6.111 Final Project Proposal
FPGA Digital Synth

Sarah Pohorecky

November 5, 2018

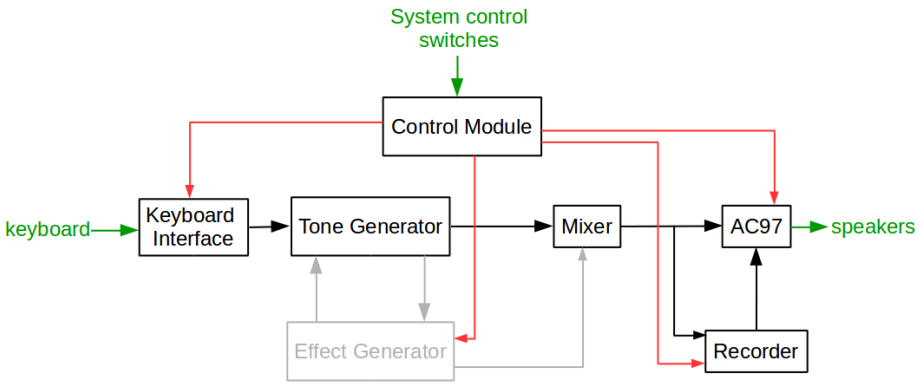

Figure 1: General System Diagram. Control Signals: Red. Direct IO: Green. Data: Black. Extension: Grey.

1 Interface

1.1 Keyboard

The main input for the synthesizer will be a 2-octave keyboard. This will either be a premade keyboard (which will likely require interfacing via MIDI), or simply 25 indivudual wires for each of the 25 notes in the two octaves centered at middle C. An "octave select" feature will allow the keys to be shifted to up or down octaces, so the user can play the full set of notes on a 5-octave keyboard.

The keyboard input will be transformed by the octave select value, and used as an index into a lookup table which will give a "tuning word" for the frequency of each note. The interface will output 25 of these 8-bit tuning words and send

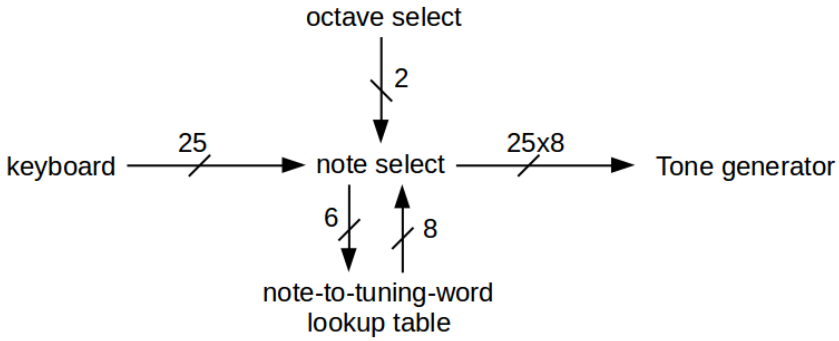

Figure 2: Keyboard Interface

them to the tone generator. This lookup table will require $61 \times 8 = 488$b to store mappings for all 61 possible notes.

1.2 Settings

The synth will use additional buttons and switches for auxillery system settings:

- **Volume**: Two labkit buttons will be used for volume input to the AC97
- **Octave Select**: Two buttons will be used to shift the octave of the keyboard to a higher or lower register
- **Instrument Select**: Will be a set of switches to select the correct number for different instruments sound effects
- **Recording**: Switch to start recording a new track
- **Playback**: Button for playback, switches for selecting which track to play back, option to clear tracks

2 Tone Generator

While a simple square wave generator may be used for initial testing of certain modules, it will be replaced with a sine wave generator that will allow mixing and more interesting audio effects.

Sine wave generation will use a CORDIC (Coordinate Rotation Digital Computer) algorithm. CORDICs function as successive approximators that only use addition and bit-shifting, instead of multiplication, making them reasonable to implement on FPGA. A CORDIC will also be more memory-efficient than just using LUTs for sine generation, as the CORDIC will be able to generate any

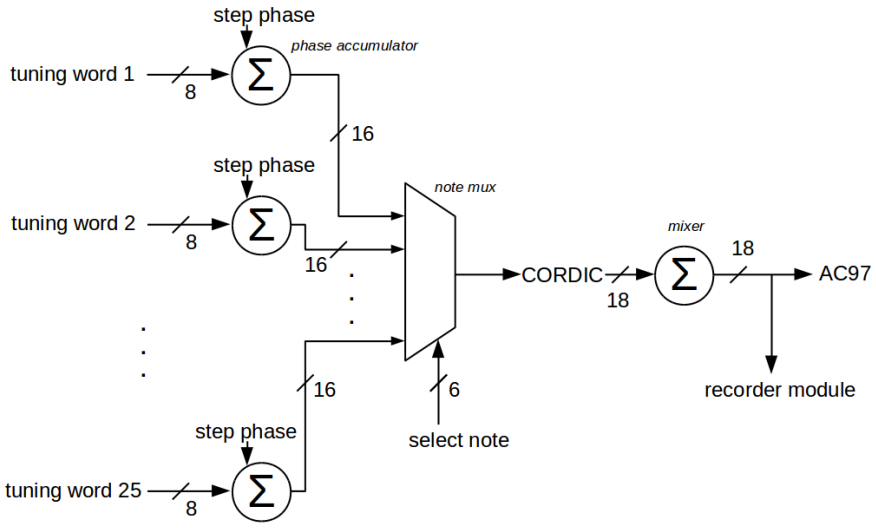

Figure 3: Tone Generator and Mixer

frequency on-demand, whereas individual LUTs would have to be built for each desired freqeuncy without the algorithm. An N-iterative CORDIC requires only N+1 pre-computed values. CORDIC is not a fast algorithm, taking about 40 cycles to converge reasonably for each desired angle, however the AC97 audio interface clocked at only 48kHz, much lower than the labkit FPGA in the MHz range, the convergence time shouldn't be a problem. However, if time allows, a lower-latency version of the CORDIC can be implemented. The CORDIC requires an input angle to generate a sine sample value. This angle comes from a phase accumulator set up for each note. The phase accumulator can be thought of as stepping around the unit circle at a rate consistent with the desired frequency. The tuning word passed to the module by the keyboard interface is the step size. Each phase accumulator is made up of a 16b counter which increments by the tuning word each time step is high and overflows when it reaches its max value. This behaviour is desirable, as it emulates the periodicity of trigonometric signals.

Since having multiple CORDICS would overly complicate the system, each output of the phase accumulator will be serially piped into the CORDIC module and its output (the sine value of the given angle) will be fed into the mixers running sum, which will be the input to the AC97 when all signals have been

processed. Most of these signals will be zero at any particular time (unless the user is keyboard-smashing), so if signals that are zero are ignored, the CORDIC should be able to process every angle within the 48 kHz clock rate of the AC97. However, if all the keys are being pressed at once, the single CORDIC may not be sufficient. If the serial CORDIC is insufficient, multiple CORDICs could be used in parallel, though there may be some tradeoff with memory. If the tradeoff isn't terrible, parallel CORDICs can be used even if the serial version is sufficient.

2.1 Instrument Generator

While simple tones corresponding to note frequencies will suffice to play music, in order to sound "musical," most people require more complex and rich sounds. In electronic synthesizers, there are several ways to do this. One is to adjust the ADSR (attack-delay-sustain-release) envelope of the sound, which changes the amplitude of a signal over time to emulate the sound characteristics of physical instruments (striking a key, and dying away, for example). Another is by adding frequency or phase modulation to the signal to emulate the tibre, or tonal quality, or the instrument and adding harmonic signals. This will likely be one of the more mathematically-intense modules, so it will need to be designed using methods that work well with FPGA (for example, minimizing multiplications and dealing with latency issues). It will take as input a note and a selection of an instrument or effect which will correspond to some logic implementing the setting. It will output a waveform that will be passed to the mixer or to the audio output.

This module will be implemented once the basic synthesizer is functional.

3 Mixer and Audio Output

In order to make more interesting music, the synth should be able to mix different tones together, allowing for additional interesting audio effects, the playing of chords, and the eventual overlay of different audio tracks. In many cases, mixing is as simple as adding together the multiple waveforms that need to be produced. More complex mixing will be an extension.

The labkit AC-97 chip will be used for audio output. This project will likely require utilization of more complex settings for the chip then were used in lab 5, so the provided AC97 modules will be used as a starting point and extended to provide neccesary functionality (for example, as may be needed to help implement the ADSR enveloping). The signal to the AC97 will be downsampled from 48kHz to 12kHz (4 times downsample), with Nyquist frequenct of 6 kHz. Since the maximum note frequency that will be generated is C_6 at 1046.502 Hz, this should not result in loss of signal. Full 18b samples will be used throughout the system in order to preserve fidelity as much as possible.

4 Recorder

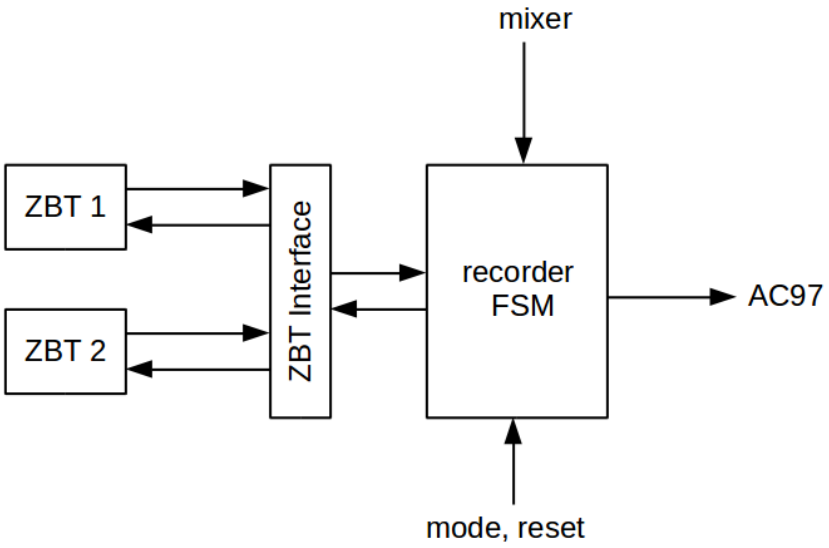

Figure 4: Recorder

The recorder will allow users to record and playback tracks. Audio output from the mixer to the AC97 will also be sent to the recorder, and it will record to the ZBT memory when record is enabled. The module will also be responsible for sending recorded tracks back to the AC97 when playback is enabled. Each ZBT memory is 512k×36. Since each sample sent to the AC97 will be 18b, two samples can be stored at each address of the ZBT (helped along by the ZBTs word write signals). Then, the total number of samples that can be stored in memory is: $512\text{k} \times 2 \times 2 = 2048000$ samples. At 12 kHz, this allows for 170 seconds of audio to be recorded.

5 References

Andraka, Ray. *A Survey of CORDIC Algorithms for FPGA Based Computers.* 1998.

Chowning, John M. *The Synthesis of Complex Audio Spectra by Means of Frequency Modulation.* Stanford Ariticial Intelligence Laboratory. Stanford, CA. 1973.

Mehra, Rajesh; Kamboj, Bindiya. *FPGA Based Design of Digital Wave Generator Using CORDIC Algorithm.*

Murphy, Ava; Slattery, Colm. *All About Direct Digital Synthesis.* Ask The Application Engineer. Analog Devices. 2004.

Oppenheim, Alan; Willsky, Alan. *Signals & Systems: Second Edition.*

Volder, Jack E. *The CORDIC Trigonometric Computing Technique.* The Institute of Electrical and Electronics Engineers. 1959.