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**.

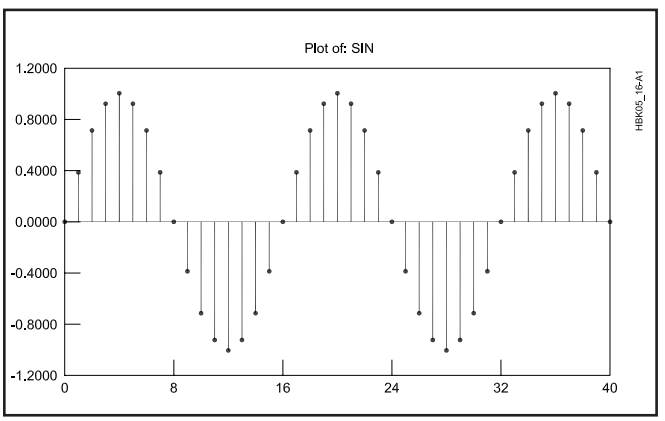


Fig 16.A1—A 40-sample sine wave.

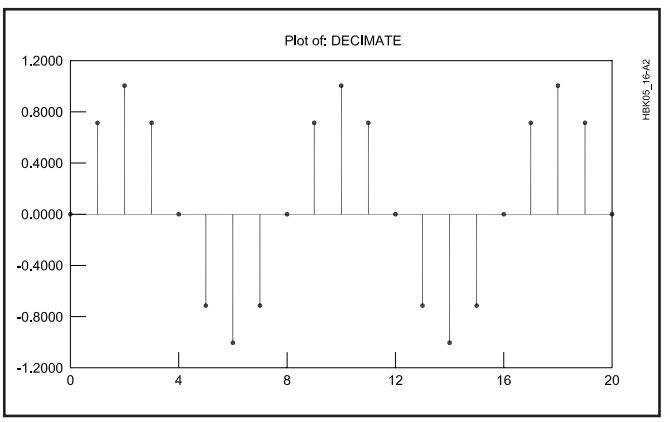


Fig 16.A2—Decimated, 20-sample sine wave.

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:

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

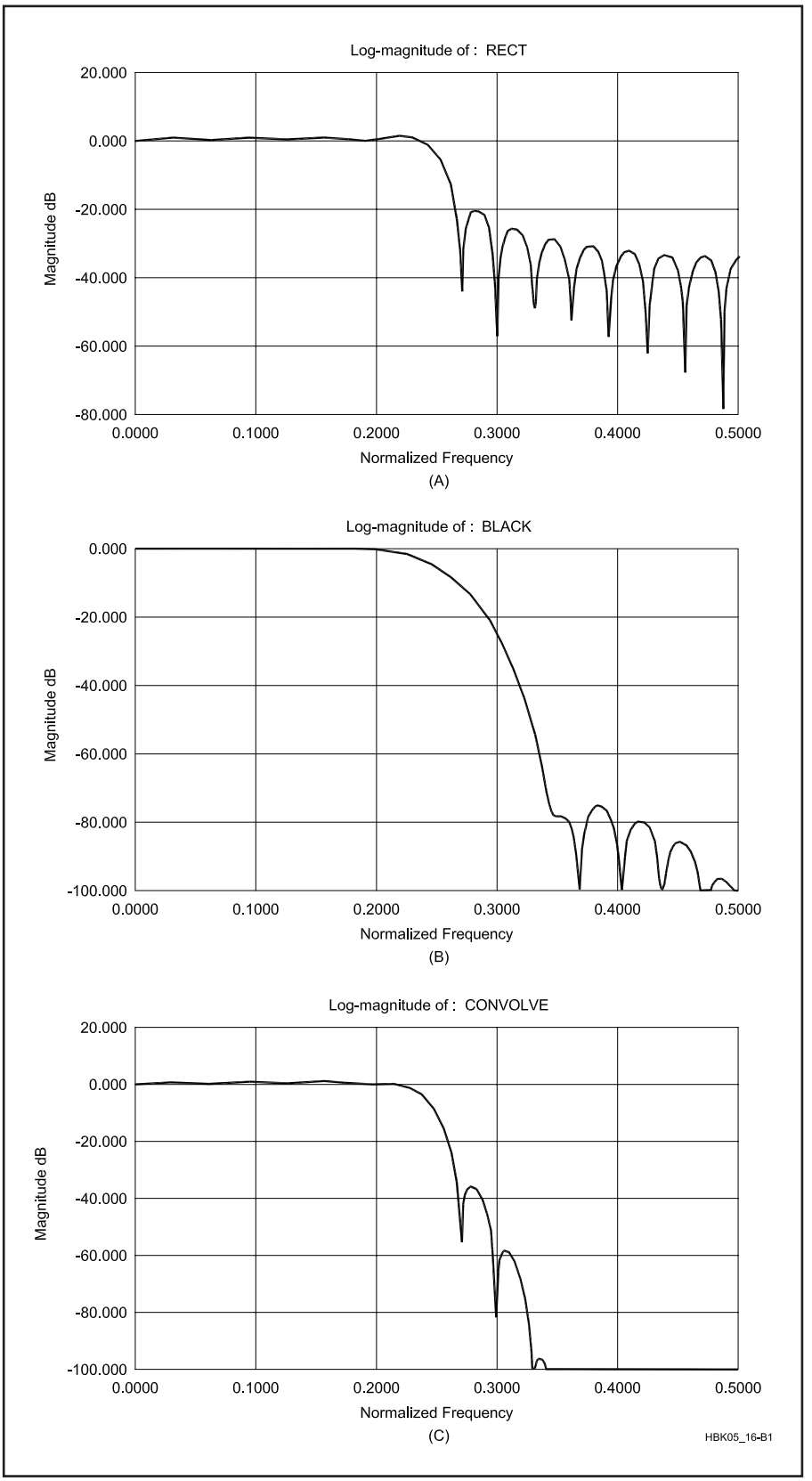


Fig 16.B1—LPF frequency response, rectangular window (A). LPF frequency response, Blackman window (B). LPF frequency response, convolution of filters shown in A and B (C).

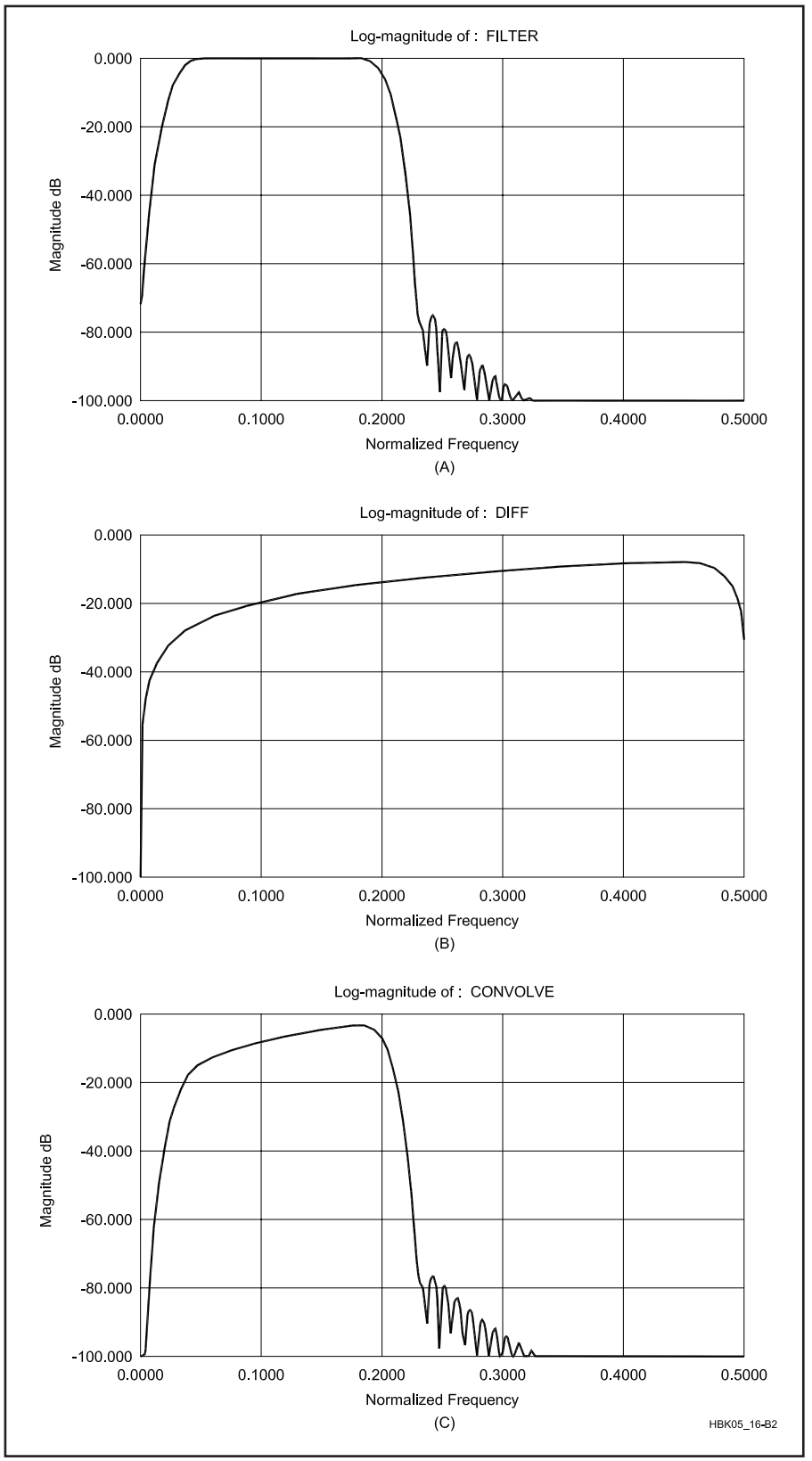


Fig 16.B2—BPF for SSB use, L = 99 (A). LPF having rising frequency response, L = 31 (B). Frequency response of convolution of filters shown in A and B (C).

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.

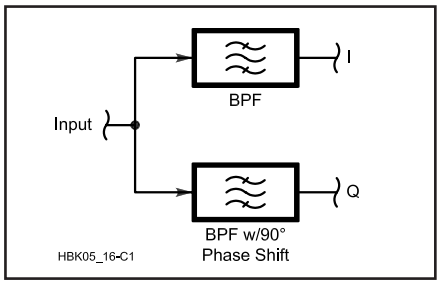


Fig 16.C1—Hilbert transformer using an analytic filter pair.

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.

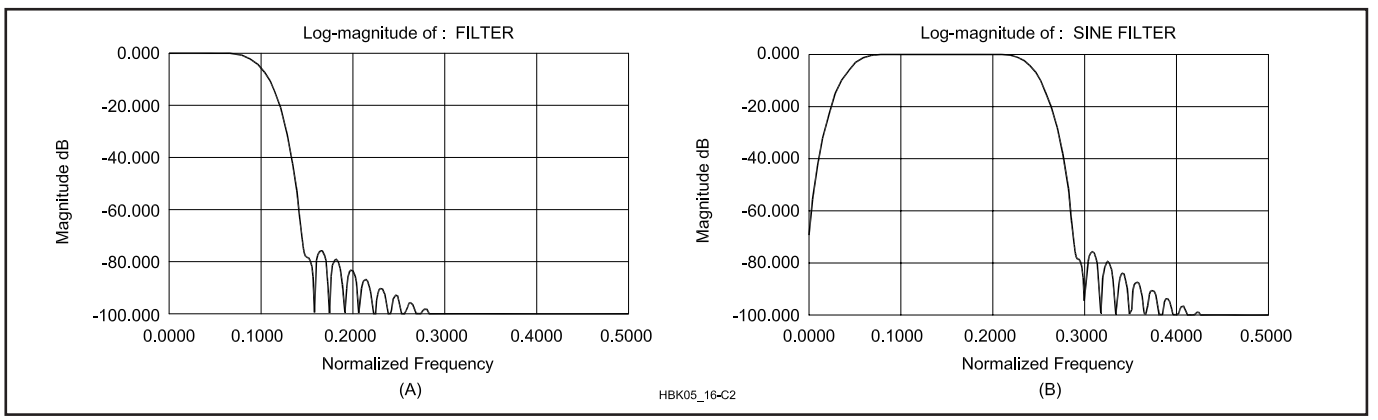


Fig 16.C2—LPF prototype frequency response (A). BPF frequency response of processed impulse response (B).

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

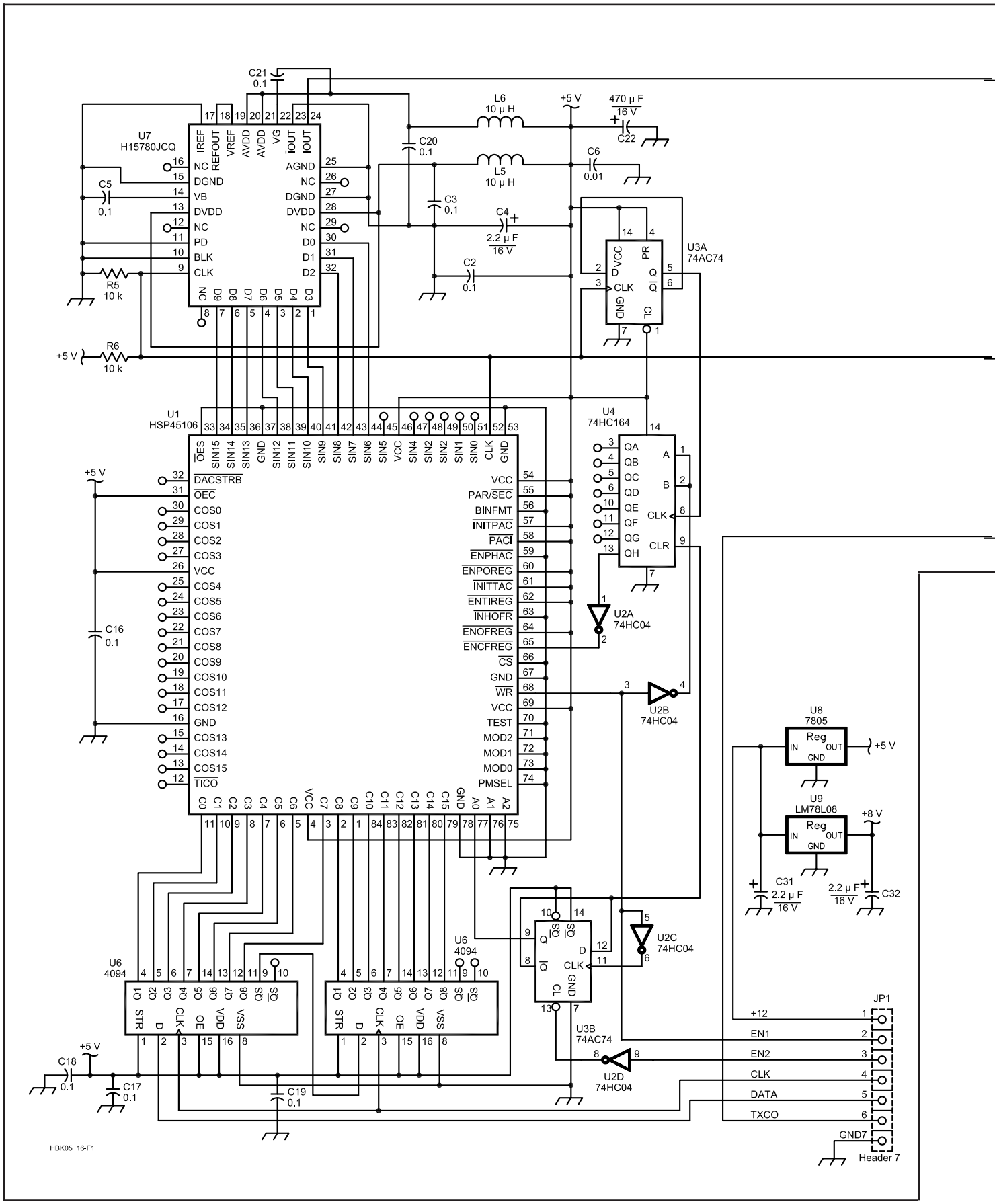
In this example of Newton's method, a generic *BASIC* program is given that computes the root of a 32-bit integer to within an error margin, DERROR. The root of a 32-bit integer is naturally a 16-bit integer. Emphasis is placed in what follows on speed of execution and accuracy as influenced by truncation 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**.


U7
H15780JCQ
C21
0.1
L6
10 µ H
+5 V
470 µ F
16 V
C22
C20
0.1
L5
10 µ H
C6
0.01
C5
0.1
C3
0.1
C4
2.2 µ F
16 V
C2
0.1
U3A
74AC74
R5
10 k
R6
10 k
U1
HSP45106
U4
74HC164
U2A
74HC04
U2B
74HC04
C16
0.1
U8
7805
U9
LM78L08
+8 V
C31
2.2 µ F
16 V
C32
U6
4094
U2C
74HC04
U3B
74AC74
U2D
74HC04
C18
0.1
C17
0.1
C19
0.1
JP1
+12
EN1
EN2
CLK
DATA
TXCO
GND7
Header 7
HBK05_16-F1

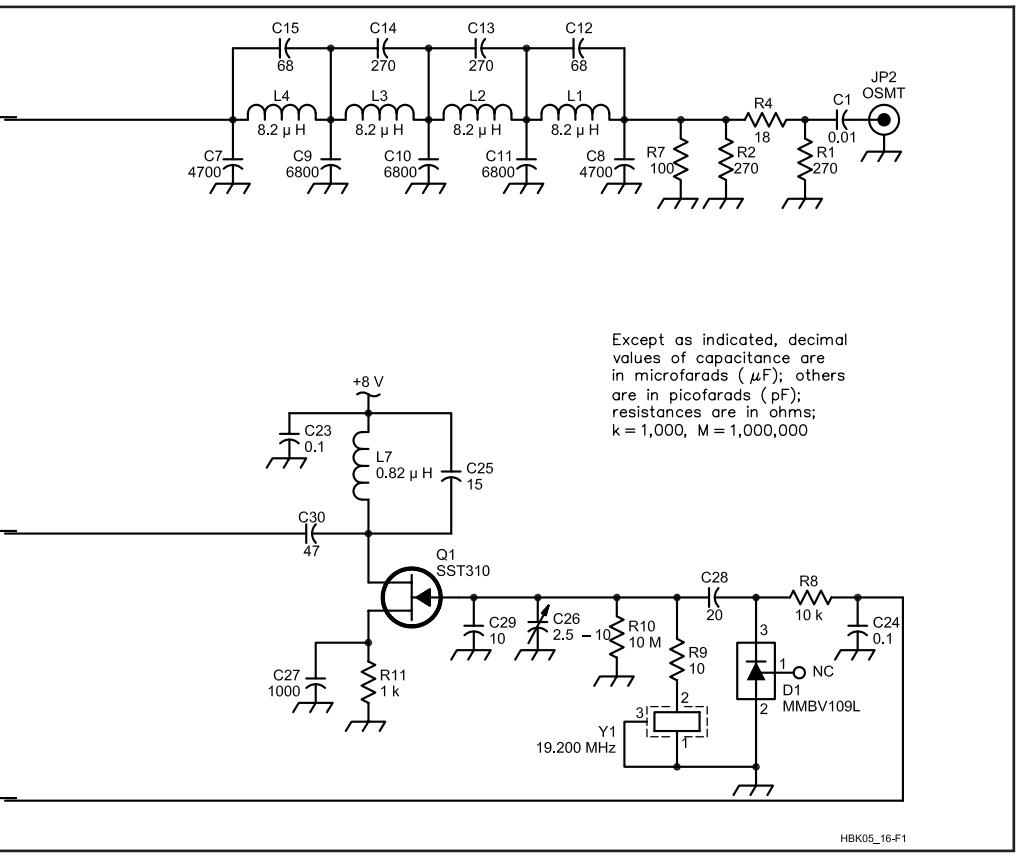


Fig 16.F1—High-performance DDS schematic diagram.

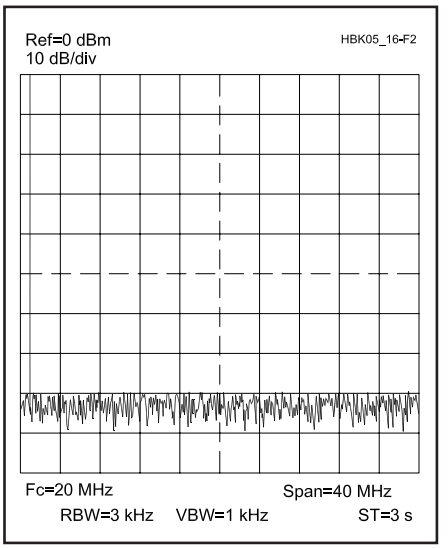


Fig 16.F2—Typical output spectrum of DDS.

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.

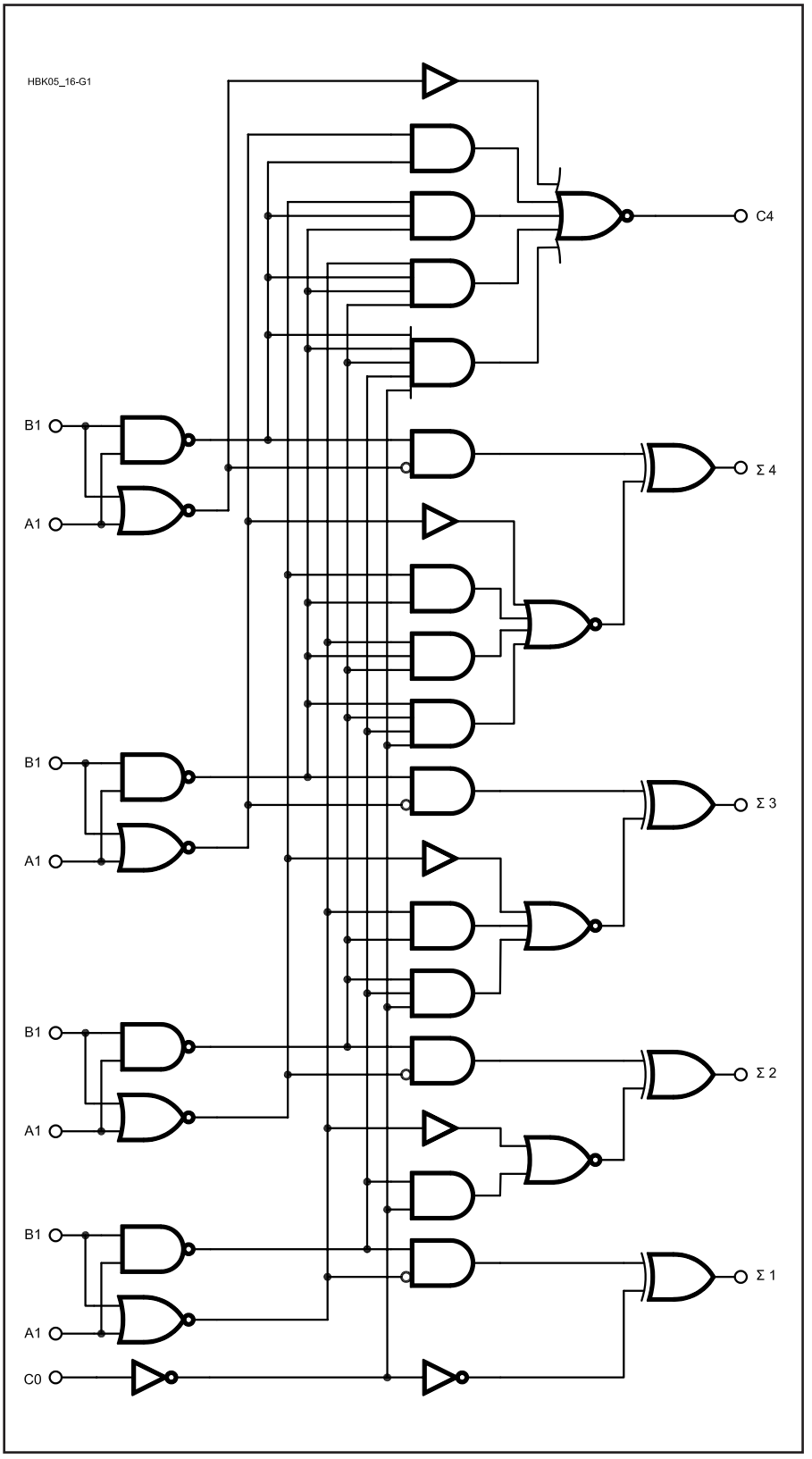


Fig 16.G1—A 4-bit adder schematic diagram.

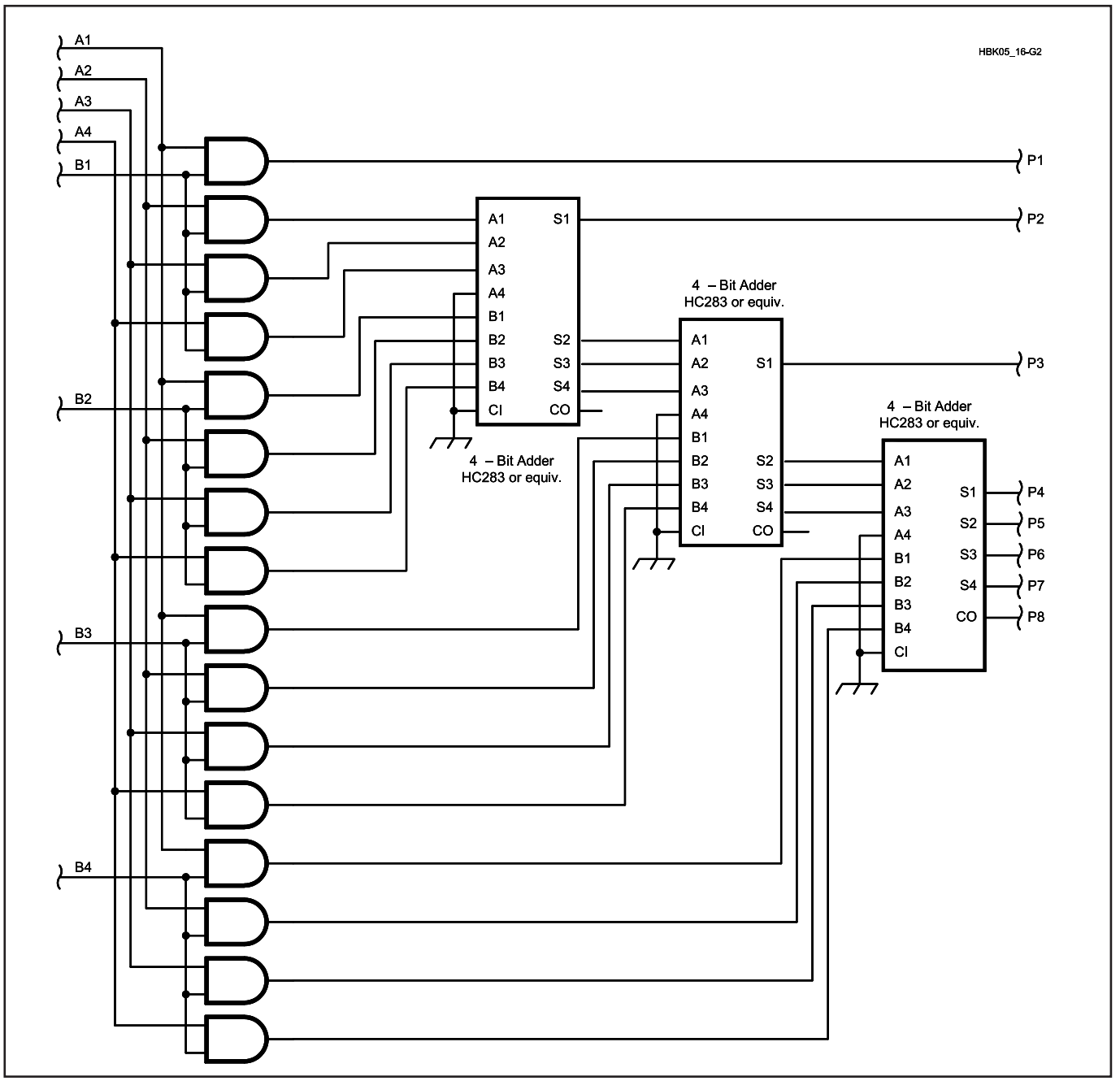


Fig 16.G2—Complete 4-bit multiplier, no pipelining.

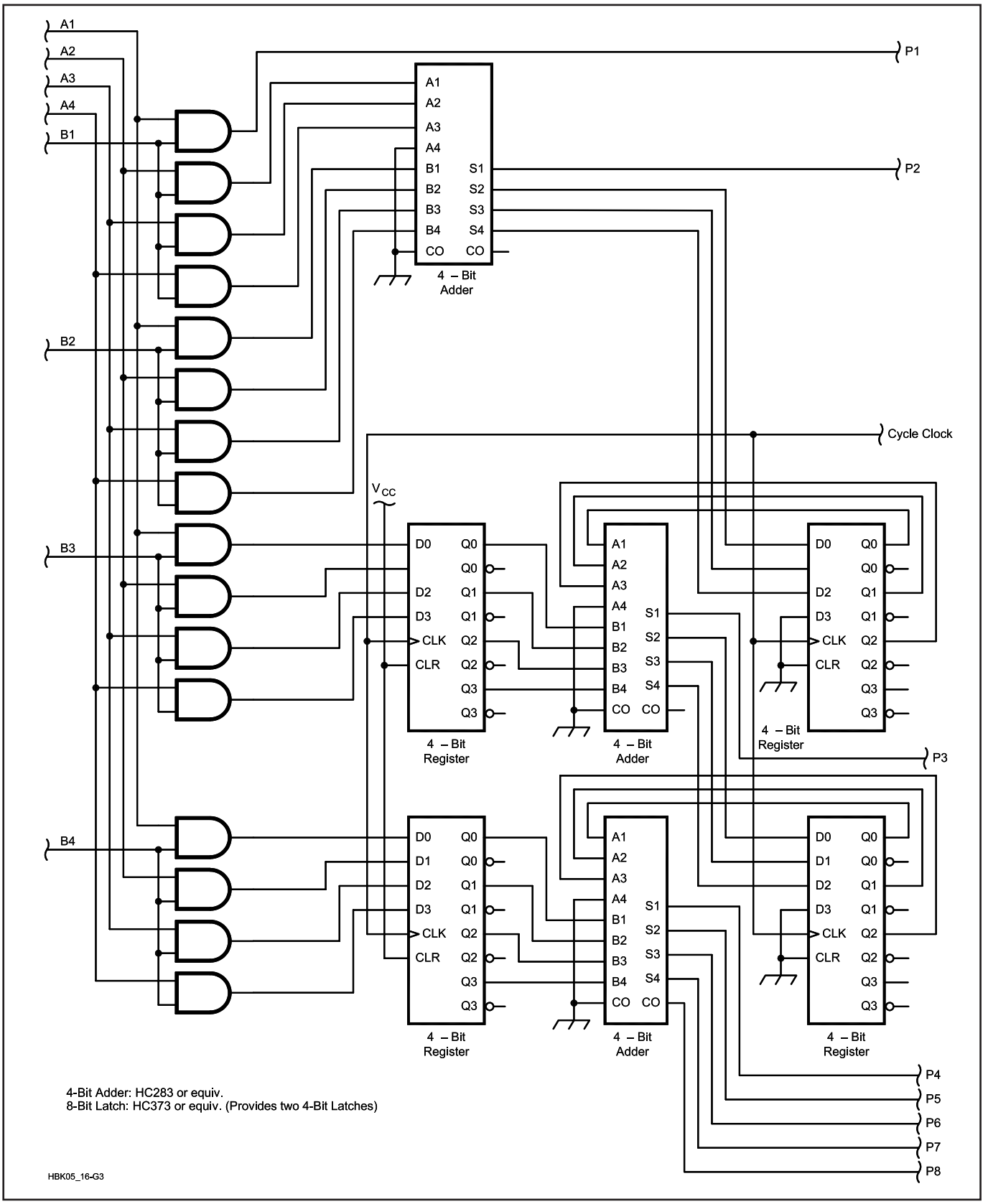


Fig 16.G3—Complete 4-bit multiplier with pipelining.