FPDJ

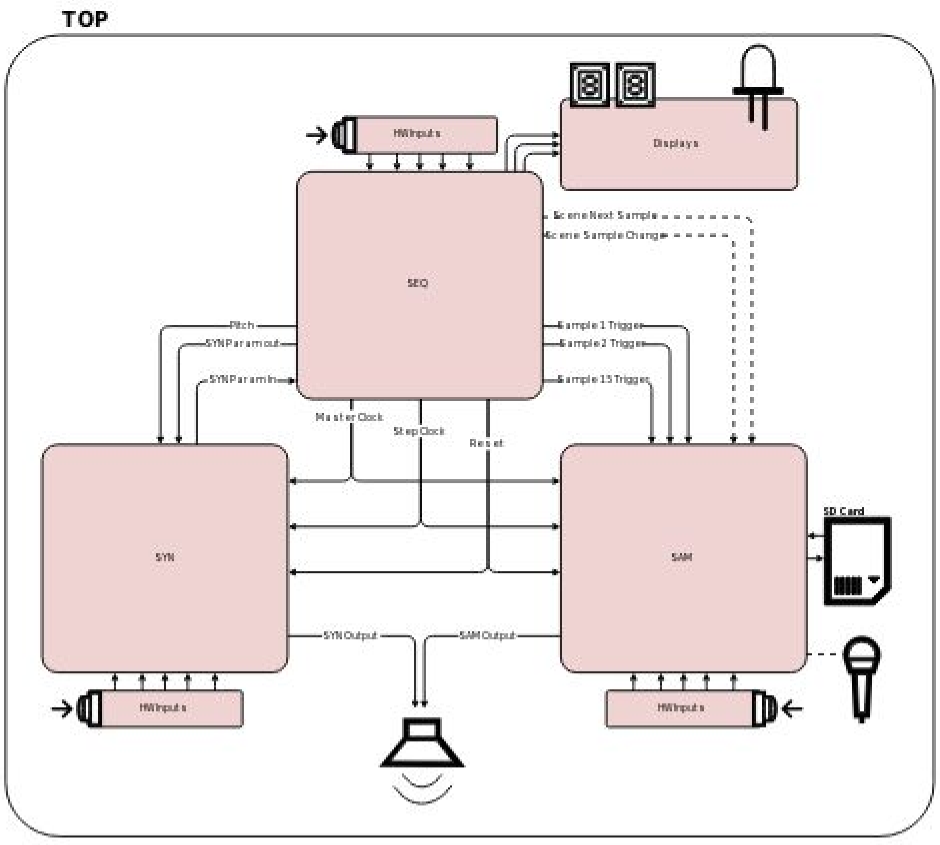
Baltazar Ortiz, Angus MacMullen, Elena Byun

Overview

As electronic music becomes increasingly prevalent, many listeners wonder how to make their own music. While there is software that allows musicians to create songs, the tangible experience that a physical instrument provides is often much more conducive to the creative process. Our project, FPDJ, is an all in one device for creating electronic music.

FPDJ provides users with a hardware interface to compose musical patterns, play sounds from an SD card, and generate other sounds in real-time.

Design

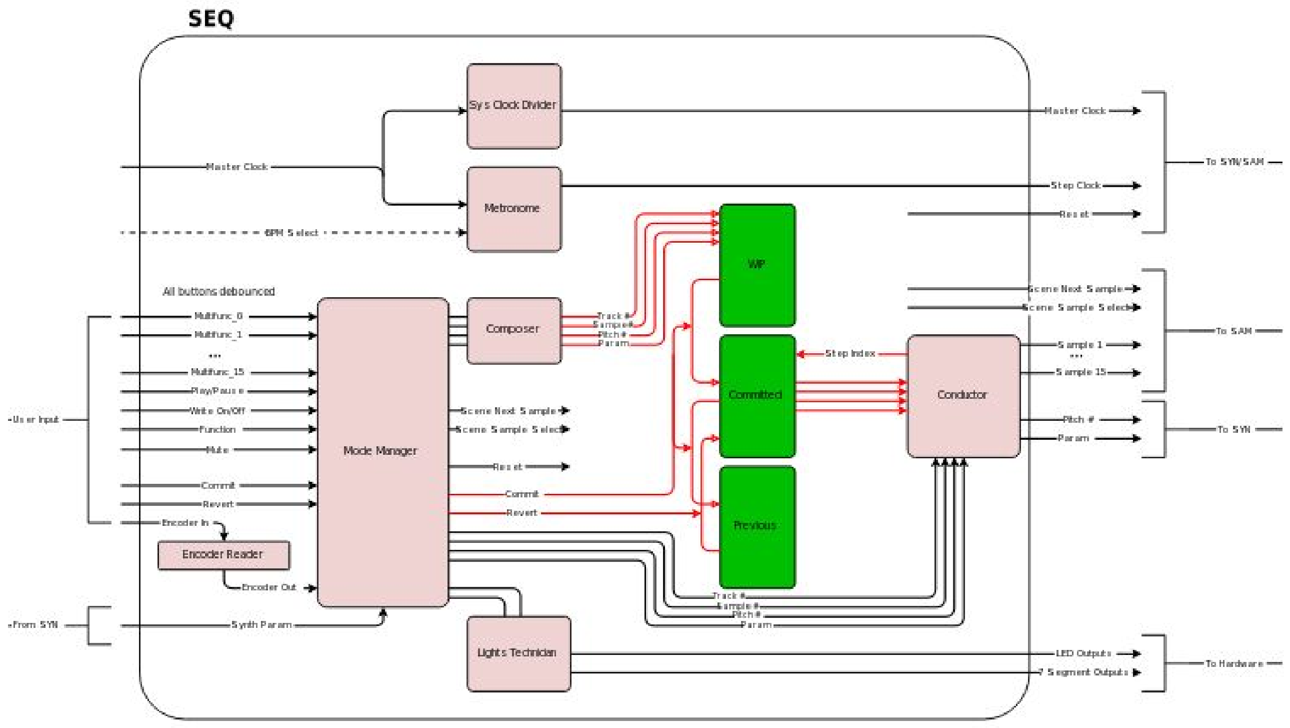


The FPDJ system is divided into three core sections: the Sequencer, the Sampler, and the Synthesizer. The Sequencer controls automated patterns and provides a master clock and a divided beat clock to the other two sections. The Sampler plays samples back from the SD card, and the Synthesizer generates melodic sounds in real-time. Elena will be responsible for the Sequencer, Baltazar will work on the Sampler, and Angus will develop the Synthesizer.

The Nexys 4 board sets a few limits for the design. The onboard RAM size limits the amount of audio that can be loaded for playback, and multiple modules trying to access the RAM at once may result in a slowdown. Also, the FFT IP core to be used in the Synthesizer will take up an appreciable amount of FPGA area.

Implementation

SEQUENCER



The Sequencer module handles the composition of rhythmic and melodic patterns. Using switches and sliders, the user sets musical samples and synthesized notes to trigger at different points in a sixteen-beat loop.

The Sequencer has the following inputs: system clock, 16 multi-function buttons, play/pause button, write on/off button, commit button, revert button, function button, mute button, a rotary encoder, and a synth param in.
The Sequencer has the following outputs: reset, master clock, step clock, SYN param out, SYN pitch out, SAM trigger out, SAM scene change sample select, SAM scene change next sample, 16 LEDs, and several 7 segment displays.

The large submodules of the Sequencer are: sys clock divider, metronome, rotary encoder reader, key debouncer, mode manager, composer, conductor, and lights technician.

Sys Clock Divider

The sys clock divider takes the fast system clock and divides it into a slower master clock to be used in all modules. The divider is fixed.

Metronome

The metronome receives the master clock and divides it into an even slower clock, which is referred to as a "step clock". The step clock is then used to determine the timing of sample and synth triggers. A reach goal is to make a metronome module that can generate step clocks for a range of bpms depending on the user input.

Rotary Encoder Reader

The rotary encoder reader reports the encoder count for a quadrature encoder.

Key Debouncer

This module adds debounce to function, mute, commit, and all multifunction buttons.

Mode Manager

The mode manager, implemented as an FSM, handles all user inputs to the system. The mode manager should perform the following:

```
If (function && multi_15)
        RESET: Send reset signals to SAM and SYN, clear memory.
If (function && multi_14)
        TRACK CLR: Wait for a track to get selected, then clear that track.
If (mute && any of multi_0~7)
        TRACK MUTE: Toggle mute/unmute of that track.
If (function && multi_13)
```

```
        SCENE CHANGE: Wait for a sample to get selected, then use encoder data to change the stored sample in SAM module via SAM scene change wires.
If (commit)
        COMMIT: Push Work in Progress memory to Committed memory, push previous Committed memory to Previous memory.
If (revert)
        REVERT: Push Committed memory back to Work in Progress, push Previous memory back to Committed memory.
If (write)
        Toggle write_on
If (play)
        Toggle play_on
If (write_on)
        If (~play_on)
                Multi_0~15 are step selectors
                If selected track is a sampler track, encoder data is used to scroll through 15 samples + no sample
                If selected track is a synth track, encoder data is used to scroll through pitches + no pitch
                Use Composer to save track #, step # and (SAM sample # or SYN pitch + param) to Work in Progress memory
                Display on 7 segment: track #, step #, sample #, pitch, param using Light Technician
                Display on LEDs: If sampler track, color-mapped samples saved in track WIP; if synth track, color-mapped pitch of LEDs in track WIP – both use Light Technician
                Display (write_on, ~play_on) using Light Technician
        Else
                Do everything in (~play_on) while also playing back Committed by exciting SAM and SYN using Conductor
                Display (write_on, play_on) using Light Technician
Else
        If (~play_on)
                If selected track is a sampler track, multi_0~15 are sample selectors. Whenever multi_0~15 are pressed, SAM is activated using Conductor.
                If selected track is a synth track, multi_0~15 are pitch selectors. Whenever multi_0~15 are pressed, SYN is activated using Conductor. Encoder data can also be used to switch octaves of pitches.
                Display on 7 segment: track #, step #, sample #, pitch, param using Light Technician
                Display on LEDs: different color for all 16 LEDs using Light Technician
                Display (~write_on, ~play_on) using Light Technician
        Else
                Do everything in (~play_on) while also playing back Committed by exciting SAM and SYN.
                Display (~write_on, play_on) using Light Technician
```

Composer

The composer has the following inputs: track #, step #, (SAM sample # or SYN MIDI pitch # + param)
The composer has the following outputs: WIP memory
The composer module is used by the Mode Manager to edit the WIP memory.

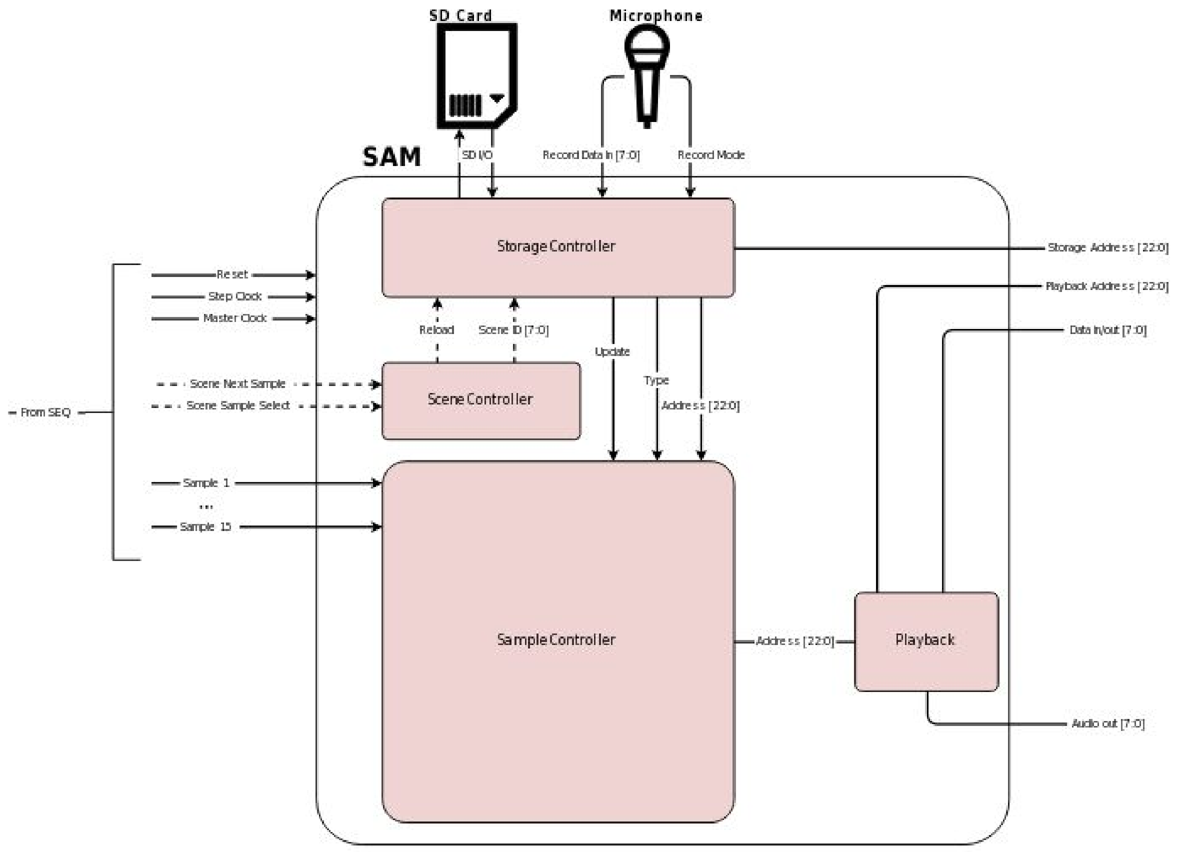
Conductor

The conductor has the following inputs: step clock, Committed memory
The conductor has the following outputs: SAM 15x trigger out, SYN pitch out, param out
The conductor module is used by the Mode Manager to activate SAM and SYN using the Committed memory.

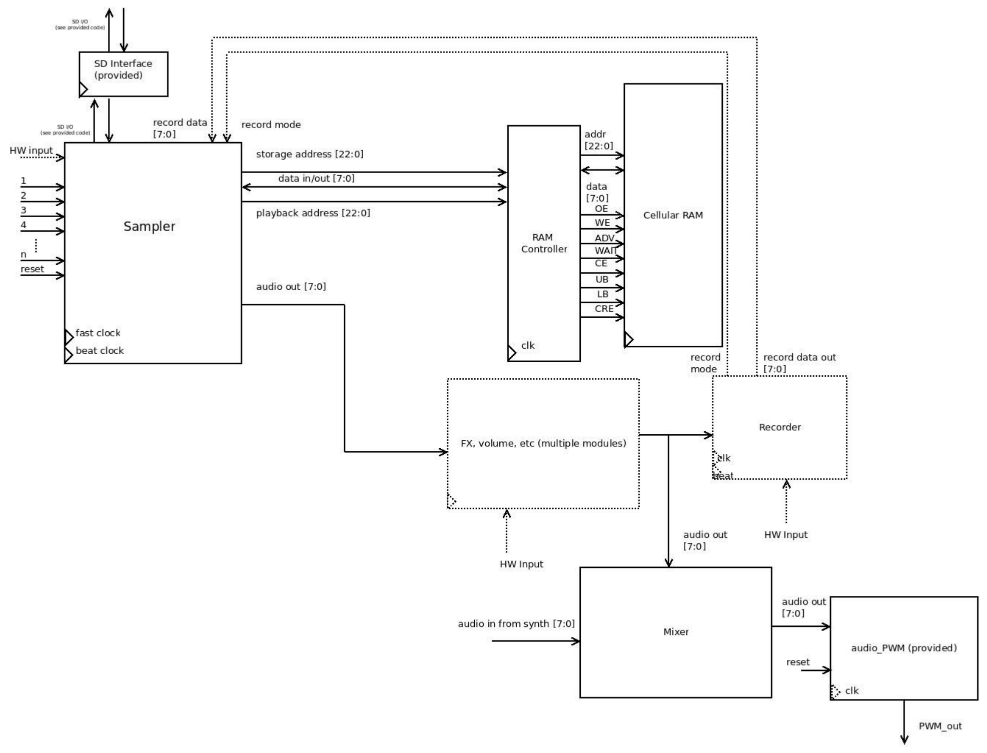
Lights Technician

The lights technician handles all output displays. The mode number, the track number, current bpm, sample number, current beat number, and the synth pitch are all displayed on the Nexys 4’s 7 segment displays. RGB LEDs also display track, pitch, sample, and Committed or WIP array information. Other LEDs are used to indicate play_on, write_on, mute key pressed, function key pressed, etc.

SAMPLER



(dotted lines represent reach goals)



(shown with auxiliary modules)

The Sampler handles the loading and playback of recorded audio clips. It retrieves audio files from the SD card on the Nexys 4 board and stores them in the board's RAM. The Sampler is connected to the Sequencer through a 15 bit wide parallel connection, providing a "scene" of 15 samples at a time. In addition, it receives a fast master clock and a slower beat clock from the Sequencer. When the Sequencer triggers a sample through the parallel connection, the sample is read from the RAM and the audio data is sent out of the Sampler to the mixer.

The Sampler consists of a sample controller, a storage controller, and a playback module. Implementing a scene control module is a reach goal.

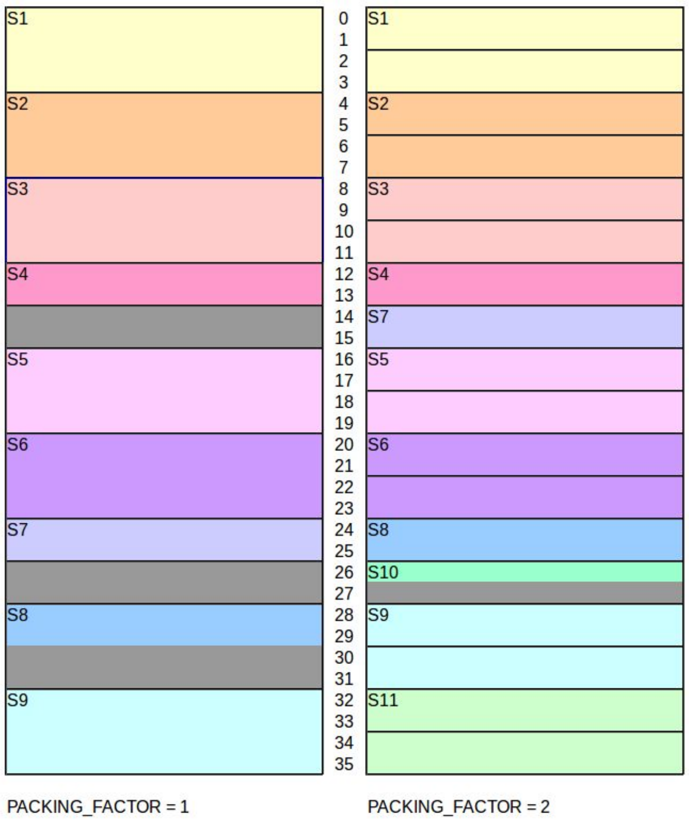
Storage Controller

The storage controller provides an interface between the SD card and the RAM. It utilizes the Nexys 4 SD interface provided on the course website. WAV audio files can be detected through their file headers, and the additional metadata is stored either in the file header or ID3 tag. The storage controller stores samples from the SD card into the RAM using a "slot" memory organization (shown below), and sends the start address and category of each sample to the sample controller. If the scene controller is implemented, the storage controller will take input from the scene controller to decide which samples to load from the SD card. The storage controller is implemented as a FSM with a submodule to utilize the provided SD interface.

The storage controller organizes samples into a “slot” system when storing them in RAM, trading some space efficiency for increased simplicity during playback.

As a reach goal, sample storage will be controlled by a PACKING_FACTOR parameter, which increases the efficiency with which the RAM is utilized at the cost of potentially slower scene load times. The packing factor must be a power of two, and is limited by the maximum sample length that is defined for the Sampler.

In the below diagram, the effects of PACKING_FACTOR = 1 (same as the basic goal behavior) and PACKING_FACTOR = 2 are demonstrated. Assume that the maximum size of each sample has been defined to be 4 memory locations long and that the SD controller reads samples in order from S1 to S11.



The bright colors in the diagram correspond to space used by actual audio data, and the gray represents space that is wasted due to the slot organization. Note that for PACKING_FACTOR = 2, the four location long sample slots are further divided into two two location long half slots. When shorter samples such as S4 are loaded, the storage controller remembers that there is still half a slot left for another short sample such as S7. This avoids some of the memory waste that occurs for PACKING_FACTOR = 1.

Sample Controller

The sample controller translates input from the Sequencer into memory addresses that are sent to the playback module.

As described above, when the storage controller loads samples into RAM, it also sends the address of the first memory location of each sample to the sample controller. Using this information, the sample controller associates each input wire with the start address of a sample.

When a single wire on the input bus is asserted at the rising edge of the beat clock, the sample controller sends the corresponding start address to the playback module. If multiple samples are triggered at once, the sample controller sends one address per fast clock cycle to the playback module. The sample controller is implemented as a FSM.

Playback

The playback module handles reading samples from the RAM and mixing their sound data together to create a final audio output for the Sampler. The playback module consists of a FSM and submodules to handle retrieving multiple samples at once from RAM and mixing the output data. If multiple samples are triggered at once, the sample controller sends one address per clock cycle, as described above. The playback submodule stores the addresses as they come in and cycles between the addresses to retrieve data from each sample that will then be mixed together for the final output.

Scene Controller (reach goal)

The scene controller module allows for the switching of sample sets during runtime. Based on a hardware interface, the user will be able to switch the set of active samples. The scene controller interfaces with the storage controller to send it a reload signal and the new scene metadata. Once implemented, the initial information about which samples to load will also be moved from the storage controller to the scene controller. The scene controller will be implemented as an FSM and will use hardware input to trigger scene loads.

RAM Controller

The RAM controller module abstracts reading and writing to the Nexys 4's RAM into a multi-port interface so that the Sampler and recorder can access memory without conflict.

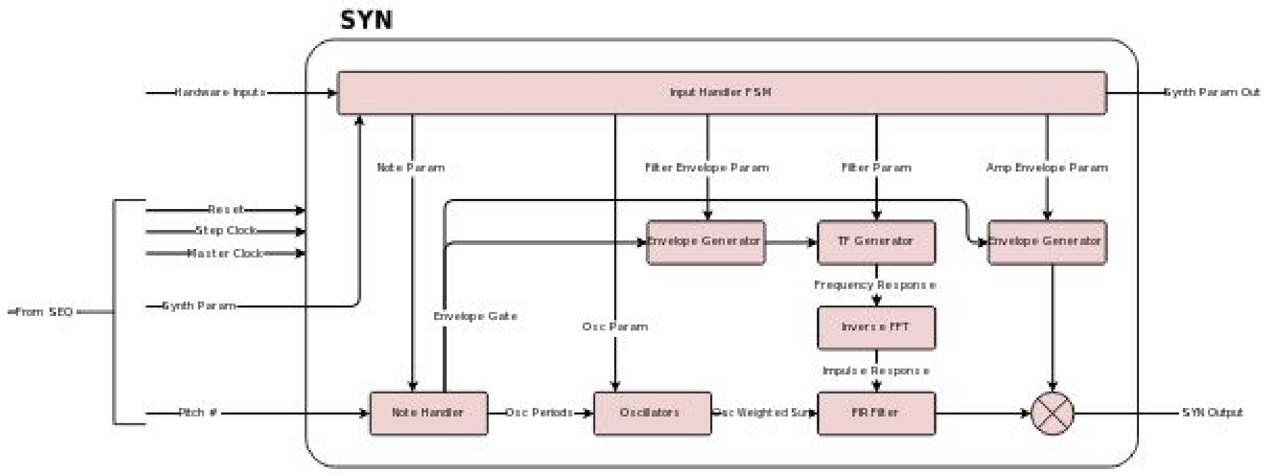
Audio Output Processing (reach goal)

A number of possible modules can be implemented to process the output of the Synthesizer and Sampler given enough time. Possibilities range from basic volume control to a variety of effects such as echo, bitcrusher, and delay.

Recorder (reach goal)

A recording module would allow the user to extend the functionality of FPDJ by recording the output of the Synthesizer and/or the Sampler (either the mixed output or one of the two) and either saving it as a new sample for further production use or continuously saving the master audio output to the SD card for playback on other devices.

SYNTHESIZER



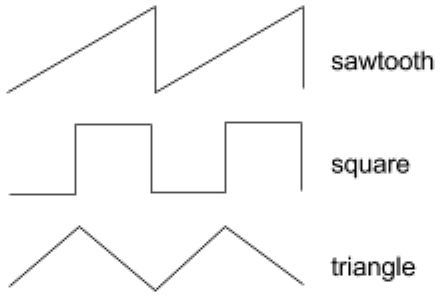
The Synthesizer is responsible for generating melodic sounds, according to the pattern written in the Sequencer. Overtone-rich waveforms are shaped with a filter to produce a wide range of timbres, modeled after typical analog subtractive synthesis techniques.

Note Handler

The note handler module receives note information from the Sequencer, as a 7-bit number using the same note mapping as the widely-used MIDI protocol. It outputs a corresponding value to the oscillators to control their pitches. The user can choose to transpose and detune the pitch of some of the oscillators, in order to build a fuller sound.

Oscillators

Each oscillator is capable of generating a tone with various waveforms: sawtooth, square, or triangle. These waveforms are chosen because they are relatively easy to generate, and are rich with overtones that can be shaped with the filter. To ensure crisp, sharp waveforms, the oscillators operate on a high-frequency clock.



An oscillator mixer performs a weighted sum of the oscillators' outputs, allowing the user to adjust the balance between various waveforms/pitches. If the sum overflows, the mixer will clip it to the minimum/maximum value, rather than wrapping around (which would produce an undesirable distortion).

The higher frequency components of the signal must be removed before the signal is passed to the main adjustable FIR filter (which operates at a lower sample rate for lower computational cost), to avoid aliasing. This is done with a fixed FIR filter stage, similar to Lab 5a, with a precomputed set of coefficients.

Reach goals for the oscillators include amplitude and/or frequency modulation between the oscillators, for additional timbral effects.

Adjustable FIR Filter

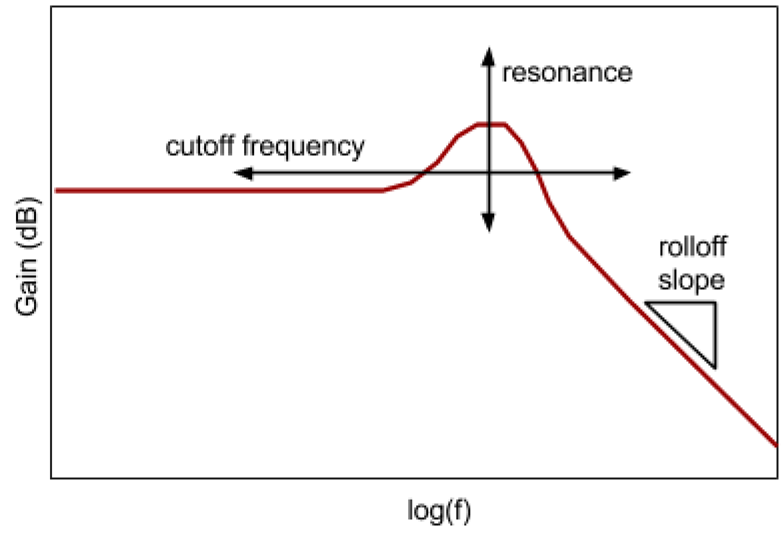
This FIR filter applies a variable impulse response to the signal generated by the oscillators. In order to affect frequencies across the audible range (approx. 20Hz-20KHz), this filter requires a longer impulse response than the fixed anti-aliasing filter.

An IFFT IP core transforms an arbitrary frequency-domain transfer function into the discrete-time impulse response used by this adjustable filter.

Filter Transfer Function Generator

This module produces frequency response parameters to be passed to the IFFT core, roughly modeled after the response of analog lowpass filters typically used in audio synthesizers. Characteristic features of these sorts of filters are: a flat response below a particular cutoff frequency, a linear rolloff (in dB gain versus logarithmic frequency) above, and optionally a peak at the cutoff frequency itself (see diagram). The timbre of the Synthesizer can be widely altered by adjusting these three filter characteristics.

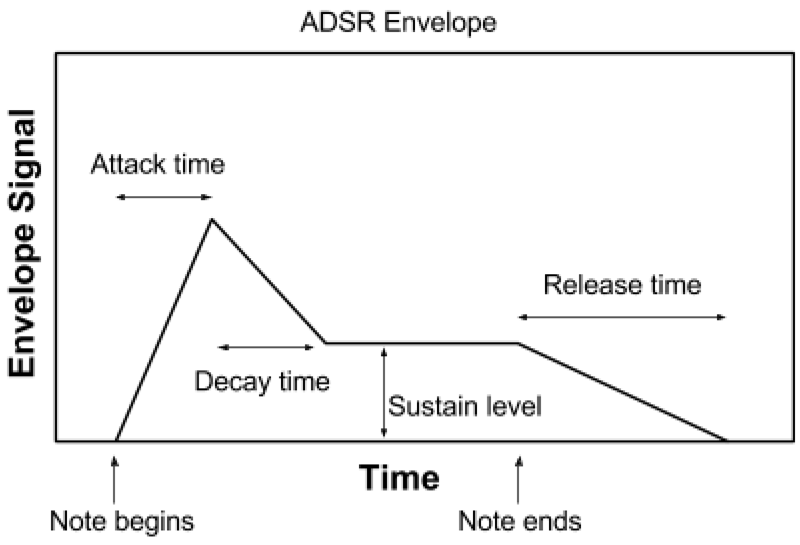
The generated transfer function will reflect this model, at minimum with a piecewise linear approximation. Reach goals include other filter types, such as highpass, bandpass, or notch filters.



Envelope Generators

The sound of the Synthesizer can be further shaped by modulating the filter cutoff frequency and output volume over time.

ADSR envelopes, which produce a transient signal based on four parameters (attack, decay, sustain, and release; see below), are often used in audio synthesizers to modulate sound over time. The envelope generator modules produce this type of signal, to control the filter cutoff and output amplitude. The effect is a transient shape to the synth’s timbre and volume.



Synth Input Handler

The parameters of the Synth are controlled by a bank of dedicated switches and potentiometers. The input handler FSM translates these physical controls to the inputs of the internal modules of the Synthesizer. Such parameters include:

- note handler: oscillator transposing/detuning
- oscillators: waveform selection, oscillator mixer levels
- filter transfer function generator: cutoff frequency, resonance, and rolloff slope
- envelope generators: attack, decay, release times; sustain level

Additionally, the synth handler relays select parameters (such as filter cutoff) to the Sequencer, so synth parameter movement can be saved into the recorded sequence. The Sequencer can play back recorded parameter movement by returning the select parameter values to the synth input handler. The input handler FSM will apply these parameter movements to the Synthesizer modules, unless the user overrides them with physical input to the Synthesizer.

Additional Reach Goals

The Synthesizer's capabilities could be further extended with other sources of modulation. Low-frequency oscillators could be mapped to pitch, filter cutoff, or output amplitude for tremolo or vibrato effects.

The Synthesizer could also be duplicated to allow for additional melodic parts. (a bassline plus a lead melody, for example). This may require the different synthesis channels to share resources such as the IFFT core or physical interface (switches, potentiometers).

Master Mixer

The master mixer module combines the outputs of the Synthesizer and the Sampler and sends it to the Nexys audio output, similarly to the Sampler's mixer submodule or the Synthesizer's oscillator mixer.

Testing

Testing for FPDJ is performed on a submodule, then major component, then system level. The Sequencer can be tested in simulation similarly to previous labs by running through its different FSM states and checking the outputs. The Sampler and Synthesizer require more testing in hardware, as they interface with external devices, such as reading from an SD card and outputting audio. However, many of their individual submodules can be verified in simulation.

Resources

In addition to the interface provided on the Nexys 4 board, the hardware interface for FPDJ will use potentiometers and buttons as input, and use LEDs to provide information to the user. We may also use an external DAC chip to improve the quality of the output audio.

Conclusion

We expect our project to be complex, but reasonable to implement in our given time frame. It will result in a fun and exciting device that could even be useful alongside other music production devices if we achieve some of our reach goals.