

DigiDrum3000 Final Report

Matthew Taylor, *Undergraduate, MIT EECS* and Charles Reischer, *Undergraduate, MIT EECS*

Abstract—We present the DigiDrum3000, a drum sampler, audio effects processor, and audio visualizer. We make use of the Real Digital Urbana development board containing an AMD Spartan-7 FPGA in order to perform real-time audio signal processing and visualization. The audio effects consist of pitch shift, bit crush, delay, low-pass filter, distortion, and reverb. These effects can be applied in any order, on both the audio and visual side*. The visualizer implements a set of video effects corresponding to each of the audio effects. By integrating the audio effects processor and visualizer into a single device, we enable highly responsive video effects that correspond to the actual effects applied to the audio signal.

Index Terms—Audio synthesis, Audio visualization, Digital signal processing, Digital systems, Field programmable gate arrays

I. PRINTED CIRCUIT BOARD (C. REISCHER)

A. Physical Board

THE user interface consists of a printed circuit board that interfaces with the PMOD connection of the Urbana FPGA board. The PCB includes 12 labeled potentiometers for volume, pitch, and effect parameter control, one TRS MIDI jack, and 11 TS jacks for patch cable connections. The MIDI input is optocoupled according to the MIDI specification, the potentiometers are read by 2 MCP3008 ADCs communicated to with SPI (5 pins), and the patch jacks use diodes to allow the FPGA to determine the patch state with 6 inout connections. In total the board uses 12 of the PMOD data pins.

B. Digital Interface

The `pcb_interface` module communicates with the MCP3008 chips through SPI to obtain the potentiometer states. The module determines the patch cable configuration through a 3-step process. First the inout connections are used

to determine which effects are downstream of each port. One inout at a time is set low, while the rest are set to pull-up. The diodes on the board only allow ports downstream of the low inout to be held low, with the effect directly following the low inout guaranteed to be below the FPGA's zero threshold (diode forward voltage drop could theoretically prevent long chains from completely being pulled low, the logic accounts for this). A reverse-order patch path ending in the "dry" port is then constructed by traversing the earlier determined port connections. Finally the input for each given port (with the special value 7 for ports unconnected or otherwise disjoint from the main path) is found from the reverse path. The entire module uses no BRAMs or DSP blocks.

C. MIDI (M. Taylor)

The DigiDrum3000 is designed to connect to the MIDI output of an electronic drum set and respond to a parameterized number of drums and cymbals. We have tested this project with the Roland TD-11 drum set module and incorporated support for bass drum, snare drum, three toms, ride cymbal, crash cymbal, opened/closed hi-hat, and hi-hat pedal. The MIDI interface follows the standard 31.25 kbaud UART protocol.

In addition to the parameter controls provided on the printed circuit board, each effect parameter may also be controlled via MIDI control change (CC) message automations in a digital audio workstation (DAW). A Python script is used to create a virtual MIDI instrument that interfaces between the DAW and the FPGA UART receiver.

II. DRAM INTERFACE (M. TAYLOR)

The DRAM interface (diagrammed in Fig. 3) is responsible for reading/writing data to/from DRAM, leveraging the UberDDR3 DRAM controller.

A. DRAM Writer

On startup, the DRAM writer awaits a UART stream of audio samples, which are all stored consecutively in the first 20 Mbits of DRAM. After the sample loading process is complete, the DRAM writer is responsible for writing video pixel data to DRAM.

B. DRAM Read Requester

The DRAM read requester requests a chunk of eight audio samples (128 bits total) from DRAM at a particular address. A global sample rate counter increments every clock cycle and resets to zero after eight audio sample periods have elapsed. The sample period ranges from 568 to 9088 cycles of the 10 ns clock and is adjustable using the pitch control knob.

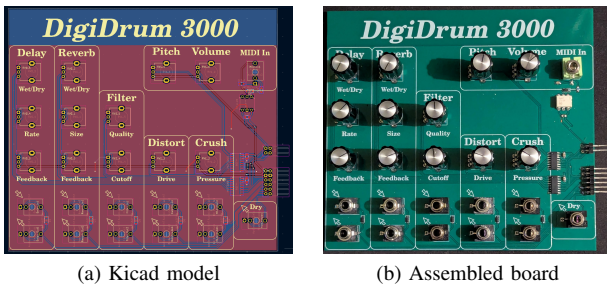


Fig. 1: Physical PCB.

C. Reischer and M. Taylor are with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 02139 USA e-mails: tmatthew@mit.edu and charl800@mit.edu.

*pitch is not orderable, delay is not orderable on visual side.

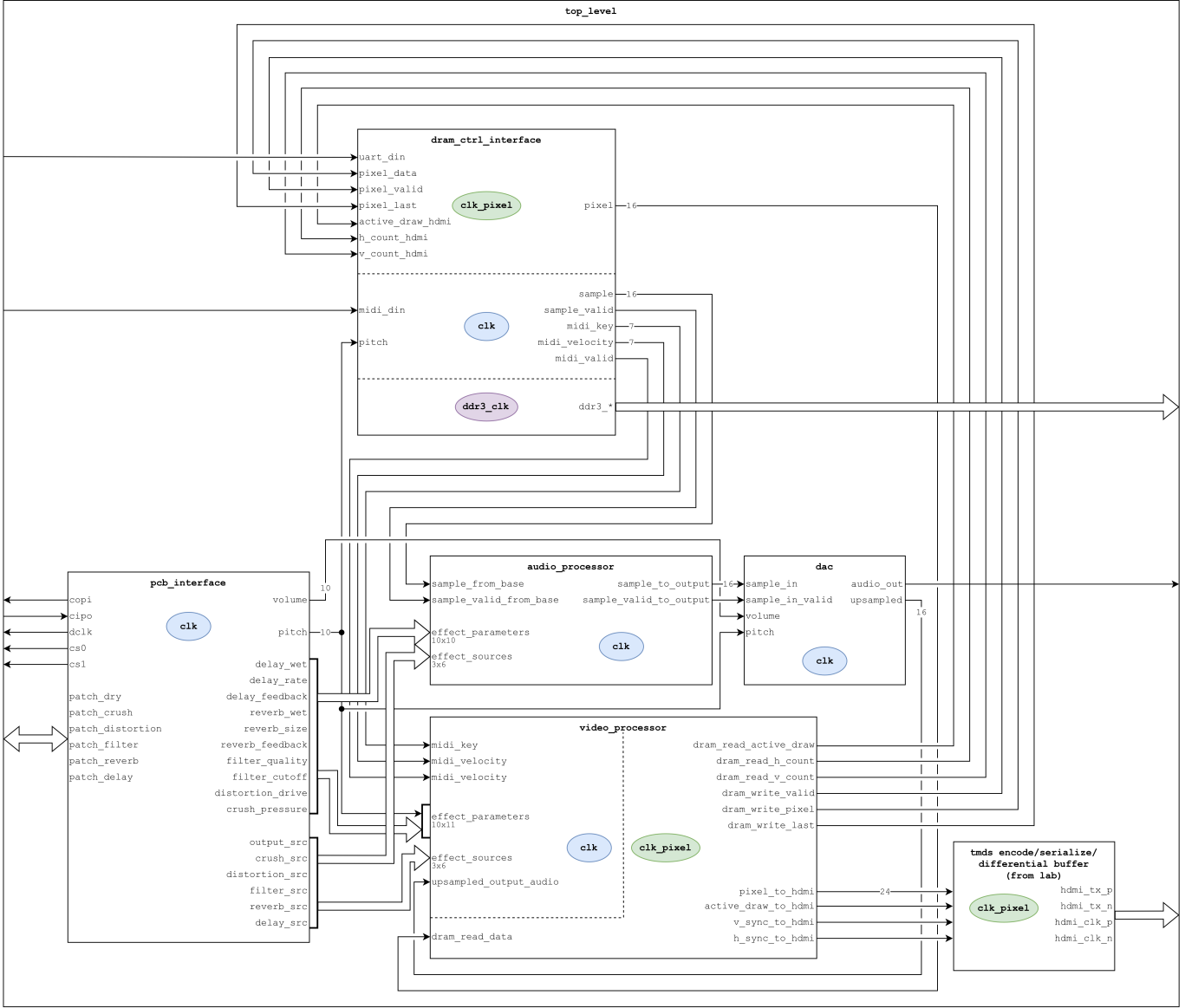


Fig. 2: Top-level block diagram.

When a particular MIDI note on message is received, a sample counter for that note is reset and is used to incrementally address DRAM at consecutive memory locations each time the sample rate counter has a value of zero.

Therefore, the maximum latency of the DRAM read requester occurs if a note is played when the sample rate counter has a value of one. This equates to a maximum latency of eight times the maximum sample period, or 727 μ s.

C. DRAM Audio Reader

The DRAM audio reader receives audio samples from DRAM and mixes samples from different drums and cymbals together. The pitch control determines the rate at which the sample mixer provides data to the audio effects processor. The sample mixer uses 1 DSP block to scale audio samples by their corresponding MIDI velocity.

To estimate the worst-case latency of a read from DRAM, we will take the extreme case of 500 DRAM controller clock

cycles (at 83.333 MHz) between a read request and a read data output. In this case, the imposed latency is 6 μ s.

D. DRAM Video Reader

The DRAM video reader receives video pixel data from DRAM and synchronizes the data into complete video frames.

E. Traffic Generator

Similar to the traffic generator module developed in Lab 6, we use a traffic generator module to manage the flow of signals to and from the UberDDR3 DRAM controller. One change to the Lab 6 traffic generator is the addition of new FSM states to support reading/writing both audio and video signals. Additionally, the value of the sample period is embedded in the command FIFO in order to synchronize the sample rate of DRAM input audio data with output audio data.

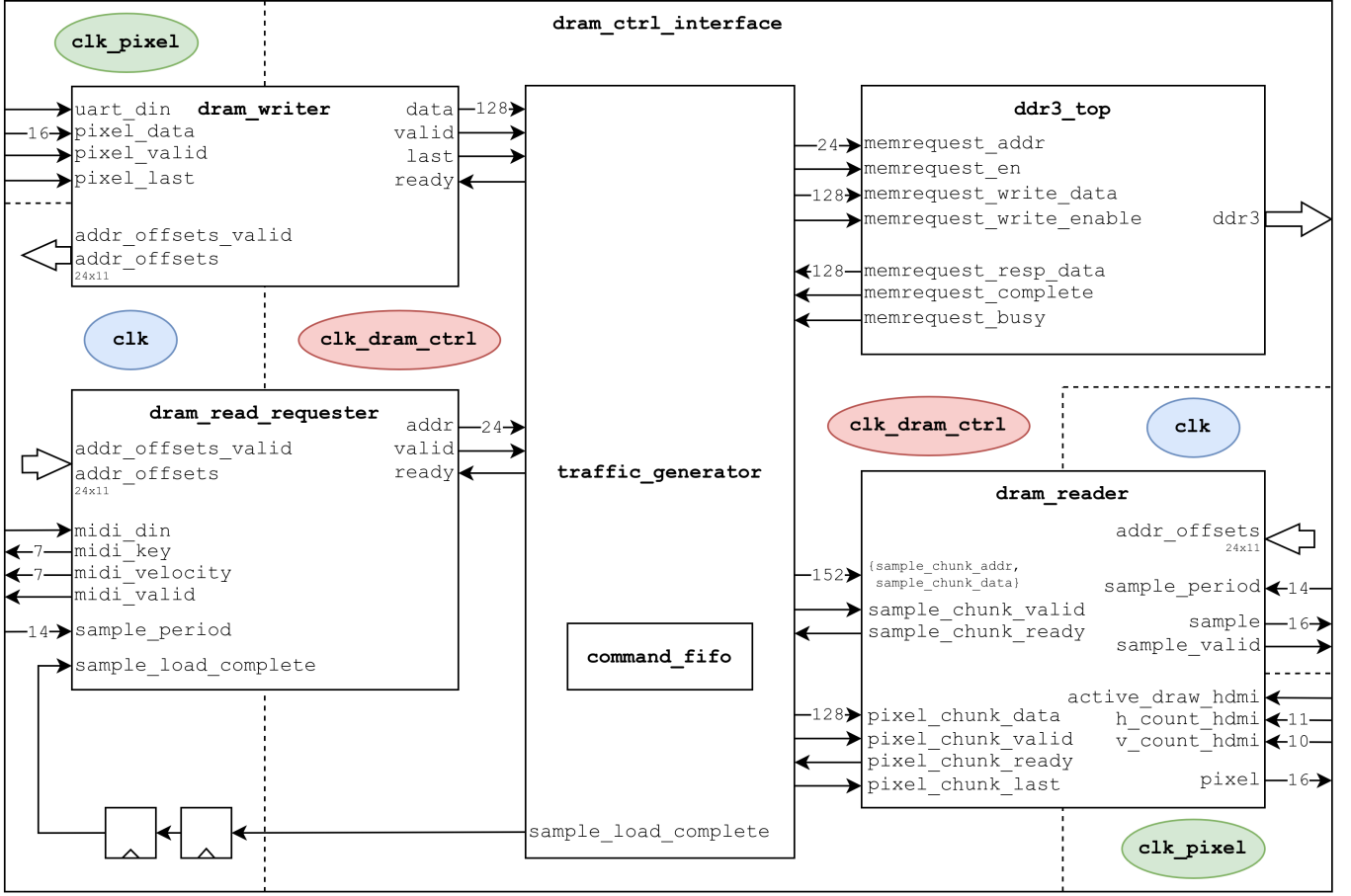


Fig. 3: Block diagram for the DRAM interface.

III. AUDIO EFFECTS PROCESSOR (M. TAYLOR)

The audio effects processor routes raw audio samples from DRAM through various effects in the same order as the patch path. All audio effects are implemented in fixed point, and simulations confirm direct match to corresponding fixed-point Python implementations and insignificant deviation from floating-point Python implementations.

Due to the loose timing constraints imposed by a 44 kHz sample rate and 100 MHz clock, finite-state machines are used extensively in the audio processor to stage calculations across multiple clock cycles. This serves to ease timing closure and enables time-division multiplexing of DSP blocks to reduce resource utilization.

A. Upsampler

At the input of the audio effects processor, a resampler is used to convert the variable sample period output of the DRAM audio reader to a fixed sample period of 2272 10 ns clock cycles (or a sample rate of about 44 kHz). The first stage of the resampler is a 3rd order Farrow structure upsampler [1], which uses Lagrange interpolation to convert the variable sample period input to an output with a fixed sample period of 9088 10 ns clock cycles. The upsampler requires 3 DSP blocks. A simulation output of the upsampler is provided in Fig. 4. Because the fractional delay of the Farrow structure

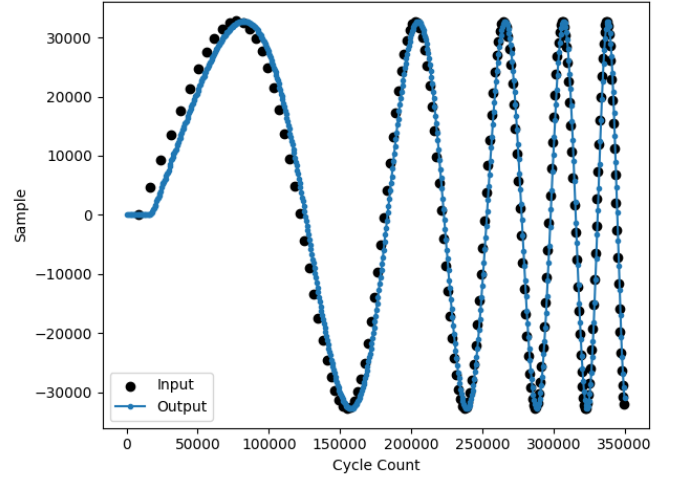


Fig. 4: Simulation of the Farrow structure upsampler with 100 MHz clock. The signal frequency at a 2272 clock cycle sample period is 1 kHz. The input sample period ranges from 568 clock cycles to 9088 clock cycles, and the output sample period is a constant 9088 clock cycles. The signal frequency depends on the input sample period, which simulates the desired pitch shifting effect.

lies between 0 and 1, the latency of the upsampler is equal to one input sample period, which is at most 91 μ s.

B. Downsampler

The second stage of the resampler is a polyphase 4x downsampler, which outputs samples at a fixed sample period of 2272 10 ns clock cycles. The downsampler requires 2 DSP blocks, 1 BRAM18 block to store the FIR filter coefficients, and 1 BRAM18 block used as an input sample buffer. The FIR filter is a Kaiser-windowed 512-tap minimum phase low-pass filter with a cutoff of 20.8 kHz for anti-aliasing. Based on the index of the filter's impulse response peak, the downsampler has a latency of about 70 μ s. This means that the maximum total latency of the resampler is about 161 μ s.

C. Audio Bit Crush

The bit crush effect applies a combination of bit-depth reduction, downsampling (without anti-aliasing), and hard clipping to the input signal. Each of these three effects are controlled by a single knob, which affects the overall fidelity of the sound. The latency of the bit crush effect is 10 ns.

D. Audio Delay

The delay effect resamples the input signal (as detailed above) and stores the audio data in a circular buffer consisting of 4 BRAM36 blocks for delayed playback. The output of the delay effect is fed back into the input to create repeated echoes, and the output is also resampled to return to a sample period of 2272 10 ns clock cycles. With a maximum input downsampling factor of 4, a maximum delay time of 744 ms can be achieved. Excluding the two resamplers, the delay effect requires 3 DSP blocks for feedback scaling and output wet/dry mixing.

The latency of the delay effect is dominated by the latency of the two resamplers. Because the sample period of the input signal is fixed at 2272 10 ns clock cycles, the first resampler has a fixed latency of about 93 μ s. The second resampler has a variable latency that depends on the delay rate and is at most about 161 μ s.

E. Audio Distortion

The distortion effect applies a soft-clipping saturator to the input signal using an approximation of the tanh function [2]:

$$\tanh(x) \approx x \frac{27 + x^2}{27 + 9x^2}$$

The approximation is related to the [3/2] Padé approximant of the tanh function, with coefficients modified to provide at $x = 3$ a value of 1 and a first and second derivative of 0. Originally, the tanh function was intended to be implemented via CORDIC. However, for the hyperbolic functions, the range of convergence in traditional CORDIC implementations is limited to $[-\pi/4, \pi/4]$. Over this domain, tanh is roughly linear and therefore not useful as a saturator.

The audio distortion effect requires 1 DSP block for scaling the input signal by the adjustable drive parameter, as well as 2 DSP blocks for implementing the tanh approximation. Additionally, the input of the effect is upsampled by a factor of 4, and the output is downsampled by a factor of 4 (both

utilizing the same filter coefficients as the downsampler described previously). Upsampling shifts the Nyquist frequency upwards to reduce the impact of aliasing caused by the effect's nonlinearity.

The upsampler and downsampler together introduce about 140 μ s of latency, and the distortion effect itself has a latency of 590 ns.

F. Audio Filter

The audio filter is a virtual analog low-pass transistor ladder filter with 4 poles [3]. The filter is implemented using a series combination of 4 1-pole low-pass filters with variable global feedback to provide an adjustable quality factor. Each 1-pole low-pass filter is implemented using trapezoidal integration in order to perform a topology-preserving bilinear transform of the modeled analog filter. Additionally, the output of the feedback and input summer is soft-clipped using the tanh approximation described previously to model analog nonlinearity.

Without incorporating the saturation element, the continuous-time audio filter has the following frequency response:

$$H(s) = \frac{1}{k + (1 + s/\omega_c)^4}$$

where $s = j\omega$, k is the feedback gain, and ω_c is the cutoff frequency. By applying the bilinear transform, the frequency response may be represented in the Z-domain with the substitution:

$$s = \frac{2}{T} \frac{z - 1}{z + 1}$$

where T is the sampling period and $z = e^{j\omega T}$. The resulting frequency response represents the frequency response

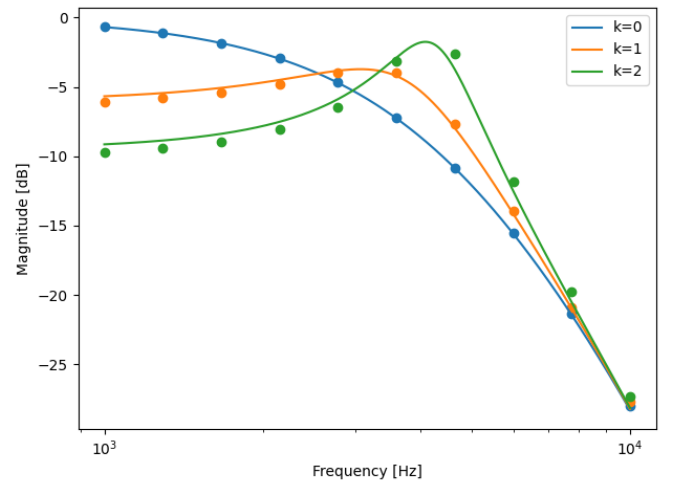


Fig. 5: Frequency response of the linear audio filter with a 5 kHz cutoff frequency and feedback gain k . The solid lines represent the theoretical frequency response of the filter topology, and the dots depict the results of simulating the fixed-point SystemVerilog implementation.

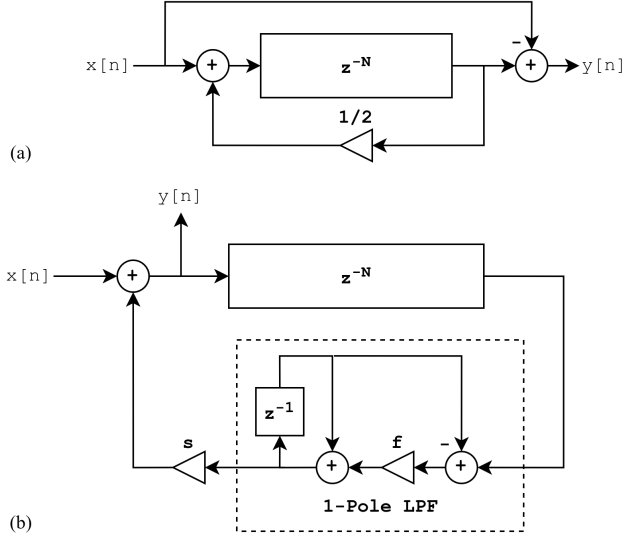


Fig. 6: (a) All-pass filter approximation used in the audio reverb. (b) Low-pass feedback comb filter used in the audio reverb. The low-pass filter cutoff parameter f is mapped to the reverb feedback control, and the feedback gain parameter s is mapped to the reverb room size control.

of the discrete-time audio filter implemented using trapezoidal integration and is plotted in Fig. 5. The discrepancy between the theoretical and simulated frequency response is primarily due to loss of precision in conversion to a fixed-point implementation with minimal resource usage. Notably, an entire division has been discarded due to its minimal contribution to the filter’s behavior across permitted operating conditions.

Including the tanh approximation, the audio filter requires 4 DSP blocks. Similar to the audio distortion effect, the audio filter also applies oversampling by a factor of 4 in order to reduce the impact of aliasing. Oversampling introduces about 140 μ s of latency, and the filter itself has a latency of 880 ns.

G. Audio Reverb

As developed by M. R. Schroeder [4], a parallel combination of comb filters followed by a series combination of all-pass filters may be used to simulate the effect of audio reverberations in a room. Comb filters have a frequency response that consists of notches which resemble the phase cancellations of echoes. Additionally, both the comb filters and all-pass filters have an impulse response that is a series of decaying echoes.

For this project, we implement the Freeverb algorithm [5], which consists of 8 low-pass feedback comb filters in parallel, followed by 4 all-pass filters in series. The structure is duplicated for the left and right audio channels, with slight variation in delay line lengths, to achieve a stereo effect. It is worth noting that the all-pass filters used in the original Freeverb implementation, as well as our implementation, do not have a flat frequency response. This is due to an approximation that reduces the number of necessary multiplications. Schematic diagrams of the all-pass and comb filters used in the audio reverb effect are provided in Fig. 6.

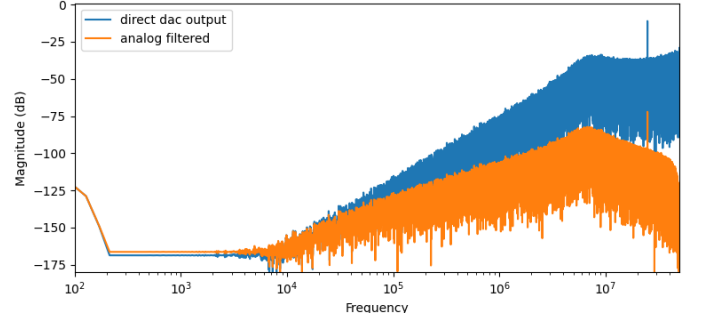


Fig. 7: DC limit cycle test of delta sigma modulator.

Each reverb channel requires 9 BRAM36 blocks total for the delay lines of the filter elements, as well as 2 DSP blocks. The latency of the reverb effect is about 1 μ s.

IV. DIGITAL TO ANALOG CONVERTER (C. REISCHER)

The final mixed digital audio signal is upsampled to roughly 700k samples/second (exactly 1 sample/142 cycles) using a minimum phase FIR filter to prevent aliasing. The upsampler introduces a latency of about 70 μ s, based on the index of the filter’s impulse response peak. The audio signal is then scaled based on the volume parameter, and fed into a second order delta-sigma modulator. The modulator has been tested and iterated extensively, and results in the noise being uncorrelated from the signal (dithered). The noise is at comparable levels to the noise present due to the analog components on the board. Theoretically the total harmonic distortion plus noise in the audible range should be -100dBFS based on simulation, although real world non-idealities (likely inter-modulation of high frequency noise) result in worse real-world performance. The entire module uses 1 BRAM18 and 2 DSP blocks, and would ideally only need 1 DSP block.

V. VIDEO EFFECTS PROCESSOR (C. REISCHER)

The video processor first generates monochrome “dry” shapes based on the MIDI signals entering the audio processor, monochrome delay shapes that move outwards from the dry shapes, and a continuous color gradient. These signals are combined, excluding the delay shapes if the delay is not included in the patch path (when delay_src is 3'b111). The dry shape generation uses 4 DSP blocks, the delay generation uses 14 DSP blocks, and the color gradient generation uses no DSP blocks. In total these modules use roughly 5200 LUTs.

The signal is then routed through the video crush, distortion, filter, and reverb modules in the same order as the patch path, including the h_count and v_count signals for proper pipelining. The h_count and v_count signals fed into the pipeline are offset to be “in the future,” so that they align with the original signal by the time they reach the end of the pipeline. This offset changes depending on which modules are currently in the patch path. Doing this allows the original sig_gen values of h_count and v_count to be used at the HDMI output, which prevents sync discontinuities when changing the patch state (some monitors will hide the screen for a second or two if this happens, presumably as an epilepsy precaution for bad connections).

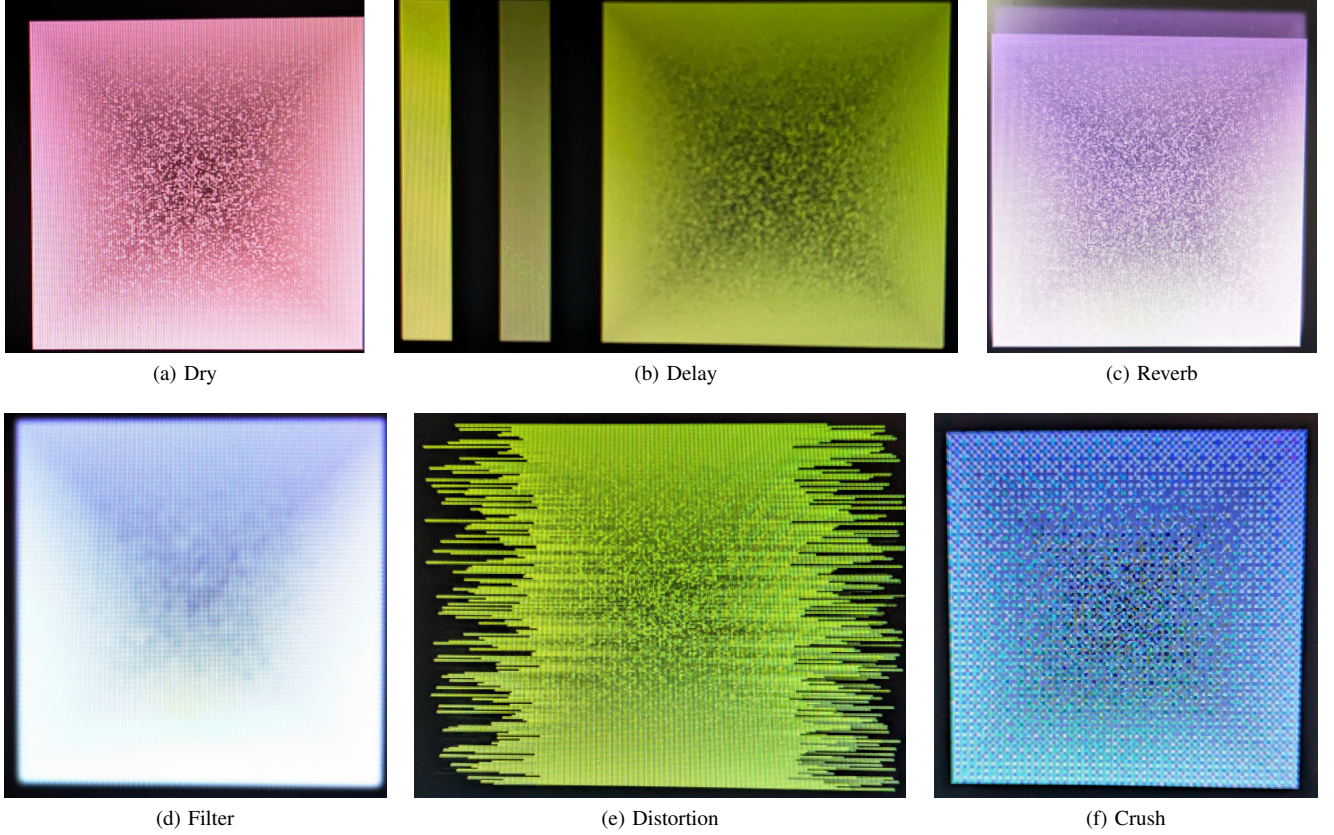


Fig. 8: Video output of square shape with each effect in isolation ((a) has no effect).

A. Video Crush

Video crush applies a continuous bit-reduction in the represented colors, i.e. $\log_2 3$ bits would represent 3 values. It does so with Bayer dithering at half resolution, leading to intentionally visible dithering patterns. Video crush uses no BRAM or DSP blocks.

B. Video Distortion

Video distortion offsets each horizontal line by a random amount, which is larger in absolute value when the drive parameter is large. It uses 4 BRAM36 and 2 DSP blocks, but could use folded multiplications in place of DSP blocks.

C. Video Filter

Video filter applies a 9x9 gaussian blur based on the filter cutoff. In order to fit within a practical amount of memory, the pixel data is compressed to 6bits per channel. Bayer dithering is used here, at full resolution. To generate the gaussian coefficients, folded CORDIC sinh and cosh (along with some other logic) is used to calculate the negative exponent at various distances from the center. The filter is applied with a 2-step method—gaussian coefficients are factorable into horizontal and vertical parts:

$$e^{-(y^2+x^2)} = e^{-y^2} e^{-x^2}$$

only needing 18 multiplications per color channel instead of 81. Video filter uses 10 BRAM36, 30 DSP blocks, and roughly 2400 LUTs.

D. Video Reverb

Video reverb applies a spacial and temporal blur, storing the current frame in DRAM and adding the previous frame to the current frame. The pixel data is only given 16 bits per pixel in DRAM, which would significantly impact the color gradient if using RGB565 or similar encoding. In order to mitigate this, the pixel are encoded using the YCoCg format [6] and the Co and Cg values are averaged across 2 pixel horizontally. Such compression is used in many formats including JPEG [7], as human perception is more sensitive to luminance than chroma in fine detail. Video reverb also 3x3 blurs the image every frame, "expanding" the stored image. This module uses 4 BRAM36, 3 DSP blocks, and the majority of DRAM bandwidth.

VI. OVERALL SYSTEM EVALUATION

A. Resource Utilization

The DigiDrum3000 uses (roughly) 26000 LUTs (80%), 19000 FFs (30%), 47 BRAM36 + 15 BRAM18 (73%), and 104 DSP blocks (87%). It also meets timing with some margin (0.276ns WNS without advanced stages of timing optimization). Our design certainly could be optimized to

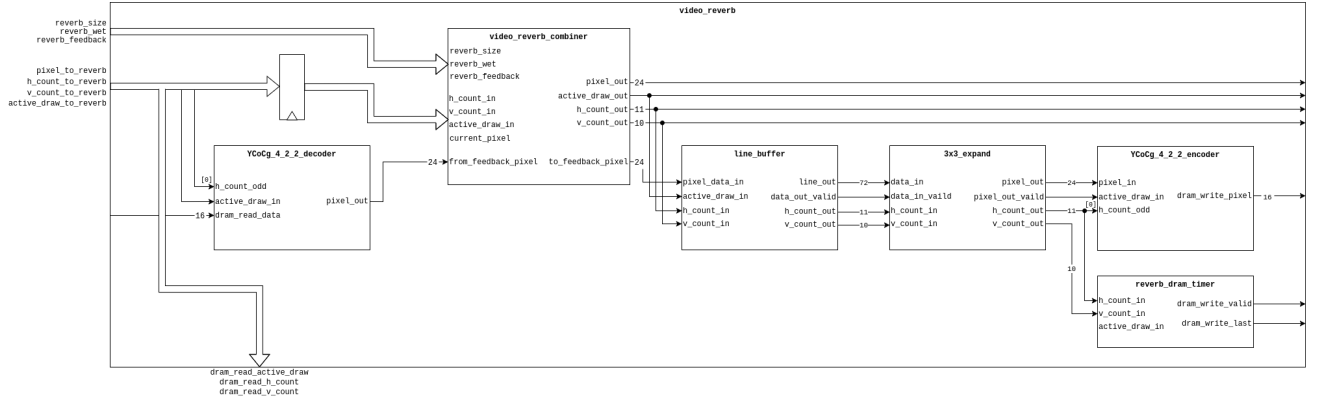


Fig. 9: Block Diagram for video reverb.

use less resources (particularly DSP blocks), but we never ran out of resources and chose to focus on completing more features instead of over-optimization. Multiplications in the video pipeline were a large contributor both to DSP and LUT utilization, due to needing single cycle throughput and 3 copies of every operation to cover all 3 RGB channels. Many DSPs were inferred from 8 bit by 8 bit multiplications, which could likely use less resources if pipelined through multiple cycles.

B. Audio Latency

We will characterize the audio latency of the design by taking the worst-case latency between a MIDI note on message being fully received and a corresponding audio sample output from the DAC upsampler. Combining the worst-case latency of the DRAM interface, audio processor, and DAC upsampler, the total worst-case audio latency of the design is about 1.5 ms. Note that the latency is dominated by the DRAM read requester latency, which is dependent on the pitch control. When no pitch shift is applied, the total audio latency drops to about 955 μ s. This value falls within our initial goal of achieving total audio latency less than the latency of a single MIDI note on message (three bytes plus six start/stop bits transferred at 31.25 kbaud, equating to a 960 μ s latency). In subjective tests, the latency introduced by the DigiDrum3000 is imperceptible.

C. Audio Fidelity

The DAC output was subjectively transparent while actually playing, although some (properly shaped) background noise can still be made out which commercially sold DACs do not suffer from (laptop headphone jacks, for example). Empirically, there was a 71dB difference in level between the background noise (nothing playing) and the peak RMS level while recording our video (RMS levels measured with REAPER). This is roughly 25dB worse than our simulation results, but still well within acceptable listening levels.

D. Feature Completion

Although the DigiDrum3000 is complete enough to present, there are many small additions, improvements, and overall polish that we did not have time to complete. These include:

- Supporting fractional bit depth in the bit crush audio effect in order to create smoother transitions when turning the control knob.
- Incorporating a high-pass filter in the feedback loop of the audio filter to reduce low-end drop-off at high quality factor.
- Image sharpening the blurred image in the video filter module based on filter quality.
- Image subsampling at low filter cutoffs in the video filter to create a more intense blur effect.
- Accounting for the reverb size parameter in the video reverb module.
- Better matching of subjective parameter intensity between audio and video.
- Coherence / low pass filter of video distortion's offset, to make lines follow more of a wavy pattern rather randomly splaying outwards.
- Chromatic aberration (different offset for different RGB channels) in video distortion.

VII. CODE REPOSITORY

<https://github.mit.edu/6205F25/fa25-6205-team24>

REFERENCES

- [1] V. Välimäki, "Discrete-Time Modeling of Acoustic Tubes Using Fractional Delay Filters," pp. 82-104, Dec. 1995.
- [2] <https://www.musicdsp.org/en/latest/Other/238-rational-tanh-approximation.html>
- [3] V. Zavalishin, "The Art of VA Filter Design," Oct. 2018.
- [4] M. R. Schroeder, "Natural Sounding Artificial Reverberation," *Journal of the Audio Engineering Society*, vol. 10, no. 3, pp. 219-223, July 1962.
- [5] <https://github.com/sinshu/freverb>
- [6] Malvar, H.S., Sullivan, G.J., Srinivasan, S., "Lifting-based reversible color transformations for image compression," *Applications of digital image processing XXXI*, vol. 7073, pp. 44-53, 2008.
- [7] S. Winkler, C. J. van den Branden Lambrecht, and M. Kunt, "Vision and Video: Models and Applications," *Vision models and applications to image and video processing*, pp. 209, 2001.