FP-DJ – An FPGA-based Digital Workstation

6.205 Final Project: Final Report
Team 02

Kofi Agyepong
Department of Electrical Engineering and Computer Science
kba@mit.edu

Deepta Gupta
Department of Electrical Engineering and Computer Science
deeptag@mit.edu

***Abstract*—We propose a digital audio workstation on the FPGA, with the ability to record digital instrument audio, apply effects (such as delay, reverb, distortion, and bass boost), play back modified sound, and display information to the user through a visual UI. Real-time effects can be applied to the instrument audio, or audio can be recorded, modified, layered together, and played back at a later time. Live audio can also be simultaneously played over pre-recorded tracks.**

I. Goals and Evaluation Metrics

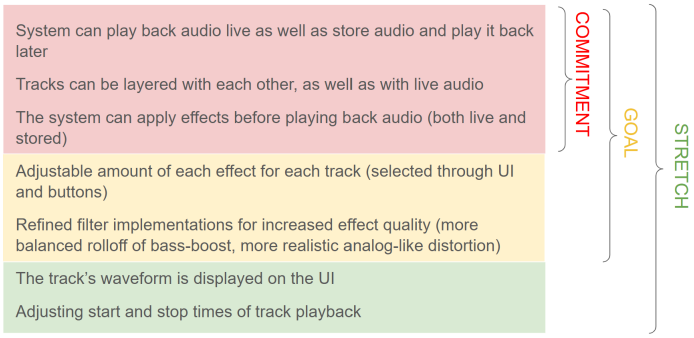

Fig. 1: Project Goals.

Our ambitions were divided into "commitment" objectives, or what we envision as our minimum viable product, "Goal" objectives, the set of objectives we felt we could reasonable complete, and "stretch" objectives, objectives we felt would be a challenge to implement but worthwhile to aim for. For efficacy evaluation, we had three major metrics:

- Latency: For synchronous playback, we aimed for delays imperceptible to the human ear (under 5 ms).
- Audio quality: we aimed to minimize unintended distortion and achieve audio quality similar to that of phone speakers.
- Resource usage: we aimed to make efficient use of FPGA resources and to avoid logic with high dead-time (that is, long periods where the logic is idle).

II. System Overview

The high level function of each block of the system is as follows:

- The I2S Decoder retrieves audio samples over I2S from a MEMs microphone.

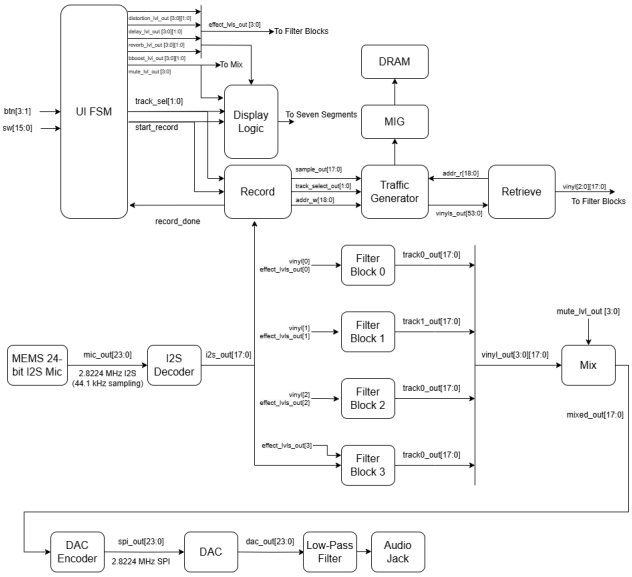

Fig. 2: Top Level.

- The UI FSM parses user input, enabling users to apply effects to tracks and record sequences of audio.
- The Display Logic block presents useful information to the user on the FPGA's seven segment displays.
- The Record block, Traffic Generator, MIG, and DRAM handle storing audio in memory, which enables recording. The Retrieve block pulls stored samples from memory, enabling asynchronous playback of audio.
- The Filter blocks apply effects to either asynchronous audio retrieved from memory (filter blocks 0-2) or synchronous audio freshly received from the I2S decoder (filter block 3).
- The Mix block superimposes all tracks together to create a single sample that will be sent to the output.
- The DAC Encoder, DAC, and Low Pass Filter translate the digital samples to an analog waveform and send it to the audio output (in this case a 3.5mm headphone jack).

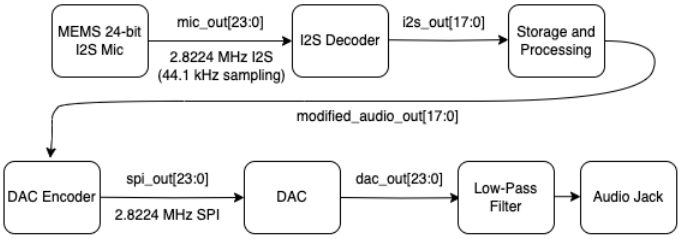

Fig. 3: Audio Input and Output.

III. Audio

A. Input/Output

To acquire audio, we are using a MEMS microphone that operates using the I2S communication protocol. The microphone provides data in a 24-bit format, and we will be using the 18 most significant bits to store our sample. The microphone's output will be fed through a decoder, which will isolate the relevant bits from each sample at a sampling frequency of 44.1 KHz (corresponding to a 2.8224 MHz clock). The I2S protocol consists of the following three signals:

- **SCK**: the serial clock (in our case, will be ran at 2.8224 MHz)
- **WS**: word select, which determines whether the data being transmitted is part of the left channel or the right channel (in our case, we will use exclusively left channel)
- **SD**: the serial data

The SD output is our sample, which is what all of the sound effects will be applied to. After the sample is modified and mixed with other tracks, it will need to be outputted. The FPGA's audio jack is a standard analog AUX port, so we have to convert our samples to an analog format before feeding them into the audio jack. To accomplish this, we are using a DAC. Since the DAC takes in 24-bit data, we first feed the 18-bit sample through a DAC encoder module, which adds on the necessary number of bits. This data then passes through the DAC using a communication protocol similar to SPI, at the same 2.8224 MHz clock rate used by the microphone. The DAC's control signals are as follows:

- **SYNC**: an active low signal that enables the transfer of data
- **SCLK**: serial clock (in our case, we will output samples at our original sampling rate of 44.1 KHz)
- **DIN**: the serial data

The output of the DAC is an analog voltage that can go through an analog low-pass reconstruction filter to eliminate any high-frequency noise that the DAC introduces. This filter consists of an op-amp and an RC network. The output of this filter will go to the audio jack, and is what the user will hear. A flowchart of the I/O handling of our audio is shown in Figure 3.

B. Digital Signal Processing

Our system is intended to be able to apply four different types of sound effect to our audio samples, namely reverb, delay, distortion, and bass-boost. Each track will be assigned a set of switches, and each effect will be assigned a switch, so whether or not an effect is applied to a specific track will be determined by the position of a set switch. This assignment scheme allows for an effect to simultaneously be applied to different tracks, and for multiple effects to simultaneously be applied to one track. If all the switches are low for a given track, the sample for that track will remain unmodified.

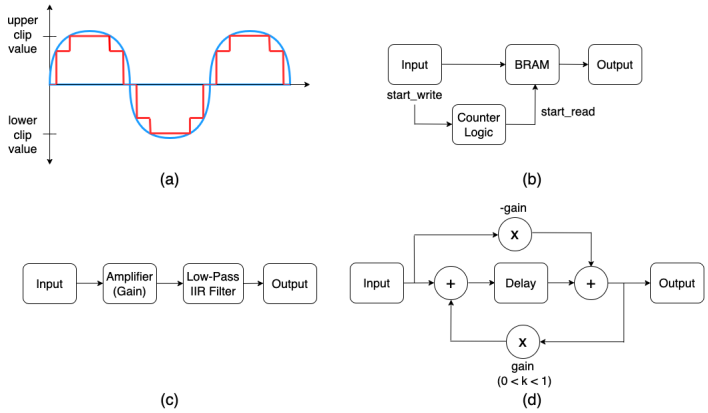

Fig. 4: Diagrams illustrating the four audio effects. (a) Distortion. (b) Delay. (c) Bass-Boost. (d) Reverb.

The four sound effects will be implemented as follows:

- **Distortion**: To create a distortion effect, we will set an upper and lower clip value to make the sine wave more of a square wave, which introduces harmonics. In order to mitigate high frequency from sharp changes in the waveform, we divide the signal into bins and clip each bin at an intermediate clipping value, creating a slightly smoother signal. A graph of what this looks like is shown in Figure 4(a).
- **Delay**: A delay effect can be made by adding clock cycles in between inputting and outputting the sample through the filter block. To do this, we store the sample in a BRAM. We have a counter with a set number of cycles, and after that number of cycles is reached, we retrieve the sample from the BRAM and output it. A block diagram of this is shown in Figure 4(b).
- **Bass-Boost**: A bass-boost effect can be achieved through filtering out the higher frequencies and amplifying the lower frequencies. In order to not lose resolution while filtering, we first amplify the entire signal, and then feed it through a low pass filter to eliminate the higher frequencies. This low-pass filter can be digitally implemented through IIR, or Infinite Impulse Response, techniques. A block diagram of this is shown in Figure 4(c)
- **Reverb**: Reverb is essentially an echo effect, where the sample is delayed several times and each delayed sample is layered on top of the others. To accomplish this, we

repeatedly feed the sample through delay block, reduce the volume of the delayed sample, and layer it on top of the previously generated sample. This will ultimate cause the echoes of the signal to fade away. We also have a feed-forward portion that mitigates any filtering effects that the feedback portion creates, ensuring that we preserve our full frequency range. A control diagram of this effect is shown in Figure 4(d).

IV. UI

The UI FSM handles parsing user input from buttons [3:1] and translating it into commands for the Record Block and Display Logic blocks. It also stores the information such as the effects currently enabled on each track and their intensity.

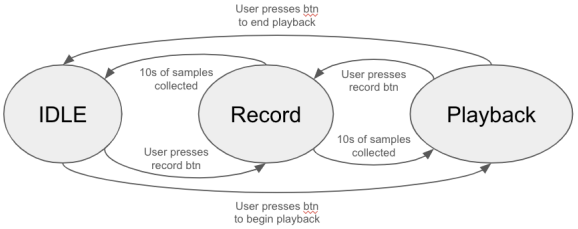

Fig. 5: UI FSM.

The FSM begins in the PLAYBACK state; in this state no commands are issued to the Record Block and the Display block is commanded to display the current state (PLAYBACK), the currently selected track (either 0, 1, 2), and the effects currently active on the selected track. While in PLAYBACK, the FSM will transition to the RECORD state upon a press of btnA.

In the Record state, the UI will command the Record block to begin recording, wait until 10s of samples have been collected (as indicated by the record_done flag), and then transition back to the PLAYBACK state. When the UI FSM transitions back to playback, it will update the value of track_valid to inform the output circuitry that the data in memory for the last recorded track is now valid. While in the Record state, the Display Block is commanded to showcase a timer with the amount of recording time remaining.

The UI FSM internally stores variables that track the intensity of the effect applied to each track. In other words, for each of the four tracks, the UI FSM keeps track of which effects are enabled and, if enabled, the intensity of said effect (which may be either Low, Medium, or High). The intensity of effects applied to each track can me edited using the 3 user input buttons. Each button has a function dependent on the current state of the FSM, which is detailed in the following diagram:

V. Memory

The memory interface consists of Record, which collects samples from the I2S decoder and sends them to memory; the Traffic generator, which handles generating memory read and write request; the DRAM and MIG, which store the samples and manage the DRAM, respectively; and Retrieve, which pulls track samples from memory and sends them to the filter blocks.

Button Function by State			
	btnA	btnB	btnC
Playback	Start Record	Cycle Track	Begin Editing Track
Edit Track	Back	Cycle Effect	Begin Editing Effect
Edit Effect	Back	Level select	Done
Record	N/A	N/A	N/A

Tracks:
- 0: Asynchronous playback
- 1: Asynchronous playback
- 2: Asynchronous playback
- 3: Synchronous playback

Effects:
- Distortion
- Delay
- Reverb
- Bass boost
- Mute

Effect Levels:
- For mute: On or Off
- For other effects: Off, Low, Medium, High

Fig. 6: UI Button Mappings

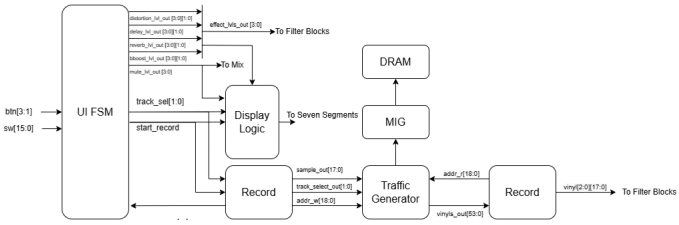

Fig. 7: Memory UI

A. Record

Record takes in 18 bit audio samples from the I2S decoder, a record signal from the UI, and a track select signal from the UI. If the record signal is low, then the Record block ignores all incoming samples. If the record signal is high, the Record block takes input samples and generates requests to the traffic generator to write them to the location in memory corresponding to the currently selected track. Record sends data to the traffic generator through a FIFO, since the two blocks run on different clocks.

B. Traffic Generator, MIG, & DRAM

The the traffic generator, MIG, and DRAM are largely be built off of the memory interface used in lab 6. The notable modifications are that:

- Each entry in memory is 18*3=54 bits wide (to hold an 18 bit sample from each of the three channels). There is a total of 10 [seconds of recording] * 44100 [samples/second] = 441,000 addresses. This corresponds to 441000 * 54 = 23.814 Mb of memory.
- Since each entry in memory contains data from all three channels, bit masking is enabled on the MIG so that a subset of the bits in an address (specifically, [53:36], [35:18], and [17:0]) can be modified without destroying the rest of the data.

C. Retrieve

Retrieve is responsible for three functions: pulling samples from memory, splitting them into three separate tracks, and routing those tracks to the appropriate buffer blocks. Retrieve perpetually cycles through the 441,000 memory address and sends the corresponding samples to the filter blocks. Retrieve accesses the data the traffic generator sends through a FIFO, since the two blocks run on different clocks.

VI. Physical System

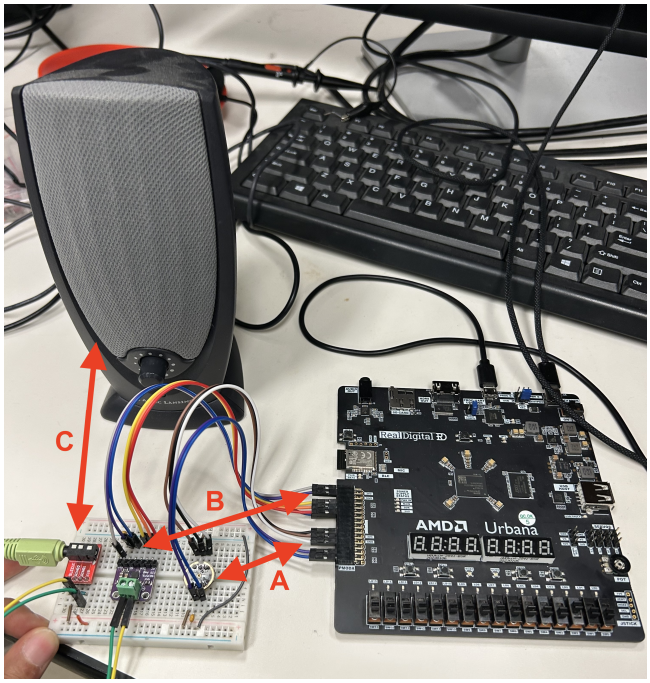

Fig. 8: Diagram of Physical System

Our physical system contains a few components external to the FPGA, shown in Figure 8. These are elaborated on below:

A: The 24-bit I2S microphone is interfaced with the FPGA through the PMODA bus. The VDD and GND pins connect to the port's 3.3V and GND pins, and the WS, SCK, and SD pins connect to the port's I/O pins. L/R is tied low so that mono input on only the left channel is received.

B: The 24-bit I2S Digital-to-Audio converter is interfaced with the FPGA through the PMODB bus. The VDD and GND pins connect to the port's 3.3V and GND pins, and the LRC, BCLK, DIN, and SD connect to the port's I/O pins. Gain is tied high to set the DAC to 6dB gain.

C: The Adafruit TRRS Audio Jack breakout board takes in analog signals from the DAC output and feeds them to a speaker. The TIP and RING1 pins are tied to each other and connect to the DAC+ output. The RING2 pin is tied low and connects to the DAC- output. The SLEEVE pin is left floating.

VII. Evaluation

A. Audio

The audio pipeline comprising of reading information from the microphone, writing information to the DAC, and transferring information from the DAC to the speaker/headphone input was successfully completed. The system is successfully able to take in external audio through the microphone and output that audio through the DAC at a roughly 44 kHz audio sampling rate. While there is some distortion observed in the output audio, and the mic's sensitivity prevents a clear recording of only the desired signal, there is minimal delay in hearing the audio being inputted into the system during live playback.

This pipeline utilized I2S, and thus the timing requirements were primarily those imposed by this protocol. The word select clock to indicate the transmission of a new data sample was constrained to a period of 64 times that of the I2S clock. Thus, to achieve a 44 kHz sampling rate, the I2S clock needed to be run at roughly 2.8 MHz, which corresponded to about 36 cycles of the FPGA's internal clock. Altogether, this means that there were 64×36, or 2304, cycles between samples. This high number of cycles meant that a sample could be stored, modified, and retrieved without the danger of being overridden somewhere in the process.

B. Filters

While initially we strived to implement four filters, namely distortion, delay, reverb, and bass-boost, only three were attempted and none of them were successful. Potential causes of failure are elaborated on in Section VIII (Insights).

C. UI and Display

Both the UI and FSM met the basic functionality goals for the project. The final implementation of the UI met the goal of parsing user input, storing and adjusting user settings, and issuing commands to the Record and Display blocks. Similarly, the display block is able to display information to the user on the seven-segment displays, although the stretch goal of integration with the monitor was not implemented.

For efficacy evaluation, latency and audio quality were of little concern with these modules, since any delays were well below the perceptible limit for humans, and the user interface and display circuitry had no impact on audio quality; the only relevant metric was resource usage. In this regard, both the UI and the Display block used a minimal amount of FPGA resources (3. 37% of the total FPGA slices for both modules combined). However, each module does spend the majority of cycles in an idle state due to the slow timescale of human perception relative to digital computation speed.

D. Memory

The memory interface met the functionality goal of being able to record and play back 10s of audio for 3 independent recordings. There are two major metrics for the efficacy of the memory interface: high enough throughput to provide samples at a sampling rate of 43.3 kHz and efficient memory usage.

The throughput requirement for the memory interface was easily satisfied. The system clock was running at 100 MHz; at a sample rate of 43.3 kHz this means there are about 2300 system clock cycles between each sample. This was ample time to retrieve samples from memory.

The minimum theoretical memory size of the system was 43.4 samples/sec * 10s * 16 bits per samples * 3 tracks ≅ 20.8 Mb. In the end, around 27.8 Mb were actually used. This is because each entry in memory was 128 bits, enough to start 8 samples total. There were 3 tracks to be recorded, so each address in memory had room for 2 samples from each track. The leftover 32 bits in each memory location could have been used to store additional samples, but this would have required asymmetric numbers of samples per track to be stored at each location in memory. This in turn would have required additional logic and/or metadata stored along with the samples to realign the data before it was sent to the filter blocks. Because the DRAM had ample memory space, we made the decision to simply leave 32 bits at each address in memory empty. While easier to implement, this does mean that around 25% of the memory used is not actually storing any meaningful data.

VIII. Insights

A. Audio

The main improvement we would like to see within our audio pipeline is noise reduction. This would allow us to get a much clearer signal, and would help with the evaluation of any effects that we apply to the signals. In addition, the noise seems to get scaled at the same or higher magnitude as the desired signal when the sample is modified. This posed problems when attempting to add samples in any way, to the detriment of our filters.

A lot of this noise appeared to stem from the physical connections, as even slightly twisting or moving certain wires led to sharp increases or decreases in the background audio and the microphone sensitivity. Several solutions were investigated to try to reduce this effect. First, we observed that there was a lot of noise within the I2S clock itself, which could easily conflate low and high signals and cause new data to be transmitted before the previous sample had finished being sent. To solve this problem, a small capacitor was added between the clock and ground to dampen the oscillations around the clock transitions. In addition, bypass capacitors were added to the microphone and DAC to reduce any noise on the power rails. In addition, digital methods were explored, such as adding an FIR low-pass filter to block out higher frequencies, and altering the bits that were written to the DAC to try to increase the number of dynamic data bits being transmitted.

However, these solutions reduced noise pretty minimally. With more time, we would have liked to further investigate the source of this noise and try to mitigate it, as this would greatly increase the quality of our system.

B. Filters

We think that one of the biggest culprits of our filters not working was signed vs. unsigned numbers. The microphone and DAC expect and output signed numbers because I2S is a signed protocol, but we think we might have lost parts of our sample or added unwanted garbarge bits while attempting to shift and add these numbers.

The problem of losing data while attempting to bit-shift impacted the performance of our delay and bass-boost filters, as well as our mix block while trying to combine the samples from the three recorded tracks.

Although we had issues developing the filters themselves, we were able to successfully integrate the filter modules for each track along with the mix module within our top level. With our skeleton complete, we believe that given more time to debug the individual filters, we would have been able to accomplish a system that contained the functionality of both our commitment and our goal, as per Figure 1.

C. Memory

The primary improvement to the memory interface would be increasing efficiency of memory usage. In our case, this could be achieved fairly easily by adding another track and using the extra 32 bits at each location in memory to store 2 of its samples. Additionally, since there is ample time to retrieve samples from memory before the next sample is requested, some of the logic surrounding the traffic generator could be trimmed down. For example, the FIFOs between the traffic generator, Record, and Retrieve could likely be replaced by a buffer of a few registers: this would still enable the clock domain crossing but eliminate the need for AXI and use less resources overall.

IX. Contributions and Acknowledgments

Deepta's contributions to the project include: writing the code for the audio pipeline (I2S decoder, DAC encoder, filter block module), constructing the physical hardware setup, and writing the code for 2 out of 3 of the attempted effects. Kofi's contributions include: writing the UI, Display, and memory interface code (UI FSM, Display logic, Record, Retrieve) and writing the code for 1 out of 3 of the attempted effects. Both of us contributed equally to system design, research, documentation, and integration. For the final report, Deepta wrote the sections and developed the figures relating to the audio pipeline, filters, and hardware setup. Kofi wrote the sections and developed the figures relating UI, display, and memory interface.

Many thanks to the 6.205 teaching staff for your support, guidance, and patience in debugging.