FPGA Implementation of Audio Effects - An EE 552 Student Application Note

by Richard Schultz

Audio Processing Unit

Duncan Campbell - Grant Cunningham - Clint Lozinsky - Richard Schultz

INTRODUCTION

Digital signal processing (DSP) is a very exciting market these days. FPGA's and ASIC's make up such a large part of this market that FPGA manufacturers are predicting their products will soon completely take over standard DSP microprocessors. While this prediction might be a little over ambitious, digital signal processing in FPGAs is gaining momentum. And while audio processing only makes up a fraction of the DSP market, it is both interesting and useful to understand how certain audio algorithms work.

The effects described here have all been implemented in the time domain. Frequency domain processing is possible for certain effects, but time domain processing is much easier. Because audio sampling is done in the time domain, it is inherently easier to process in this domain as it does not require hardware to transform the signal. Using a few basic elements outlined below one can easily and effectively implement a digital effects processor.

THE BUILDING BLOCKS

The Circular Buffer

The first critical element in effects implementation is a circular buffer. Circular buffers are a crucial component to any digital signal processing application. They permit data to be continually updated, and overwrite the oldest data in memory. Looking at figure 1, data is originally written to the very first memory location. As new data comes in, it is written to the next available memory address. Once the address pointer

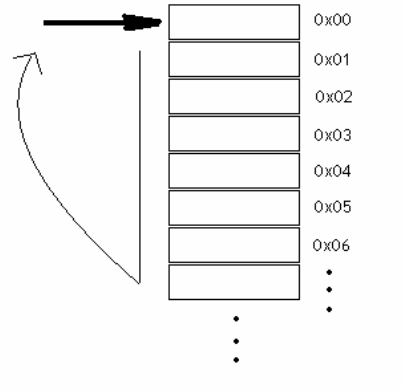

FIGURE 1 – A CIRCULAR BUFFER

reaches the end of the circular buffer, it immediately wraps around and starts writing to the first memory address again. Storing data in this manner, 2^N samples are always available in the buffer, and all that is necessary is controlling the pointers. This can be most easily done with a simple *N*-bit counter.

For the implementation of the effects described here, one write pointer and two read pointers are used as shown in figure 2. Using these three pointers, one is able to obtain an input, $x[n]$ and two outputs, $y_1[n]$ and $y_2[n]$. Both $y_1[n]$ and $y_2[n]$ are time delayed versions of the input, and can be written as:

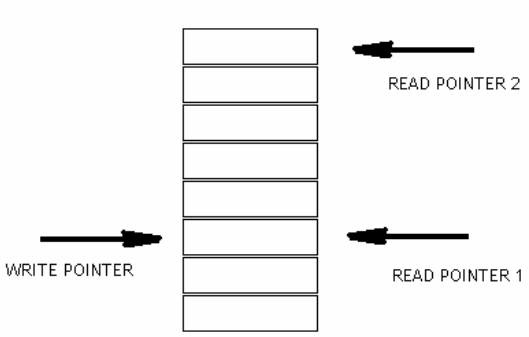

FIGURE 2 – A CIRCULAR BUFFER FOR AUDIO EFFECTS

$$y_1[n] = x[n - t_{d1}]$$
$$y_2[n] = x[n - t_{d2}]$$

where t_{d1} and t_{d2} are constant delay factors. t_{d1} is the difference between the first (current) read pointer and the write pointer. Generally this could be as little as one clock cycle (giving adequate time to store a value before immediately reading it back) but for purposes of this document it is assumed to be zero. The human ear does not notice time delays less than about 50ms, so this is a reasonable assumption. Using this, we can simplify our outputs to:

$$y_1[n] = x[n] \qquad \textbf{(1)}$$
$$y_2[n] = x[n - t] \qquad \textbf{(2)}$$

With t being a value corresponding to the time difference of the two samples. This difference is the product of the sample rate of the digital audio stream, and the number of samples. Also note that while n corresponds to an index in the circular buffer and is thus constrained to *N* (total number of samples stored), the circular buffer treats the data as continuous. The only precaution that must be taken is to keep $t < N$.

The Mixer

The next element required to implement a few basic effects is a mixer. This is a very simple element that in effect takes in two signals and outputs the weighted sum of them. The two inputs to the mixer are the values from the circular buffer, so the output of the mixer is given by the equation:

$$y_{out}[n] = ay_1[n] + by_2[n]$$

Which, from Equations 1 and 2, can be simplified to yield:

$$y_{out}[n] = ax[n] + bx[n - t] \qquad \textbf{(3)}$$

Where a and b are the weighting coefficients.

With Equation 3, one can easily implement a large range of effects by changing only the values α, β and τ.

THE EFFECTS

Echo

The echo effect is the easiest of the effects to implement. This effect is created by adding the current sample to a previous sample. Using Equation 3, it is easy to see that this effect is created by keeping T constant. Therefore an echo effect is simply described by the equation:

$$y_{echo}[n] = ax[n] + bx[n - t_{echo}] \qquad \textbf{(4)}$$

Where t_{echo} is the echo length.

Obviously, this is accomplished using only the mixer and circular buffer outlined in equation 3 above. The only thing that is necessary to accomplish this is to subtract a value from the current read pointer and use it as the second read pointer.

Chorus

The chorus effect is only slightly more complicated than the echo. In a similar manner to an echo, the chorus is produced when you add the current sample to a previous sample, only the amount of delay is varied sinusoidally. By varying the delay from 40ms to 60ms continuously at a rate of 0.25 Hz, you have a standard chorus effect. The equation that describes this is:

$$y_{chorus}[n] = ax[n] + bx[n - t(n)] \qquad \textbf{(5)}$$

Where $t(n) = A\sin(2pf)$, A = Constant multiplier and f = frequency of variation

To implement this effect, a sine look up table can be generated, and this can be subtracted from the current read pointer. Numerous HDL design libraries include sine-cosine lookup tables, and their usage is rather simple.

Flange

The flange effect is very similar is structure to the chorus effect. In fact, the only thing that changes is the amount of varying delay, and the rate at which it occurs. For a standard flange effect the delay generally varies from 0ms to 10ms at a rate of 0.5Hz. Obviously the equation for a flange effect is the same as that for a chorus effect:

$$y_{flange}[n] = ax[n] + bx[n - t(n)] \qquad \textbf{(6)}$$

Where, once again $t(n) = A\sin(2pf)$, A = Constant multiplier and f = frequency of variation. Cleary this is the same as the chorus effect in equation 5. The only difference between implementation is the specification of the A and f values.

Phaser

The phaser effect is the result of two identical, yet out of phase, signals being added together. This produces various notches in the phase response and has a canceling effect which is audible to the human ear. In essence, this is basically the same effect as the flange and chorus, only with different parameters once again. The equation for a phaser is:

$$y_{phaser}[n] = ax[n] - bx[n - t(n)] \qquad \textbf{(7)}$$

In this case, $t(n)$ can be anything from a sinusoid to a saw-tooth or even a constant value. The only major difference is the sign of B, which results in the phase canceling effect. The varying delay isn't completely necessary for this effect, but it does have improved tonal qualities when it is changed by some small factor.

Once again, this can be implemented in the same manner as the flange and chorus effects. The only necessary precaution is to ensure that the mixer is capable of signed arithmetic. If using VHDL, the std_logic_vector array type has signed arithmetic capabilities built in, so one does not need to generally worry about this too much.

OVERALL IMPLEMENTATION

Certain other factors need to be taken into consideration when designing these effects into an FPGA project. As is clearly evident, the circular buffer needs to be large enough to be able to produce a noticeable echo effect. This requires the usage of memory, and there are numerous types of memory available to the FPGA designer. The Block RAM found on Xilinx boards is generally of

sufficient size for this, and is easy to use. For this reason, as well as its lack of external components, it is the obvious solution. SDRAM (Synchronous Dynamic RAM) is another option, although this requires the usage of a separate SDRAM controller to handle refresh cycles and other control aspects. If the project is not already using SDRAM this is perhaps not feasible in all instances. However, if SDRAM is available, its larger sizes make it preferable for total flexibility of the effects. SRAM (Static RAM) or Flash memory is yet another option, as it does not have the stringent control requirements SDRAM has. However, one may find that the board they have to work with doesn't allow them to use the SRAM, or its resources may be used by other FPGA elements. SRAM is also more expensive than SDRAM, so this may make it less feasible in certain instances.

Aside from that, implementation is rather straightforward. As the sampling rate of the audio is generally much lower than the clock rate for an FPGA project, all memory read and write operations, as well as any mixing and other post-processing operations can generally be done with extra clock cycles to spare. Even at a standard sampling rate of 44-48 kHz and a conservative clock of 25MHz, there are approximately 500 clock cycles to carry out the required operations.

CONCLUSIONS

As described, implementing standard audio effects in an FPGA is not as complicated as one might assume. It is also very rewarding, as the results of the implementation are noticeable by anyone. These few basic effects are used widely in the music industry to process vocals and instruments, so their application gets heard by literally millions of ears every day. As well, the way the methods used to implement the effects are applicable to various other fields of digital signal processing. Based on the ease of implementation in an FPGA, it's no wonder that FPGA's occupy a large share of the DSP market.

References

[1] Micea , Mihai V., Stratulat, Mircea, Ardelean, Dan, and Aioanei, Daniel *Implementing Professional Audio Effects with DSPs*. University of Timisoara, Romania, 2001

[2] Smith, Steven W. *The Scientist and Engineer's Guide to Digital Signal Processing, Second Edition.* San Diego, CA: California Technical Publishing, 1999