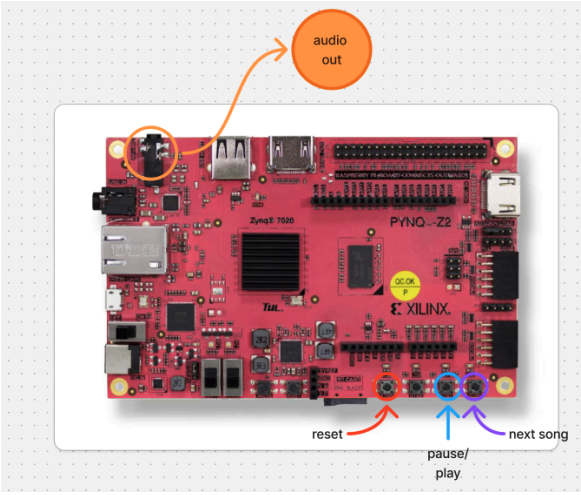
EE108 Final Report — Taylor Tam, Benji Warburton, Danica Sun

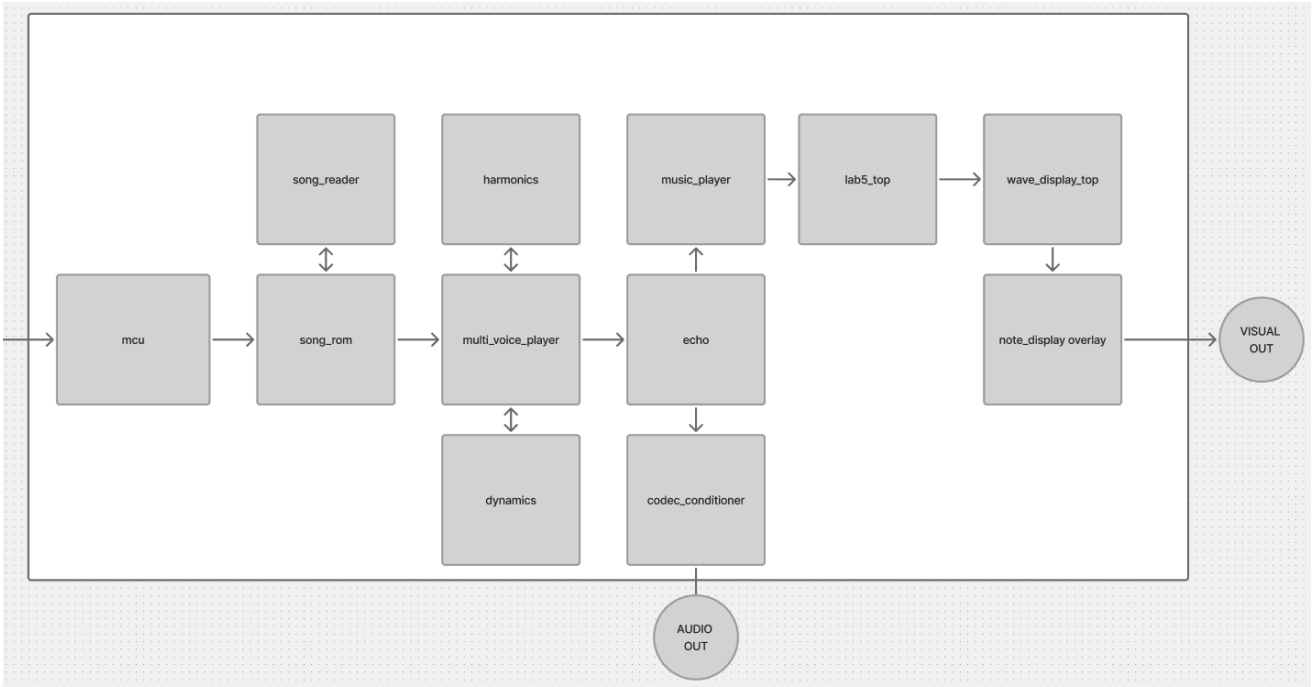
Usage

We made no major usage changes; ‘reset’ resets the system, ‘pause/play’ pauses and plays a song, and ‘next song’ advances through songs.



High-Level Implementation and Description

The MCU selects the song to be played, and the song_reader steps through the song note-by-note by fetching data from song_rom. Each note is synthesized in the audio path, where harmonics enriches the waveform, dynamics applies an ASDR envelope, and multi_voice_player combines the voices into a multi-chorded output. These are passed to echo, which adds a delayed, attenuated copy using a circular buffer. We can read songs from ROM, play up multiple simultaneous harmonically shaped notes with ASDR dynamics, and output the resulting audio to codec while displaying past, present, and future notes on a display.



Module Specifications

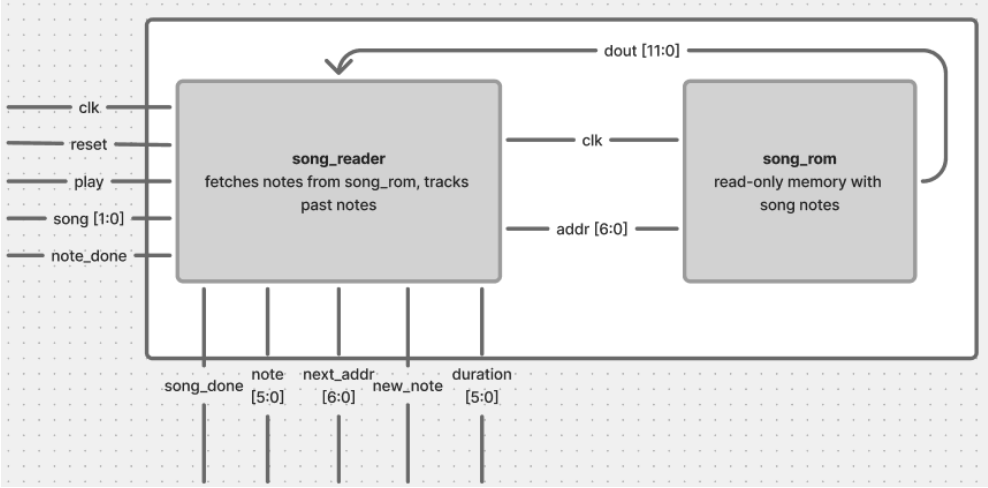
---

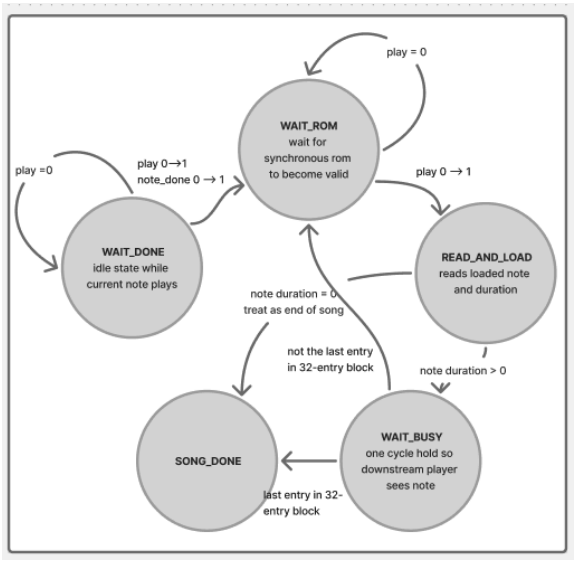
1 – Chords (4 points)

Functionality - We will allow our music player to play at least three notes simultaneously. This requires generating multiple waveforms, one for each note in the chord, and combining them into a single audio sample to feed to the codec.

Unique implementation challenges & design - A unique challenge was generating three simultaneous notes without degrading audio quality or breaking timing consistency. Since each chord requires three independent waveforms to be produced in parallel, we instantiated three note-generation paths, kept them synchronized to the same new_note, beat, and sample timing signals, and then multiplied the signed audio samples. The main difficulty was summing full-amplitude voices, which caused clipping and led to low-register chords sounding staticky. We therefore introduced multi_voice_player, which assigns each 'voice' a chord tone, attenuates the harmonies before mixing, and uses wider signed intermediate signals to avoid overflow.

Block Diagram and song_reader FSM -





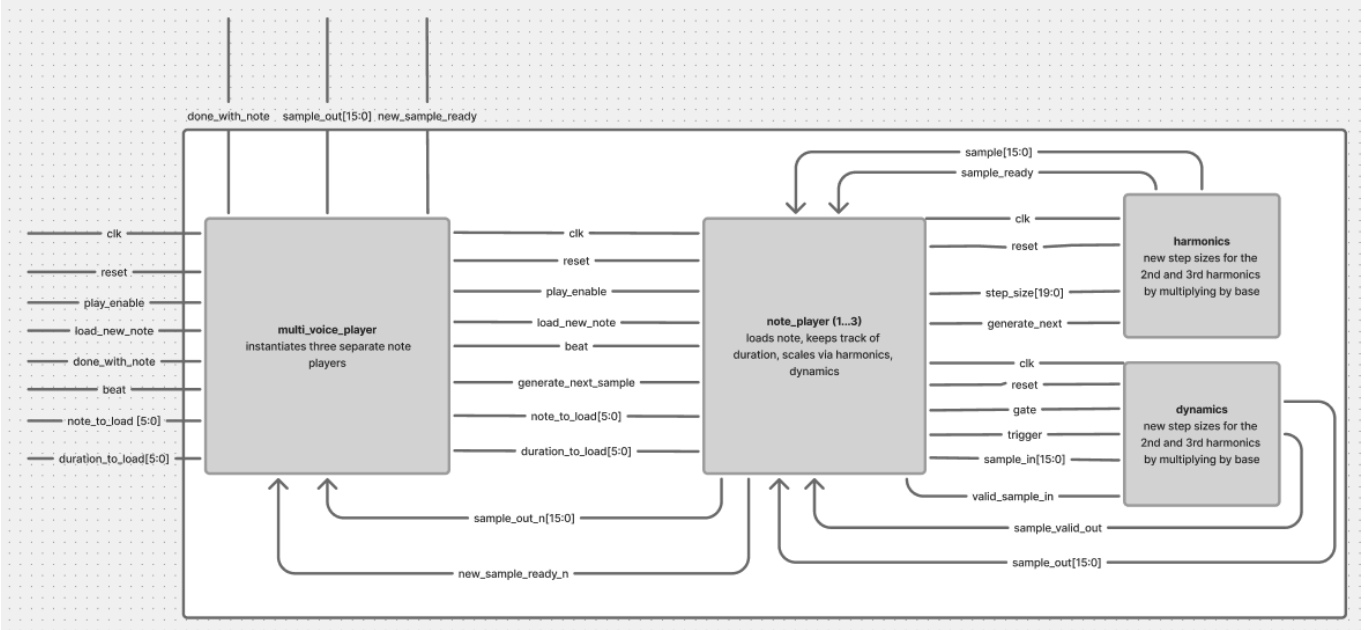
This controlled how notes were fetched from song_rom and handed to the playback path. We wait until playback is enabled and the current note has finished, we then move to wait (since song_rom is synchronous and needs 1 clk cycle to return data). We then read the ROM note, split into duration and note, and either load or transition to done (if duration is 0). We then stay in song_done until reset or a song change occurs. Therefore, we have a simple fetch-and-dispatch controller that sequences the note playback one ROM entry at a time.

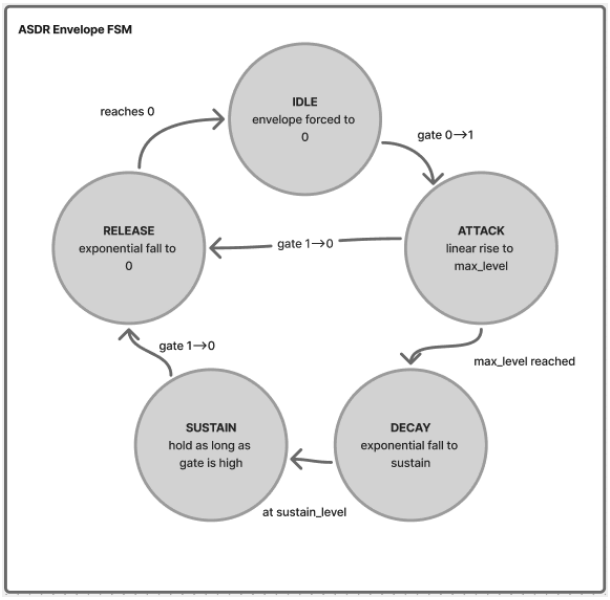
2, 3 – Weighted Harmonics (3 points) and ASDR Dynamics (4 points)

Functionality - We will add weighted harmonics (multiples of the fundamental frequency) to the note being played. Different weights will give different instrument sounds to the music. You can make a flute sound, a violin sound, a trumpet sound, or an electric guitar with distortion with the proper use of harmonics. Additionally, via dynamics, we will modulate the signal amplitude during the time a note is played. To achieve the full 4 points, we will implement a complex Attack-Decay-Sustain-Release (ADSR) envelope rather than simple exponential decay.

Unique implementation challenges & design - After loading a note, we convert the note index into a corresponding phase step size and then pass this to harmonics. Our harmonics module generates sine waves in parallel, corresponding to the fundamental and selected higher partials, and then combines them with reduced weights to shape the timbre without overflowing the audio range. Then, that waveform is passed into dynamics which applies an ASDR envelope via an FSM. A key challenge here was adapting the ASDR FSM to advance only when a valid audio sample is produced rather than on every sys clk edge. Advancing the envelope at 100MHz caused timing mismatches and sounded incorrect; by synchronizing the envelope to the sample stream, we created a smoother note shape.

Block Diagram and dynamics ASDR Envelope FSM –





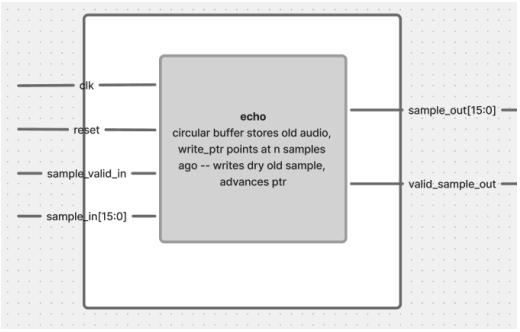
The ASDR envelope controls the note's amplitude over time. When a new note is triggered, the FSM leaves idle and enters ATTACK, where the amplitude rises linearly. It then moves to DECAY after reaching a maximum, where the amplitude falls exponentially to a sustained level, which triggers the SUSTAIN state. Once the note becomes inactive, the FSM transitions to RELEASE. In this implementation, the envelope level is multiplied by the harmonic waveform sample-by-sample, so the resulting audio has a shaped onset and fade-out rather than abrupt starts/stops.

4 – Echo (2 points)

Functionality – We'll add an "echo" to the sound being played by simultaneously playing a delayed version of the sound being fed to the output. The delay will be in the range of 100ms to 500ms and the delayed sound will be attenuated.

Unique implementation challenges & design - We implemented this as a separate module after the full synthesis and right before codec. Echo uses a circular delay buffer: one each sample, it reads an older sample from the buffer, writes in the current sample, advances the pointer, and attenuates the delayed copy, then mixes it back with the dry signal before clipping the result into a signed 16-bit value. Our main challenge was ensuring our buffer advanced on audio sample events rather than the sys clk and handled signed overflow correctly when mixing samples.

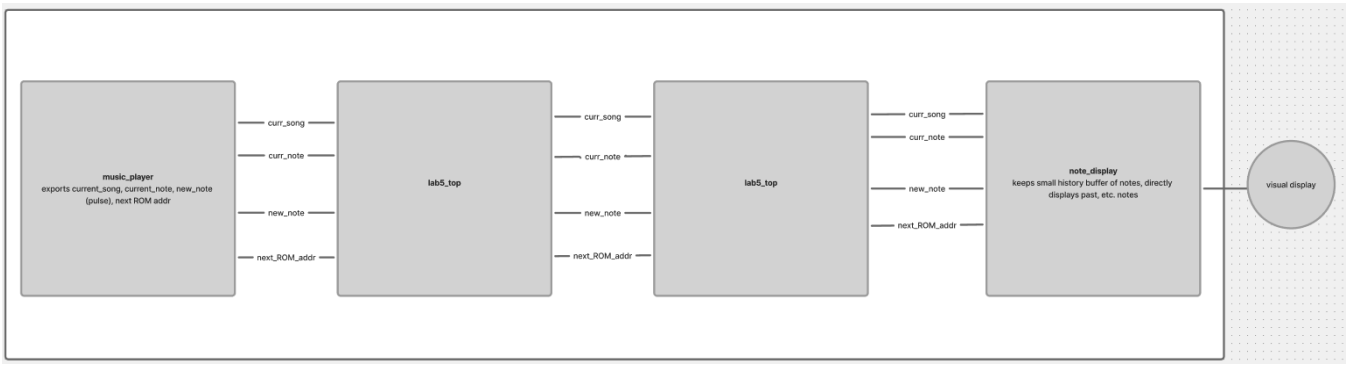
Block Diagram



5 – Past, Future, and Present Note Display (4 points)

Functionality – We will display currently playing notes or chords as text on the screen. We will also display the past few notes that have finished playing and notes in the future yet to be played to earn the full 4 points.

Unique implementation challenges & design - We implemented this as a separate visualization path running in parallel to our audio pipeline. This module overlays text onto the VGA output using the current song, note, new_note pulse, and next ROM address exported by music_player. A small register-based history shifts on each new_note to track past notes, while several parallel song_rom lookups preview future notes directly from read-only memory. We then convert each note number into note-name text, reconstruct the displayed chord voicing to match the audio path, and then use the existing font ROM to draw the results into a boxed region on the screen. Our main challenge was keeping the display consistent with the playback pipeline, our display had to mimic multi_voice_player's chord logic and not interfere with the audio path.

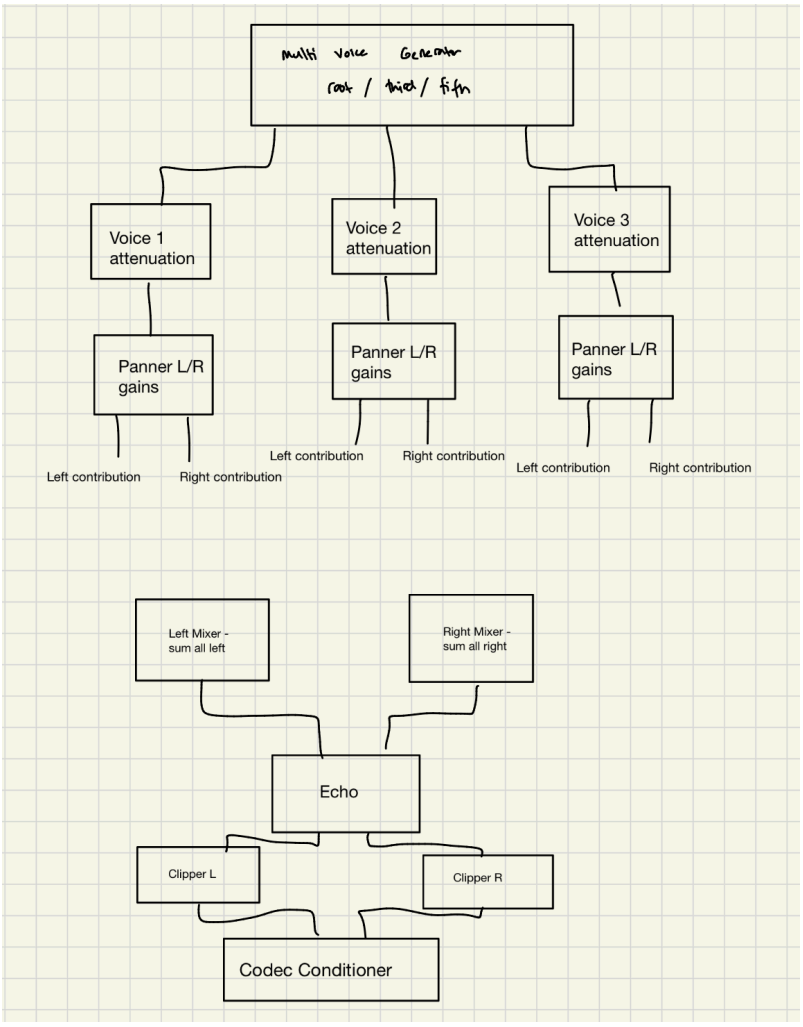


6 - Stereo Effects

Functionality –

Mono audio is a single stream, the same sound sample goes to both L/R speakers while stereo is two streams (left and right), so sounds can be positioned across the listening field. We kept the original mono mix for compatibility, but also generated a stereo mix by panning each voice into L and R with different gains.

We chose root centered, third left, fifth right so the chord spreads naturally across the stereo field without losing the main melody. The root carries the harmonic foundation, so keeping it centered preserves clarity. Offsetting the third and fifth slightly left/right makes the chord feel wider and helps listeners distinguish the individual note voices.



Unique implementation challenges & design -

The main challenge was adding true L/R behavior without disrupting the existing timing or the mono stream used elsewhere. We kept the stereo changes simple by doing the changes where the three voices are already mixed. Each voice is adjusted for low-register balance, then split into a left share and a right share using fixed-point gains, and those shares are summed into separate left and right outputs and clipped to 16-bit. The original mono mix still exists for compatibility, but now the left and right channels sound different, giving a wider image. The echo stays mono and is added equally to both sides, so timing and the codec interface stay the same while the stereo spread comes from the dry voice mix.