Project 2: Wavetable Synthesis

In this project, we will code a synthesizer capable of producing sounds similar to a musical instrument. The technique we will use is called "wavetable synthesis" and forms the basis behind virtually all electronic music synthesizers, PC sound cards, and arbitrary waveform generators (AWG). In order to do this we will need to understand how to synthesize or generate waveforms (signals) with the DSP and the mechanics of sound generation (why certain instruments sound the way they do).

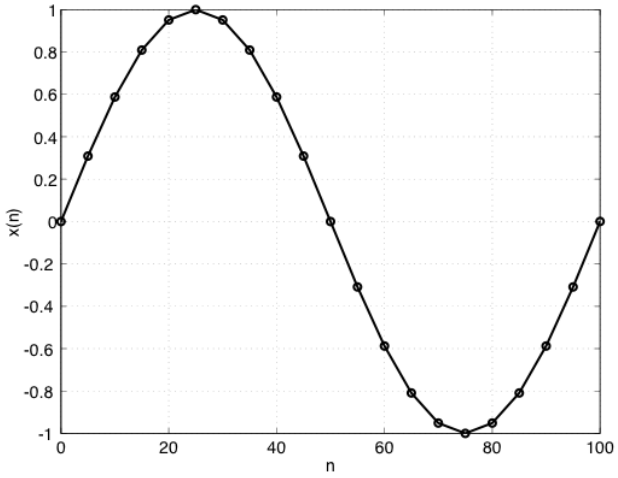
Sine Wave Synthesis

The generation of sinusoids or tones or waves is a widely used function of DSPs in audio, communications, and control applications. High-speed, high-precision DSPs are capable of synthesizing stable and low distortion sinusoids of any frequency. These sinusoids can be produced digitally using either truncated Taylor series or "Lookup Tables." We will examine the later.

Suppose we calculate the values of L evenly-spaced points on a sinusoid

$$x[n] = \sin(2\pi n / L), \quad 0 \le n \le L-1$$

and store them in memory to form a lookup table.



Base Address	0
Base Address + 1	$\sin(2\pi/L)$
Base Address + 2	$\sin(4\pi/L)$
	.
	.
	.
Base Address + L – 1	$\sin[2\pi(L-1)/L]$

If we read out one table entry every sample period, the frequency of the sinusoid produced at the D/A is then

$$\omega_0 = 2\pi / L \quad \text{(rads/sample)}$$

or

$$\omega_0 = \frac{2\pi}{L} f_s \quad \text{(rads/s)}$$

or

$$f_{\text{FUND}} = \frac{f_s}{L} \quad \text{(Hz)}$$

where f_s is the sample rate of the D/A. When one table entry is read out every sample period, the frequency of the sinusoid is referred to as the *fundamental table frequency* (FTF) or f_{FUND}. The *fundamental period*, T_{FUND} of the sinusoid is given by

$$T_{\text{FUND}} = 1/ f_{\text{FUND}}.$$

Example: Assume the sampling rate is f_s. = 16,000 samples/second and the lookup table has L = 256 entries. Then

$$f_{\text{FUND}} = 16000/256 = 62.5Hz$$

and

$$T_{\text{FUND}} = 256/16,000 = 16 \text{ ms}.$$

Other frequencies can be generated from the lookup table through the use of a *table increment*. If the table increment, Δ = 1, the table entries are read consecutively; if Δ = 2, every other table entry is read; if Δ = 3, every third table entry is read; and so on. Using a table increment, we cycle through the lookup table Δ-times faster and thus the frequency of the sinudoid at the D/A will be

$$f = \Delta f_{\text{FUND}},\ 1 \le \Delta \le L/2.$$

The maximum value Δ can be is L / 2 since at least two samples per cycle are required to synthesize a sine wave without aliasing (result of Nyquist theory). The total harmonic distortion (THD) of the synthesized wave depends on the length of the table, L and the accuracy (number of bits of precision) of the data stored in the lookup table. Clearly, if Δ is an integer, only those frequencies which are integer multiples of the fundamental frequency can be generated

Δ	f (f_s = 16kHz, L = 256)
1	62.5
2	125.0
.	.
.	.
.	.
L / 2	8,000.0

Wavetable Synthesis for Motorola DSP56300

Generating the Lookup Table

The first step in synthesizing sinusoids is to create a lookup table containing the values of the sinusoid using

$$\texttt{BASE\_ADDRESS} + l = \sin(2\pi l / L)$$

where BASE_ADDRESS is the base address of the lookup table. The following macro stored as **`sinelut.asm`** and called before the main event loop, will load the values defined by (7.6) into memory.

```
sinelut     macro       SineTableLen,SineBaseAddr
; sinelut               - macro to generate sine coefficient lookup table
; SineTableLen          - Length of Sine table
; SineBaseAddr          - base address in Y memory of sine lookup table

pi    equ   3.141592654
freq  equ   2.0*pi/@cvf(SineTableLen)

      org   y:SineBaseAddr
count set   0
      dup   SineTableLen
      dc    @sin(@cvf(count)*freq)
count set   count+1
      endm

      endm  ;end of sinelut macro
```

Sinusoid Synthesis with Integer Delta

The first step in the sinusoid synthesis with integer Δ is to define the lookup table length and allocate memory for the circular queue to store the table values. This is done in the **`pass_sin.dat`** file:

```
LUT_LENGTH  equ   256         ;length of lookup table 256
.
.
.
      org y:$000000
SINE        dsm   LUT_LENGTH
DELTA       dc    8           ;integer delta
```

Next, in **`pass_sin.asm`** we insert lines to include the **`sinelut.asm`** macro file and call the macro in order to generate table values. We also include the **`singenid.asm`** file which generates the sinusoid sample:

```
.
.
.
      include 'pass_sin.dat'        ;include user data

      include 'sinelut.asm'
      sinelut LUT_LENGTH,SINE       ;build sine lookup table with macro

      org p:$100
.
.
.
      include 'ada_init.asm'
      include 'proginit.asm'
      include 'procster.asm'
      include 'singenid.asm'        ;integer delta
echo
      end
```

Next, we include lines to initialize a pointer into the table with the correct modifier and offset values in the **`proginit.asm`** file:

```
proginit
      move #SINE,r4
      move #LUT_LENGTH-1,m4
      move y:DELTA,n4
      rts
```

Next, in **`singenid.asm`** we place our sinusoid subroutine in order to read the lookup table:

```
sinusoid
      move y:(r4)+n4,x0 ;read sample from table, post increment by delta
      rts
```

Finally, we include a line to call the sinusoid subroutine in the **`procster.asm`** file:

```
process_stereo
      jsr sinusoid
```

```
        rts
```