Adding a Virtual Tape Loop

Audio Processing with a Microprocessor

Tom O'Haver
Dept of Chemistry
University of Maryland
College Park MD 20742

There is a lot of talk about digital audio processing, but talk is not the same as practical action. As the prices of microprocessor systems and interface devices continue to drop, such applications are sure to become quite common, even among amateurs. This article describes a few of the possibilities of the use of a small low cost microprocessor system for digital processing of audio signals. The effects described involve echo, reverb, fuzz, time delay, phase phlanging, mono-to-enhanced-stereo conversion, and frequency multiplication. These effects could also be quite useful for the experimentally inclined audio enthusiast or music group.

Hardware Requirements

To run the programs given here, you will need a 6502 (or equivalent) processor with from 1 K to 5 K bytes of programmable memory, an 8 bit input port connected to a fast 8 bit analog to digital converter (ADC), and a latched 8 bit output port connected to an 8 bit digital to analog converter (DAC). An additional output port and digital to analog converter are required for stereo applications. The basic hookup for a simple monaural system is shown in figure 1. The signal from the preamp is amplified, low pass filtered, converted to digital by the analog to digital converter, processed in the microcomputer, converted back into analog by the digital to analog converter, and then filtered some more before going to the power amp. The success of such a system in audio processing depends upon its ability to operate at ultrasonic speeds; that is, the rate at which the audio signal is digitized, processed, and output must (or should) be as far above the upper limit of the audio spectrum as possible. Thus the speed of each of these steps is critical. We'll consider each step individually.

Next to the microcomputer itself, the analog to digital converter is really the most critical component. It must be a fast one; a conversion time of 50 *μs or less* is necessary to allow sufficiently high sampling rates. I have been using a Datel Model E8HB1, an 8 bit successive approximation analog to digital converter with a 4 μs conversion time (available from Datel Systems Inc, 1020G Turnpike St, Building S, Canton MA 02021, for $85 in unit quantities). I recommend this unit. Its conversion time is probably faster than you will need, but at least you won't have to buy a new one when microsystems get faster (as they certainly will). In addition to speed, the Datel analog to digital converter has two other features you should look for in a converter. First, it is bipolar, which means it is capable of accepting both the positive and the negative excursions of the audio waveform. Otherwise, you would have to add some offset to the incoming signal. (The programs in this article assume offset binary coding.) Second, it *clips* on overload rather than wrapping around. That is, if the input audio signal exceeds the dynamic range of the analog to digital converter, the digital output simply stops at full scale rather than wrapping around or folding back to zero. The reason this is a useful feature is the fact that an audio signal contains many peaks and transients which greatly exceed the average signal amplitude. With only an 8 bit system, there is really no way to keep these transients from exceeding the range of the converter, at least occasionally; if you try to prevent it by adjusting the average amplitude to a very low level, you'll get too much quantization noise. Clipping the peaks may offend the audio purist, but I'll guarantee you that it sounds a *lot* better than wrapping around. Of course, a better solution would be to use 12 bit converters and a 12 or 16 bit (or faster 8 bit) computer. Sufficiently fast 12 bit analog to digital converters are available for about $150, and 12 bit digital to analog converters are typically about $30. But without a 12 or 16 bit processor, all processing would have to be done in double precision, which might slow things down too much (unless you have a 4 MHz 6502, which I do not). Anyway, an 8 bit converter which clips is good enough for the time being.

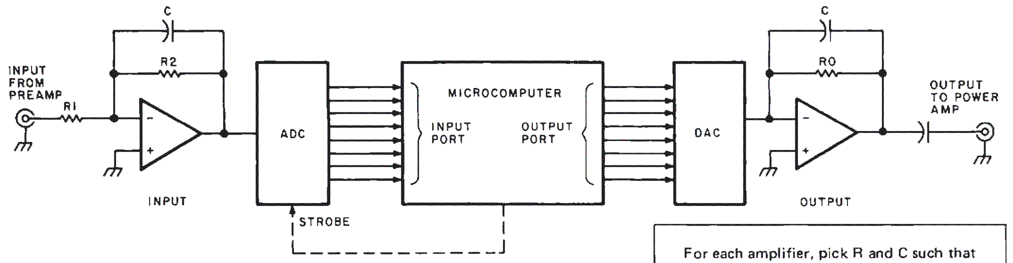
Selection of a digital to analog converter is much easier, since several fast, low cost 8 bit units are available. The digital to analog converter needn't be bipolar, since a DC blocking capacitor can be added easily. The Hybrid Systems 371-8 at $10 is a good choice, as is the Motorola MC 1408L8 at about $5. I've used both successfully. The Hybrid Systems unit is more convenient because it has a built-in reference supply, while you will have to supply an external (2 V) reference for the Motorola unit. This must be very well filtered but not necessarily well regulated for audio applications. (An advantage of the Motorola units is that they can be used as multiplying digital to analog converters. If you drive the reference input of one converter from the output of another converter, then the output of the first converter will be the *product* of the digital inputs to the two converters. This allows you to obtain automatic level control, compression, expansion, fading, and amplitude modulation effects without relying on much slower software multiplication routines and without getting into trouble with quantization noise.)

As for the processor itself, almost any 6502 system should do with the examples I've included in this article: KIM-1, Jolt, Ebka, OSI, PAIA, Apple-II, PET 2001, etc. I've used both the Ebka and the OSI systems with good results. OSI has a particularly convenient analog IO board (Model 430) which can be populated with two MC1408L8 8 bit digital to analog converters, an 8 bit analog to digital converter, and their associated latches and address decoding logic. The OSI Model 430 analog to digital converter circuit is of the synchronous tracking (up-down counter) type. Be warned, however, that this analog to digital converter wraps around on overrange. It also requires some individual tweaking of component values to get it to work. If you want to use the OSI 430 board, I strongly recommend that you replace their analog to digital circuit with a better one, such as the Datel E8HB1. Other than that, the OSI board is just fine.

One very important concept which you must understand is the relation between sample rate, aliasing, and low pass filtering. If you don't understand these terms and their significance, then before you go on you should read the article by Hal Chamberlin on page 62 of the September 1977 issue of BYTE. For the programs presented in this article, the sampling rate will fall between 20 and 40 kHz with a 1 MHz processor clock frequency, assuming you are using a sufficiently fast analog to digital converter (less than 50 μs sampling time). To control aliasing, you have to roll off the high frequency response of the *input* signal to the analog to digital converter at a frequency no higher than about 1/4 of the sampling frequency, ie: about 5 kHz for a 20 kHz sampling rate. This may not sound much like "hi-fi," but actually it sounds better than you might think. For better highs, you need faster processing and maybe a faster input converter. The 6502 is pretty good in this respect; it's available in versions at least up to 4 MHz. This would give you a sampling rate of 80 to 160 kHz for the programs given here and would extend the highs to the 20 to 40 kHz range. Now, *that's* a high fidelity computer!

The sampling rate therefore determines the frequency at which the response of the system must be rolled off (by means of appropriate low pass filters) in order to reduce aliasing to a tolerable level. In the simple circuit of figure 1, the only roll off is that provided by the capacitors in the feedback loops of the two op amps. Although this circuit is satisfactory for experimental purposes, the cutoff rate of the high frequency rolloff is not sharp enough for first class results. If you're really serious, you'll want more sophisticated, sharp cutoff filters. Hal Chamberlin gives the circuit of an excellent filter in his article in September 1977 BYTE, mentioned previously. The unpopulated printed circuit board, as well as an assembled and tested unit, is available from Hal. The cutoff frequency of this filter is 3 kHz, probably too low if you have a reasonably fast processor, so you might want to modify it or roll your own based on the designs in Don Lancaster's *Active Filter Cookbook* or other reference sources. Only one sharp cut filter is needed, between the preamplifier and the input analog to digital converter, to reduce aliasing. The filter on the output, between the digital to analog converter and the power

Figure 1: The system design of an audio processing test bed requires two simple peripheral devices and a computer. The input device is an analog to digital converter (ADC in this diagram) preceded by a filter. The output device is a digital to analog converter (DAC in this diagram) driving another filter. Source material from (for example) a broadcast program is input through the ADC, processed in real time by the program in the computer, then output in real time to the DAC where it (for example) goes to your audio power amplifier and speaker system. The program in the computer can be as simple as an unprocessed transfer from input to output, or as complex a transfer function as the constraints of real time will allow, given the speed of the computer.



For each amplifier, pick R and C such that

$$f_c = \frac{1}{(2\pi RC)}$$

where

f_c is cutoff frequency (kHz)
R is kilohms
C is microfarads

amplifier, needn't have as sharp a cutoff, but it should have the same cutoff frequency as the input filter. For the simple filters in figure 1, the cutoff frequency is equal to $1/2\pi RC$, where R and C are the values of the feedback resistor and capacitor, respectively.

One last thing to consider about the hardware is the level (amplitude) of the audio signal. In order to avoid excessive quantization noise, the input signal must be amplified enough to utilize the whole dynamic range of the input analog to digital converter. In figure 1, the first op amp provides gain in addition to filtering. The gain of this amplifier, which is equal to R_2/R_1, will have to be adjusted for your particular system. Given choices of R and C for filter cutoff, R_1 can be chosen given a desired gain level. For example, if your preamp provides a maximum output signal of 0.2 or 1 V peak-to-peak, and your input converter has an input voltage range of ±5 V (10 V peak-to-peak), then a gain of 10 V/1 V = 10 is appropriate. Also, the maximum output signal of the digital to analog converter must not be allowed to overload the power amplifier. This will dictate the selection of the feedback resistor R_0 of the output op amp in figure 1; the output voltage is directly proportional to the value of this resistor.

Software Considerations

So what about the software? First, let's see how to get data in from the input converter and out to the output converter without any processing at all. If your analog to digital conversion device (which I reference symbolically as CONV) is connected to an input port whose address is F800, then to load one sample of the audio signal into the accumulator (A) register of a 6502 requires one instruction, thus:

```
LDA CONV
```

This is all you need if you're using a tracking converter such as that on the OSI board, but if you're using a strobed converter, you'll have to give the converter a strobe pulse first, allow it time to convert, then load the A register. The fastest way to do this is to assign the input conversion strobe to an unused address, decode that address, and use the address select line (address "strobe," as it is sometimes called) as the pulse which strobes the converter. I have used the latter approach for my strobed analog to digital converter and have (arbitrarily) assigned address EC00 (which I call STROBE symbolically) to the address strobe. With this arrangement the converter is strobed by any instruction which references that address; for example an STA as shown here:

```
STA STROBE
__ }
__ }  several instructions executed
__ }  while conversion occurs.
LDA CONV
```

The above routine strobes the converter and then loads the data into the A register. The dashes represent intervening instructions which take up enough time to allow the analog to digital converter to complete its conversion. This will always be a useful code, rather than just no operation instructions (NOPs) or a wait loop. Conversion times of commercial analog to digital converters vary all over the place. As I mentioned before, for audio processing you'll need a fast one which converts in a time of 50 μs or better. Just make sure there are enough instructions between referencing STROBE and the loading from CONV to give the input converter time to convert.

To output one sample to the digital to analog converter (called DAC symbolically) is quite simple. For example, if the converter is connected to an output port whose address is F900, then all I have to do is store the sample:

```
    8D 00 F9     STA DAC
```

To test the proper operation of the input and output converters we can write a "straight wire" program which simply transfers the data from the input to the output without change. Listing 1 shows 6502 position independent code for such a program.

Note that the input conversion strobe instruction is placed right *after* the load CONV instruction. This may seem backwards but it gives the analog to digital converter a total of seven machine cycles (an STA and a JMP) to convert before it will load into A. On a 1 MHz machine, this means the conversion time could be as long as 7 μs. If your converter is slower than this, put some NOP instructions or a wait loop right before the CLC instruction. The other programs in this article execute much more code between strobing and loading the input analog to digital converter and will usually allow you to get by with no additional instructions intended specifically to slow down execution.

The program of listing 1 is good for testing out the hookup to your audio system. The sound quality of music played "through your computer" this way may be better than you would expect, considering that the audio waveforms are being sliced up into discrete samples, converted into binary numbers, and then converted back into an analog audio waveform!

So what kind of audio processing can you do? I'll resist the temptation to say that the applications are limited only by your imagination. They are not. They are limited by your programming skill, your processor speed, and your system's programmable memory capacity. You can never have too much of these. I'll not claim to have even scratched the surface of potential applications in this article. I'll just tell you about a few things I've done, mostly because they were easy to program. If you don't come up with better ideas than these I'll be disappointed.

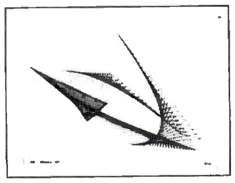
```
AD  00  F8  START LDA CONV        get new data from converter
8D  00  EC        STA STROBE      strobe input conversion
8D  00  F9        STA DAC         output to DAC
18                CLC             unconditional
90  F4            BCC START       branch to START
```

Listing 1: A 6502 "straight wire" program loop. In order to simply listen to the input data on the output channel without any processing, we must enter a tight machine coded loop which reads the input converter, then stores the input data into the output converter. This 6502 program assumes an input digital to analog converter at address space location F800 (CONV), an input conversion strobe which occurs on reference to address space location EC00 (STROBE) and an output digital to analog converter at location F900 in address space (DAC). These same assumptions about IO apply to listings 2 thru 6 as well. With hardware like figure 1, try running this program using an audio signal from your favorite record album. The results will probably be of higher quality than you might have expected.

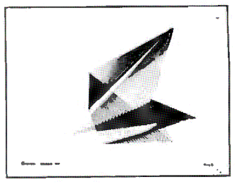
Waveform Modification

A very easy class of audio processing functions are those which are intended to distort the audio waveform. Believe it or not, distortion is actually considered desirable by musicians in some cases for obtain-

Art-by-Computer tm is here!



File 1



File 3

Black-on-white, 9 3/4H by 12 3/4W reproductions of computer generated designs See Jan 78 Byte or phone or write for a brochure.
Set of 12: $20.00; singles $2.00 each. Minimum order- 2 prints. Orders less then $10.00 and all orders from outside continental U.S. add $2.00 postage and handling. Payment must be in U S. dollars California residents add 6% sales tax Send check or money order to

Leland C Sheppard
PO Box 60051, Dept B
Sunnyvale, California 94086.

Also available at selected retail outlets.

© LCS, 1978.

```
AE  00  F8  START LDX CONV        Put converter data into x register
8D  00  EC        STA STROBE      Strobe converter.
BD  00  03        LDA TABLE, X    Look up xth byte of TABLE
8D  00  F9        STA DAC         Output to DAC
18                CLC             Repeat
90  F1            BCC START
```

Listing 2: Waveform modification. This 6502 program, obtained by modifying the program of listing 1 slightly, uses the input sample from the analog to digital converter (value 0 to 255) to look up the output sample in a "transfer function" table located at 0300 in memory and referenced with the name TABLE. The key to what the distorted output sounds like relative to the input is the data stored in TABLE (see text and figure 2).

ing special effects (such as "fuzz") with electric guitars and other electronic instruments. The computer can perform a rather elegant general purpose distortion function by utilizing a stored transfer function as illustrated in the program of listing 2.

To use this algorithm, you must set up a table in memory (on page 03 in this example) which serves as the *transfer function*. Each sample of the input waveform obtained from the input converter is used as an index to look up a corresponding byte in the table, which is then used as the output value. In this example the table is just 256 bytes long and is indexed by the 6502 processor's X register. Depending upon what we store in the table we can get any kind of distortion effect we want. A trivial case would be to use a straight line function, ie: put 00 in 0300, 01 in 0301, 02 in 0302. . .and FF in 03FF as shown in figure 2a. This would yield no effect at all; the output would be identical to the input as was the case with the program of listing 1. But if we use anything *other* than a straight line, we'll get distortion. Several possibilities are shown in figure 2. Figure 2b would give a square wave output while 2c would yield a somewhat less strongly distorted output. With the function shown in figure 2d, we would get a frequency doubling effect; that is, if the input were a sine wave of one frequency, the output would be an approximate sine wave of twice that frequency (one octave higher). With figure 2e, we'd get an output two octaves higher. This can be extended even further with the appropriate transfer function (eg: figure 2f). The effect on the sound of an electric guitar is quite remarkable, particularly as the frequency multiplication factor is a function of the amplitude of the input signal and changes at the input decays.

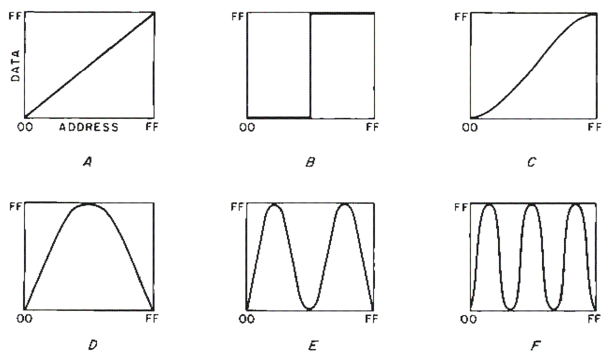


Figure 2: Examples of transfer functions for use with the waveform modification technique of listing 2. These curves are produced by plotting the data at an address versus the address within the table, with both values having a range of 00 to FF hexadecimal (eight bits worth). (A) is the simple linear transfer function. (B) is a transfer function which is equivalent to a saturated clipping amplifier: it puts out a square wave. (C) represents a slight distortion of the linear response of (A). (D) is a transfer function which effectively doubles the frequency of the input waveform, while (E) quadruples the input frequency and (F) multiplies the frequency by a factor of 8 (ie: three octaves higher).

Quite apart from its potential uses in music recording or performance, the above technique is a neat way to teach (or learn) about the effect of transfer characteristic nonlinearity on audio distortion. Just put in the characteristic under consideration and listen to the effect it has on the audio.

Time Delay, Phase Shift and Reverb Effects

If we store digitized audio in an array in programmable memory, and read it out to the digital to analog converter at a later time, we have a time delay effect which can be used for phase shift and reverb. The maximum time delay you can achieve depends on the sampling rate and the amount of available memory; but even with only 256 bytes you can get some pretty good phase shift and phase "phlanging" effects. With 4 K bytes you can get a good reverb.

The essential programming technique behind all of these effects is quite simple: output a byte from the data buffer to the digital to analog converter; input a new sample from the analog to digital converter and put it in the same location in the data buffer as the byte just output; increment the pointer modulo the length of the buffer and repeat. (Thus, when you get to the end of the data buffer you reset the pointer to the beginning and continue.) By scaling and adding the new data from the input converter to the old data from the data buffer, we can generate a range of effects depending on the length of the time delay.

The routine of listing 3 adds the audio signal to a slightly delayed version of itself and outputs the scaled sum to the digital to analog converter. In this routine, page 03 serves as the data buffer and x as the pointer. The pointer is initialized to DELAY, decremented until it gets to zero, and then reset to DELAY. This results in a sort of circular data buffer which acts as a first in last out shift register. The new and old (delayed) data are added and sent to the output converter. (Note that to prevent overflow, the data are divided by two *before* adding.) The time delay, determined by DELAY, can be adjusted from 1 to 225 samples. Such short delays do not result in a perceptible echo. The effect is rather that of a "comb filter" with multiple peaks and dips distributed throughout the audio spectrum. This is due to the fact that there will be a cancellation at every frequency whose period is an integral multiple of twice the time delay and a reinforcement at every frequency whose period is an integral multiple of the time delay. This rearranges the amplitude and phase relationships of the harmonics of music and speech and has a quite noticeable effect on the sound, variously described as a "resonant" or "twangy" effect. (If you have a hum problem in your audio setup, you might try to find the value of DELAY which puts a dip right at the hum frequency.)

The above idea can be extended and the effect made much stronger by causing DELAY to change continuously in real time. This would cause the peaks and dips to sweep through the audio spectrum. This effect is called "phase phlanging" by some people. An easy way to do it (not necessarily the best way, however) is shown in listing 4. This is the same as the previous program except that the DEC DELAY instruction has been added to reset the buffer pointer to a different value each cycle through the buffer. The effect of this routine on voice and music is quite dramatic. With speech and solo singing it gives a kind of voice doubling effect, as if two people were speaking or singing in synchronization. It makes a 6 string guitar sound reminiscent of a 12 string guitar. A concert piano comes out distinctly like a questionably tuned honky tonk piano. The effect on organ music is unreal and unpleasant. If you play the guitar and sing, or think you do, try processing a tape recording of yourself this way. It will sound better, or at least different (which in my case is the same thing).

If you have two output ports and two digital to analog converters, you can generate two channels of audio output. For example, you can convert a monaural source to "pseudostereo" with a further

```
A6  10       RESET LDX DELAY       initialize pointer to buffer length
BD  00  03    NEXT LDA BUFFER, X   get oldest data
4A                 LSR A           divide by 2
85  11             STA TEMP        keep
AD  00  F8         LDA CONV        get new data
8D  00  EC         STA STROBE      (strobe converter)
9D  00  03         STA BUFFER, X   replace old data with new
4A                 LSR A           divide by 2
18                 CLC
65  11             ADC TEMP
8D  00  F9         STA DAC         output to DAC
CA                 DEX             advance the buffer
D0  EF             BNE NEXT        pointer and repeat
18                 CLC
90  E2             BCC RESET
```

Listing 3: Time delays are possible with a buffer. Using the memory located at hexadecimal 300 to 3FF as a 256 byte delay buffer, a number of interesting effects can be achieved. This program supports a delay of up to 255 inner loop periods, too short to be perceptible as a delay per se, but it does transform signals by adding the delayed sample's points to the new input samples, producing an interesting filtered result. The delay buffer length is set by the value loaded into the X index register from location DELAY in the first instruction of the program. As in all the examples of this article, this 6502 program is position independent and can be loaded at any arbitrary place in memory address space which contains programmable memory not conflicting with IO or data storage locations.

```
A6  10       RESET LDX DELAY
BD  00  03         LDA BUFFER, X
4A                 LSR A
85  11             STA TEMP
AD  00  F8         LDA CONV
8D  00  EC         STA STROBE
9D  00  03         STA BUFFER, X
4A                 LSR A
18                 CLC
65  11             ADC TEMP
8D  00  F9         STA DAC
CA                 DEX
D0  E7             BNE NEXT
C6  10             DEC DELAY
18                 CLC
90  E0             BCC RESET
```

Listing 4: Modifying the processing done by the delay program of listing 3 to sweep the time delay value results in this "phase phlanging" program. The difference between this program and that of listing 3 is the DEC instruction which changes the value of the delay parameter DELAY each time it is reloaded. The effects must be heard to be believed.

```
A6  10       RESET LDX
BD  00  03    NEXT LDA BUFFER, X
8D  00  EF         STA DAC2        delayed sound to one channel
AD  00  F8         LDA CONV
8D  00  EC         STA STROBE
9D  00  03         STA BUFFER, X
8D  00  F9         STA DAC 1       direct sound to other channel
CA                 DEX
D0  EB             BNE NEXT
C6  10             DEC DELAY
18                 CLC
90  E4             BCC RESET
```

Listing 5: Modifying the program of listing 4 to turn it into a pseudostereo processor. Here, the delayed data is sent to a second channel, with the amount of delay swept as it was with the phase phlanger approach. But instead of adding the two channels together, they are kept separate and sent to the left and right stereo speakers.

modification of the program in listing 3 (see listing 5).

In this example the additional DAC is connected to an output port whose address is EF00. Instead of being added together, the direct and delayed signals are simply sent to the two different channels. The result is a sort of stereo phase phlanging effect which sounds much like a "rechanneled for stereo" disk recording. Try this through stereo headphones. So now you can have a stereo electric guitar. Would anyone like to extend it to quadraphonic?

If you have at least 4 K bytes of memory available in your system for your buffer, then you can obtain echo and reverberation effects quite readily. The idea is basically the same as the phase shifter routines just

```
                         AUDIO REVERB SIMULATION

                                 LABELS

F800            CONV         Address of 8 bit analog to digital converter
EC00            STROBE       Converter strobe line
F900            DAC          Address of 8 bit digital to analog converter
0009            FIRST        Lowest page number in data buffer
0010            LAST         1 + highest page number in data buffer
00A0            PNTRL        Low half of data buffer pointer
00A1            PNTRH        High half of data buffer pointer
0011            TEMP         Temporary storage

                              PROGRAM CODE

A0  00                 LDY #0          Set pointer to zeroth byte of page
                                       "FIRST".
A9  00                 LDA #0
85  A0                 STA PNTRL
A5  09         RESET   LDA FIRST
85  A1                 STA PNTRH
8D  00  EC     NEXT    STA STROBE      Strobe converter.
B1  A0                 LDA (PNTR), Y   Get oldest byte.
4A                     LSR A           Divide by 2.
85  11                 STA TEMP        Save.
AD  00  F8             LDA CONV        Get new byte.
4A                     LSR A           Divide by 2.
18                     CLC             Add to oldest byte, and return to
                                       data buffer.
65  11                 ADC TEMP
91  A0                 STA (PNTR), Y
8D  00  F9             STA DAC         Output.
C8                     INY             Go to next point.
D0  E9                 BNE NEXT        Increment pointer (double
                                       precision).
E6  A1                 INC PNTRH
A5  10                 LDA LAST        When end of data buffer is reached,
                                       reset pointer to FIRST and
                                       continue.
C5  A1                 CMP PNTRH
D0  E0                 BNE NEXT
18                     CLC
90  DA                 BCC RESET
```

Listing 6: The use of large amounts of memory can lead to interesting effects, for example this reverberation program. Here a 4 K byte buffer from address space locations 2000 to 2FFF is used to store delayed samples obtained from the input converter at location CONV. This code for the 6502 processor is position independent, provided it is not loaded in the same region as the delay buffer, the page zero constants, or the IO device addresses.

discussed, except that a much larger data buffer is used. Here we can use the indirect form of the LDA and STA instruction, and we maintain a 16 bit pointer in page zero (unlike the 6800, the 6502 has only 8 bit index registers). The routine of listing 6 yields a reverberation time which is adjustable up to about 0.5 seconds. The data buffer is assumed to be the 4 K byte block from addresses 2000 to 2FFF. On each cycle through the buffer, the old data is divided by two, added to the new data output, and returned to the buffer. Thus, the old signals (ie: the echo) die off by a factor of two each time they are heard. You can hear about five or six echos before they drop below audibility.

If START is set to 20, using the whole 4 K buffer, the effect is something like that of a large hall or perhaps an old railroad terminal. The difference is that the computer produces a clear, clean echo at very precisely timed intervals and with a precisely controlled decay rate. Compare this with either a natural reverberation situation or a mechanical unit: the result is a more mechanical sound, much like a tape loop reverb device, without the false resonances of a spring type device. The advantage over a tape loop device is, of course, that it will never wear out or get out of alignment.

Several useful modifications of this program can be made. For example, you could utilize a second digital to analog output and a stereophonic sound system to achieve spacial separation between the direct and "reflected" sound. You could then apply some filtering to the reflected sound channel to simulate selective absorption by the room furnishings. You could also improve the realism of this effect by writing the routine to provide more than one delay time, for example by maintaining two or more buffer pointers which would allow the incoming data to be added to several points in the data buffer. You'll need a fast processor to keep the sampling frequency up, however. Finally, by simply dropping the LSR and ADC instructions in the program of listing 6, you can get a simple time delay effect; say a word and it is repeated immediately. Great for language study; listen to and critique your pronunciation without wearing out your tape recorder. Or if you have lots of memory (at least 32 K), you can get delays long enough to allow you to sing a round with yourself! I won't comment on the frightening social significance of this.

GLOSSARY

Analog to digital converter (often abbreviated **ADC**): Integrated circuit or hybrid module which converts an analog voltage into a parallel digital number, usually in a binary or binary coded decimal format; characterized principally by the number of bits of parallel binary output (the more the better) and the conversion time (the shorter the better). Most commercially available analog to digital converters have from six to 14 bits, convert in 0.5 μs to 200 ms, and cost from $12 to $300 each.
Address decoding: Logic circuitry present in all microcomputer systems which looks for certain addresses on the address bus and outputs a pulse (address strobe) whenever those addresses occur. Used to select individual IO ports, sections of memory, and devices tied to the data bus.
Address strobe: A pulse or logic level generated by the address decode logic in response to the occurrence of a particular address or a range of addresses in a microcomputer.
Aliasing: An instrumental artifact, caused by sampling a periodic waveform less than twice per period, which results in an apparent reduction in the frequency of the waveform. (The effect is quite analogous to the use of a stroboscope to "slow down" the action of periodic mechanical motion.) In audio processing, aliasing sounds like a gross distortion.
Conversion time: The time it takes an analog to digital converter to convert an analog voltage to a binary number. Specifically, it is defined as the time between the strobe pulse and the instant that the digital output is valid.
Cut off frequency: The frequency at which a low or high pass filter *begins* to cut off a signal (which means to reduce its amplitude).
Cut off rate: Also called attenuation rate. The rate at which the response of a low or high pass filter increases attenuation as you go to higher or lower frequencies. The response of a simple single section low pass RC filter drops off only at the rate of a factor of two for every factor of two increase in frequency (called −6 dB per octave in engineering jargon). More sophisticated "active" filters employing operational amplifiers can have much faster cut off rates. These have the advantage of extending the high frequency response as far as possible while still reducing aliasing to an acceptable level.
Digital to analog converter (frequently abbreviated **DAC**): An integrated circuit or hybrid module which converts a parallel binary or binary coded decimal number to an analog voltage or current proportional to the number. Commercially available digital to analog converters have resolutions from eight to 16 bits and cost from $5 to $100.
Data buffer: A section of programmable memory used to store data, usually temporarily.
Fuzz: A kind of distortion occasionally used by electric guitarists for special effect.
Offset binary coding: An arrangement for operation of a bipolar analog to digital converter in which a zero input voltage corresponds to a midscale digital output. For an 8 bit converter with a ±5 V input range, an input of 0 V would be converted to hexadecimal 80, −5 V to hexadecimal 00, and +4.96 V to hexadecimal FF. (This differs from two's complement coding.)
Peak-to-peak: The voltage difference between the average positive excursion and the average negative excursion of an AC signal.
Phase shift: A (usually small) time delay between two similar periodic waveforms.
Phlanging: An audio effect originally produced by playing duplicate tape or disk recordings in almost, but not quite exact, synchronization.
Quantization noise: The noise caused by the conversion of a smooth, continuous analog waveform into a "stair step" approximation in the process of digitization. It adds a "hiss" to audio signals, technically called "white noise." Like any other type of hiss, it can only be partially removed by filtering. The smaller the steps, the less the noise. Thus an 8 bit digitization, yielding 256 discrete steps or "quantization levels," results in a slightly noticeable quantization noise, but in a 12 bit conversion (4096 steps), the effect is quite negligible.
Sample rate: The rate at which the signal waveform is digitized. The larger the number of samples per period of the waveform, the closer the digitized waveform will be to the original analog waveform. The sample rate must be at *least* twice the highest frequency to be digitized in order to prevent aliasing. In this article, the sampling rates are determined by the execution times of the inner loops of the programs.
Strobe: In general, a pulse used for time synchronization of some event. In the context of an analog to digital converter, the term refers to the "start conversion" pulse applied to the converter to initiate the conversion process. In this article, the input conversion strobe is supplied by the microcomputer under software control.
Successive approximation: A popular type of analog to digital converter. Most fast converters are of this type. It performs the conversion bit by bit, starting with the most significant bit and progressing to the least significant bit. Although generally fine for audio processing applications, this type of converter can exhibit nonlinearity (and therefore distortion) if the input signal changes appreciably during conversion. To prevent this, a sample-and-hold circuit can be used ahead of the converter, or, as in this article, one can reduce the problem to insignificance by using a converter with a conversion time much less than the period of the highest frequency passed by the input low pass filter.
Tracking analog to digital converter: A low cost type of converter which uses an up-down binary counter to track or follow the analog input. Its advantage is that it requires no strobe pulse, as its output is always trying to keep up with the output. This type is often implemented in software when conversion time is not particularly important.
Transfer function (or characteristic): The functional relationship between the output and the input of a device.
Wrap around: What happens to your car's mileage indicator after you've driven 99999.9 miles. It "wraps around" to 00000.0. The same thing occurs in electronic counters; in an 8 bit device, the next count after hexadecimal FF wraps it around to 00. Some analog to digital converters do this when the input voltage exceeds full scale. It must be prevented in audio processing. ■