Modular Synthesizer

Tejasvi Vishwanadha
Andrew Muth
Michael Miller

Summary

A modular synthesizer is a musical instrument that produces sounds by routing independent modules together. Modules such as oscilla

Modules

Audio Modules

Oscillator

The oscillator is one of the basic building blocks of the modular synthesizer. It produces a variety of simple waveforms given a frequency input. The 'type' input selects between pulse, triangle, saw, ramp, and sine waves. Using the 'sync' line, two oscillators can be hooked together so that they produce synchronized waveforms to prevent beating or create strange effects. The 'width' input allows for precise control of the pulse widths when using the pulse waveform type.

Filter

The filter is another basic module in the synthesizer. It provides a one-band parametric equalizer filter with four different modes of operation: high pass, low pass, band pass, and notch. The 'q' input controls the Q ("quality factor") of the filter, and 'level' determines how much the signal is boosted or cut at the specified 'cutoff' frequency. Equalizer filters can be used for many purposes, ranging from noise and rumble removal by cutting the unwanted frequencies to tonal enhancement by emphasizing desired frequencies.

Sequencer

The last of the core modules, the sequencer is a modifiable list of numbers. Its output produces the next number based upon the 'speed' input. Normally, this is just the next number sequentially, but the 'random' switch makes the sequencer produce a random element from the list. As well, the 'end' input controls whether the sequencer stops once it has traversed the list, loops, or loops in reverse. This setting applies mainly when the 'random' switch is off. The sequencer module can be used to pass a series of frequencies to an oscillator, for example, to produce arpeggios or melodies. It could also be used to control another module and producing interesting effects such as a shifting filter whose cutoff frequency keeps moving.

Slew Limiter

The slew_limiter module simply limits the rate of change of a signal. The 'amount' of slew controls how spread out the input signal becomes. If the input signal were a sequence of frequencies and the output was connected to an oscillator, the resulting sound would slide from specified frequency to specified frequency rather than having a sharp transition. As well, the 'direction' input controls whether the spreading is applied only in the upward direction, downward direction, or both.

Amplifier

The amplifier module allows precise control over the amplitude of an input signal. Besides simply increasing or decreasing a waveform's volume ('level'), the amplifier can also be used to modulate the signal's amplitude via the 'control_in' signal and the 'control_level.' This allows for a variety of interesting effects like vibrato using a sine wave control signal or a signal that fades in or out using a ramp or saw wave.

Envelope

The envelope module manipulates the attacks, sustain, and release of an input pulse-like signal. The 'attack' input specifies the amount of time after the input change before the value reaches the new value. This functionality is very similar to the slew_limiter. Once the 'attack' duration has passed and the signal has reached the input value, over the next 'decay' milliseconds, the signal will slowly decay to the sustain 'level' value. While the input pulse remains high, the output persists at the sustain 'level.' Once the input returns low, the output decays for 'release' milliseconds until it reaches the low value. envelopes are used primarily as control signals for other modules such as the amplifier. It allows a signal to have a harder (or tonally different) attack from its sustained sound, and then have a smooth fade out or release trail.

Noise

The noise module creates one of three 'type's of noise: pink, white, or low frequency. Noise can be used to add randomness to a signal or to add some grit to a sound.

Ring Modulator

The ring_modulator is the simplest of the modules in the synthesizer. It multiplies two signals in the time domain. This effect can often produce bell-like tones with interesting overtone structures since this operation is equivalent to a convolution in the frequency domain.

MDelay

The delays module produces a delayed copy of the input signal 'delay' milliseconds later, mixed with the original signal based on the 'wetdry' percentage. Wet refers to the delayed signal, while dry is the input signal. The 'level' of the delayed signal can also be controlled, as well as how much the delayed signal feeds back into the original ('feedback'). This is a common tool in audio design and production. It often is used to provide a sense of "space," but is also an effect.

Reverb

Like delay, reverb is used to create space. The most realistic digital reverbs to date use convolution. Given the impulse response for a space, one can place a sound in that space simply by convolving that signal with the impulse in the time domain. The reverb plug-in includes a variety of preset impulses to choose from. The modules also provides the 'time' input to control how long the reverb trail lasts, and the 'wetdry' to control the mixing of the reverb and the input signal.

Pitch Corrector

A specialty, complex module, the pitch_corrector takes an input signal and snaps its detected frequencies to the pitch classes 'allow'ed and ignores those in 'bypass.' The 'retune' input controls how quickly the pitch is adjusted to its corrected value. Pitch detection will be accomplished using either an FFT or autocorrelation, depending on which method seems to generate better or faster results. Pitch shifting will then be done using a time-independent method. This module is the most complex of any in the synthesizer.

Vocoder

The vocoder is another complex module. It modulates a ‘carrier’ tone using the frequency spectrum of a ‘modulator’ signal. This can be used to create effects where a synthesizer tone seems to “speak” when it is modulated by a voice singing or talking.

N-to-1 Mixer

The mixer module simply takes N inputs and produces one output, combining the inputs at the given input ‘level’s. This module allows two signals to be combined in an additive fashion and balanced against each other.

Control Modules

Module Control Interface

All of the audio modules in this proposal require a number of control values that specify (within their respective ranges) the parameters of their operation. Using the oscillator module as an example, the module would need control values for the waveform type (sine, saw, square, etc), frequency, amplitude, input-output control (VCO, etc). We are currently planning to implement this required functionality in a manner similar to how most peripheral chips work, namely by having each module have a parallel control interface and a set of addressable internal registers. The values in these registers would have a default startup state and will control all of the parameters mentioned above. To accommodate the required precision the input data stream will be 16 bits wide, with each state register being 16 bits wide as well -- thus, each audio module will have a (n x 16) bit memory footprint for control registers. The number of address lines will be determined on a case-by-case basis as different modules will require different numbers of control registers.

Beyond this data interface, each audio module will have 3 control signals which govern the module's behavior. A <SET> strobe will be used to write the value presented on the <INPUT_DATA> port to the register at location <ADDRESS>. A <RUN/#STOP> signal will control if the module is currently generating a processed output or simply acting as a "proxy" for the input data stream. Finally, a <BYPASS/#MUTE> signal will select the output behavior in the #STOP state -- if BYPASS is selected, the input is simply copied to the output; if #MUTE is selected the output will be 0x0 for all input values.

Master Controller

As mentioned above, all of the audio processing modules will require initialization and control signals to determine their output behavior. However, this information will be coming from an external source -- initially a PC terminal over RS-232, and later from a dedicated U/I unit (still over RS-232). Choosing to implement an external control interface using the simple serial protocol will minimize the amount of effort and testing/debugging needed to get the audio units up and running while still allowing the possibility for "drop-in" U/I units later on.

Internally, this master controller module has 3 separate subsystems:
- RS-232 serial parsing Finite State Machine
- shadow memory buffer of all audio module's control registers
- control interface implementation

In a nutshell, this module will parse the RS-232 data stream and act on its commands to update an internal memory array which shadows all of the control registers in all of the instantiated audio modules. When commands come in to change a particular module's control settings, those updates will be written to the shadow memory array which represents the state of the entire synthesizer. Independently, the control interface subsystem will see that a shadow memory value has changed and update the appropriate audio processing unit.

User-Interface Controller

On a real analog synthesizer, all of the control values mentioned above would be controlled by a large variety of analog knobs, sliders, and switches which would set and display the current system state. However, in this project's implementation using such an analog interface would require many ADCs and expensive analog components, so we will be using instead a variety of digital inputs and a small number of analog data sources which are multiplexed to control any selected audio module.

Digital input will occur using a bank of 8 digital switches and 7 input buttons. We will also use a pair of analog potentiometers arranged as a 2-axis (XY) joystick with 10-bit precision to get precise analog control. The joystick pots are connected to an SPI-based ADC with 10-bit precision and a well-defined control interface. Therefore, using this controller will require implementing
an SPI controller in the FPGA.

Beyond these inputs, our major source of input will be a resistive analog touchscreen mated to our primary data display, a 4.3" LCD panel. The touchscreen is read using an Analog Devices AD7843-ARU-Z controller, which speaks a simple TTL serial dialect to provide press locations and an interrupt output. The LCD (480x272) is driven using a simple digital interface at a low clock frequency (<10 MHz). These parts are already within our personal inventory as well.

Depending on the issues seen during testing and debugging of our LCD/touchscreen control panel, we will balance the amount of control signals generated by this unit versus the labkit's hardware (aforementioned switches, buttons, joysticks). However, preliminary investigation suggests implementing this system using the LCD and touchscreen should not pose a substantial challenge.

Inter-Module Dynamic Router

To allow for dynamic intra-synthesizer module routing behaviors, we will attempt to implement a ring-buffer with a 16-bit data bus and n-bit address bus, where n is the number of address lines needed to address all of the audio modules. Each audio module will need a "wrapper" module which implements a simple state machine to load only the appropriate input data value while passing along the remainder.

Special cases will be implemented for modules which may take in multiple data packets (for example, adders or subtractors).

MIDI Interface Controller

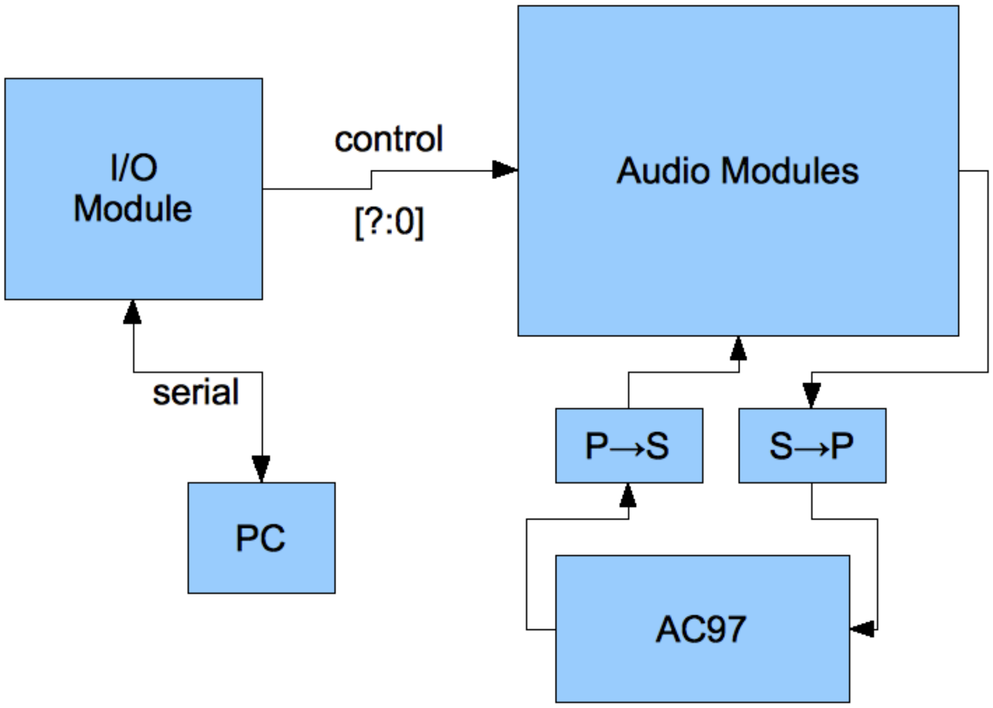
In the unlikely situation that all of the above systems are implemented and operating correctly with time to spare before Demo-Day, a MIDI input interface will be constructed to allow basic commands from a MIDI source (for example, a keyboard) to drive an oscillator within the synthesizer. This MIDI implementation will only look for a very small subset of the MIDI specification's commands (currently NOTE-ON and NOTE-OFF), as these are the only commands which apply to the synthesizer's functionality.

Misc Modules

Serial to Parallel/Parallel to Serial

Data to and from the AC97 module and Composite Video Module will probably be passed as parallel bits of data. In order to use this as input and output to our audio modules, we will need to convert them into in serial bits and then back from serial bits.

Block Diagram


I/O Module
control
[?:0]
Audio Modules
serial
PC
P→S
S→P
AC97

Division of Labor/Priorities

Mike and Teja will be working on the audio and miscellaneous modules while Andrew will handle the input/output modules. Teja will work with Andrew to implement the ring buffer. Our first priorities are to implement the core audio modules and an interface to control them, consisting of the master controller and dynamic router. These components take the highest priority because they are required for early-stage testing and will allow Mike and Teja to work on the complex audio systems while Andrew works independently on the U/I. After this core functionality is achieved, the audio team will focus first and foremost on implementing the full set of audio modules. Similarly, Andrew will work on building the tactile (non-PC-based) user interface and display system using his LCD, touchscreen, and digital I/O. As more audio components are added, the user display will be updated and the master controller will be connected to them. By building on our core "required" functionality in two independent paths we will avoid blocking issues due to one group's systems depending on the other's being fully up and running.

Should we be able to implement all of these modules without running out of time, the audio team will begin work on the vocoder and pitch correcter, while Andrew will begin implementing a MIDI I/O interface. We consider the those last two audio modules "reach goals," since they are extremely complicated and difficult. Our hope is to include at least one before Demo-Day.