Generating/mixing Sine Waves

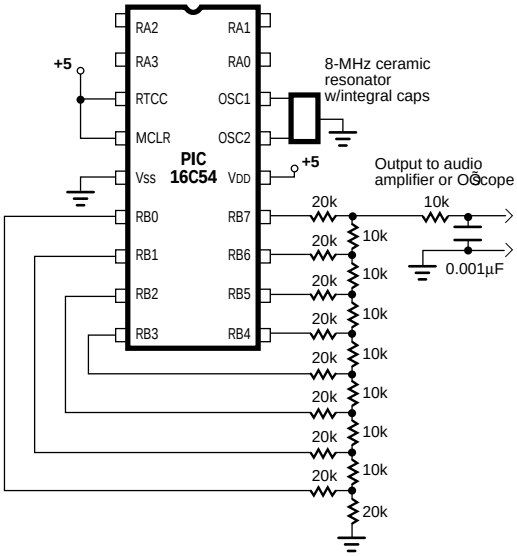
Introduction. This application note illustrates methods for generating audio-frequency waveforms using lookup tables.

Background. Digitally generating sine waves or other complex waveforms may seem like wasted effort compared to using a few discrete components or an IC. If the goal is just to make a pure tone, that view is probably true. However, if you need precise control over the frequency, duration and phase of a waveform, digital generation becomes downright cheap and simple compared to the analog alternatives.

The basic procedure for generating a sine wave is easy enough: At precise time intervals, the program looks up a value from a table, delivers it to a D/A converter, then waits another interval to put out the next value. At the output of the D/A converter, the signal is a connect-the-dots version of a sine wave. The addition of a low-pass filter smoothes the jaggies, and we have a real sine wave.

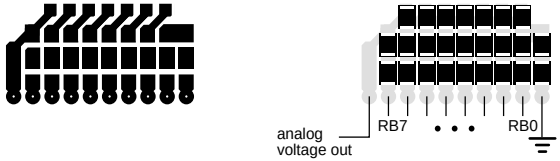
Other waveforms can be made in a similar way, if their patterns repeat within a time interval that can be stored in a reasonably sized lookup table, or computed on-the-fly. If they cannot, then they must be synthesized by combining multiple sine waves.

The program in the listing demonstrates both techniques. When the frequency constants *freq1* and *freq2* are set to the same value, the circuit's output is a pure sine wave. That's because the two pointers *phase1* and *phase2* move through the table of sine values at the same rate. However, if the *freq* constants are different, the circuit produces a mixture of two sine waves.



How it works. The resistors in figure 1 are connected in an array known as an R-2R ladder. This arrangement has an interesting property. It divides each voltage present at its inputs by increasing powers of two, and presents the sum of all these divided voltages at its output. Since the PIC is capable of driving its outputs from rail to rail (0 to +5 volts), the R-2R ladder converts the binary number present on port B into a proportional voltage from 0 to 5 volts in steps of approximately 20 millivolts.

Most commercial D/A converters work on this same principle, but have internal voltage regulators, logic, and latches. Since our demonstration doesn't require any of those things, we can use the resistor array alone. Unfortunately, it can be difficult to find a prefabricated R-2R ladder at the volume parts houses. Figure 2 shows how to construct one from 25 surface-mount 10k resistors and a tiny printed circuit pattern. Alternatively, you may adapt the program listing to drive your favorite packaged D/A converter.



Actual-size pcb pattern Parts placement

Figure 2. Constructing an R-2R digital-to-analog converter.

The program that drives the D/A converter begins by setting port rb to output and clearing its variables. Next it enters this loop:

- Add *freq1* to *acc1*
  > If *acc1* overflows (generates a carry), increment *phase1*
- Put the sine value corresponding to *phase1* into *temp*
- Add *freq2* to *acc2*
  > If *acc2* overflows (generates a carry), increment *phase2*
- Add the sine value corresponding to *phase2* into *temp*
- Copy *temp* to the output
- Do it again.

The loop always takes the same amount of time to execute, so how do *freq1* and *freq2* control the frequency? If the value of *freq1* is 1 and *acc1* starts out at 0, the program will complete 255 loops before acc1 overflows, generating a carry. *Phase1* will increment once in every 256 loops. This will produce a low output frequency. If *freq1* is 100, *acc1* will overflow on the third trip through the loop, so *phase1* will increment much faster, producing a higher frequency.

This scheme, while compact, is not perfect. Take that example of *freq1* = 100. If *acc1* starts off at 0, the sequence goes as shown in the table to the right.

acc1	carry
0	0
100	0
200	0
44	1
144	0
244	0
88	1
188	0
32	1
132	0
232	0
76	1
176	0
20	1

The rate at which carries occur is, on average, proportional to *freq1*. However, the interval between carries can be 1, 2, or 3 trips through the loop. At higher frequencies, these variations become noticeable as a thickening or jittering of traces on the oscilloscope screen. If your application requires particularly pure output, keep this in mind.

Modifications. Play with the values of *freq1* and *freq2* and observe the results on the oscilloscope. Compare the results to a textbook discussion of frequency mixing.

Try removing the resistor/capacitor low-pass filter from the output of the R-2R ladder and look at the change in the output waveform. If you are listening to the output through an audio amplifier, you may find that you prefer the sound of the unfiltered D/A converter. The harmonics it produces give the tone more sparkle, according to some listeners.

Perhaps the most useful modification is this: Set *freq1* and *freq2* to the same value, say 100. Run the program and listen to the tone or observe it on the ‘scope. If you use a scope, make note of the peak-to-peak amplitude of the tone. Now substitute *mov phase1, #7* for *clr phase1* at the beginning of the program. Run the program again. Do you hear (see) how the amplitude of the tone has fallen off? Try once more with *mov phase1,#8*. No tone at all. This is a graphic illustration of how out-of-phase sine waves of the same frequency progressively cancel each

other. The 16 sine table entries represent amplitudes at intervals of 22.5 degrees. When *phase1* is initialized with a value of 8, while *phase2* is zero, the phase shift between the two is 22.5 x 8 = 180 degrees. Two sine waves 180 degrees out of phase cancel each other totally. Smaller phase shifts produce proportionally less attenuation.

Given the PIC's limited math capabilities, this phase shifting technique is an easy way to control the overall amplitude (volume) of the tones you generate.

Program listings. This program may be downloaded from our Internet ftp site at ftp.tech-tools.com. The ftp site may be accessed directly or through our web site at http://www.tech-tools.com.

```
; Program: Synthesizing sine waves (SINE.SRC)
; This program mixes two sinewave signals and outputs them to port b for
; conversion to analog by an R-2R ladder D/A converter.

; Device and reset vector (remember to change if programming a different part).
                device  pic16c54,xt_osc,wdt_off,protect_off
                reset start

output          =       rb              ; Ladder DAC on port b.
freq1           =       100             ; Sets frequency 1.
freq2           =       50              ; Sets frequency 2.

; Set aside variable space above special-purpose registers.
                org     8
acc1            ds      1               ; Accumulator for phase 1.
acc2            ds      1               ; Accumulator for phase 2.
phase1          ds      1               ; Position in sine table, p1.
phase2          ds      1               ; Position in sine table, p2.
temp            ds      1               ; Temporary variable.

                org 0

start           mov     !rb,#0          ; Make rb an output.
                clr     acc1            ; Clear the accumulators.
                clr     acc2
                clr     phase1          ; Clear phase pointers.
                clr     phase2
:loop           add     acc1,#freq1     ; Add freq1 to acc1
                addb    phase1,c        ; If carry, increment phase1.
                AND     phase1,#00001111b ; Limit to 0-15.
```

```
        add     acc2,#freq2         ; Do the same for
        addb    phase2,c            ; phase2.
        AND     phase2,#00001111b
        mov     w,phase1            ; Look up sine value in table.
        call    sine
        mov     temp,w              ; Store 1st sine value in temp.
        mov     w,phase2
        call    sine                ; Get value from table.
        add     temp,w              ; Add sines together.
        mov     output,temp         ; Output to D/A.
        goto    :loop               ; Do forever.

; Notice that the sine values in the table below are scaled so that no pair adds to
; more than 254. This prevents overflowing the variable temp when the phases are
; added together. If you add more phases, adjust the maximum values in this table
; appropriately. The values are in increments of 22.5 degrees (0.3927 radians).
sine    jmp     pc+w
        retw    64,88,109,122,127,122,109
        retw    88,64,39,19,5,0,5,19,39
```