120th ASEE Annual Conference & Exposition
FRANKLY, WE DO GIVE A D*MN
June 23-26, 2013

Paper ID #5740

A Low-Cost Programmable Arbitrary Function Generator for Educational Environment

Mr. Mani Dargahi Fadaei, Azad University

Mani Dargahi Fadaei received B.S. in electrical engineering from Azad University of Kermanshah, Kurdistan in 2010. Currently, he is pursuing M.S. in electrical engineering in Azad university of Iran, Tehran Markazi branch. His research interests include wireless network, ultra low power and ultra low voltage amplifier design.

A Low-Cost Programmable Arbitrary Function Generator for Educational Environment

Abstract

This paper presents the design, implementation, and operation of a low-cost programmable arbitrary function generator intended for use as a plug-in board in a personal computer. Since a digital computer is used, capabilities such as programmability and signal recording are available with this system. Therefore, this instrument can generate any function of time that can be represented mathematically. The ability to generate arbitrary waveforms makes this instrument more versatile that an ordinary function generator that can produce only a few different waveforms. This design offers a significant advantage to educators in underdeveloped countries with limited resources to obtain a low-cost instrument that can be used in undergraduate laboratories where more expensive commercially arbitrary function generators are not available. One interesting application of this instrument is the synthesis of sound. If the equation for a particular sound is known, the sound can be produced when this function generator is connected to an audio amplifier and a speaker.

Introduction

This paper describes the design, implementation and operation of programmable function generator intended for use with a personal computer (PC). Because a digital computer is used, capabilities such a programmability and signal recording are available in this system. The ability to generate arbitrary waveforms makes this instrument more versatile than an ordinary function generator that can produce only three or four different waveforms.

This design offers two significant advantages to educators: (1) it provides a low-cost instrument that can be used in undergraduate laboratories where more expensive commercial arbitrary function generators are not available; and (2) it is suitable for use as a student project. One interesting application for this system is the synthesis of sound. If the equation for a particular sound wave is known, that sound can be produced when this function generator is connected to an audio amplifier and speaker.

Methodology

The design of this waveform generator utilizes the theory of sampled-data systems. Shannon's sampling theorem states that a time-dependent function $f(t)$ that is limited in bandwidth to f_m and sampled at a rate $f_s > 2f_m$ can be completely reconstructed from its samples[1].The sampled waveform is a discrete time signal that possesses all of the frequency components in the interval $0 < f < f_m$. However, additional frequency components that are imposed by the sampling process appear in the interval $f > (f_s - f_m)$, and these undesirable components are separated from the original frequency by a bandwidth of $f_s - 2f_m$. If $f(t)$ is sampled at a rate less than $2f_m$ samples per second, the undesirable components overlap the original components in the frequency spectrum, and consequently $f(t)$ cannot be recovered from the original sampled signal. This phenomenon is known as aliasing. To reconstruct a function that has been sampled at the rate $f_s > 2f_m$, the sampled signal must be directed through a low-pass filter that has a cutoff frequency of f_m. The magnitude plot of the low-pass filter response must have a slope that is sufficiently steep in the cutoff region to reduce the magnitude of the lowest undesirable frequency component $f_s - f_m$ to an acceptable level.

Utilizing this sampled-data scheme, a programmable arbitrary function generator was designed to interface with a PC. This specific computer platform was chosen because it is very common in both industrial and scientific environments[2]. A flowchart of the operation of this function generator is shown in figure 1. An arbitrary function is defined; band-limited is necessary, then sampled according to Shannon's sampling theorem entirely in software. The values of these samples are then stored in a data file. These stored samples are then directed to an output device at the rate that they were sampled. This device reconstructs the discrete time-sampled signal and processes it through low-pass filters to recover and display a close approximation of the original defined function.

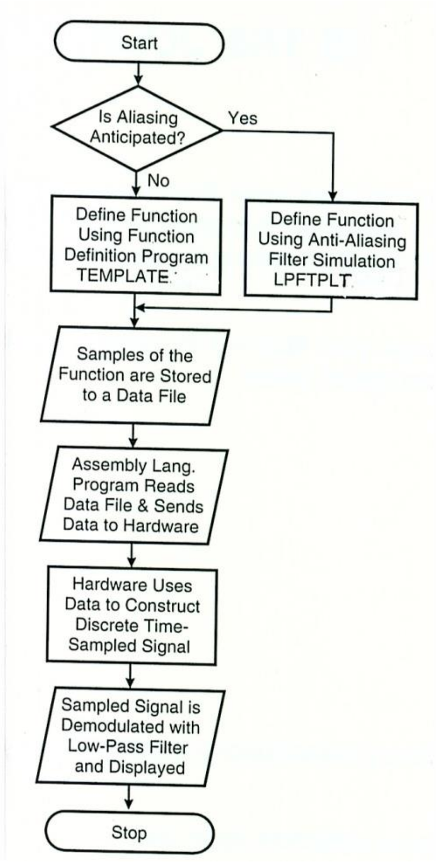
Start
Is Aliasing Anticipated?
Yes
No
Define Function Using Function Definition Program TEMPLATE
Define Function Using Anti-Aliasing Filter Simulation LPFTPLT
Samples of the Function are Stored to a Data File
Assembly Lang. Program Reads Data File & Sends Data to Hardware
Hardware Uses Data to Construct Discrete Time-Sampled Signal
Sampled Signal is Demodulated with Low-Pass Filter and Displayed
Stop

Figure 1. Flowchart of arbitrary function generator

Software Components

This programmable function generator was designed to generate waveforms in the audio spectrum. Therefore, defined functions are limited in bandwidth to 18 kHz and are sampled at a rate of 50,000 samples/sec, resulting in a guard bandwidth of 14 kHz. Reconstruction of the sampled signal is accomplished using a four-pole low-pass filer with a slope of 80 dB/decade, as shown in figure 2, that attenuates the lowest undesirable frequency component by approximately 20 dB. The sampling rate is low enough to satisfy the speed require of the assembly language program used to generate the sample.

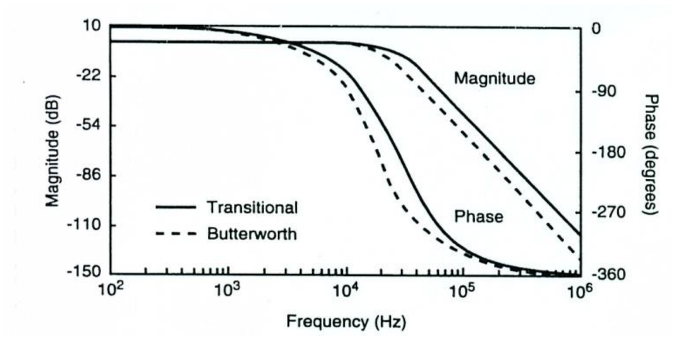

Figure 2. Bode Plots for Gaussian transitional and Butterworth low-pass filters

The arbitrary function generator operates in the following sequence:

1. First, the generating function is defined in a program called TAMPLATE. If is anticipated that the desire function will have frequency components above in the bandwidth limit of 18kHz, and thus will be subject to the effects of aliasing, the function can be defined in the alternate program called LPFTPLT.

2a**.** TAMPLATE evaluates the function at time intervals that are small enough to satisfy Shannon's sampling theorem. The decimal value of each sample is quantized, encoded into an 8-bit value, and stored in a data file.

2b**.** LPFTPLT applies an anti-aliasing low-pass filter simulation to the function. The decimal values of these filtered samples are quantized into 8-bit value, and stored in a data file.

3. The stored data file contains amplitude samples for the defined function. The utility program GRAPH uses this data to display the sampled function on the PC monitor.

4. The function is then generated using a program called FE. This program prompts the user for the name of the data file that contains the sampled values and the number of

times to repeat the waveform. The program then uses this information to execute the assembly language program ASSEM.EXE.

5. ASSEM.EXE accesses the 8-bit values stored in the data file and directs them to the hardware device used to generate the defined function.

6. This hardware device constructs a discrete time-sampled (SAM) signal from these 8-bit values. This sampled signal is then sent through a transitional low-pass filter to generate a continuous time signal that is a close approximation to the original defined function.

The program LPFTPLT is not only used to define, sampled, quantize, and store desired function, but also to limit the bandwidth of the function. This program also solve the necessary difference equations used to simulate a low-pass filter with the transfer function of

$$\frac{\boldsymbol{Y(s)}}{\boldsymbol{R(s)}} = \frac{\boldsymbol{a_0}}{\boldsymbol{s^4 + a_3 s^3 + a_2 s^2 + a_1\, s + a_0}} \qquad (1)$$

Where a_n = filter parameters.

In the LPFTPLT program, the user must define a number of variables: (1) the transfer function parameters a_n, where $n = 0, \ldots, 3$, (2) the desired waveform function. (3) the name of the data file, (4) the integration step size h, and (5) the sampling period s_p, which is typically 20μs for a sampling rate of 50,000 samples/sec. Note that the value s_p must be a multiple of the value of h.

For the accurate and stable simulation, the value of h must be chosen so the product of $\lambda_m h$ lies within the relatively stable region of numerical integration, where $\lambda_m h$ is the magnitude of the largest pole of the transfer function[3,4]. The stable region of Euler's numerical integration routine is a unit circle that is centered at $(-1 + j0)$ on the s-plane. Therefore, to ensure accurate simulation when using Euler's one-step integration routine, h must be less than$1/\lambda_m$. Furthermore, since transient behavior of a low-pass filter can include overshoot, a scaling factor is needed to ensure that the output of the filter simulation lies within the allowable range of -1 to +1.

Difference equations are used in the LPFTPLT program to simulate the transfer function of equation (1). The dynamic behavior of this low-pass filter can be described by the following state variable equations:

$$\begin{aligned}
\dot{\boldsymbol{x}}_1 &= \boldsymbol{x_2} \\
\dot{\boldsymbol{x}}_2 &= \boldsymbol{x_3} \\
\dot{\boldsymbol{x}}_3 &= \boldsymbol{x_4} \\
\dot{\boldsymbol{x}}_4 &= \boldsymbol{r(t) - a_3 x_4 - a_2 x_3 - a_1 x_2 - a_0\, x_1} \\
\boldsymbol{y(t)} &= \boldsymbol{a_0 x_1},
\end{aligned} \qquad (2)$$

Where
$\dot{\boldsymbol{x}}_{\boldsymbol{n}}$ = state variables
$\boldsymbol{r(t)}$ = input
$\boldsymbol{a_n}$ = filter parameters
$\boldsymbol{y(t)}$ = function after processing by the low-pass filter.

Using Euler's one-step explicit integrator, which is defined as

$$\boldsymbol{x(k+1) = x(k) + h\dot{x}(k)} \tag{3}$$

The state variables given in equation (2) are transformed into the difference equations

$$\begin{aligned}
&\boldsymbol{x_1(k+1) = x_1(k) + hx_2(k)}\\
&\boldsymbol{x_2(k+1) = x_2(k) + hx_3(k)}\\
&\boldsymbol{x_3(k+1) = x_3(k) + hx_4(k)}\\
&\boldsymbol{x_4(k+1) = x_4(k) + h[r(kh) - a_3x_4(k) - a_2x_3(k) - a_1x_2(k) - a_0\,x_1(k)]}\\
&\boldsymbol{y(k) = a_0x_1(k)}
\end{aligned} \tag{4}$$

that are programmed in LPFTPLT.

Once the function has been sampled and stored into a data file, the program GRAPH can be used to display the resulting waveform. This utility program can accept data files that were created by either TEMPLATE or LPFTPLT [5]. The program allows the user to scale both the horizontal and vertical axes to magnify or reduce any portion of the display in a manner similar to the vertical amplifier and sweep speed adjustments available on an oscilloscope. To display the intended waveform, the user must enter the name of the data file, the upper and lower limits of the horizontal and vertical axes, and optional graph title. The graph of the function is then display on the monitor, and the user can print this graph with the print-screen key.

Functions can be also be generated from a data file using a program called GEN.EXE. This program prompts the user for the name of the data file and the number of waveform repetitions, then encodes the repetition number into a 16-bit value and stores it to a file called GEN.EXE, then copies the contents of the data file to a file called DATA and executes the output module ASSEM.EXE.

The output module ASSEM.EXE loads both the 16-bit repetition value found in the REPSFILE file and 8-bit function amplitude value found in the DATA file into RAM. The program display a message indicating that all data has been loaded, then output this data to the PC's hardware interface. When ASSEM.EXE detects the data file's end-of-file marker, it either repeats the process for the selected number or repetitions or terminates. ASSEM.EXE was written in Assembly Language[6,7], the flowchart for the program shown in figure 3.

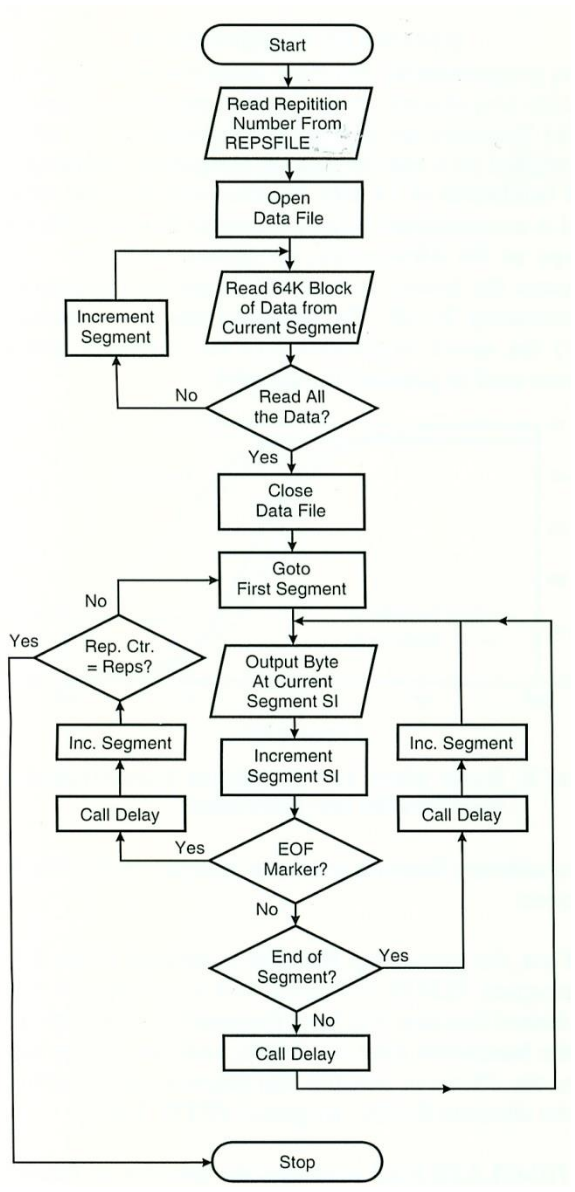

Figure 3. Flowchart of ASSEM.EXE output program

Hardware Components

The block and schematic diagrams of the hardware used to reconstruct functions are shown in figures 4 and 5, respectively. All bus lines in the PC are buffered using 74LS244 tri-state buffers for the address and strobe lines, and 74LS245 bi-directional buffers for the data lines[8]. A 74LS688 8-bit comparator and three 2-input OR gates are used to decode ten address lines and two strobe lines. When the address lines correspond to 0300 (hex) and the IOW and LEN strobe lines are active, the decoding circuit generates a clock pulse (active low) signifying that the data on these data lines is valid. This pulse clocks the byte-wide D flip-flop inside and AD-558 digital-to-analog converter (DAC), capturing successive bytes of data from data lines. The discrete time-sampled signal is obtained from the output of this DAC. This signal is filtered to recover a continuous signal using a four-pole, Gaussian-to-12-dB, transitional low-pass filter with cutoff frequency of 18 kHz[9] and parameters

$a_3 = 417.6 \text{ x } 10^3$, $a_2 = 106.3\times10^9$, $a_1 = 13.14\times10^{15}$, and $a_0 = 706.3\times10^{18}$.

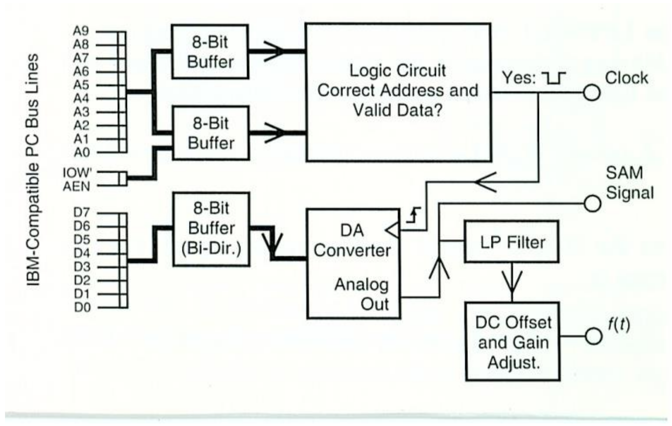

Figure 4. Block diagram of hardware interface

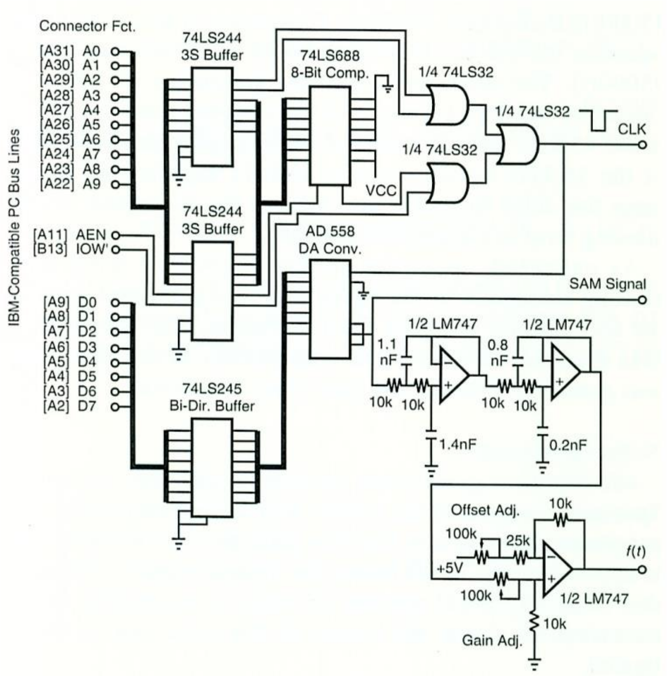

Figure 5. Schematic diagram of hardware interface

When selecting a filter for this type of application, both magnitude and phase characteristics must be carefully considered. The low-pass filter must sufficiently attenuate all of the harmonics imposed by the sampling process without introducing nonlinear group delay in the pass-band of the function. The obscure Gaussian transitional filter was chosen to meet these requirements. Comparison of transitional and Butterworth filters can be seen in figure 2.

The effects of nonlinear group delay should be considered when designing the low-pass filter. If the phase shift of the filter is linear with frequency, the propagation delay imposed by the filter is identical for all frequencies. This phenomenon is known as uniform group delay. A graph of filter's phase response with respect to a linear frequency axis may be helpful when estimating the group delay of the filter. Note that a Gaussian-to-12-dB transitional filter exhibits nearly linear phase shift while a Butterworth filter exhibits exceptional nonlinear phase shift throughout the pass-band.

To illustrate the effects of nonlinear phase shift, the propagation delay of the first five harmonics of a 2-kHz square wave imposed by both a Butterworth filter and a Gaussian filter is giving in table 1. When the period and phase shift of the waveform are known, the propagation delay can be calculated using the equation

$$\boldsymbol{P_T = -T_0 \times \frac{\theta_p}{360}} \quad (5)$$

Where T_0 is the period and θ_p is the phase shift.

Table 1. Propagation Delay of First Five Harmonics of 2-kHz Square Wave through Butterworth and Gaussian Transitional Filters

	Butterworth		Transitional	
Frequency (kHz)	Phase (degrees)	Prop. Delay (μs)	Phase (degrees)	Prop. Delay (μs)
2	-16	22.2	-13	18.1
6	-51	23.6	-40	18.5
10	-87	24.2	-68	18.8
14	-130	25.0	-94	18.6
18	-180	27.0	-119	18.4

As expected, the nonlinear phase of the Butterworth filter causes an increase in propagation time for each of successive harmonics, ranging from 22 to 27μs through the ninth harmonic. These large variations in propagation delay cause ringing in the square wave output. In comparison, the linear phase response of the Gaussian transitional filter exhibits relatively small deviation in propagation time, or uniform group delay, for these same harmonics. The trade-off for the linear phase response of the transitional filter is a roll-off in magnitude response that is less steep in the 3-dB to 12-dB region than of the Butterworth filter.

Finally, a differential amplifier is used to adjust the direct current (DC) offset and swing of the function generator output. Potentiometers are used to set the DC offset to 0 volts and swing to 2 volts peak-to-peak.

Experimental Results

Function Generation and Analysis

The function generator described in this paper was used to produce basic signal such as sinusoidal, triangular, square, and trapezoidal waveforms, as well as to produce more complicated signals such as double sideband, suppressed carrier (DSBSC) and amplitude modulated (AM) waveforms. All of these signals were displayed in time and analyzed in the frequency domain using an oscilloscope and a spectrum analyzer, respectively.

Frequency domain analysis of the DSBSC and AM signals provided an interesting application of the commonly used trigonometric identity

$$\cos(\alpha)\cos(\beta) = [1/2][\cos(\alpha+\beta) + \cos(\alpha-\beta)]. \quad (6)$$

To demonstrate this application, the function generator was first programmed to generate an AM signal

$$f_{AM}(t) = [1 + \cos(\omega_m t)] \cos(\omega_c t).$$

Where ω_m is the frequency of the modulating signal ω_c is the frequency of the carrier signal. It can be shown from equation (6) that the frequency spectrum of this AM signal has a carrier frequency with a unity magnitude and two one-half amplitude sidebands with frequencies of

$\omega = \omega_c \pm \omega_m$.

Next the function generator was programmed to generate a DSBSC signal

$$f_{DSBSC}(t) = \cos(\omega_m t) \cos(\omega_c t). \quad (7)$$

Equation (6) indicates that the frequency spectrum of this DSBSC signal does not include a carrier frequency and exhibits only two one-half amplitude sidebands with frequencies of

$\omega = \omega_c \pm \omega_m$.

A carrier signal of 5 kHz and a modulating signal of 50 Hz were used for both the AM and DSBSC signals. The anti-aliasing low-pass filter simulation template was not used in this demonstration because equations (6) and (7) indicate that these two signals do not contain frequency components above the Nyquist limit. As expected, spectrum analysis revealed the presence of the carrier frequency in the AM signal and the absence of the carrier frequency in the DSBSC signal. Time domain plots of both signals are shown in figure 6.

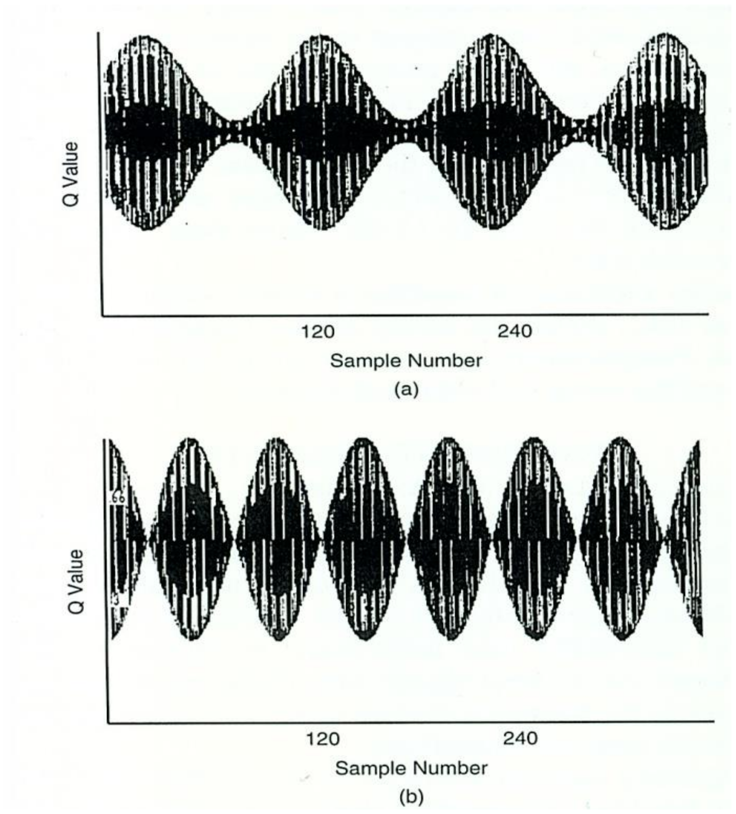

Figure 6. Time domain plots of generated functions: (a) AM signal; (b) DSBSC signal.

The programmable function generator was also used to study the effects of aliasing imposed by the sampling process. A 15-kHz square wave was sampled both with and without the anti-aliasing low-pass filter simulation found in the LPFTPL program. An aliasing frequency of 5 kHz was anticipated in the signal that was processed without the filter. The Fourier series for this square wave is

$$f_{sq}(t) = A_0 + A_1\cos(2\pi \times 15000t) + A_3\cos(2\,\pi\, x\ 45000t)\ + \ldots \quad (8)$$

For the 50-kHz square wave sampling signal, the Fourier series is

$$f_{sm}\,(t)= B_0 + B_1\cos(2\,\pi \times 50000t) + B_3\cos(2\,\pi \times 150000t) + \ldots \quad (9)$$

The sampled signal is the product of equations (8) and (9), and one of the terms of this product is $[A_3\cos\,(2\pi \times 45000t)][B_1\cos(2\,\pi \times 50000t)]$. Expansion of this term using equation (6) yields **(1/2)** $A_3B_1[\cos\,(2\pi \times 5000t) + \cos(2\pi \times 95000t)]$**.** The undesirable aliasing frequency of 5 kHz is caused by third harmonic of the sampled function, which is the 45-kHz term in equation (8). Limiting the bandwidth of the 15-kHz square wave to 18 kHz significantly attenuate this third harmonic and reduce the magnitude of the aliasing term to a negligible amount.

As expected, spectrum analysis revealed the 5-kHz aliasing frequency in the signal that was generated without the anti-aliasing low-pass filter. However, this unwanted 5-kHz sinusoid was attenuated considerably in the signal that was generated with the low-pass filter simulation.

Noise Generation

All function generators produce some type of noise. Spectrum analysis of the signals that were produced by this programmable arbitrary function generator reveals noise that is approximately 30 dB below the desired signals. Although this relatively small amount of noise is acceptable in the reconstructed signal, the causes of this noise can be investigated.

This particular function generator is susceptible to four different types of noise. First, aliasing effects can cause noise in the sampled signal. Although this source of noise can be reduced considerably by using anti-aliasing low-pass filters, the aliasing frequencies are still present in the reconstructed signal. Higher-order low-pass filters can be utilized to reduce noise levels of this type. Secondly, capacitive coupling from the nearby switching circuitry of the computer can cause noise in the signal. This source of noise can be reduced by using shielded cable on all analog leads in the system. A possible third source of noise is small variations in the time intervals between each sample. For a sampling rate of 50 kHz, the time interval between samples is 20 μs with a variation of approximately ±0.5 μs, which is less than ±3%. The fourth source of noise can result from distortion caused by sampled signals with square-top pulses, which are common in digital storage or communication systems, rather than signals that conform to natural sampling, where the tops of the pulses "follow" the sampled signal. Further, note that the digitization noise caused by the limited 8-bit resolution used in this system was negligible.

Conclusion

This paper describes the design and operation of low-cost, programmable arbitrary function generator suitable for use in undergraduate laboratories as an analytical tool or as a student design project. Using custom software and a personal computer, this system can generate any function of time that can be represented mathematically.

References

1. Schwartz, M. *Information Transmission, Modulation, and Noise*. San Francisco: McGraw-Hill Book Company. 1959
2. Burr. K., and J. Brown. *The Handbook of Personal Computer Instrumentation*. New York: Simon and Schuster. 1986.
3. Lambert, J. D., C*omputational Methods in Ordinary Differential Equations,* New York: Wiley, 1973.
4. Rahrooh, A., and T. T. Hartley, “Adaptive Matrix Integration for Real-Time Simulation of Stiff Systems,” *IEEE Transaction on Industrial Electronics,* 36, no.1 (February 1989): 18-24.
5. Mosher, F.E., and D. I. Schneider, *Using turbo programming,* San Francisco, Calif.: McGraw-Hill, 1988.
6. Vielleford, C. *programming the 80286,* San Francisco, Calif.: Sybex, Inc., 1987.
7. Norton, P. and J. Socha, *Peter Norton’s Assembly Language Book for the PC,* New York: Simon and Schuster Inc., 1989.
8. Tomplins, W. J., and J. G. Webster, *Interfacing Sensors to the IBM PC,* Englewood Cliffs, N.J.: Prentice Hall, 1988.
9. Williams, B. *Electronic Filter Design Handbook,* St. Louis: McGraw-Hill Book Company, 1981.