Two methods of calculating square roots in DSP applications are described in the projects section (see separate PDF file), and these are presented here.

Project D: Newton's Method for Square Roots in QuickBasic 4.5

The file, *nwtnsqrt.bas* is an example of Newton's method. This is a generic *BASIC* program that computes the root of a 32-bit integer to within an error margin, DERROR. The root of a 32-bit integer is naturally a 16-bit integer. Emphasis is placed in what follows on speed of execution and accuracy as influenced by truncation and rounding. 32-bit integer variables are defined DEFLONG, 16-bit integers are DEFINT. Integer math in *QuickBasic* is much faster than floating-point math.

Newton's method iteratively converges on a result. Experience has shown that three to six iterations are necessary to obtain best accuracy for a 16-bit result, but here we execute as many iterations as necessary to obtain accuracy DERROR, initially defined to be one least-significant bit or $1/(2^{15}) \approx 30 \times 10^{-6}$. Note that if DERROR is small or zero, convergence may never be reached because of quantization noise. A loop counter, K, is established to count iterations. The program displays on the computer screen the argument, its root and the iteration count. Users may readily modify the program to use random numbers as arguments to time the number of roots per second it calculates.

As an experiment, try changing the line of code that produces the initial guess, OLDGUESS. An interesting try would be to set it to a low constant such as zero. When this is done, the iteration count rises for almost all arguments. This illustrates the value of a good initial guess with Newton's method.

Project E: A Fast Square-Root Algorithm Using a Small Look-Up Table

The file, *fsqrt.txt* is a machine-language example of a fast square-root algorithm. The target processor in this case is the Motorola MC68HC16Z1, a 16-bit, fixed-point DSP.

Appendix: DSP Projects

PROJECT A: DECIMATION

This project illustrates the concept of decimation using Alkin's *PC-DSP* program, included with the book of that name listed in the **Bibliography**. First, generate 40 samples of the sinusoid $y(n) = \sin(n/4)$, where $0 \leq n \leq 39$. This sequence may be generated using the "Sine" function of the "Generate" sub-menu under the "Data" menu, with parameters Var1 = SIN, A = 1, B = 0.25, C = 0 and #Samples = 40. Press F2 to display the data, which should match **Fig 16.A1**.

Next, decimate the sequence by a factor of 2 using the "Decimate" function found in the "Process" sub-menu under the "Data" menu. Use parameters Var1 = SIN2, Var2 = SIN, Factor = 2. Display the new sequence by pressing F2. It should match **Fig 16.A2**.

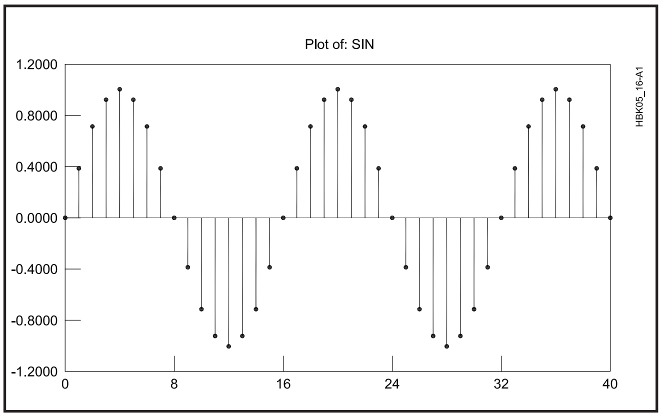


Fig 16.A1—A 40-sample sine wave.

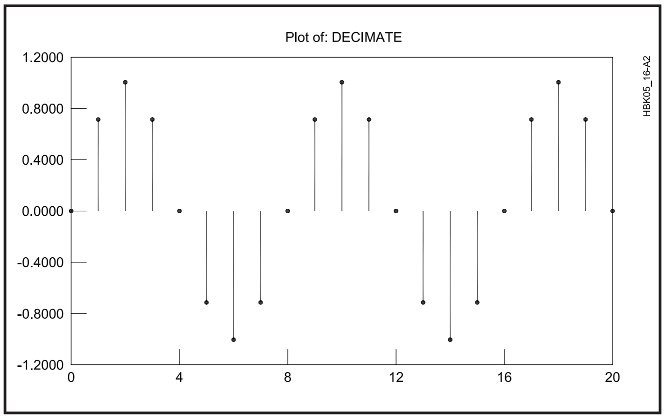


Fig 16.A2—Decimated, 20-sample sine wave.

PROJECT B: FIR FILTER DESIGN VARIATIONS

An FIR filter's ultimate attenuation and its transition BW are largely determined by the filter's length: the number of taps used in its design. Fourier and other design methods do not always readily optimize the trade-off among transition BW, ultimate attenuation and ripple. One way to achieve better ultimate attenuation at the expense of passband ripple is to convolve the impulse responses of two short filters to obtain a longer filter. The two impulse-response sequences are processed by precisely the same convolution sum that is used to compute FIR filter outputs (Eq 3 in the main text).

A filter obtained by convolving two filters of length L has length 2L –1. In one example, two LPFs of length 31 may be convolved to produce a filter of length 61. The resulting frequency response, plotted against that of a LPF designed with Fourier methods for an identical length of 61 taps, would show that the ultimate attenuation of the convolved filter is 20 dB or 10 times greater than that of the plain, Fourier-designed filter. Also, the convolved filter would have a greater passband ripple and a narrower transition region. Quite often, filters that were designed using different window functions may be convolved to get some of the benefits of each in the final filter.

A look back at Fig 16.29 reveals that different window functions achieve different transition BWs and values of ultimate attenuation. The rectangular window attains a narrow transition BW, but a poor ultimate attenuation; the Blackman window, on the other hand, has nearly optimal ultimate attenuation and a moderate transition BW. Let's see what happens when we convolve the impulse responses of filters designed using each method. We will constrain ourselves to filters with odd numbers of taps so that the convolved impulse response will also have an odd number of taps.

Using your favorite filter-design software, first design a LPF by the Fourier method with a length of 31, using a rectangular window, and a cut-off frequency (–6 dB point) of $0.25f_s$. Its frequency response is shown in **Fig 16.B1A**. We produce a second filter having the same cut-off frequency of $0.25f_s$ using a Blackman window, whose response is shown in **Fig 16.B1B**. The response of the filter formed by the convolution of the two filters is shown in **Fig 16.B1C**, along with that of a standard Fourier-designed LPF. The final filter has length 61 taps. Notice that the filter obtains the benefits of the rectangular window's sharp transition region and those of the Blackman window's good ultimate attenuation.

A second advantage may be garnered by convolving two different filters in that their responses may be governed separately, while producing desired changes in frequency (or phase) response. A good example of this arises when it is desired to alter the audio response of an SSB transmitter (or receiver), but keep the ultimate attenuation characteristics the same. A long BPF with excellent transition properties may be convolved with a much shorter filter that is manipulated to provide the desired passband response.

FIR filters used in Amateur Radio transceivers must usually have at least 60 dB ultimate attenuation. This generally requires at least 63 taps. As our second FIR filter variation, let's consider a case wherein we want to customize an IF-DSP transmitter's frequency response without impacting opposite-sideband rejection. We will use a 99-tap BPF in each leg of a Hilbert transformer (as part of an SSB modulator) whose response is convolved with that of a 31-tap filter describing the variation in frequency response we want. The 99-tap fixed filter has the frequency response shown in **Fig 16.B2A**. The 31-tap filter has been designed using Fourier methods to have a 6 dB/octave rise in its frequency response, as shown in **Fig 16.B2B**.

The frequency response of the convolution of the two filters' impulse responses is shown in **Fig 16.B2C**. It is important to note that the net response is that of the *product* of the two filters' frequency responses; that is, if $H_1(\omega)$ and $H_2(\omega)$ are the two frequency response functions, the final response is simply:

$$H_{composite}(\omega) = H_1(\omega) H_2(\omega) \qquad (B1)$$

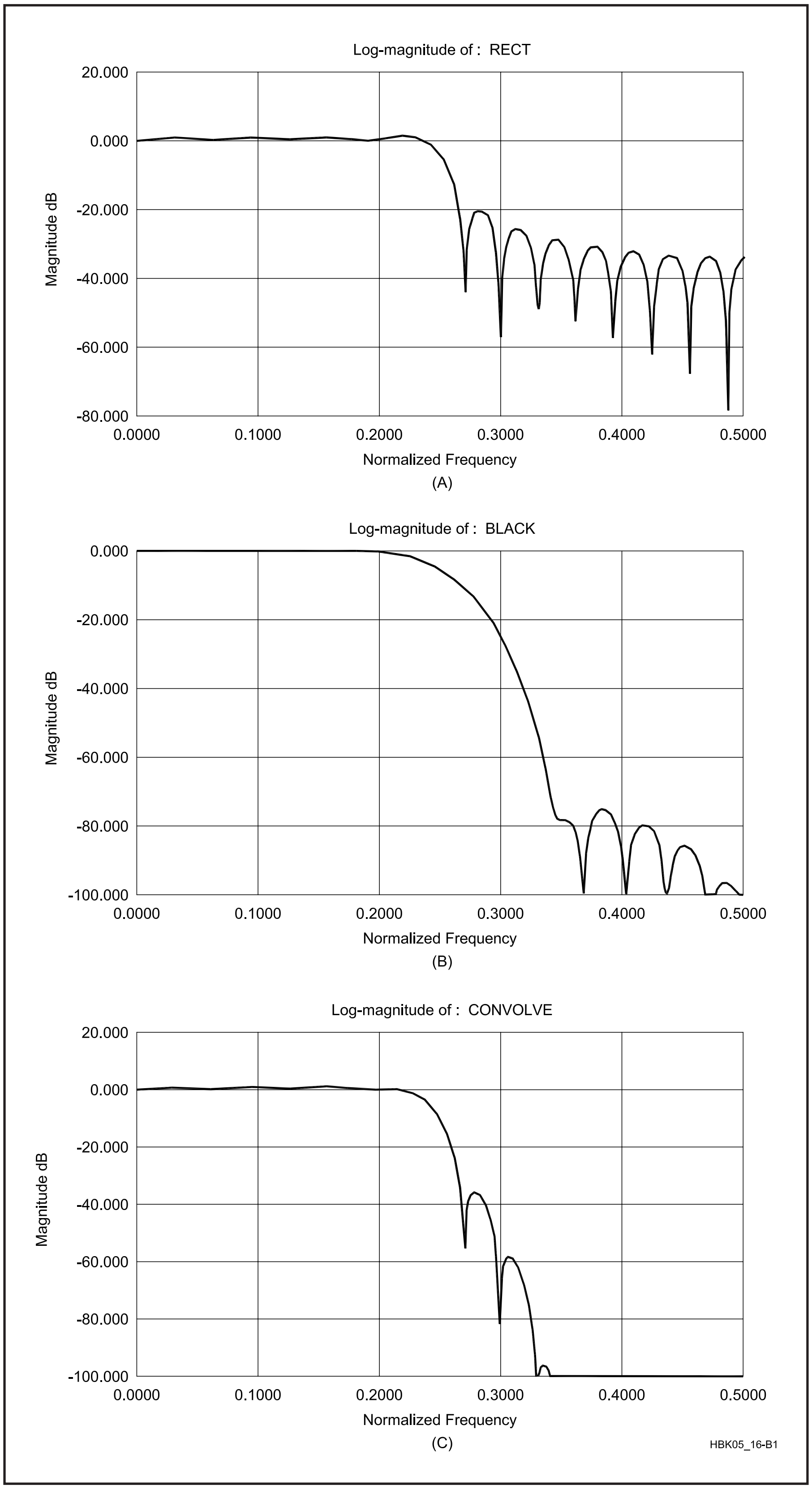


Fig 16.B1—LPF frequency response, rectangular window (A). LPF frequency response, Blackman window (B). LPF frequency response, convolution of filters shown in A and B (C).

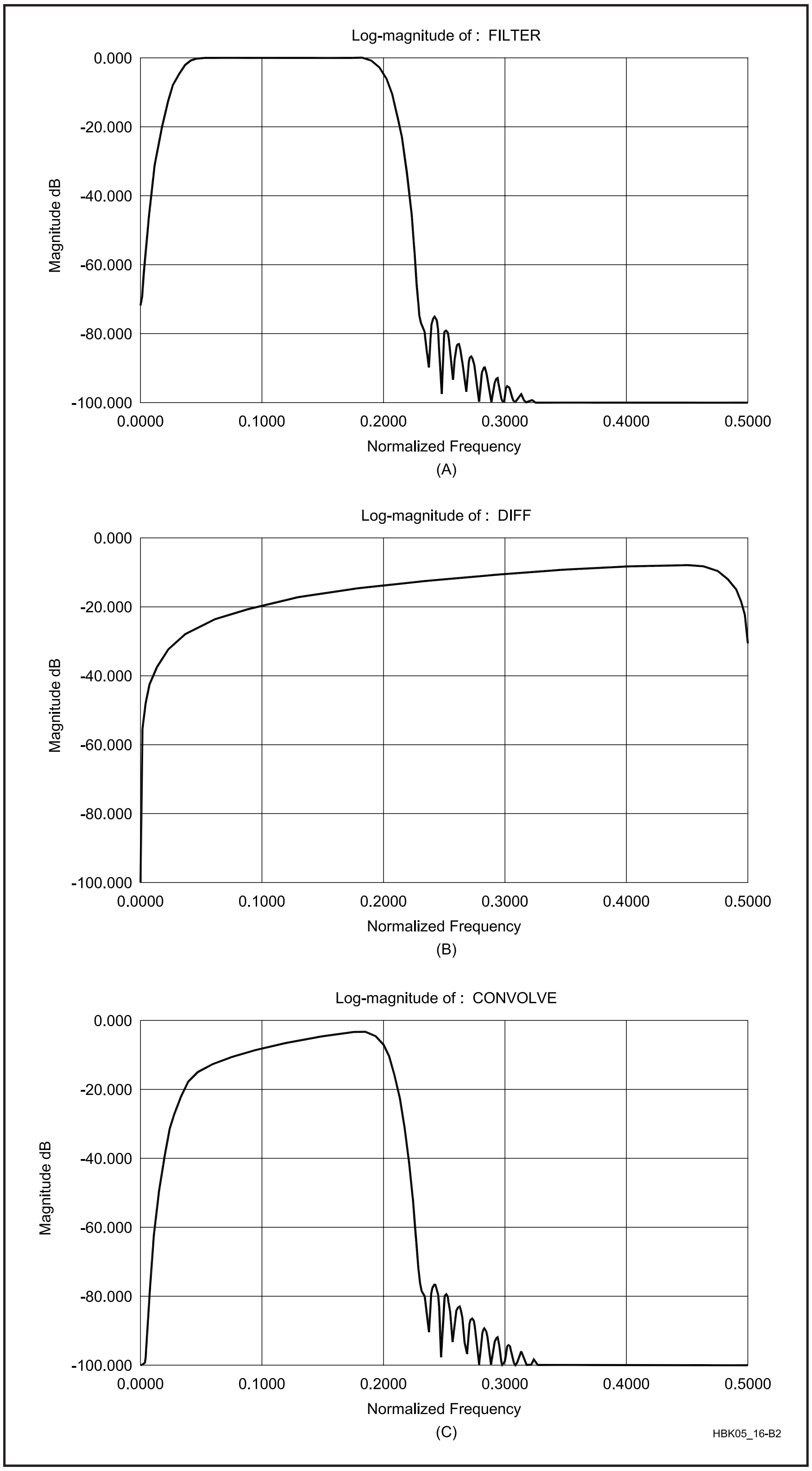


Fig 16.B2—BPF for SSB use, L = 99 (A). LPF having rising frequency response, L = 31 (B). Frequency response of convolution of filters shown in A and B (C).

PROJECT C: ANALYTIC FILTER PAIR GENERATION

Frequency-translation properties of complex multiplication work just as well on the responses of filters as they do on real signals. In this project, we will explore just how these properties are applied to the generation of analytic filter pairs. Analytic filter pairs are used to produce complex signals from real signals for the purposes of modulation, demodulation, and other processing algorithms.

An analytic filter pair consists of two filters (usually BPFs) whose frequency responses are identical, but whose phase responses differ at every frequency by 90°. These filters are used in legs of a Hilbert transformer, as shown in **Fig 16.C1**. The creation of these filters begins with the design of a LPF prototype having the desired passband, transition-band, and stopband characteristics. Such a prototype filter, as might suffice for an SSB receiver, would have a frequency response such as that shown in **Fig 16.C2A**.

The filter's impulse response (L = 63) is then multiplied by a sine-wave sequence (also L = 63) whose frequency represents the amount of upward translation applied to the LPF's frequency response. If the sine wave is high enough in frequency, the resulting impulse response is a BPF filter centered on ω_0, the sine wave's frequency. See **Fig 16.C2B**. Likewise, the prototype LPF's impulse response is multiplied by a cosine-wave sequence to produce a filter having the same frequency response as that of the sine-wave filter, but with a phase response differing by 90°. Sample-by-sample multiplication occurs according to Eq 21 in the main text.

When an analytic filter pair is used in a demodulator, IF shift may be included by varying the frequency of w_0. In combination with various filter BWs, IF shift is useful in avoiding interference by modifying a receiver's frequency response. Further modification may be obtained by convolving each filter in the analytic pair with a filter having the desired characteristic. The phase relation between the filters in the pair will not be altered by the convolution.

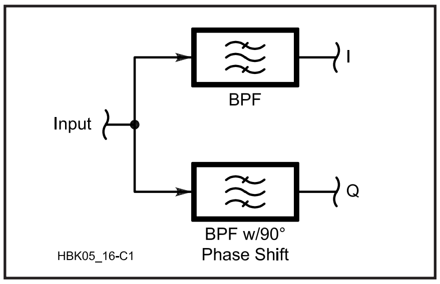


Fig 16.C1—Hilbert transformer using an analytic filter pair.

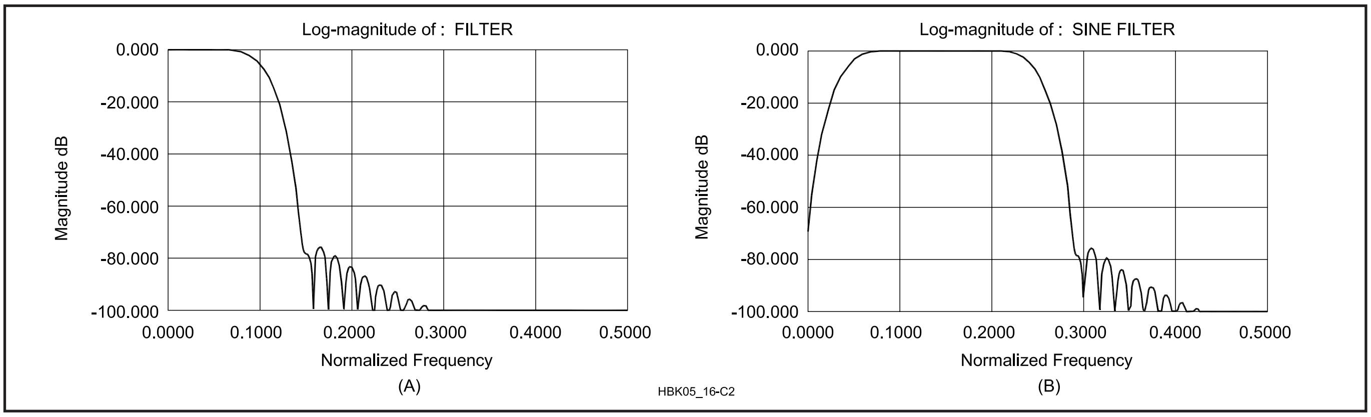


Fig 16.C2—LPF prototype frequency response (A). BPF frequency response of processed impulse response (B).

PROJECT D: NEWTON'S METHOD FOR SQUARE ROOTS IN QUICKBASIC 4.5

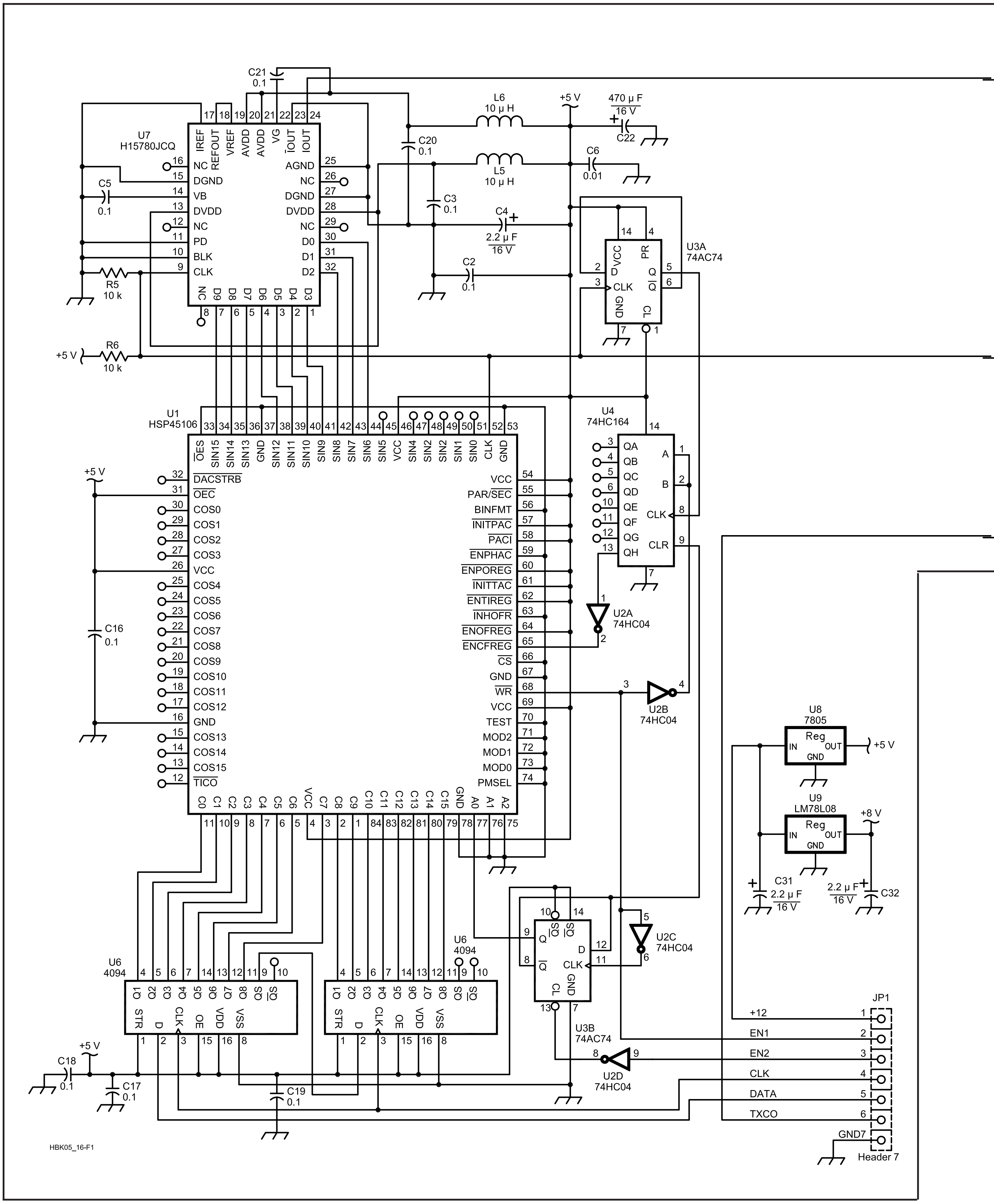
In this example of Newton's method, a generic *BASIC* program is given that computes the root of a 32-bit integer to within an error margin, DERROR. The root of a 32-bit integer is naturally a 16-bit integer. Emphasis is placed in what follows on speed of execution and accuracy as influenced by truncation and rounding. 32-bit integer variables are defined DEFLONG, 16-bit integers are DEFINT. Integer math in *QuickBasic* is much faster than floating-point math.

As described in the **AM Demodulation** section in the main text, Newton's method iteratively converges on a result. Experience has shown that three to six iterations are necessary to obtain best accuracy for a 16-bit result, but here we execute as many iterations as necessary to obtain accuracy DERROR, initially defined to be one least-significant bit or $1/(2^{15}) \approx 30 \times 10^{-6}$. Note that if DERROR is small or zero, convergence may never be reached because of quantization noise. A loop counter, K, is established to count iterations. The program displays on the computer screen the argument, its root and the iteration count. Users may readily modify the program to use random numbers as arguments to time the number of roots per second it calculates.

The program is included in the *2002 ARRL Handbook* companion software. The software is available for free download from *ARRLWeb* at: **www.arrl.org/notes**.

PROJECT E: A FAST SQUARE-ROOT ALGORITHM USING A SMALL LOOP-UP TABLE

This project is a machine-language example of a fast square-root algorithm. The target processor in this case is the Motorola MC68HC16Z1, a 16-bit, fixed-point DSP. The method is depicted in **Fig 16.16** in the main text. Like the previous project, this is included in the *2002 ARRL Handbook* companion software. The software is available for free download from *ARRLWeb* at **www.arrl.org/notes**.



C21 0.1
U7 H15780JCQ
IREF REFOUT VREF AVDD AVDD VG IOUT IOUT
NC DGND VB DVDD NC PD BLK CLK
AGND NC DGND DVDD NC D0 D1 D2
NC D9 D8 D7 D6 D5 D4 D3
C5 0.1
R5 10 k
R6 10 k
+5 V
L6 10 μ H
L5 10 μ H
C20 0.1
C3 0.1
C4 2.2 μ F 16 V
C2 0.1
C6 0.01
470 μ F 16 V C22
U3A 74AC74
U1 HSP45106
U4 74HC164
QA QB QC QD QE QF QG QH A B CLK CLR
U2A 74HC04
U2B 74HC04
U2C 74HC04
U2D 74HC04
U3B 74AC74
C16 0.1
OES SIN15 SIN14 SIN13 GND SIN12 SIN11 SIN10 SIN9 SIN8 SIN7 SIN6 SIN5 VCC SIN4 SIN2 SIN2 SIN1 SIN0 CLK GND
DACSTRB OEC COS0 COS1 COS2 COS3 VCC COS4 COS5 COS6 COS7 COS8 COS9 COS10 COS11 COS12 GND COS13 COS14 COS15 TICO
VCC PAR/SEC BINFMT INITPAC PACI ENPHAC ENPOREG INITTAC ENTIREG INHOFR ENOFREG ENCFREG CS GND WR VCC TEST MOD2 MOD1 MOD0 PMSEL
C0 C1 C2 C3 C4 C5 C6 VCC C7 C8 C9 C10 C11 C12 C13 C14 C15 GND A0 A1 A2
U8 7805 Reg IN OUT GND
U9 LM78L08 Reg IN OUT GND
+8 V
C31 2.2 μ F 16 V
C32 2.2 μ F 16 V
U6 4094
Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 QS QS STR D CLK OE VDD VSS
C18 0.1
C17 0.1
C19 0.1
JP1
+12 EN1 EN2 CLK DATA TXCO GND7
Header 7
HBK05_16-F1

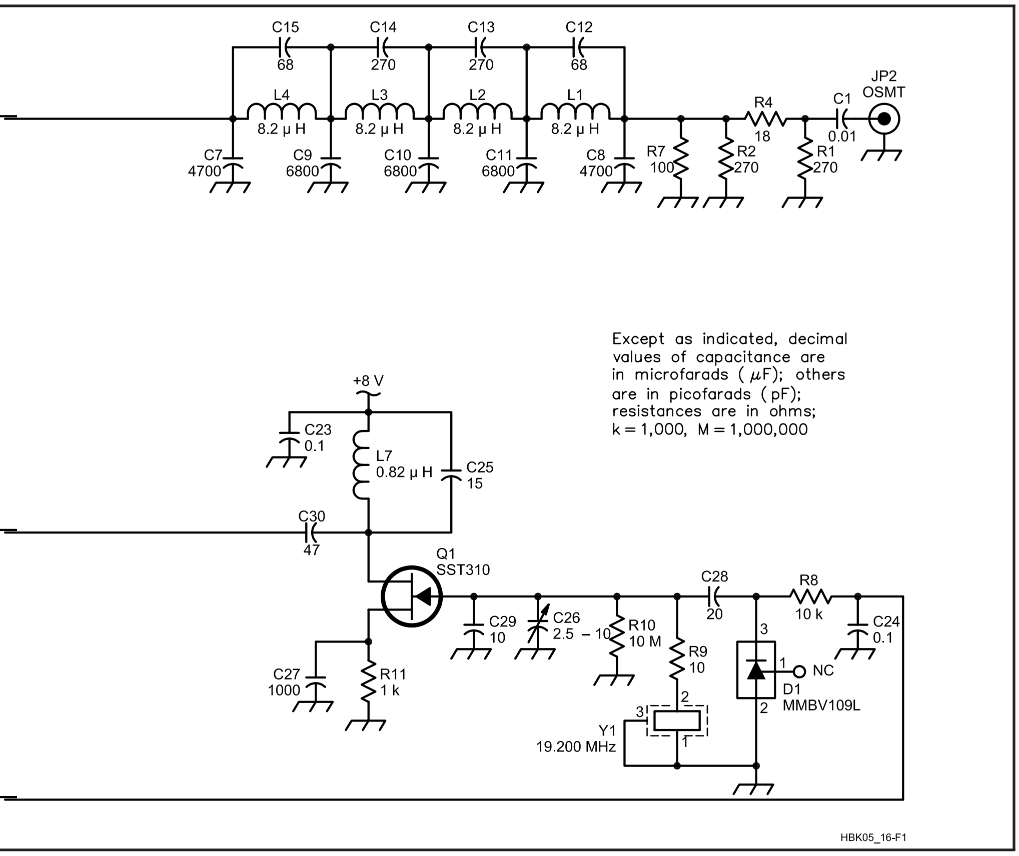


Fig 16.F1—High-performance DDS schematic diagram.

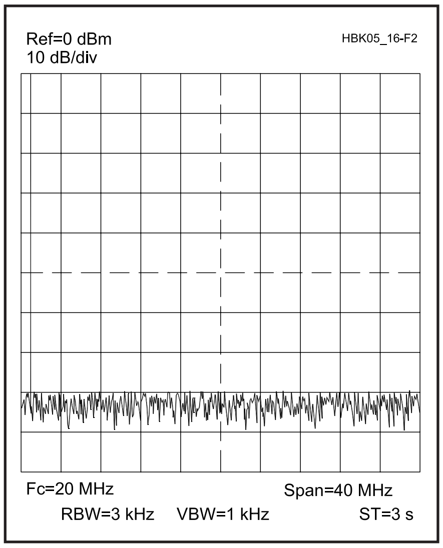


Fig 16.F2—Typical output spectrum of DDS.

PROJECT F: A HIGH-PERFORMANCE DDS

A DDS is described below that is used as a reference for a PLL. See **Fig 16.F1**. This DDS is designed to cover a small range of frequencies near 1 MHz. A crystal-oscillator clock at 19.2 MHz is applied to both the DDS, a Harris/Intersil HSP45106, and the DAC, a Harris/Intersil HI5780. Making the DDS output frequency a small fraction of the clock frequency makes it relatively easy to obtain excellent spurious performance. PM spurs are limited to –90 dBc and AM spurs to about –60 dBc. If the output is not squared at the input to a PLL chip, an external Schmitt-trigger squaring stage may be added, eliminating virtually all the AM spurs prior to the LPF.

The LPF at the output of the circuit is a 4-section elliptical type. Design impedance is 100 Ω. This filter cuts out many high-frequency spurs and stops clock feed-through. The DAC's 10 input lines are fed from the 10 most-significant bits of one of the DDS's outputs. The HSP45106 has two 16-bit outputs (sine and cosine) to accommodate the needs of complex-mixer designs, but only one is being used here.

The DDS chip itself is programmed using a 16-bit parallel interface. This is transformed into a serial interface by shift registers U5 and U6, divider U3 and counter U4. Each time the frequency is changed, an internal 32-bit phase-increment accumulator must be updated. The phase increment is just f_{out}/f_{clk}, expressed as a 32-bit, unsigned fraction. This value is written into the chip in two 16-bit segments, most-significant bit of the most-significant word first.

During serial programming, a data bit is placed on the DATA line by the host microprocessor; the clock line is toggled high, then low to shift the bit into the shift registers. After the first 16 bits have been shifted, they are written into the DDS by toggling the ENABLE line. Counter U4 supplies the necessary write pulse with appropriate timing. The remaining 16 bits are then shifted and written to the chip, completing the operation.

An example of the output spectrum of this circuit is shown in **Fig 16.F2**. Components are surface-mount types and care must be exercised during construction. See Ulbing's article in the **Bibliography** for information on surface-mount soldering techniques.

PROJECT G: A FAST BINARY MULTIPLIER IN HIGH-SPEED CMOS LOGIC

In this project, a fast 4-bit binary multiplier is described that may be constructed from 'HC-series logic gates or programmed into an FPGA. Two variations are explored: one without pipelining, and one with pipelining. Pipelining is employed where the propagation delays of gates limit throughput.

As seen in Fig 16.45 in the main text, a 4-bit multiplication may be broken into several 4-bit additions. In our circuit, 4-bit adders are used to add rows of bits in the summation, each one producing a single output bit. The diagram of a fast, 4-bit adder with look-ahead carry is shown in **Fig 16.G1**.

In this multiplier, 4-bit adders are used to add adjacent rows of bits in the traditional way. A multiplier connected this way is shown in **Fig 16.G2**. Not all bits in each addend have mates in the other, so 4-bit adders suffice. In the case where execution speed exceeds the reciprocal of the total propagation delay, pipelining must be employed to avoid error.

To use pipelining, we place storage registers between the stages of addition and one interim result is held by each stage at each clock time. See **Fig 16.G3**. The result is the same, but appears only after a latency of three clock times. When maximum gate delays are well known, this approach also yields more predictable performance because the latency is independent of the input data.

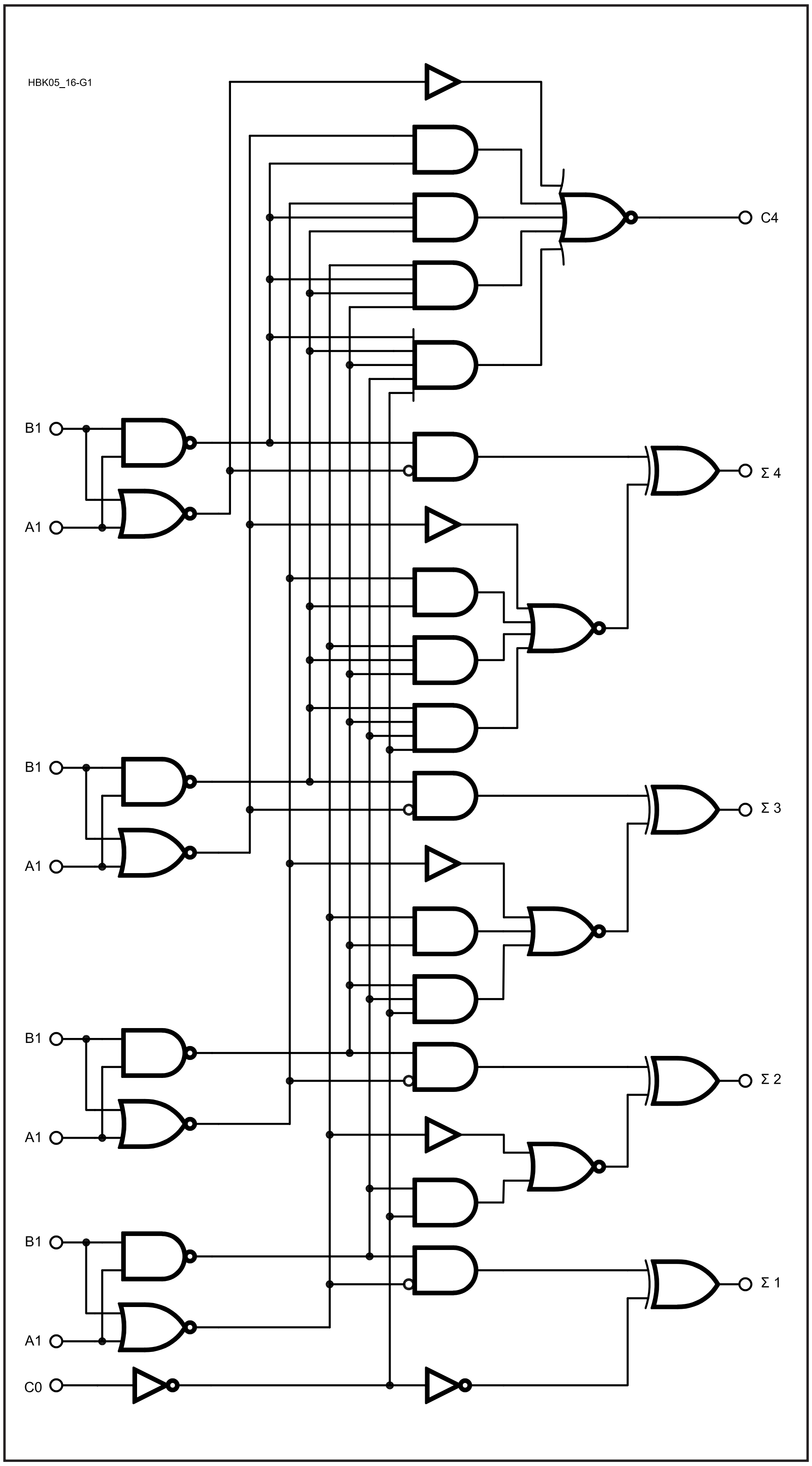


Fig 16.G1—A 4-bit adder schematic diagram.

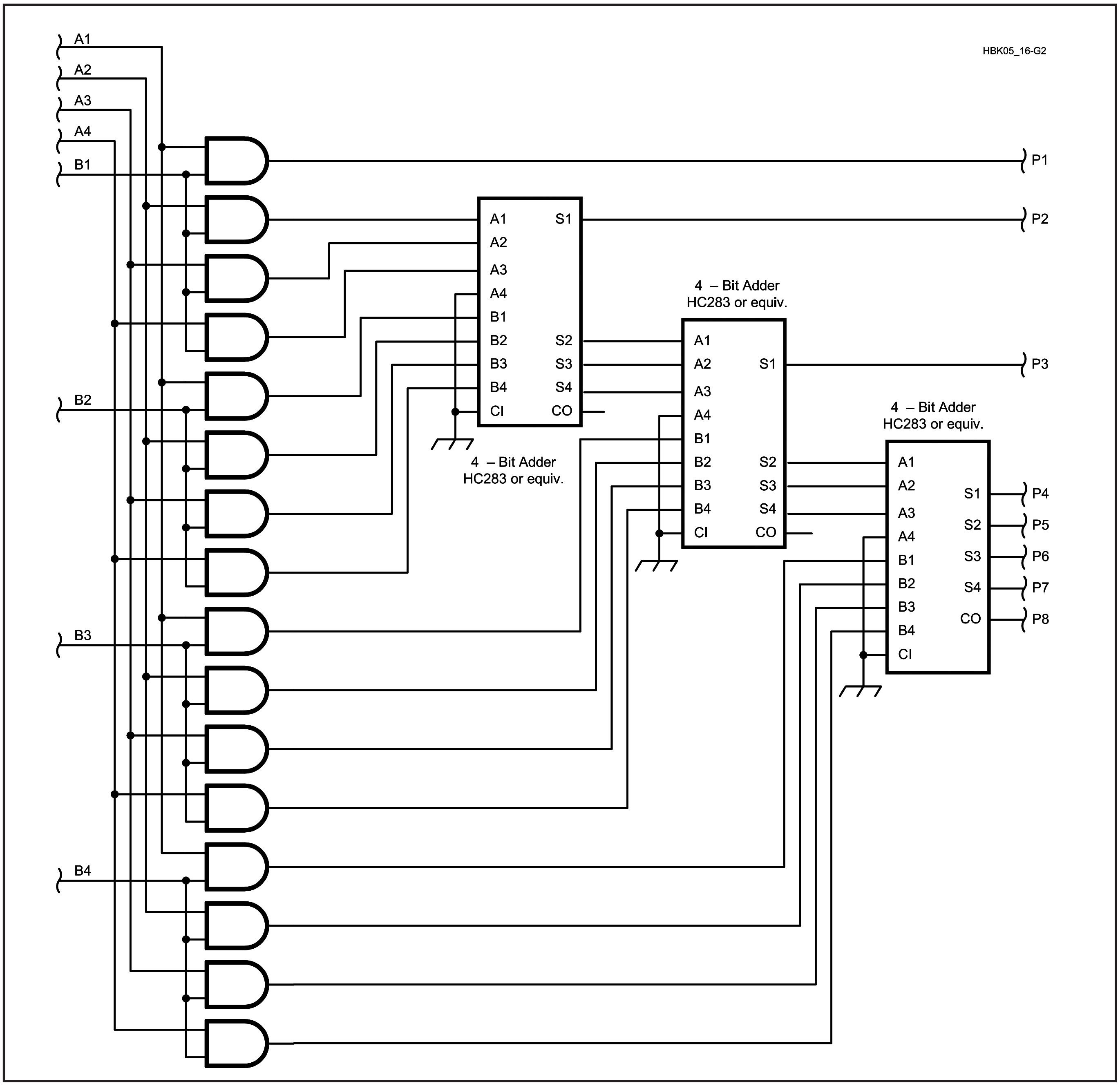


Fig 16.G2—Complete 4-bit multiplier, no pipelining.

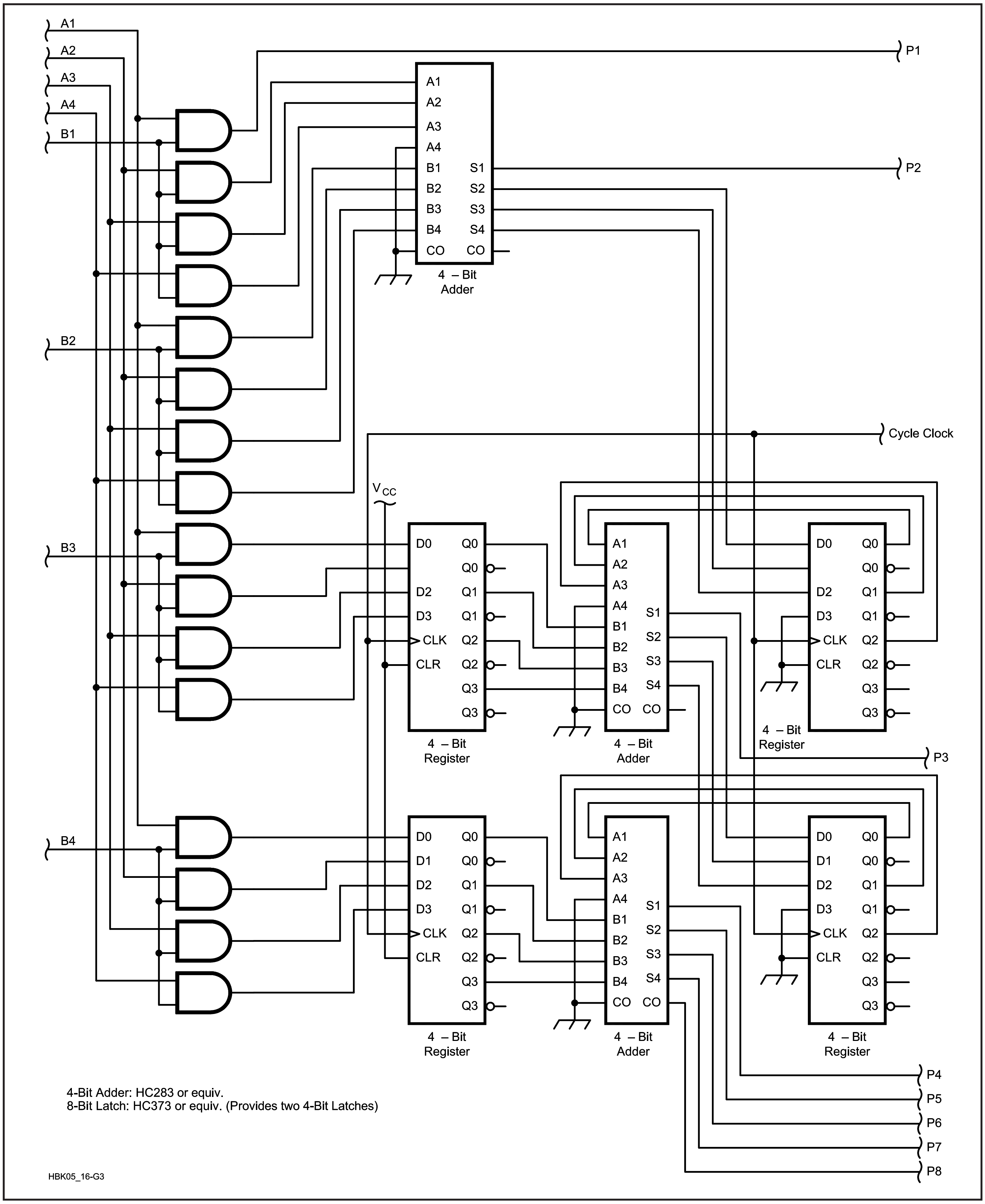


Fig 16.G3—Complete 4-bit multiplier with pipelining.

The files in this directory present sample calculations to supplement the material in chapter 15, "DSP and Software Radios". The .mcd files were generated with Mathcad 7 and the .pdf files are Adobe Acrobat versions of those files.

      exp_decay.mcd exp_decay.pdf
Compares the exponential decay generated by a digital and an analog circuit.

      fourier_conv.mcd fourier_conv.pdf
Demonstrates how to convolve sine waves with a signal to determine its frequency content. This is the same basic method used by the Fourier transform.

      quant_rand.mcd quant_rand.pdf
Quantizing a random signal results in quantization noise.

      quant_sine.mcd quant_sine.pdf
Quantizing a periodic signal results in quantization spurs. The spurs can be reduced by dithering.

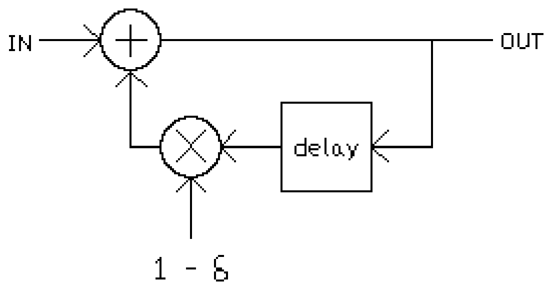
      sampling.mcd sampling.pdf
Sampling in the time domain results in periodicity in the frequency domain (alias frequencies).

      sinc.mcd sinc.pdf
Linear and dB plots of the sinc function.

      sine_aprox.mcd sine_aprox.pdf
Generating a sine wave using a fifth-order curve fit.

      windowing.mcd windowing.pdf
Demonstration of the windowing method of calculating FIR filter coefficients.

Exponential decay implemented digitally N1AL 6/8/2009



Let's compare the exponential decay of the above circuit to that of an analog R-C network to see if we have the correct equation for the time constant.

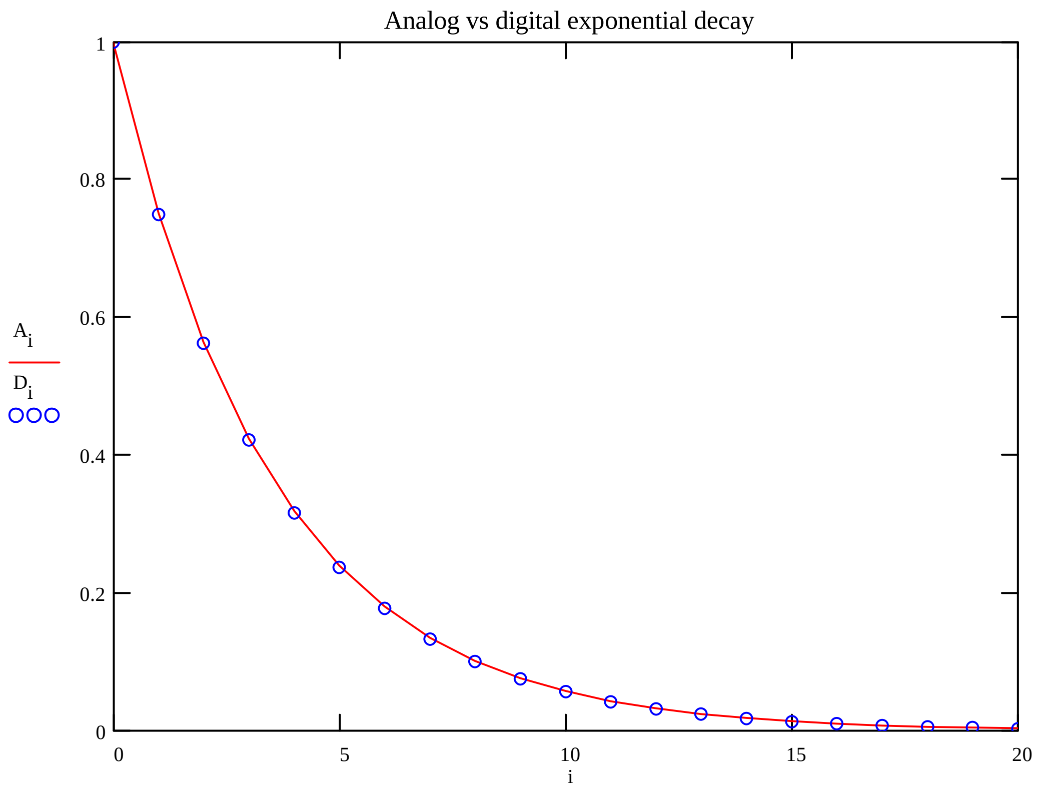
$n := 100$ $i := 0 .. n$ The analog circuit starts at time zero (with value 1.0).

$j := 1 .. n$ For the digital circuit, we assume there was a pulse at the input with value 1.0 at time zero.

$\delta := \frac{1}{4}$ $\tau := \left(\frac{1}{\delta} - \frac{1}{2}\right)$ This is the equation for time constant that we want to check. The time constant τ is in units of sample period.

$D_0 := 1$ $D_j := (1 - \delta) \cdot D_{j-1}$ Digital circuit response.

$A_i := e^{\frac{-i}{\tau}}$ Analog circuit response



It works pretty well even for large δ. (The approximation is best for small δ / large time constant, τ.)

Fourier transform - convolving sine waves N1AL 6/8/2009

To demonstrate the principle of the Fourier transform, we will create a periodic test signal that we know in advance consists of a single sine wave at the second harmonic. (That is, there are two cycles within each repetition period of the signal.) Then we test it with sine waves of different frequencies by convolving each sine wave with the test signal, which means multiplying them together and taking the average (DC component) of the result.

$i := 0 .. 1000$ $p1 := 1000$ The period of the fundamental frequency is 1000 samples.

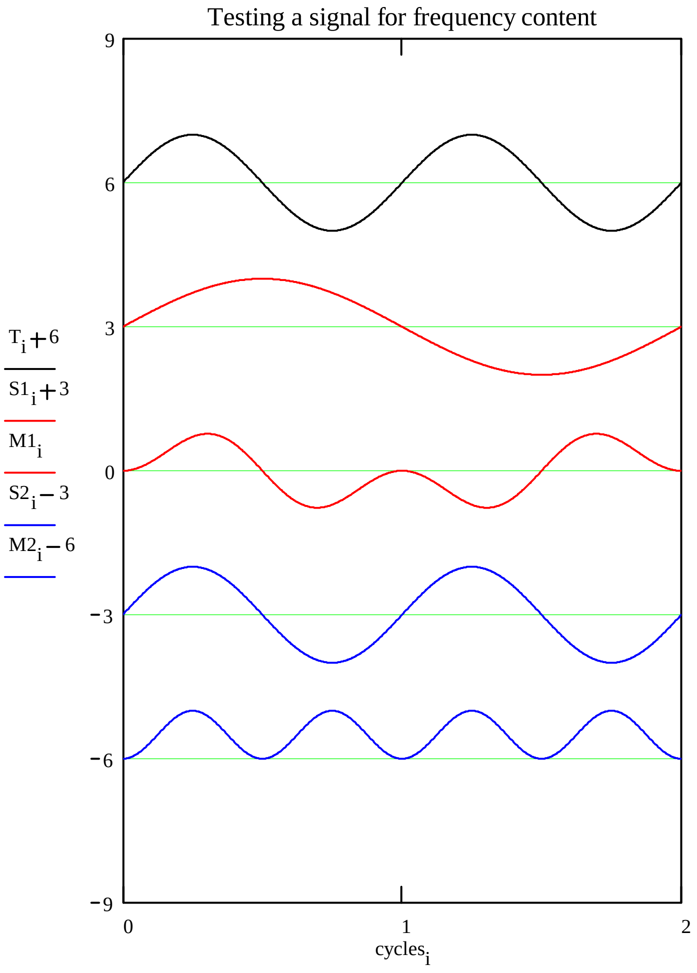
$S1_i := \sin\left(2\cdot\pi\cdot\frac{i}{p1}\right)$ First harmonic $S2_i := \sin\left(2\cdot\pi\cdot\frac{2\cdot i}{p1}\right)$ Second harmonic

$T_i := S2_i$ Signal to be tested

$M1_i := S1_i \cdot T_i$ Test with first harmonic

$M2_i := S2_i \cdot T_i$ Test with second harmonic

$cycles_i := \frac{2\cdot i}{p1}$



Signal to be tested, T

"First harmonic", S1

Test with first harmonic, M1=S1*T

$\frac{1}{1000}\cdot\left(\sum_i M1_i\right) = 0$ Zero "DC" value

Second harmonic, S2

Test with 2nd harmonic, M2=S2*T

$\frac{1}{1000}\cdot\left(\sum_i M2_i\right) = 0.5$ Large "DC" value

Only the second harmonic has a non-zero convolution, so the test signal contains only second harmonic.

Quantizing a random signal - N1AL 6/5/2009

Compared to a periodic waveform like a sine wave, a voice signal is much more random.
We will simulate a voice signal with random noise filtered to a 1 kHz bandwidth.
Then we will quantize it with 8 bits of resolution and compare the spectra of the continuous and the quantized signals.

$fs := 10000$ Sample rate $\quad f := 1000$ Bandwidth $\quad bits := 8$ Resolution

$n := 10000$ Number of samples $\quad i := 0 .. n-1$ Sample index

$A_i := \mathrm{rnd}(1) - 0.5$ 10,000 samples of unfiltered random noise

$fwidth := 500$ Filter width $\quad j := 0 .. n - fwidth - 1$ Fewer filtered samples to account for filter width

$k := 0 .. fwidth - 1$ Filter index

$$sinc_k := \frac{\sin\left[\pi \cdot \left(k - \frac{fwidth}{2} + 0.5\right) \cdot \frac{2 \cdot f}{fs}\right]}{\pi \cdot \left(k - \frac{fwidth}{2} + 0.5\right) \cdot \frac{2 \cdot f}{fs}}$$

An FIR filter with an impulse response in the shape of a sinc function theoretically results in a "brick wall" ideal low-pass filter.

$$hann_k := \frac{\left[1 + \cos\left[\pi \cdot \left(k - \frac{fwidth}{2} + 0.5\right) \cdot \frac{2}{fwidth}\right]\right]}{2}$$

The impulse response must be windowed to reduce passband and stopband ripple due to the finite impulse response length. We will use a Hann (Hanning) window.

$$Afilt_j := \sum_k A_{j+k} \cdot sinc_k \cdot hann_k$$

Convolve the windowed filter impulse response with the signal.

$$RMS := \sqrt{\frac{\sum_j \left(Afilt_j\right)^2}{n - fwidth}}$$

Calculate the RMS value of the filtered signal.

And normalize by the RMS value: $Afilt_j := \frac{Afilt_j}{RMS}$

$scale := 2^{bits-1} - 1 \quad scale := \frac{scale}{\max(Afilt)}$

Scale the signal so that the peak is at full scale of the analog-to-digital conversion.

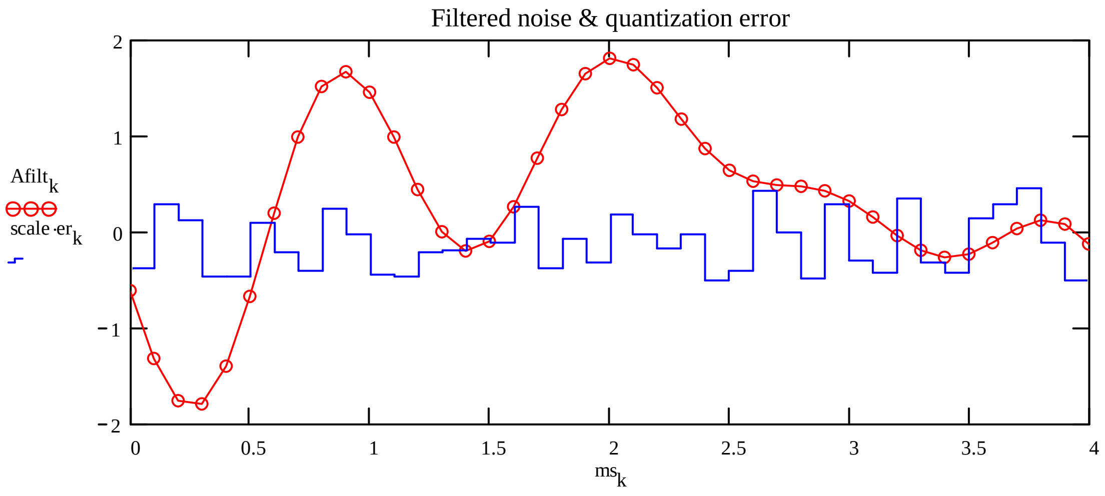
$$Adig_j := \frac{\mathrm{floor}\left(scale \cdot Afilt_j + 0.5\right)}{scale}$$

Digitize the signal = convert from analog to digital.

$er_j := Adig_j - Afilt_j$ The error signal is the difference between the original signal and the quantized signal

$start := 70$

$k := start .. start + 40$ Counter for points to plot. $\quad ms_k := (k - start) \cdot \frac{1000}{fs}$ Time in milliseconds



Now let's plot the frequency spectra of the original and quantized filtered-noise signals.
First, we'll need to window the entire 9500-point sequence before doing the FFT.
Let's use a Hann window again:

$$\mathrm{hann}_j := \frac{\left[1+\cos\left[\pi\cdot\left(j-\frac{n-\mathrm{fwidth}}{2}+0.5\right)\cdot\frac{2}{n-\mathrm{fwidth}}\right]\right]}{2}$$

Window the original signal:

$\mathrm{Afilt}_j := \mathrm{Afilt}_j\cdot\mathrm{hann}_j$

Window the digitized signal:

$\mathrm{Adig}_j := \mathrm{Adig}_j\cdot\mathrm{hann}_j$

The regular FFT functions in Mathcad require that the input be a real sequence of length 2^n.
Our sequence is real but of the wrong length so we have to use the "complex FFT" cfft():

$S := \mathrm{cfft}(\mathrm{Afilt})$ Frequency spectrum of undigitized signal.

$Sd := \mathrm{cfft}(\mathrm{Adig})$ Frequency spectrum of digitized signal.

$m := \mathrm{ceil}\left(\frac{n}{2}\right)$

$j := 0 .. m$

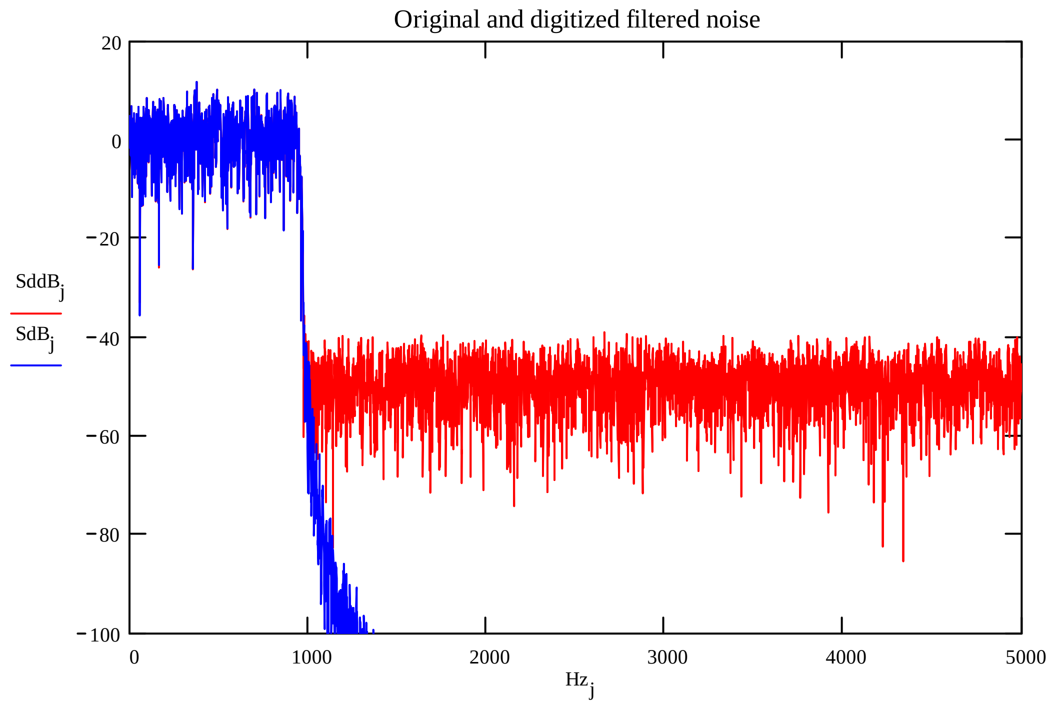
The Fourier transform of a real signal gives a symmetrical frequency spectrum. That is, the spectrum from 0 to fs/2 is the mirror image of the spectrum from -fs/2 to 0 (which is the same as the spectrum from fs/2 to fs). So the output of the normal FFT function has only half the samples since the other half is the same anyway. The complex FFT gives the entire spectrum but we will only use half of it since we actually have a real input.

$\mathrm{Hz}_j := j\cdot\frac{fs}{n}$ Frequency in Hz as a function of FFT sample.

$\mathrm{SdB}_j := 20\cdot\log\left(\left|S_j\right| + 0.000001\right)$

$\mathrm{SddB}_j := 20\cdot\log\left(\left|Sd_j\right| + 0.000001\right)$

Convert frequency spectra into dB.
The 0.000001 fudge factor is to prevent taking the log of zero in case some sample is zero.



With an 8-bit quantizer, the quantization noise should theoretically be
$SNR_{eff} = 1.76 + 8 * 6.02 + 10 \log(10{,}000/(2*1000)) = 56.9$ dB.

However, that assumes a sine-wave signal whose RMS power is only 3 dB below the peak. With a noise signal, the peak-to-RMS ratio is greater, so SNR_{eff} is somewhat worse.

Quantizing a sine wave - N1AL 6/5/2009

Quantizing a complex signal generally results in quantization noise. Quantizing a periodic signal results in quantization spurs - discrete tones on frequencies related to the signal frequency and the sample rate. Let's see how that works with a sine wave.

$fs := 9500$ Sample rate

$f := 1000$ Sine wave frequency

We choose a sample rate that is not an integer multiple of the sine wave frequency so that the spurs don't end up on harmonics only.

$bits := 8$ Resolution

$n := 9500$ Number of samples

$i := 0 .. n-1$ Sample index

By sampling for an integer number of sine-wave cycles, we won't have to window the FFT.

$A_i := \sin\left(2 \cdot \pi \cdot \frac{f}{fs} \cdot i\right)$ Undigitized sine wave

$scale := 2^{bits-1} - 1$ Scale the signal so that the peak is at full scale of the analog-to-digital conversion.

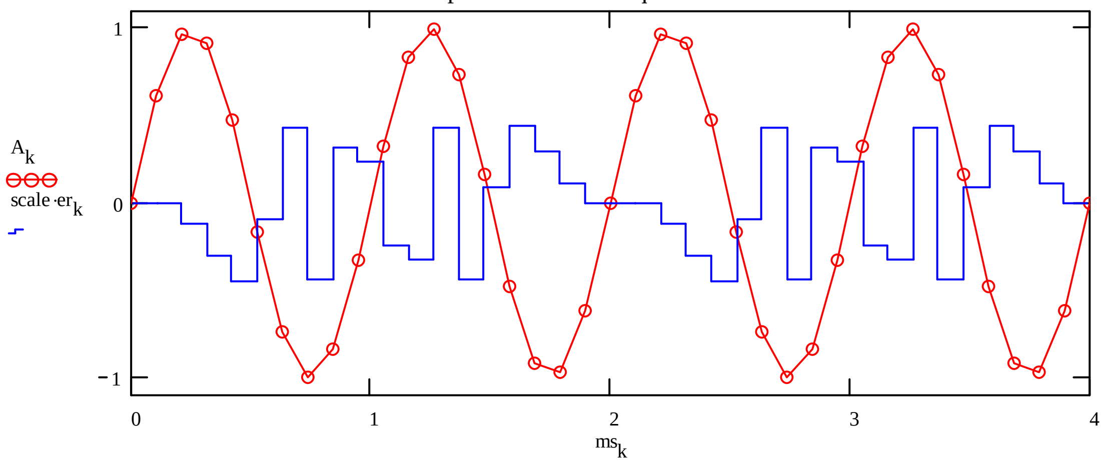
$Adig_i := \frac{\mathrm{floor}(scale \cdot A_i + 0.5)}{scale}$ Digitized sine wave

$er_i := Adig_i - A_i$ The error signal is the difference between the original signal and the quantized signal

$k := 0 .. 100$ Counter for points to plot.

$ms_i := i \cdot \frac{1000}{fs}$ Time in millisconds

Sampled sine wave & quantization error



Notice how the error signal repeats every two cycles.

Calculate the frequency spectrum using the fast Fourier transform.

The regular FFT functions in Mathcad require that the input be a real sequence of length 2^n.
Our sequence is real but of the wrong length so we have to use the "complex FFT" cfft():

$S := \mathrm{cfft}(A)$ Frequency spectrum of undigitized signal.

$Sd := \mathrm{cfft}(Adig)$ Frequency spectrum of digitized signal.

$m := \mathrm{ceil}\left(\frac{n}{2}\right)$

$j := 0..m$

The Fourier transform of a real signal gives a symmetrical frequency spectrum. That is, the spectrum from 0 to fs/2 is the mirror image of the spectrum from -fs/2 to 0 (which is the same as the spectrum from fs/2 to fs). So the output of the normal FFT function has only half the samples since the other half is the same anyway. The complex FFT gives the entire spectrum but we will only use half of it since we actually have a real input.

$f\_index := \mathrm{floor}\left(f \cdot \frac{n}{fs} + 0.5\right)$ $norm := \left|S_{f\_index}\right|$ $S_j := \frac{S_j}{norm}$

$normd := \left|Sd_{f\_index}\right|$ $Sd_j := \frac{Sd_j}{norm}$

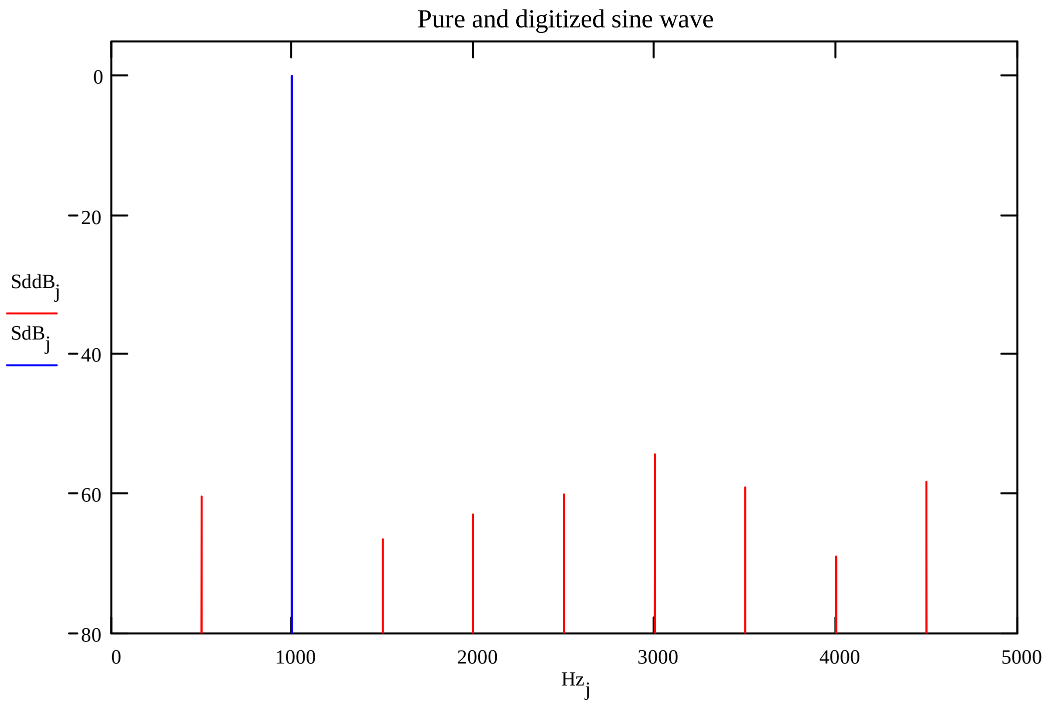
Normalize the spectra so that the value at the sine-wave frequency is at zero dB.

$SdB_j := 20 \cdot \log\left(\left|S_j\right| + 0.000001\right)$

$SddB_j := 20 \cdot \log\left(\left|Sd_j\right| + 0.000001\right)$

Convert frequency spectra into dB.
The 0.000001 fudge factor is to prevent taking the log of zero in case some sample is zero.

$Hz_j := j \cdot \frac{fs}{n}$ Frequency in Hz as a function of FFT sample.



Because the error has a period of two sine-wave cycles, the spurs are at f/2 and harmonics.

Now let's try dithering. By adding a small amount of noise to the signal before quantization the quantization error is no longer periodic. The quantization spurs are converted to quantization noise, which is preferable in many applications even though the total spurious signal power is greater.

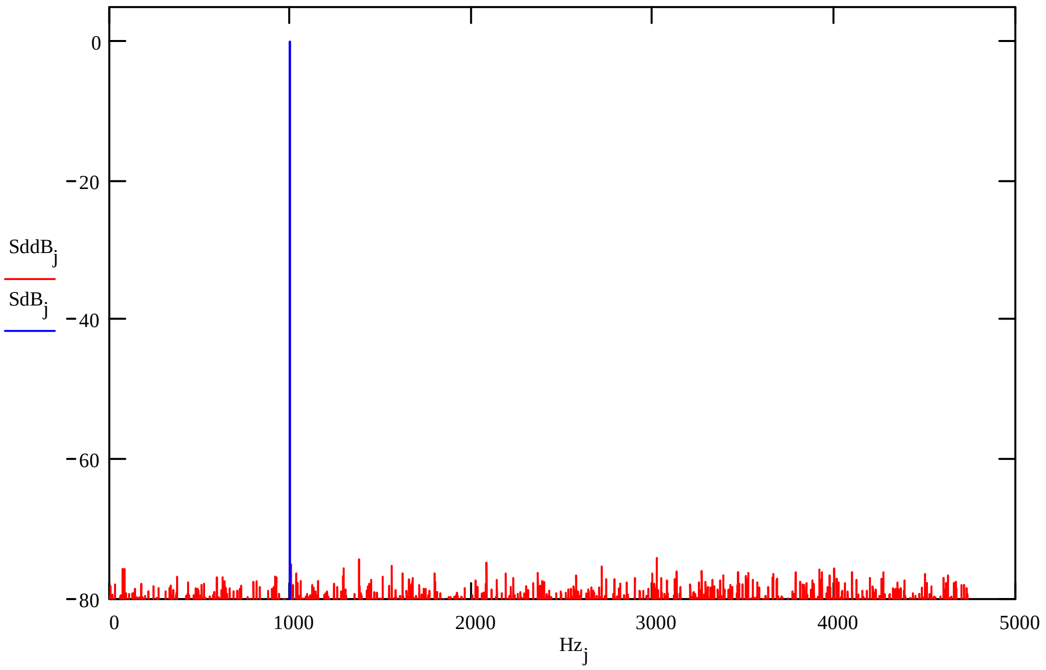
$$Adig_i := \frac{\text{floor}(\text{scale} \cdot A_i + \text{rnd}(1))}{\text{scale}}$$ Add 1 LSB of uniformly-distributed noise.

$$Sd := \text{cfft}(Adig)$$ Re-calculate the frequency spectrum

$$normd := \left| Sd_{f\_index} \right| \qquad Sd_j := \frac{Sd_j}{norm}$$ and re-normalize.

$$SddB_j := 20 \cdot \log\left(\left| Sd_j \right| + 0.000001 \right)$$ Convert to dB.

Pure sine wave &digitized with dithering



The discrete spurs have been "smeared out" into random noise.

Sampling in time domain = periodicity in frequency domain   N1AL 6/9/2009

Sampling a signal in the time domain causes that signal to be repeated in the frequency domain. These alias products appear at frequencies equal to each harmonic of the sample rate plus and minus the frequency of the original signal. Here we show how one set of samples corresponds to sine waves of three different frequencies.

$\mathrm{SamplePeriod} := 100$ 100 display points in the graph per sample

$\mathrm{SampleRate} := \frac{1}{\mathrm{SamplePeriod}}$ Sample rate (samples per second) = 1 / period (seconds per sample)

$\mathrm{OffsetFreq} := 0.2 \cdot \mathrm{SampleRate}$ Frequency difference between a harmonic of the sample rate and an alias frequency

$N := 400 \qquad i := 0 .. N$

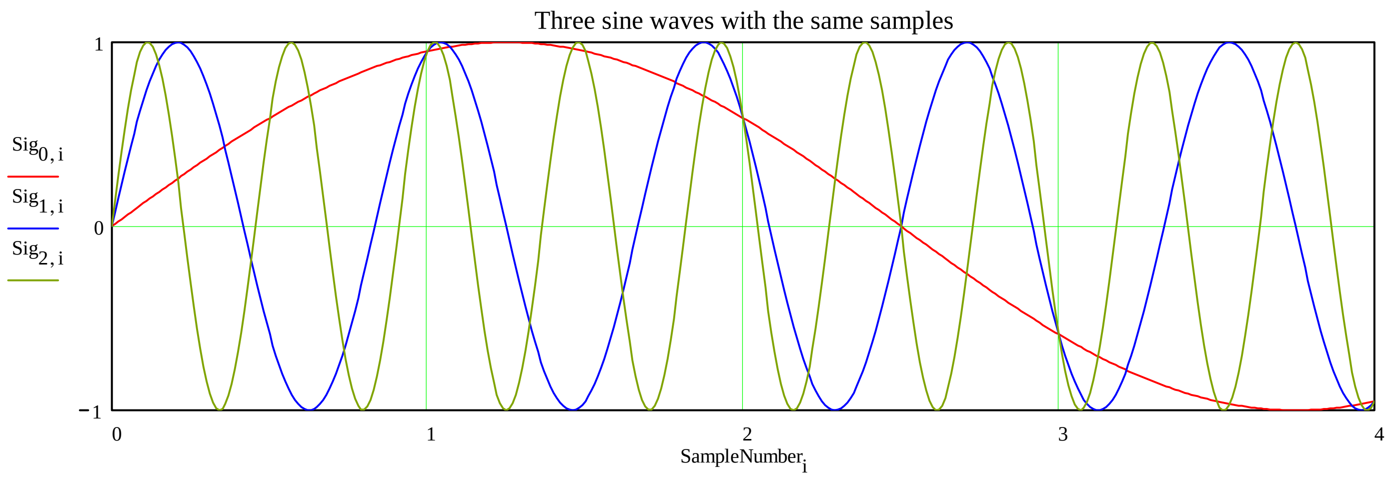
$\mathrm{Freq}_0 := 0 \cdot \mathrm{SampleRate} + \mathrm{OffsetFreq} \qquad \mathrm{Sig}_{0,i} := \sin(2 \cdot \pi \cdot \mathrm{Freq}_0 \cdot i)$

$\mathrm{Freq}_1 := 1 \cdot \mathrm{SampleRate} + \mathrm{OffsetFreq} \qquad \mathrm{Sig}_{1,i} := \sin(2 \cdot \pi \cdot \mathrm{Freq}_1 \cdot i)$

$\mathrm{Freq}_2 := 2 \cdot \mathrm{SampleRate} + \mathrm{OffsetFreq} \qquad \mathrm{Sig}_{2,i} := \sin(2 \cdot \pi \cdot \mathrm{Freq}_2 \cdot i)$

All three of these signals have exactly the same sampled sequence. The same sampled sequence corresponds to many different frequencies.

$\mathrm{SampleNumber}_i := \frac{i}{\mathrm{SamplePeriod}}$

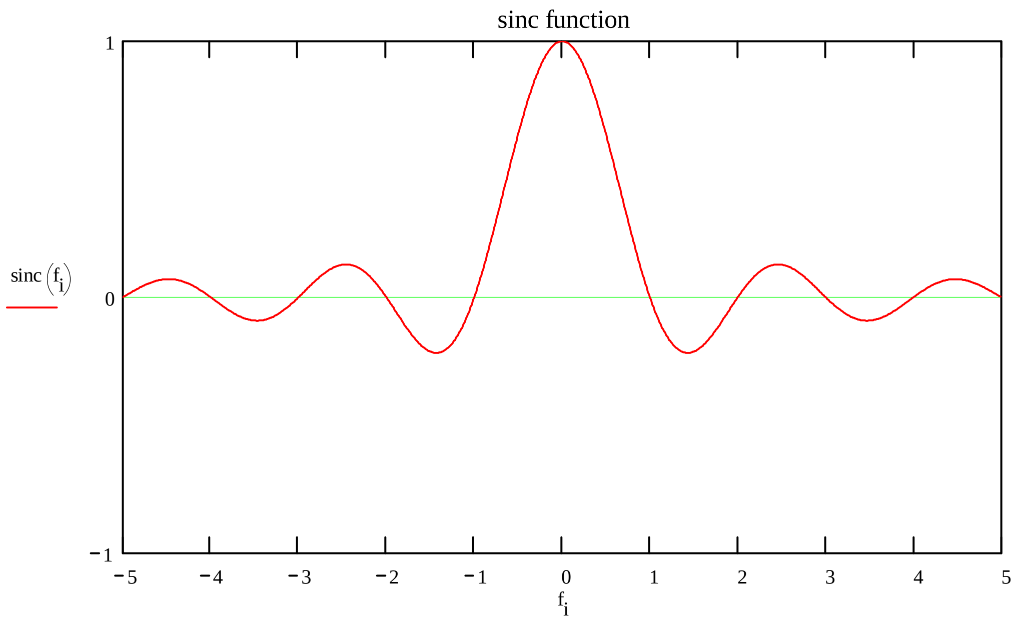


Sinc function N1AL 6/8/2009

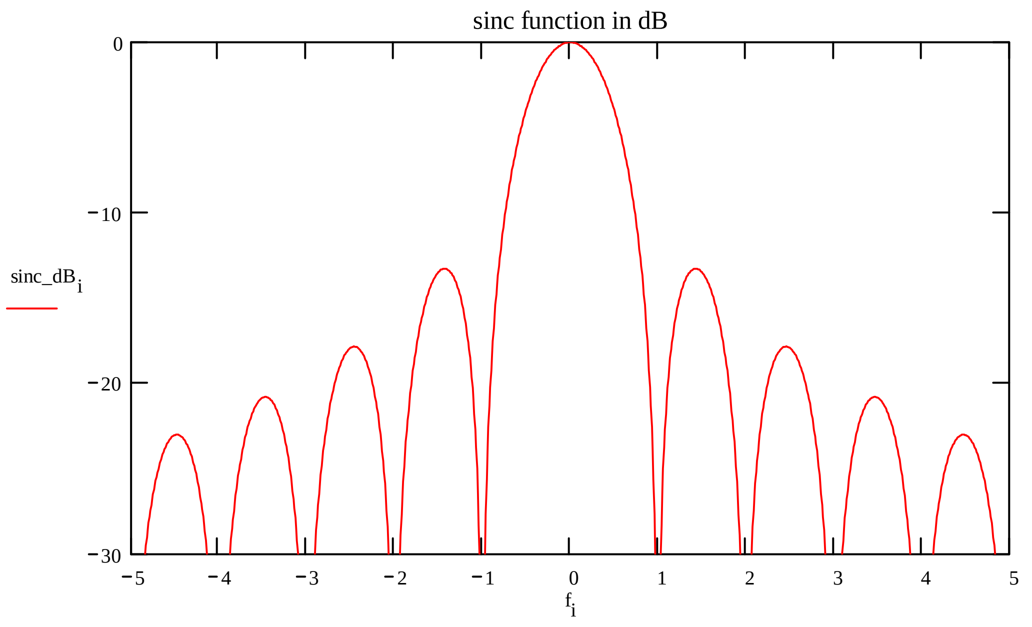
$\mathrm{sinc}(x) := \mathrm{if}\left(x = 0, 1, \frac{\sin(\pi \cdot x)}{\pi \cdot x}\right)$ Define the sinc function.
(The "if" statement is required to avoid dividing by zero.)

$n := 1000 \quad i := 0 .. n - 1 \quad len := 10$ Set up constants for the graphics plot

$f_i := \left(i - \frac{n}{2}\right) \cdot \frac{len}{n}$ Offset the frequency to get f = 0 at the center
and scale the frequency to plot a length of (len).



$\mathrm{sinc\_dB}_i := 10 \cdot \log\left(\mathrm{sinc}(f_i)^2 + 10^{-6}\right)$ Convert to dB. (The 10^{-6} is to prevent taking the log of zero.)



Fifth-order sine approximation N1AL 5/28/2009

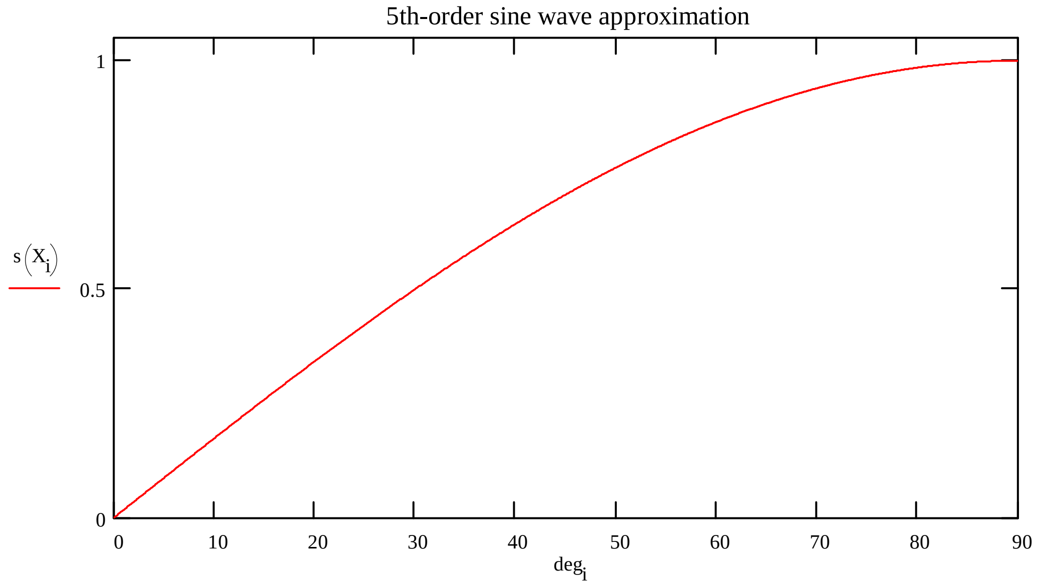
It is possible to obtain quite a good approximation to a sine wave between 0 and 90 degrees using a fifth-order curve fit. Although this only works for the first quadrant of one complete 360-degree cycle, the other quadrants are all reversed and/or inverted versions of the first, so with additional hardware or software the other three quadrants can be calculated from the first.

$$C := \begin{bmatrix} 0 \\ 3.14062500 \\ 0.02026367 \\ -5.32519600 \\ 0.5446778 \\ 1.800293 \end{bmatrix}$$

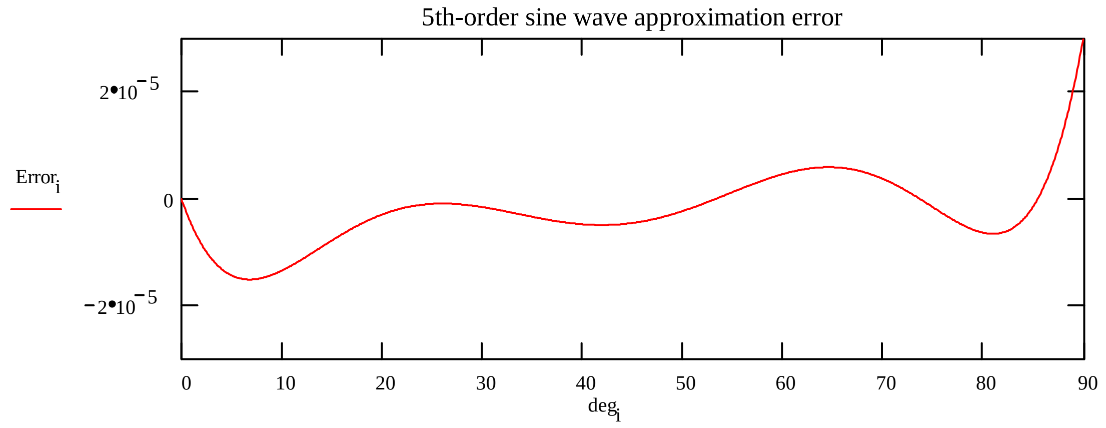
The coefficients are from the Analog Devices ADSP2100 Users Guide

$$s(x) := C_1 \cdot x + C_2 \cdot x^2 + C_3 \cdot x^3 + C_4 \cdot x^4 + C_5 \cdot x^5$$ Fifth order approximation.

$\text{size} := 1024$ $i := 0 .. \text{size}$ $X_i := \dfrac{i}{\text{size} \cdot 2}$ X=1.0 corresponds to 180 degrees. $\text{deg}_i := X_i \cdot 180$



$$\text{Error}_i := s(X_i) - \sin(\pi \cdot X_i)$$



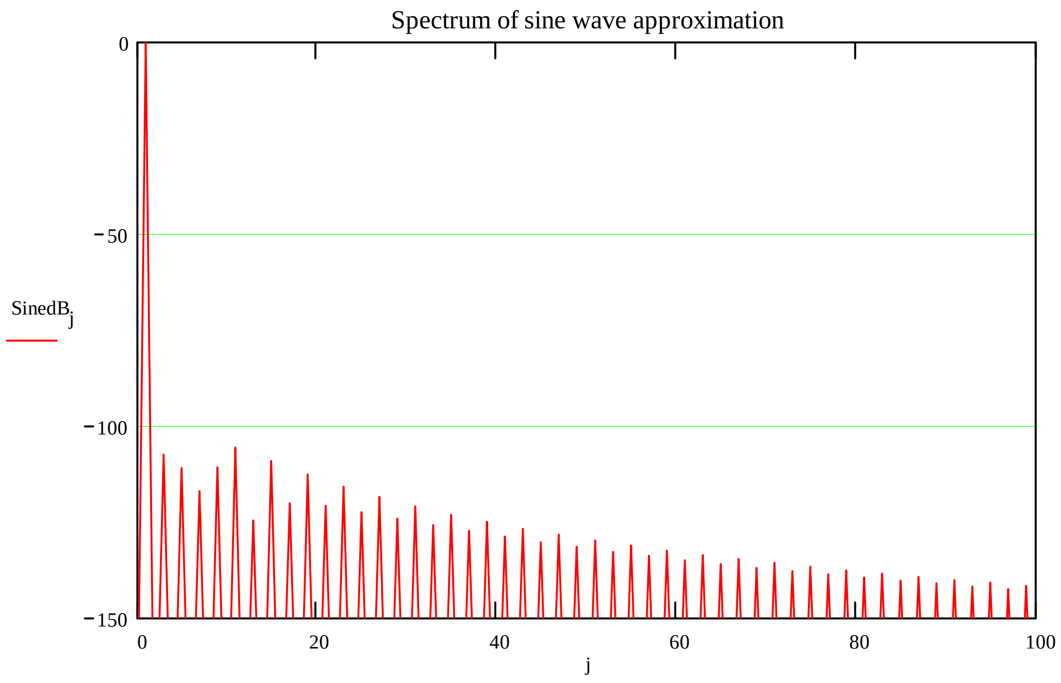
Create the four quadrants of one cycle of the sine wave:

$\text{sine}_i := s(X_i)$ $\quad \text{sine}_{i+\text{size}} := \text{sine}_{\text{size}-i}$ $\quad i := 0..\,2\cdot\text{size}-1$ $\quad \text{sine}_{2\cdot\text{size}+i} := -\text{sine}_i$

$\text{Sine} := 2\cdot\text{FFT}(\text{sine})$ Compute the frequency spectrum

$j := 0..\,2\cdot\text{size}$ There are 4*size points in the time sequence but only 2*size points in the frequency spectrum. That is because with a real sequence, the frequency spectrum is symmetrical, so only half the points are needed.

$\text{SinedB}_j := 20\cdot\log\left(\left|\text{Sine}_j\right| + 10^{-10}\right)$ Convert the frequency spectrum to dB.
(The 10^{-10} is to avoid taking the log of zero.)



All harmonics are over 100 dB down. Note that only odd harmonics are present

Windowing filter coefficients   N1AL 6/9/2009

The filter coefficients for an FIR filter are just the impulse response of the filter. One method to compute the filter coefficients is to write an equation to describe the desired frequency response and then do an inverse Fourier transform to get the imulse response. However, an impulse response of finite length has ripples in the pass-band and stop-band. To reduce the ripples, the impulse response must be windowed.

As a demonstration of the technique, we define an ideal "brick-wall" filter in the frequency domain and run it through an inverse Fourier transform to get the impulse response (even though we already know that the answer is a sinc function). Then we window the result and do a forward Fourier transform on both the windowed and non-windowed versions so we can compare the amount of ripple.

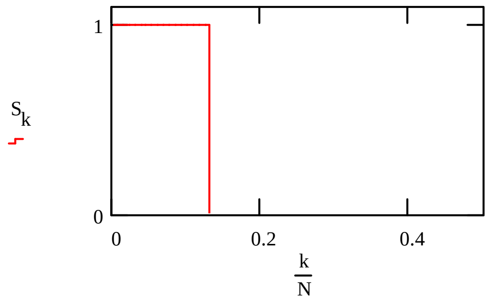
$N := 128$	Number of filter coefficients	$Fc := 0.125$	Filter cutoff frequency
$n := 0 .. N-1$	Coefficient index	$k := 0 .. \frac{N}{2}$	Frequency index

Define the frequency response of an ideal low-pass filter:

$S_k := 0 \qquad i := 0 .. N \cdot Fc \qquad S_i := 1.0$

The frequency response is first set to all zero.

Then the frequencies up to Fc = 0.125 are set to 1.0.



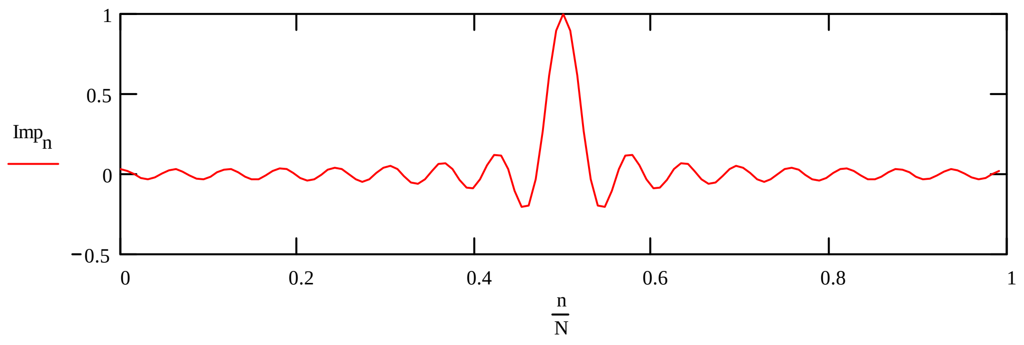
Now calculate the impulse response:

Get impulse response with inverse Fourier transform:

$I := \mathrm{IFFT}(S)$

Shift the coefficients to put the peak at the center:

$j := 0 .. \frac{N}{2} - 1 \qquad Imp_j := \frac{I_{j+\frac{N}{2}}}{I_0} \qquad Imp_{j+\frac{N}{2}} := \frac{I_j}{I_0}$



Extend the impulse response with zeros to simulate a non-repeating impulse response:

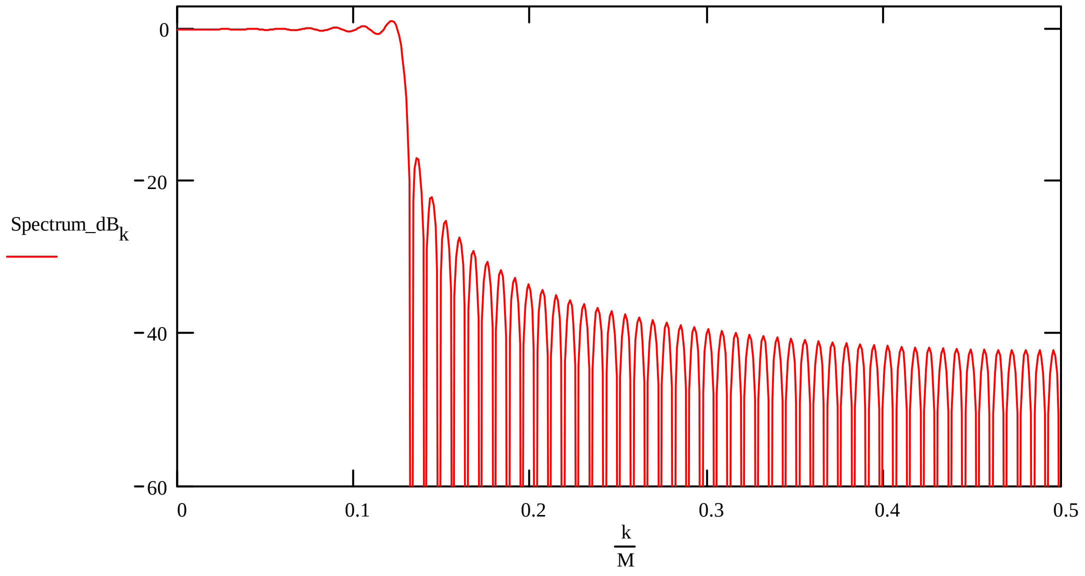
$M := 1024 \quad i := 0 .. M-1 \quad \text{Impulse}_i := 0 \quad \text{Impulse}_n := \text{Imp}_n$

Now let's see how we did.

$\text{Spectrum} := \text{FFT}(\text{Impulse}) \quad k := 0 .. \frac{M}{2}$ Only half the coefficients are needed in the frequency domain since the spectrum is symmetrical.

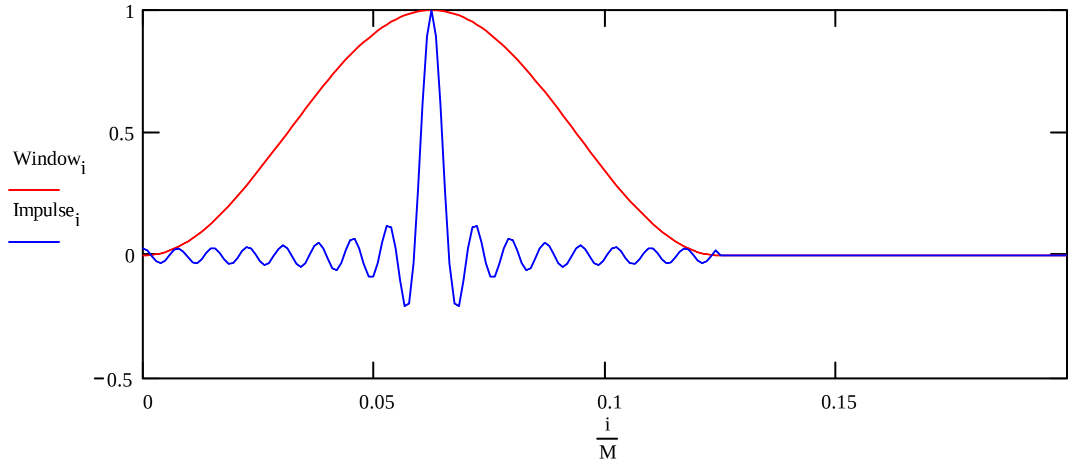
$$\text{Spectrum\_dB}_k := 20 \cdot \log\left(\left|\frac{\text{Spectrum}_k}{\text{Spectrum}_0}\right| + 10^{-7}\right)$$ Convert to dB units

Unwindowed filter frequency response



Now window the impulse response:

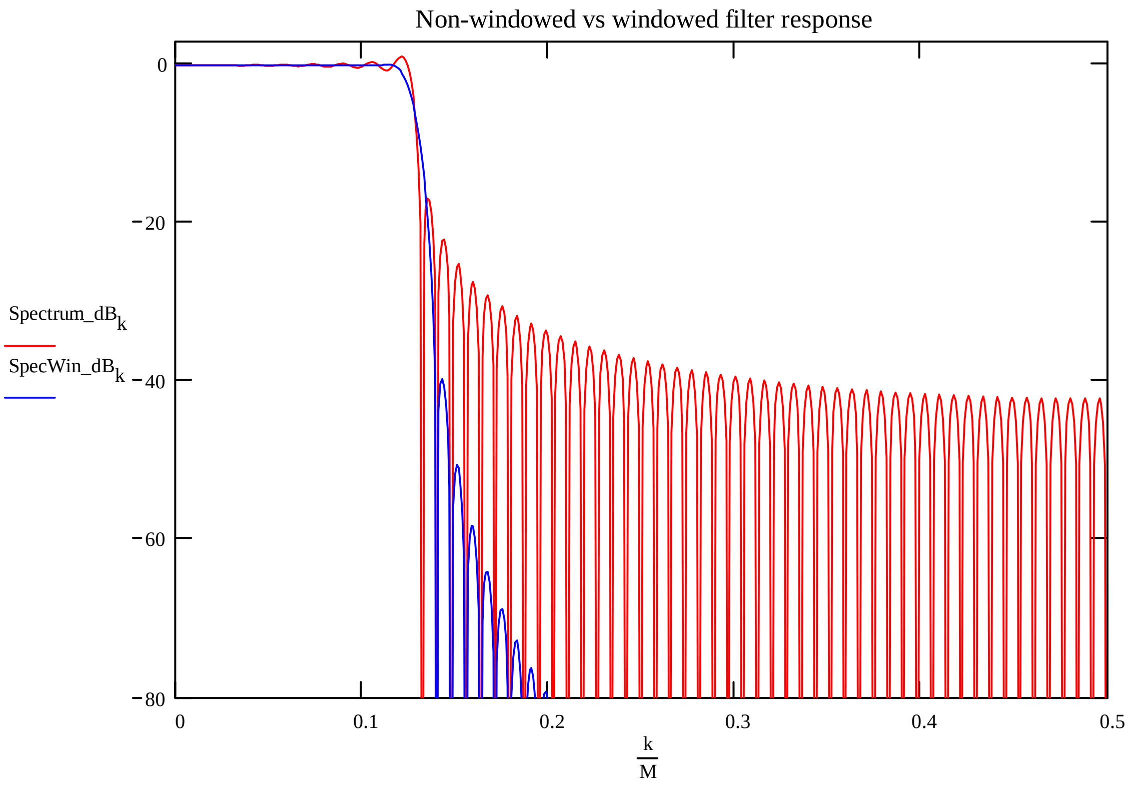
$\text{Window}_i := \frac{1}{2} \cdot \left[1 - \cos\left[2 \cdot \pi \cdot \left(\frac{1}{N} \cdot i\right)\right]\right]$ Hanning window $\quad j := N .. M-1 \quad \text{Window}_j := 0$



$\text{WindowedImpulse}_i := \text{Window}_i \cdot \text{Impulse}_i$ Apply the window

$\text{SpectrumWin} := \text{FFT}(\text{WindowedImpulse})$ $k := 0 .. \frac{M}{2}$

$$\text{SpecWin\_dB}_k := 20 \cdot \log\left(\frac{\left|\text{SpectrumWin}_k\right|}{\left|\text{SpectrumWin}_0\right|} + 0.0000001\right)$$



Ripple in both the passband and stopband is greatly reduced through windowing but the passband-to-stopband transition is not as sharp..