Radar Emitter Simulation Using The E8267C PSG Vector Signal Generator

Application Note

Introduction

Historically, simulating radar emitters has been an expensive and time consuming process. However, using off-the-shelf vector signal generators, it is now possible to produce the complex pulsed waveforms. The advent of microwave signal generators and spectrum analyzers with vector capability, allows engineers to generate pulsed microwave signals with precise control over output power, amplitude envelope, and modulation within the pulse. These precision signals can be used as a standard to evaluate the performance of subsystems and to troubleshoot system problems. The devices to be tested are typically radar warning receivers and elint receivers.

The purpose of this application note is to help the design engineer generate and evaluate complex radar signals using standard microwave test equipment. This application note shows how MATLAB® and the Agilent E8267C PSG vector signal generator can be used to create signals and simulate complex radar emitters. The appendices contain the complete MATLAB code for generating the pulse signals described in this application note.

Experience has shown that this type of series of complex pulse patterns will enable the user to perform roughly 80% of the tests necessary to evaluate the performance of an electronic warfare system. The final 20% of the receiver testing is typically done on a test range using real emitters.

Table of Contents

Agilent Technologies

Equipment Configuration

The general-purpose test equipment needed to evaluate a receiver requires two key components: a microwave signal generator capable of producing the signals required for the test, and a microwave spectrum analyzer capable of verifying the signal's characteristics. The equipment should cover a frequency range of 0.5–18 GHz. If the receiving system must process phase or frequency coded pulses, then the generator must be able to produce these signals.

Vector signal generator

The Agilent E8267C vector signal generator that covers 250 KHz to 20 GHz meets the needs for a general-purpose signal source to test radar warning receivers and elint systems. The generator is a member of the ESG/PSG line and provides excellent output power, low phase noise option, analog modulation, and digital communication modulation common to that line. The generator also provides I/Q modulation with an internal arbitrary waveform generator providing 80 MHz of modulation bandwidth. The E8267C's internal arbitrary waveform generator to produce complex radar signals.

Software programming tools

Creating custom radar signals requires a software-programming tool capable of dealing with complex array math and displaying the information in a usable format. While there are several tools available, MATLAB is widely used and commonly available. Agilent has chosen to support MATLAB as a waveform builder for the PSG.

To move the digital waveforms from MATLAB into the arbitrary waveform generator inside the PSG vector signal generator, Agilent has developed Download Assistant. The program enables users to easily download their IQ waveforms into the arbitrary waveform generator's memory. In addition, it allows the user to send any standard commands for programmable instruments (SCPI) command to the signal generator to control the instrument state. Download Assistant adds keywords to MATLAB to format and download arrays of data through common GPIB interface cards or a LAN interface into the signal generator. The examples used in this paper demonstrate how to use Download Assistant with MATLAB 6.5 or later revision to create, download, and generate radar signals. Download Assistant and the programming examples used in this paper can be obtained for free at the Agilent web site: http://www.agilent.com/find/psg.

Performance spectrum analyzer

To capture and demodulate signals the E4440A PSA Series spectrum analyzer is used with the 89601A vector signal analysis (VSA) software. This configuration provides frequency coverage of 26.5 GHz with up to 36 MHz of analysis bandwidth. Wide bandwidth configurations are also available, refer to **http://www.agilent.com/find/89600** for more information.

Generating a Simple Pulse

To adequately test elint receiver performance, a wide variety of test signals are needed. The user may need to simulate various types of radar emitters or to simulate the multiple modes of operation for a single type of radar. This requires the test engineer to control the basic pulse parameters: center frequency, power, pulse width (PW), and pulse repetition interval (PRI). Doing a reasonable simulation of a simple emitter also requires control of the rise time and fall time of the pulse. Shaping the rising and falling edge of the pulse enables the user to control the frequency spectrum of the waveform.

Generating a signal

To generate the signal, we will build an array in MATLAB that describes the in-phase and quadrature time domain waveforms, and download the arrays into the signal generator. Figure 1 shows a plot of a typical pulsed waveform.

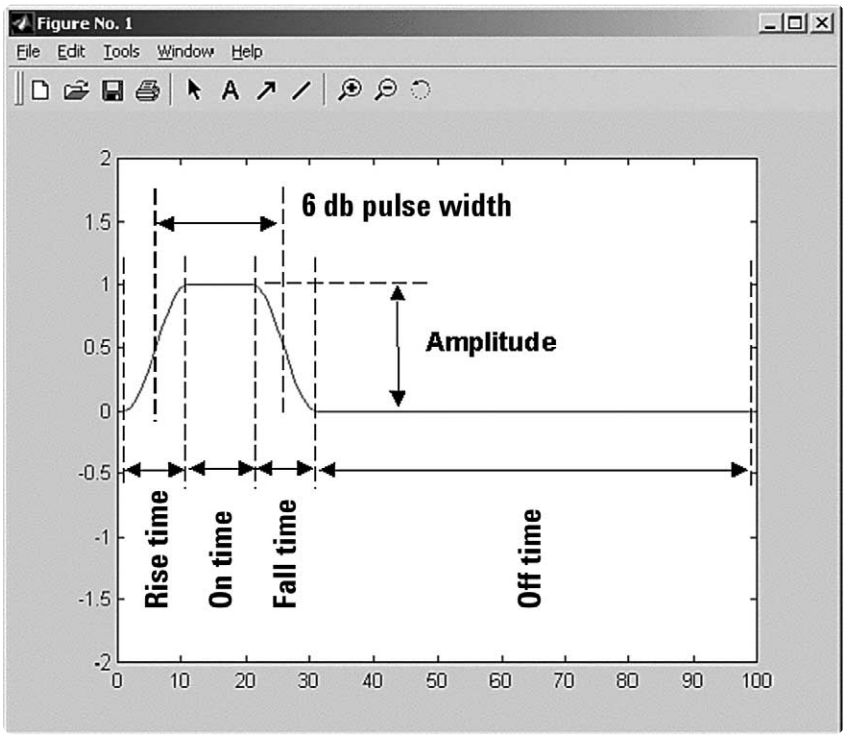


Figure 1: Typical pulsed waveform plot

The following block contains a subset of the code in Pulse.m

```
 sampclk = 100e6;              % ARB Sample Clock for playback

n=10;                          % number of pts in the rise & fall time
ramp=-1:2/n:1-2/n;             % ramp from -1 to almost +1 over n pts
rise=(1+sin(ramp*pi/2))/2;     % raised cos rise-time shape
on=oneon=ones(1,10);           % on-time characteristics
fall=(1+sin(-ramp*pi/2))/2;    % raised cos fall-time shape
off=zeros(1,70);               % defines the off-time characteristics

% build the pulse envelop
i=[rise on fall off];

% plot the i-samples and scale the plot
plot(i)
axis ([0 length(i) -2 2])

% set the q-samples to all zeroes
q=zeros(1,length(i));

IQData=[i + (j * q)];
```

In the program, to build the waveform the pulse is broken down into four parts: rise, on, fall, and off. The on and off sections of the pulse are built using the ones (1,10) and zeros (1,70) functions. In this case the ones command creates a 1 by 10 array and fills it with 1s. Likewise, the zeros command creates a 1 by 70 element array and fills it with 0s. This method provides a simple way to establish the on-time and off-time of the pulse.

The rising and falling edges of the pulse are shaped using a raised cosine function. To build the two cosine waveforms, the program starts by building a linear ramp from –1 to almost +1, using the function $ramp = -1:2/n:1-2/n$. For the ramp function, if the linear ramp were to continue, the point following the last point in the array would be exactly 1. Ramp functions are often multiplied by some multiple of π (represented in MATLAB by the variable pi) as part of a function to build sine waves. Given that –π and π represent the exact same point on the unit circle, when the ramp is multiplied by π and the sine or cosine taken of the array, a perfect sinusoidal waveform is produced. This idea will be used in several of the example programs.

In the case of the rising edge of the pulse, multiply the ramp with ±π/2 then take the sine of the result. This will produce the center of the sine wave with a first point of –1 and a final point of almost +1. Adding one to the result and dividing by two produces the desired waveform. The final equation takes the form:

rise=(1+sin(ramp*pi/2))/2

The falling edge is simply the negative of the rising edge. The amplitude envelop of the final pulse is built by concatenating the four arrays using the equation:

i=[rise on fall off]

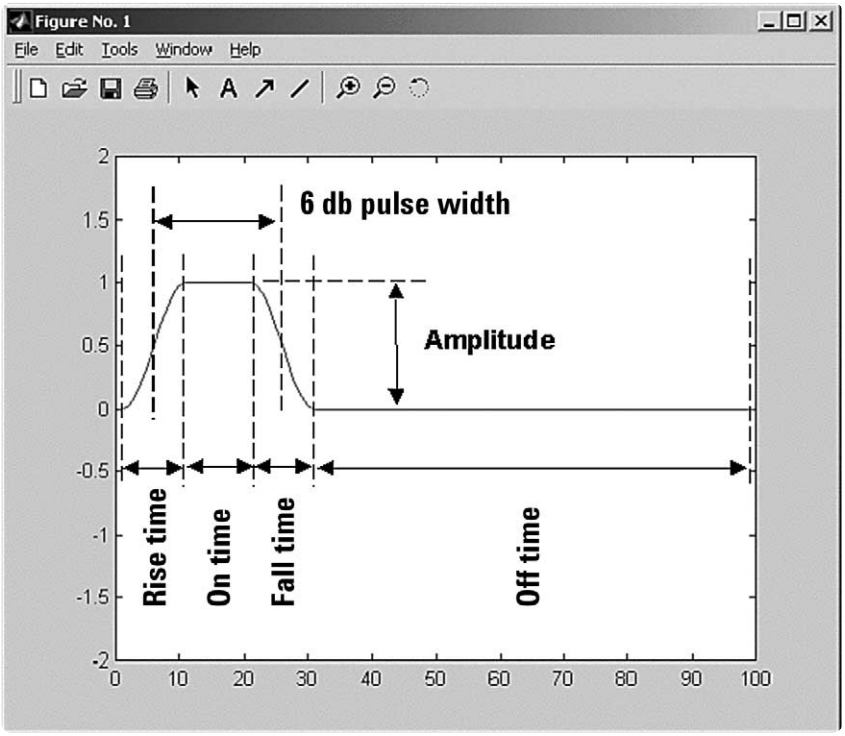


Figure 2: Final pulse plot

Because the phase of the pulse will be constant in this example, the imaginary portion of the array is set to zero using the formula:

q=zeros(1,length(i))

A single complex array is built from the two arrays using the formula:

IQData=[i + (j * q)]

Note that multiplying by j in this equation is the equivalent of multiplying by the square root of –1. This infers that q is the imaginary portion of the waveform. The waveform is now ready to be downloaded into the signal generator.

Setting the clock

The variable $smplclk = 100e6$ is used to set the clock frequency for the arbitrary waveform generator to 100 MHz. This allows time to be associated with each point in the waveform. Each point within the waveform will occupy 1/smplclk or 10 ns of time. The important timing characteristics of the pulse can be calculated using this information. The 0% to 100% rise-time and fall-time of the pulse is 10 ns*n or 100 ns where n describes the number of points in the arrays rise or fall.

While the 0% to 100% is useful during the construction of the waveform, it cannot be measured accurately on the microwave pulse. The 10% to 90% rise-time is a common pulse parameter and can easily be measured using standard test equipment. Given that the rising and falling edges of the pulse are built from raised cosine functions, it can be shown that the 10% to 90% rise-time is equal to .59 times the 0% to 100% rise-time. In this case, the 10% to 90% rise-time would equal 59 ns. Thus the rise-time of the pulse can be set and very accurately calculated by setting the value of n and $smplclk$.

In general, to insure the final output signal's rise-time is controlled by the calculated waveform and not the rise-time of the anti-alias filters following the arbitrary waveform generator in the signal generator, when the sample clock is set to its maximum value of 100 MHz, use four or more points in the rise-time waveform.

Pulse width and repetition

The pulse width and pulse repetition interval can easily be calculated. Typically, the pulse width is calculated based on the points 0.5 down from the amplitude of the pulse in a linear display, or 6 dB down from the amplitude of the pulse in a log display. Because the raised cosine function is symmetric around this point, the number of points in the 6 dB pulse width can be exactly calculated as the on-time, plus half of the rise-time, plus half of the fall time. The equation to calculate the pulse width in seconds would be:

pulse_width=(length(rise)/2)+length(on)+(length(fall)/2))/smplclk

The number of points in the pulse repetition interval is the rise-time, plus the on-time, plus the fall-time, plus the off-time. The equation to calculate the pulse width in seconds would be:

pulse_repetiton_interval=(length(rise)+length(on)+length(fall)
+length(off))/smplclk

This is an exact calculation and can be used as a standard when evaluating the performance of the signal processing within a receiver.

Controlling output power

To test a receiver's performance, it is critical to have known pulse power at the receiver input. Controlling the output power of the PSG from MATLAB is straightforward. When:

$$\sqrt{i^2+q^2} = 1$$

the output power of the signal generator will equal the front panel power level. For purposes of this paper, refer to the power level set at the front panel as the reference power level. For our example waveform, that sets the pulse amplitude in the real array to one and in the imaginary array to zero. If we set the output power of the signal generator to 0 dBm, then the peak power of the pulse will equal 0 dBm. The command from MATLAB (which uses Download Assistant) to set the output power is:

[status, status_description]=agt_sendcommand(io,'POWer 0');

Reducing the amplitude of the waveform below one will reduce the output power. However there are several scaling factors that must be addressed.

Automatic loop control

The first issue to deal with is automatic loop control (ALC) in the output of the signal generator. This feedback loop is used during the normal operation of a continuous wave (CW) source to hold the output power at a known level. For pulsed signals generated by the IQ modulator, the ALC will tend to drive the average power of the signal to equal the reference power driving the peak power well above the reference power. This becomes a real problem when the signal becomes more complex. It is good practice to turn off the ALC by putting the calibration process into manual mode. This can be done from the front panel of the instrument or from MATLAB. The command in MATLAB (which uses Download Assistant) to turn off ALC is:

```
[status, status_description]=agt_sendcommand(io,'POWer:ALC:STATe OFF');
```

Note that the ALC is part of the signal generator calibration process. Even when the ALC is turned off, the value for the output gain correction is held (but not updated) in a digital-to-analog converter and applied to the output. If the ALC is turned off for an extended period of time, the output calibration may drift. About once per day in laboratory conditions, it is a good practice to turn off the IQ modulation and press the **Manual Calibration** softkey under the **Power** hardkey to update the calibration.

Scaling factor

The second issue to deal with is real time IQ scaling, which is expressed as a linear percentage of the reference level. This scaling factor is applied to all IQ waveforms to easily enable users to specify a known back off for the arbitrary waveform generator drive level into the IQ modulator. Since the worst-case compression occurs at maximum input power for the IQ modulator, specifying a value below 100 percent may reduce the non-linear distortion produced by the IQ modulator. A value of 70 percent will reduce the output power by 3 dB. The command in MATLAB (which uses Agilent Download Assistant) to set the real time scaling value is:

```
[status,status_description]=agt_sendcommand(io,'RADio:ARB:RSCaling100')
```

Calibration

The final issue to deal with is user calibration. This is a feature that allows the user to compensate for frequency-dependent loss between the signal generator and the device under test (DUT). The automatic calibration process within the signal generator uses a GPIB power meter to measure the power at the DUT input to generate the user calibration array. When the signal generator is set to a new frequency, the processor within the PSG will correct for the losses and provide the displayed power at the DUT input. For detailed information about using this feature, refer to the E8267's User's Manual.

Running Pulse.m

The file Pulse.m is a complete MATLAB program used to generate and download a simple pulse into the PSG. A printout of the program can be found in Appendix A.

Note that the pulse parameters were modified from the simple pulse example illustrated above to produce a more realistic signal. The pulse is 1 ms wide with a pulse repetition interval of 10 ms. The reference level for the signal generator is set to 0 dBm, but the peak pulse amplitude of the waveform is set to 0.707, producing an output power of –3 dBm. Figure 3 illustrates the signal.

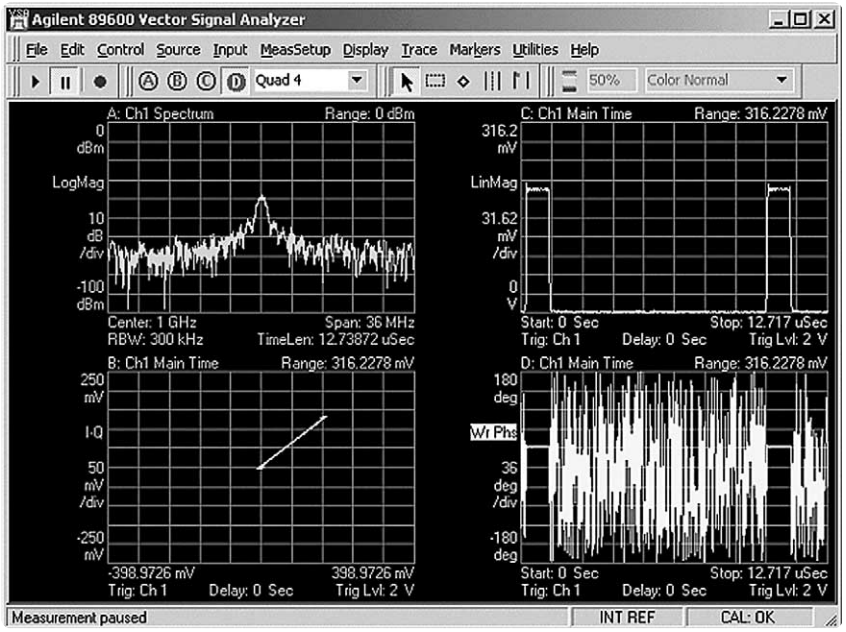


Figure 3: Screen capture from the VSA signal

The user should be able to justify the displayed results on the analyzer with the MATLAB program Pulse.m. The upper left hand plot shows the frequency spectrum of the signal. The upper right hand plot shows the linear amplitude time domain waveform. The lower left hand plot shows the IQ vector for the signal. The lower right hand plot displays the phase of the signal versus time. Observe the linear amplitude plot and note that the amplitude of the signal is constant within a single pulse and between pulses. From the phase plot versus time; note that the phase of the pulse is constant within a single pulse and between pulses. This infers that the PSG vector signal generator is coherent in frequency and phase with the vector signal analyzer.

Generating a Pulse Doublet (Doublet.m)

Having done the hard work of building a pulsed waveform and describing how each part of the program operates to create the signal, new waveforms may easily be built to stress some aspect of the system under test. The next signal is a pulse doublet, which are two pulses placed very close in time. When testing an elint system, a doublet is used to verify the ability of the system to correctly identify two closely spaced pulses rather than a single long pulse. Often a series of doublets will be created with a different amount of separation between each doublet. Also an amplitude change will be introduced between the pulses to place additional stress on the system.

Generating a pulse doublet using Pulse.m as a starting point is fairly straightforward. Define three new variables: pulse1, pulse2, and separation. The variables pulse1 and pulse2 will contain the amplitude envelope of each pulse. The amplitude of each pulse can be scaled independently of the other. The variable separation will define the number of points (and the time) between the two pulses. Finally, concatenate the new pulses with separation between them.

The following block contains a subset of the code in Doublet.m

```
separation=zeros(1,128);      % separation between the pulses

% define arrays which contain the pulse envelope for each pulse
pulse1 = [rise on fall];
pulse2 = .5*[rise on fall];

% concatenate and scale the pulses
i = .707*[pulse1 separation pulse2 off];
```

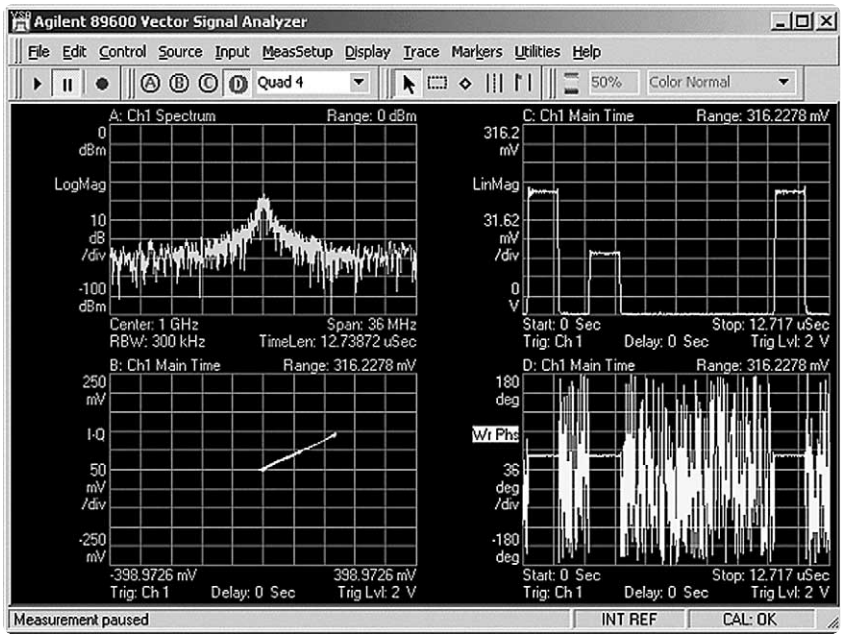


Figure 4: Screen capture of pulse doublet signal

Generating Phase Shift Between Pulses (PM_Doublet.m)

The next example will add a π/2 phase shift (90°) between the two pulses. Note that in the previous examples, the amplitude envelope was placed in the real array and zeros were placed in the imaginary array. For signals that require no phase or frequency modulation, that technique works fine and simplifies the math. For this example, the signals will be specified in terms of an am and pm array and converted into IQ.

The array for pm contains a constant but different phase during the on-time of each pulse. *Pulse1* is set to 0° and *pulse2* is set to π/2. Multiplying the am waveform times the sine or cosine of the pm waveform performs the IQ conversion. The period following the variable am instructs MATLAB to multiply the arrays on an element by element basis.

Note that in Figure 5 the π/2 phase shift between pulse1 and pulse2 shown in both the IQ plot and the phase versus time plot.

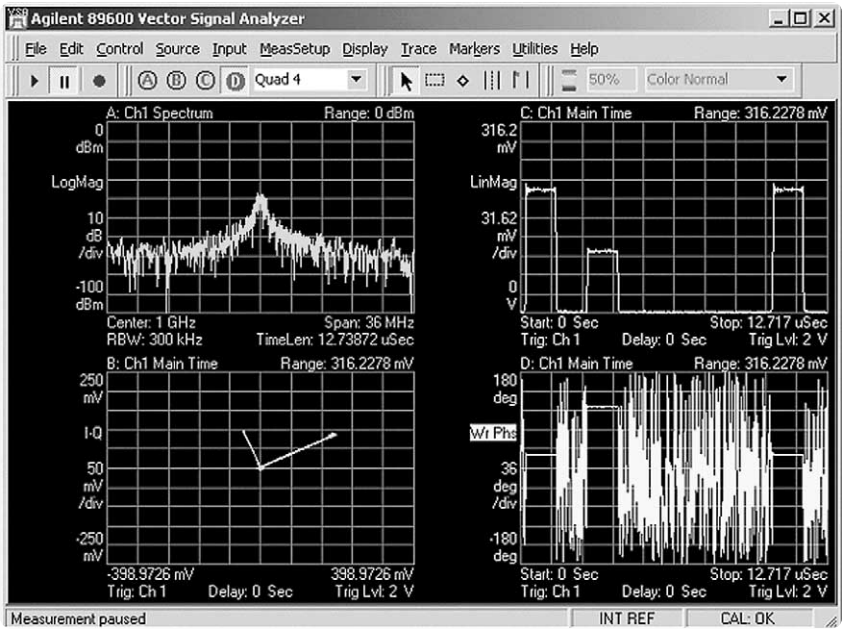


Figure 5: Screen capture of phase shift between pulses

The following block contains a subset of the code in PM_Doublet.m

```
% set the phase of the two pulses
pm = [0*ones(1,length(pulse1)) separation
(pi/2)*ones(1,length(pulse1)) off];

% convert am and pm to i and q
i=.707*am.* cos(pm);
q=.707*am.* sin(pm);
```

Creating Doppler Shift (Doppler.m)

Creating a pulsed waveform with a constant doppler frequency shift requires the introduction of a new technique. The idea is to build a waveform that contains the doppler offset frequency from the carrier versus time, then integrate the waveform to produce phase versus time. Remember that phase is simply the integral of frequency. In Figure 7 an array fm is produced that contains a constant offset frequency in hertz. The length of the array is equal to the entire am pulsed waveform. The fm waveform is integrated using the function cumsum, which is the cumulative sum of the elements of the array. The new array must be scaled by **2*π/sampclk** to obtain units of radians. The am and pm waveforms are converted into IQ and scaled for downloading into the signal generator.

Note that the math shown in the block below allows the user to specify the doppler offset frequency in hertz versus time and, given the sample clock in hertz, calculate the pm waveform. This is very powerful. While this example produces a static doppler offset within a single pulse, the technique can easily be extended to produce a doppler trajectory for a moving emitter. The primary limitation of the technique is the 64 MSa of memory within the signal generator to play the waveforms. With 100 MHz sample clock, the of memory will allow the production of 640 ms of unique signal.

The following block contains a subset of the code in Doppler.m

```
doppler_freq = 100e3;      % defines the doppler offset freq in Hz

% define an array which contains the doppler freq in each sample
fm=doppler_freq*ones(1,length(am));

% use an integral to translate from fm to pm
pm=(2*pi/sampclk)*cumsum(fm);

% convert am and pm to i and q and scale amplitude
i=.707*am.* cos(pm);
q=.707*am.* sin(pm);
```

Now to discuss the two vector displays of the doppler signal. Figure 6 shows the vector signal analyzer tuned to the exact center frequency of the signal generator. The phase versus time plot in the lower right hand corner shows a ramp in phase versus time during the on-time of the pulse. The IQ display in the lower left hand corner shows an arc of phase. From the parameters in the doppler.m MATLAB program, the pulse width can be calculated to be 1 ms and the doppler shift set to 100 KHz. The phase shift generated by a 100 KHz doppler shift over 1 ms should be:

(100 KHz)*(360º/cycle)*(1e–6 sec) = 36º

Note that the displays show 36º of phase shift during the on-time of the pulse.

In Figure 7 the vector signal analyzer has been tuned to the doppler offset frequency. Note that the phase of the pulse is constant and stable over at least two pulses. This infers that the signal generator and arbitrary waveform generator are coherent with the vector signal analyzer. This also provides confirmation that the math used to calculate the doppler waveform is correct.

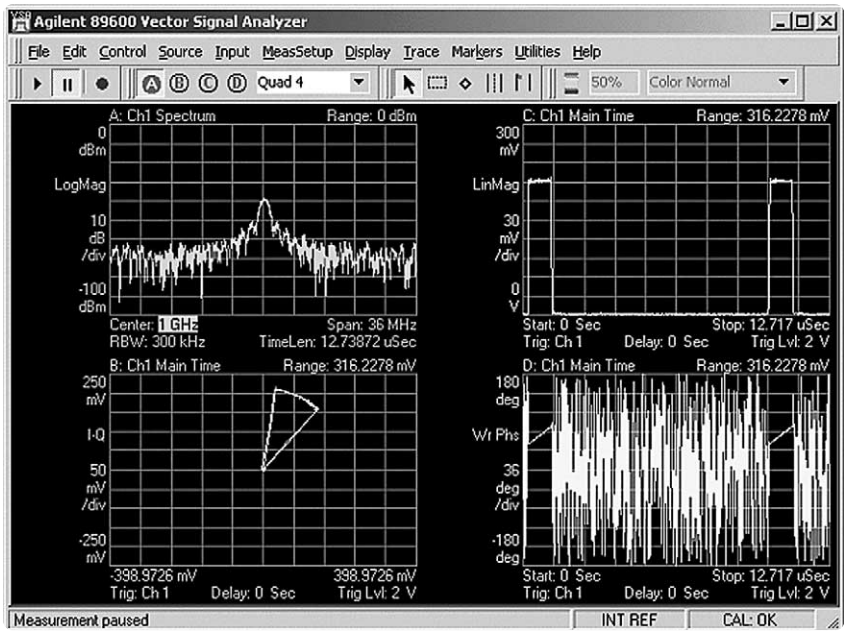


Figure 6: Doppler signal when VSA is tuned to center frequency of the signal generator

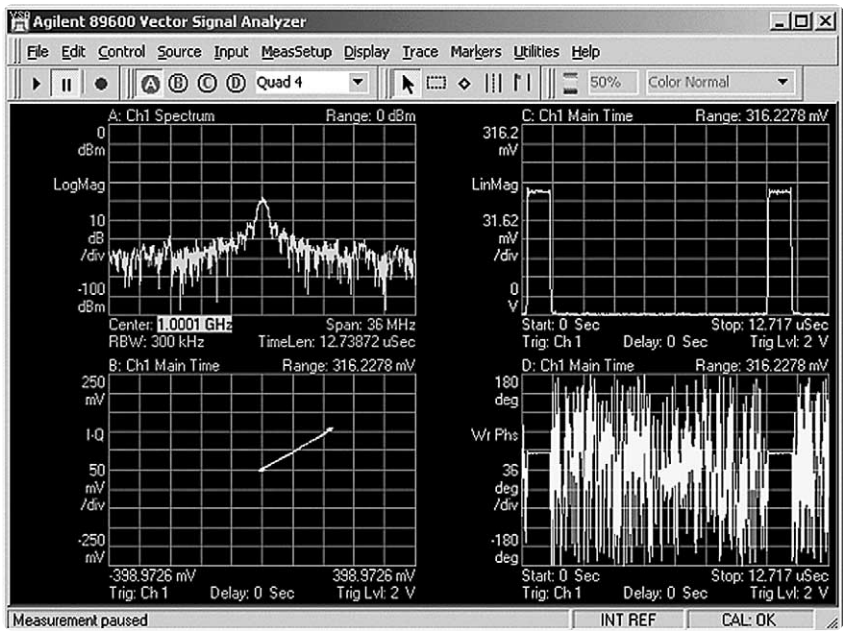


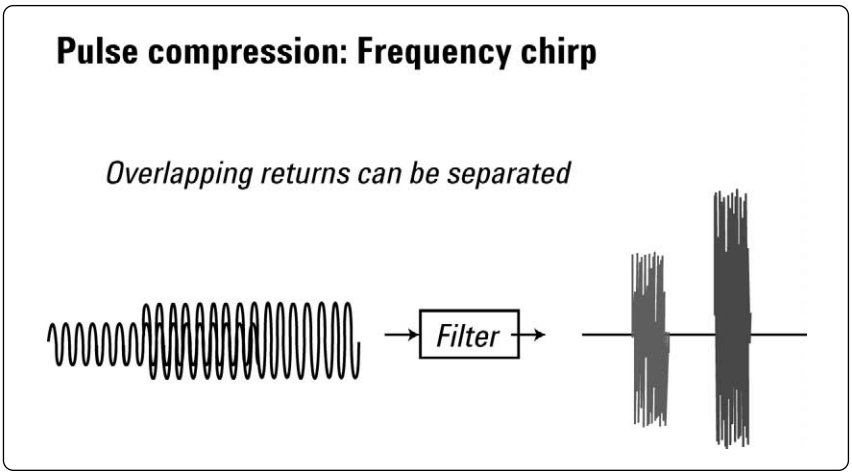
Figure 7: Doppler signal when VSA tuned to dopper offset frequency

Building Pulse Compression Signals

Having provided examples of simple pulsed waveforms with control of the amplitude and phase of the pulse, the next step will be to produce modulation within the pulse. Both phase and frequency modulation are used by radar systems to improve range and resolution. Radar systems that use modulation within the pulse are referred to as pulse compression radar systems.

The radar range equation points out a basic engineering trade-off between range and resolution and the need for pulse compression. To build a long-range radar (or for the radar to 'see' a great distance) the radar needs high average output power. To obtain good resolution, the radar needs a narrow pulse that reduces the average output power. Pulse compression provides a path around this trade-off. Pulse compression radars will transmit a long pulse with modulation inside the pulse. The returns are processed through a filter that is matched to the characteristics of the modulation compressing the pulse in time. This compression allows the radar to separate overlapping returns while transmitting a high average power.

It is important for elint and radar warning receivers to correctly process these types of signals. The modulation type and deviation of the signals provide important information about the purpose and intent of an observed system. It is often difficult to obtain a signal source with the appropriate characteristics to verify the performance of the elint system.



Generating a Barker Coded Pulse (Barker.m)

Barker coded signals are typical in pulse compression radar systems. Barker codes are binary numbers containing between 2 and 13 bits that have unique auto correlation functions. The points adjacent to the peak of the correlation function equal zero. This is very useful in a radar system since any spurious response can be misinterpreted as a target. A Barker coded pulse typically uses binary phase modulation. The chip rate is the dwell time for each bit within the pulse. In this example, we will build a 7-bit Barker coded waveform. The 7-bit Barker code contains the bits [+1 +1 +1 –1 –1 +1 –1].

To build the phase waveform, the seven bits of information must correctly encode into a binary phase shift keyed waveform and deal with the speed of the phase transitions. The transition time between phase states will at least, in part, determine the occupied bandwidth of the signal.

Within the program, first define the possible phase states and transitions as individual arrays, and then concatenate them into the final waveform. The two possible states the waveform can occupy are positive and negative, or +1 and –1. There are four possible transitions: negative to positive, positive to negative, negative to negative, and positive to positive. The states are built with a constant value over the chip period. The transitions are built using raised cosine functions. Note that the array rise was built as part of the am array, but it is used here as a 0 to 1 phase transition. At the end of the code sequence, the function rise-1 is used to provide a –1 to 0 transition. Having constructed the array with +1 and –1 states, multiply the waveform by $\pi/2$ to provide the appropriate phase deviation. The resulting waveform is converted to IQ and downloaded into the signal generator.

The following block contains a subset of the code in Barker.m

```
neg_pos=(1+sin(ramp*pi/2))-1;
pos_neg=(1+sin(-ramp*pi/2))-1;
pos_pos=ones(1,4);
neg_neg=-ones(1,4);
pos=ones(1,13);
neg=-ones(1,13);

pm=(pi/2)*[0 0 0 ...
    [rise pos]...       %Bit 1  high
    [pos_pos pos]...    %Bit 2  high
    [pos_pos pos]...    %Bit 3  high
    [pos_neg neg]...    %Bit 4  low
    [neg_neg neg]...    %Bit 5  low
    [neg_pos pos]...    %Bit 6  high
    [pos_neg neg]...    %Bit 7  low
    rise-1 0 0 off];

i=.707*am.* cos(pm);
q=.707*am.* sin(pm);
```

Figure 8 shows the demodulated signal. The lower right hand plot displays the demodulated phase versus time. Note that during the on-time of the pulse, the seven bits of the code are clearly visible. Markers may be used to verify the phase state accuracy and timing. The noise in phase between the pulses is due to the fact that during the off-time of the pulse the phase of the signal is undefined.

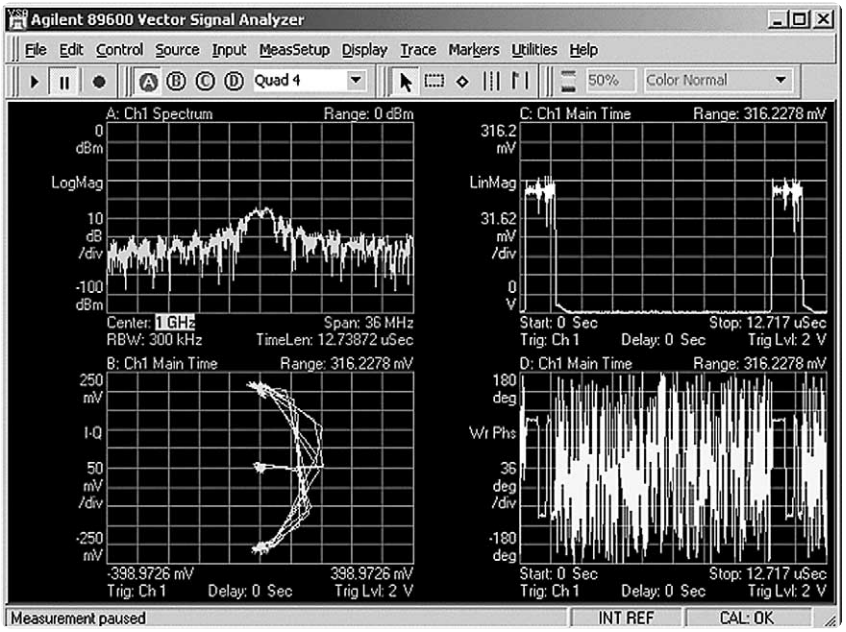


Figure 8: Demodulated signal

Generating a Linear FM Chirp (LFM_Chirp.m)

To demonstrate a linear fm chirp signal, the objective is to build an fm waveform that will linearly sweep the frequency across a known deviation. This example uses the technique introduced in the doppler example that integrates the fm waveform to produce a pm waveform which can be converted over to IQ. The advantage of this process is that it allows the user to build waveforms in frequency versus time to create arbitrary fm waveforms that can be converted and downloaded into the signal generator.

As the code below illustrates, the fm chirp is built using a ramp that starts at exactly –1 and ends at exactly +1. This allows the user to easily scale the waveform by multiplying by the desired frequency deviation divided by two. Note that the frequency will sweep both above and below the carrier frequency. Using the internal arbitrary waveform synthesizer, the signal generator can produce up to an 80 MHz chirp. To eliminate an unnecessary frequency step at the beginning and end of the chirp, the fm waveform is held at the frequency of the chirp endpoints during the rise-time and fall-time of the pulse.

In Figure 9 note the demodulated fm waveform in the lower right hand plot. The chirp is linear and the deviation is equal to the 10 MHz defined in the program (±5 MHz). The values in the fm demodulator are only defined during the on-time of the pulse. In the upper right hand plot, note that the amplitude of the pulse is flat during the pulse on-time. Because the signal is being swept across a 10 MHz frequency span, this indicates the IQ modulator within the signal generator provides a flat frequency response across that bandwidth. The spiral effect in the IQ plot in the lower left hand corner is due to the frequency offset from the carrier during the rise and fall time of the pulse.

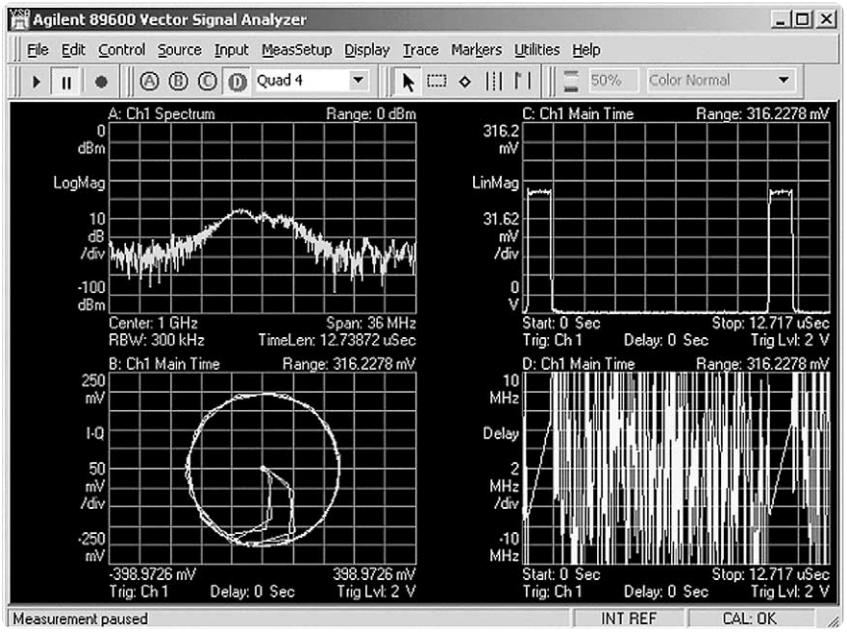


Figure 9: FM chirp signal

The following block contains a subset of the code in LFM_Chirp.m

```
chirp_dev = 10e6;          % defines the total chirp deviation in Hz

% define an array which contains the chirp waveform
fm=(chirp_dev/2)*([-ones(1,n) (-1:2/(length(on)-1):1) ones(1,n)
ones(1,length(off))]);

% use an integral to translate from fm to pm
pm=(2*pi/sampclk)*cumsum(fm);

% convert am and pm to i and q and scale amplitude
i = .707*am.* cos(pm);
q = .707*am.* sin(pm);
```

Generating a Non-Linear FM Chirp (NLFM_Chirp.m)

The final example will demonstrate how to add a known amount of non-linear distortion to the fm chirp waveform. The non-linear distortion will be produced by the single cycle of a sine wave scaled to fit within the on-time of the pulse. The amount of non-linearity is set by scaling the amplitude of the sine wave as some percentage of the total deviation. Because the value of the sine wave is zero at its end points the maximum deviation of the chirp will not change. The resulting S-shape of the waveform is typical of the distortion seen in non-synthesized chirped signals.

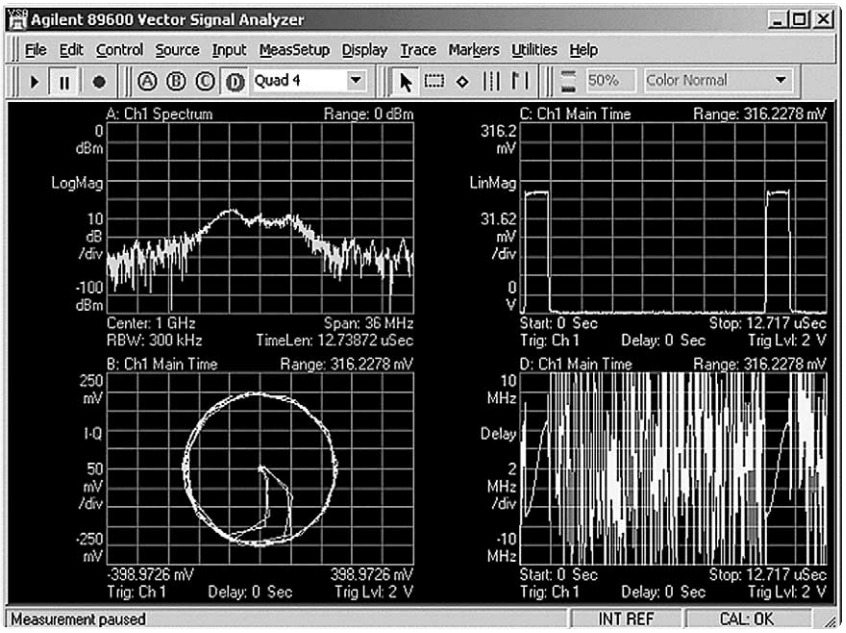


Figure 10: Non-linear fm chirp

The following block contains a subset of the code in NLFM_Chirp.m

```
chirp_dev = 10e6;       % defines the total chirp deviation in Hz

% create some non-linear distortion to add to the chirp
nonlinear=.2*sin((pi)*(-1:2/(length(on)-1):1));

% add the nonlinearity to the chirp and concatenate the sections
fm=(chirp_dev/2)*([-ones(1,n) nonlinear+(-1:2/(length(on)-1):1)
    ones(1,n) ones(1,length(off))]);

% use an integral to translate from fm to pm
pm=(2*pi/sampclk)*cumsum(fm);

% convert am and pm to i and q and scale amplitude
i =.707*am.* cos(pm);
q=.707*am.* sin(pm);
```

Appendix A: Simple pulse

```
% Script file: Pulse.m
%
% Purpose:
%   To calculate and download an arbitrary waveform file to generate a
%   simple pulsed signal with the PSG vector signal generator.
%
%   Record of revisions:
%
%   Date       Programmer       Description of change
%   ====       ==========       =====================
%   4/15/2002  Randal Burnette  Version for 2002 AD Symposium in MatLab/VEE
%   8/14/2002  John Hutmacher   Added comments and Download Assistant
%   9/4/2002   Randal Burnette  Added Preset, turned ALC off, and IQ Scaling
%   6/26/2003  Randal Burnette  Added Modulation ON
%
%   Define variables:
%
%   n          -- counting variable (no units)
%   ramp       -- ramp from -1 to almost +1; used to build sine waves
%   rise       -- raised cosine pulse rise-time definition (samples)
%   on         -- pulse on-time definition (samples)
%   fall       -- raised cosine pulse fall-time definition (samples)
%   off        -- pulse off-time definition (samples)
%   inphase    -- in-phase modulation signal (samples)
%   quadrature -- quadrature modulation signal (samples)
%   IQData     -- complex array containing both i and q waveform samples
%   Markers    -- array containing markers for Event Markers 1 and 2
%   sampclk    -- clock freq for the D/A converters in the IQ modulator


sampclk = 100e6;           % ARB Sample Clock for playback

n=4;                       % number of points in the rise & fall time
ramp=-1:2/n:1-2/n;         % ramp from -1 to almost +1 over n points
rise=(1+sin(ramp*pi/2))/2; % defines the raised cos rise-time shape
on=ones(1,92);             % defines the on-time characteristics
fall=(1+sin(-ramp*pi/2))/2; % defines the raised cos fall-time shape
off=zeros(1,900);          % defines the off-time characteristics

% build the pulse envelop
inphase = [rise on fall off];

% plot the i-samples and scale the plot
plot(inphase)
axis ([0 length(inphase) -2 2])

% set the q-samples to all zeroes
quadrature = zeros(1,length(inphase));

% define a composite iq matrix for download to the PSG using the
% PSG/ESG Download Assistant
IQData = [inphase + (j * quadrature)];
```

Appendix A: Simple pulse (continued)

```
% define a matrix and activate a marker for the beginning of the waveform
Markers = zeros(2,length(IQData));%fill Marker array with zero ie. no markers set
Markers(1,1:10) = 1;%set Marker to first ten points of playback

% make a new connection to the PSG over the GPIB interface
io = agt_newconnection('gpib',0,19);

%verify that communication with the PSG has been established
[status, status_description,query_result] = agt_query(io,'*idn?');
if (status < 0) return; end

% preset the instrument
[status, status_description] = agt_sendcommand(io,':STATus:PRESet');

% set carrier frequency and power on the PSG using the PSG Download Assistant
[status, status_description] = agt_sendcommand(io, 'SOURce:FREQuency 1e9');
[status, status_description] = agt_sendcommand(io, 'POWer 0');

% put the ALC into manual control
[status, status_description] = agt_sendcommand(io, 'POWer:ALC:STATe OFF');

% set the IQ real time scaling to 70.7% or -3dB
[status, status_description] = agt_sendcommand(io, 'RADio:ARB:RSCaling 70.7');

% download the iq waveform the PSG baseband generator for playback
[status, status_description] = agt_waveformload(io, IQData, 'pulse', sampclk,
'play', 'no_normscale', Markers);

% Turn on modulation
[status, status_description ] = agt_sendcommand( io, 'OUTPut:MODulation:STATe ON');

% Turn on RF output power
[status, status_description ] = agt_sendcommand( io, 'OUTPut:STATe ON');
```

Appendix B: Pulse doublet

```
%  Script file: Doublet.m
%
%  Purpose:
%    To calculate and download an arbitrary waveform file to generate a
%    doublet (two simple pulses) within a single PRI with the PSG
%    vector signal generator.
%
%    Record of revisions:
%
%    Date          Programmer         Description of change
%    ====          ==========         =====================
%    4/15/2002     Randal Burnette    Initial version for 2002 AD Symposium
%    8/14/2002     John Hutmacher     Added comments and Download Assistant
%    9/4/2002      Randal Burnette    Added Preset, turned ALC off, and IQ Scaling
%    6/26/2003     Randal Burnette    Added Modulation ON
%
%    Define variables:
%
%    n             -- counting variable (no units)
%    ramp          -- ramp from -1 to almost +1; used to build sine waves
%    rise          -- raised cosine pulse rise-time definition (samples)
%    on            -- pulse on-time definition (samples)
%    fall          -- raised cosine pulse fall-time definition (samples)
%    off           -- pulse off-time definition (samples)
%    i             -- in-phase modulation signal (samples)
%    q             -- quadrature modulation signal (samples)
%    IQData        -- complex array containing both i and q waveform samples
%    Markers       -- array containing markers for Event Markers 1 and 2
%    seperation    -- array containing the time separation between the pulses

n=4;                      % defines the number of points in the rise-time & fall-time
ramp=-1:2/n:1-2/n;        % ramp from -1 to almost +1 over n points
rise=(1+sin(ramp*pi/2))/2;% defines the raised cos rise-time shape
on=ones(1,120);           % defines the on-time characteristics
fall=(1+sin(-ramp*pi/2))/2;% defines the raised cos fall-time shape
off=zeros(1,640);         % defines the off-time sample points
seperation=zeros(1,128);  % defines the seperation between the pulses

% define arrays which contain the pulse envelope for each pulse
pulse1 = [rise on fall];
pulse2 = .5*[rise on fall];

% concatenate and scale the pulses
i = [pulse1 seperation pulse2 off];

% plot the i-samples and scale the plot
plot(i)
axis ([0 length(i) -2 2])

% set the q-samples to all zeroes
q = zeros(1,length(i));
```

Appendix B: Pulse doublet (continued)

```
% define a composite iq matrix for download to the PSG using the
% PSG/ESG Download Assistant
IQData = [i + (j * q)];

% define a marker matrix and activate a marker to indicate the beginning of the waveform
Markers = zeros(2,length(IQData)); %fill Marker array with zero ie. no markers set
Markers(1,1:10) = 1; %set Marker to first ten points of playback

% make a new connection to the PSG over the GPIB interface
io = agt_newconnection('gpib',0,19);

% verify that communication with the PSG has been established
[status, status_description,query_result] = agt_query(io,'*idn?');
if (status < 0) return; end

% preset the instrument
[status, status_description] = agt_sendcommand(io,':STATus:PRESet');

% set carrier frequency and power on the PSG using the PSG Downlaod Assistant
[status, status_description] = agt_sendcommand(io, 'SOURce:FREQuency 1e9');
[status, status_description] = agt_sendcommand(io, 'POWer 0');

% put the ALC into manual control and set the IQ real time scaling
[status, status_description] = agt_sendcommand(io, 'POWer:ALC:STATe OFF');
[status, status_description] = agt_sendcommand(io, 'RADio:ARB:RSCaling 70.7');

% defines the ARB Sample Clock for playback
sampclk = 100000000;

% download the iq waveform the PSG baseband generator for playback
[status, status_description] = agt_waveformload(io, IQData, 'doublet', sampclk, 'play',
'no_normscale', Markers);

% Turn on modulation
[status, status_description ] = agt_sendcommand( io, 'OUTPut:MODulation:STATe ON');

% Turn on RF output power
[status, status_description ] = agt_sendcommand( io, 'OUTPut:STATe ON' );
```

Appendix C: Pulse doublet with phase offset

```
% Script file:  Phase_Offset_Doublet.m
%
% Purpose:
%   To calculate and download an arbitrary waveform file to generate a
%   doublet (two simple pulses) within a single PRI and that has pi/2
%   (or 90 deg) phase offset between the pulses.
%
%   Record of revisions:
%
%   Date          Programmer         Description of change
%   ====          ==========         =====================
%   4/15/2002     Randal Burnette    Initial version for 2002 AD Symposium
%   8/14/2002     John Hutmacher     Added comments and Download Assistant
%   9/4/2002      Randal Burnette    Added Preset, turned ALC off, and IQ Scaling
%   6/26/2003     Randal Burnette    Added Modulation ON
%
%   Define variables:
%
%   n          -- counting variable (no units)
%   ramp       -- ramp from -1 to almost +1; used to build sine waves
%   rise       -- raised cosine pulse rise-time definition (samples)
%   on         -- pulse on-time definition (samples)
%   fall       -- raised cosine pulse fall-time definition (samples)
%   off        -- pulse off-time definition (samples)
%   i          -- in-phase modulation signal (samples)
%   q          -- quadrature modulation signal (samples)
%   IQData     -- complex array containing both i and q waveform samples
%   Markers    -- array containing markers for Event Markers 1 and 2
%   separation -- array containing the time separation between the pulses
%   am         -- amplitude envelope for the pulse, linear units
%   pm         -- phase of the pulse vs time in rads

sampclk = 100e6;  % defines the ARB Sample Clock for playback

n=4;                      % defines the number of points in the rise-time & fall-
time
ramp=-1:2/n:1-2/n;        % ramp from -1 to almost +1 over n points
rise=(1+sin(ramp*pi/2))/2; % defines the raised cos rise-time shape
on=ones(1,120);           % defines the on-time characteristics
fall=(1+sin(-ramp*pi/2))/2; % defines the raised cos fall-time shape
off=zeros(1,640);         % defines the off-time sample points
separation=zeros(1,128);  % defines the separation between the pulses

% define arrays which contain the pulse envelope for each pulse
pulse1 = [rise on fall];
pulse2 = .5*[rise on fall];

% concatenate and scale the pulses
am = [pulse1 separation pulse2 off];

% set the phase of the first pulse to 0 rad and the second to pi/2 rad
pm = [0*ones(1,length(pulse1)) separation (pi/2)*ones(1,length(pulse1)) off];
```

Appendix C: Pulse doublet with phase offset (continued)

```
% plot the amplitude envelope and scale the plot
% plot(am)
% axis ([0 length(am) -2 2])

% convert am and pm to i and q
i = am.* cos(pm);
q = am.* sin(pm);

% define a composite iq matrix for download to the PSG using the
% PSG/ESG Download Assistant
IQData = [i + (j * q)];

% define a marker matrix and activate a marker to indicate the beginning of the waveform
Markers = zeros(2,length(IQData)); %fill Marker array with zero ie. no markers set
Markers(1,1:10) = 1; %set Marker to first ten points of playback

% make a new connection to the PSG over the GPIB interface
io = agt_newconnection('gpib',0,19);

% verify that communication with the PSG has been established
[status, status_description,query_result] = agt_query(io,'*idn?');
if (status < 0) return; end

% preset the instrument
[status, status_description] = agt_sendcommand(io,':STATus:PRESet');

% set carrier frequency and power on the PSG using the PSG Download Assistant
[status, status_description] = agt_sendcommand(io, 'SOURce:FREQuency 1e9');
[status, status_description] = agt_sendcommand(io, 'POWer 0');

% put the ALC into manual control and set the IQ real time scaling
[status, status_description] = agt_sendcommand(io, 'POWer:ALC:STATe OFF');
[status, status_description] = agt_sendcommand(io, 'RADio:ARB:RSCaling 70.7');

% download the iq waveform the PSG baseband generator for playback
[status, status_description] = agt_waveformload(io, IQData, 'phase_offset',
sampclk, 'play', 'no_normscale', Markers);

% Turn on modulation
[status, status_description ] = agt_sendcommand( io, 'OUTPut:MODulation:STATe ON');

% Turn on RF output power
[status, status_description ] = agt_sendcommand( io, 'OUTPut:STATe ON' );
```

Appendix D: Pulse with doppler frequency offset

```
% Script file:  Doppler.m
%
% Purpose:
%   To calculate and download an arbitrary waveform file that simulates a
%   simple pulse signal with a fixed doppler frequency offset from the
%   center frequency of the signal generator using IQ modulation.
%
%   Record of revisions:
%
%   Date          Programmer         Description of change
%   ====          ==========         =====================
%   4/15/2002     Randal Burnette    Initial version for 2002 AD Symposium
%   8/14/2002     John Hutmacher     Added comments and Download Assistant
%   9/4/2002      Randal Burnette    Added Preset, turned ALC off, and IQ Scaling
%   6/26/2003     Randal Burnette    Added Modulation ON
%
%   Define variables:
%
%   n             -- counting variable (no units)
%   ramp          -- ramp from -1 to almost +1; used to build sine waves
%   rise          -- raised cosine pulse rise-time definition (samples)
%   on            -- pulse on-time definition (samples)
%   fall          -- raised cosine pulse fall-time definition (samples)
%   off           -- pulse off-time definition (samples)
%   i             -- in-phase modulation signal (samples)
%   q             -- quadrature modulation signal (samples)
%   IQData        -- complex array containing both i and q waveform samples
%   Markers       -- array containing markers for Event Markers 1 and 2
%   am            -- amplitude envelope for the pulse, linear units
%   pm            -- phase of the pulse vs time in rads
%   fm            -- offset frequency from carrier vs time in Hz
%   sampclk       -- clock freq for the D/A converters in the IQ modulator
%   doppler_freq  -- doppler offset frequency in Hz

sampclk = 100e6;              % defines the ARB Sample Clock for playback
doppler_freq = 100e3;         % defines the doppler offset freq in Hz

n=4;                          % defines the number of points in the rise & fall time
ramp=-1:2/n:1-2/n;            % ramp from -1 to almost +1 over n points
rise=(1+sin(ramp*pi/2))/2;    % defines the raised cos rise-time shape
on=ones(1,92);                % defines the on-time characteristics
fall=(1+sin(-ramp*pi/2))/2;   % defines the raised cos fall-time shape
off=zeros(1,900);             % defines the off-time sample points

% concatenate the parts of the amplitude of the pulse into a single array
am = [rise on fall off];

% plot the am-samples and scale the plot
% plot(am)
% axis ([0 length(am) -2 2])
```

Appendix D: Pulse with doppler frequency offset (continued)

```
% define an array which contains the the doppler freq in each sample
fm=doppler_freq*ones(1,length(am));

% use an intergral to translate from fm to pm
pm=(2*pi/sampclk)*cumsum(fm);

% convert am and pm to i and q
i= am.* cos(pm);
q= am.* sin(pm);

% define a composite iq matrix for download to the PSG using the
% PSG/ESG Download Assistant
IQData = [i + (j * q)];

% define a marker matrix and activate a marker to indicate the beginning of the waveform
Markers = zeros(2,length(IQData)); %fill Marker array with zero ie. no markers set
Markers(1,1:10) = 1; %set Marker to first ten points of playback

% make a new connection to the PSG over the GPIB interface
io = agt_newconnection('gpib',0,19);

% verify that communication with the PSG has been established
[status, status_description,query_result] = agt_query(io,'*idn?');
if (status < 0) return; end

% preset the instrument
[status, status_description] = agt_sendcommand(io,':STATus:PRESet');

% set carrier frequency and power on the PSG using the PSG Downlaod Assistant
[status, status_description] = agt_sendcommand(io, 'SOURce:FREQuency 1e9');
[status, status_description] = agt_sendcommand(io, 'POWer 0');

% put the ALC into manual control and set the IQ real time scaling
[status, status_description] = agt_sendcommand(io, 'POWer:ALC:STATe OFF');
[status, status_description] = agt_sendcommand(io, 'RADio:ARB:RSCaling 70.7');

% download the iq waveform the PSG baseband generator for playback
[status, status_description] = agt_waveformload(io, IQData, 'doppler',
sampclk, 'play', 'no_normscale', Markers);

% Turn on modulation
[status, status_description ] = agt_sendcommand( io, 'OUTPut:MODulation:STATe ON');

% Turn on RF output power
[status, status_description ] = agt_sendcommand( io, 'OUTPut:STATe ON' );
```

Appendix E: Pulse with barker code

```
% Script file:  Barker.m
%
% Purpose:
%   To calculate and download an arbitrary waveform file that simulates a
%   simple 7 bit barker RADAR signal to the PSG vector signal generator.
%
%   Record of revisions:
%
%   Date          Programmer         Description of change
%   ====          ==========         =====================
%   4/15/2002     Randal Burnette    Initial version for 2002 AD Symposium
%   8/14/2002     John Hutmacher     First draft
%   9/4/2002      Randal Burnette    Added Preset, turned ALC off, and IQ Scaling
%   6/26/2003     Randal Burnette    Added Modulation ON
%
%
%   Define pulse variables:
%
%   n          -- counting variable (no units)
%   ramp       -- ramp from -1 to almost +1; used to build sine waves
%   rise       -- raised cosine pulse rise-time definition (samples)
%   on         -- pulse on-time definition (samples)
%   fall       -- raised cosine pulse fall-time definition (samples)
%   off        -- pulse off-time definition (samples)
%   i          -- in-phase modulation signal (samples)
%   q          -- quadrature modulation signal (samples)
%   pm         -- phase modulation
%   sampclk    -- clock freq for the D/A converters in the IQ modulator
%   neg_pos    -- transition from low bit to high bit
%   pos_neg    -- transition form high bit to low bit
%   pos_pos    -- defines high bit
%   neg_neg    -- defines low bit
%   pos        -- defines high bit
%   neg        -- defines low bit

sampclk = 100e6;  % defines the ARB Sample Clock for playback

n=4;                      % defines the number of points in the rise-time & fall-time
ramp=-1:2/n:1-2/n;        % number of points translated to time
rise=(1+sin(ramp*pi/2))/2; % defines the pulse rise-time shape
on=ones(1,120);           % defines the pulse on-time characteristics
fall=(1+sin(-ramp*pi/2))/2;% defines the pulse fall-time shape
off=zeros(1,896);         % defines the pulse off-time characteristics
am=[rise on fall off];    % defines the pulse envelope

neg_pos=(1+sin(ramp*pi/2))-1;   %
pos_neg=(1+sin(-ramp*pi/2))-1;
pos_pos=ones(1,4);
neg_neg=-ones(1,4);
pos=ones(1,13);
neg=-ones(1,13);
```

Appendix E: Pulse with barker code (continued)

```
pm=(pi/2)*[0 0 0 ...
     [rise pos]...              %Bit 1  high
     [pos_pos pos]...           %Bit 2  high
     [pos_pos pos]...           %Bit 3  high
     [pos_neg neg]...           %Bit 4  low
     [neg_neg neg]...           %Bit 5  low
     [neg_pos pos]...           %Bit 6  high
     [pos_neg neg]...           %Bit 7  low
     rise-1 0 0 off];

% plot the pm-samples and scale the plot
plot(pm)
axis ([0 length(pm) -2 2])

% convert am and pm to i and q
i= am.* cos(pm);
q= am.* sin(pm);

% define a composite iq matrix for download to the PSG using the
% PSG/ESG Download Assistant
IQData = [i + (j * q)];

% define a marker matrix and activate a marker to indicate the beginning of the waveform
Markers = zeros(2,length(IQData)); %fill Marker array with zero ie. no markers set
Markers(1,1:10) = 1; %set Marker to first ten points of playback

% make a new connection to the PSG over the GPIB interface
io = agt_newconnection('gpib',0,19);

% verify that communication with the PSG has been established
[status, status_description,query_result] = agt_query(io,'*idn?');
if (status < 0) return; end

% preset the instrument
[status, status_description] = agt_sendcommand(io,':STATus:PRESet');

% set carrier frequency and power on the PSG using the PSG Download Assistant
[status, status_description] = agt_sendcommand(io, 'SOURce:FREQuency 1e9');
[status, status_description] = agt_sendcommand(io, 'POWer 0');

% put the ALC into manual control and set the IQ real time scaling
[status, status_description] = agt_sendcommand(io, 'POWer:ALC:STATe OFF');
[status, status_description] = agt_sendcommand(io, 'RADio:ARB:RSCaling 70.7');

% download the iq waveform the PSG baseband generator for playback
[status, status_description] = agt_waveformload(io, IQData, 'barker',
sampclk, 'play', 'no_normscale', Markers);

% Turn on modulation
[status, status_description ] = agt_sendcommand( io, 'OUTPut:MODulation:STATe ON');

% Turn on RF output power
[status, status_description ] = agt_sendcommand( io, 'OUTPut:STATe ON' );
```

Appendix F: Pulse with linear fm chirp

```
% Script file:  LFM_Chirp.m
%
% Purpose:
%   To calculate and download an arbitrary waveform file that simulates a
%   pulsed signal with a linear fm chirp to the PSG vector signal generator.
%
%   Record of revisions:
%
%   Date          Programmer         Description of change
%   ====          ==========         =====================
%   4/15/2002     Randal Burnette    Initial version for 2002 AD Symposium
%   8/14/2002     John Hutmacher     Added comments and Download Assistant
%   9/4/2002      Randal Burnette    Added Preset, turned ALC off, and IQ Scaling
%                                    and corrected fm to pm integration calc
%   6/26/2003  Randal Burnette       Added Modulation ON
%
%   Define variables:
%
%   n             -- counting variable (no units)
%   ramp          -- ramp from -1 to almost +1; used to build sine waves
%   rise          -- raised cosine pulse rise-time definition (samples)
%   on            -- pulse on-time definition (samples)
%   fall          -- raised cosine pulse fall-time definition (samples)
%   off           -- pulse off-time definition (samples)
%   ontime        -- total number of points in the rise + on + fall
%   i             -- in-phase modulation signal (samples)
%   q             -- quadrature modulation signal (samples)
%   IQData        -- complex array containing both i and q waveform samples
%   Markers       -- array containing markers for Event Markers 1 and 2
%   am            -- amplitude envelope for the pulse, linear units
%   pm            -- phase of the pulse vs time in rads
%   fm            -- offset frequency from carrier vs time in Hz
%   sampclk       -- clock freq for the D/A converters in the IQ modulator
%   chirp_dev     -- total chirp frequency deviation in Hz

sampclk = 100e6;  % defines the ARB Sample Clock for playback
chirp_dev = 10e6; % defines the total chirp deviation in Hz

n=4;                     % defines the number of points in the rise-time & fall-time
ramp=-1:2/n:1-2/n;       % ramp from -1 to almost +1 over n points
rise=(1+sin(ramp*pi/2))/2; % defines the raised cos rise-time shape
on=ones(1,92);           % defines the on-time characteristics
fall=(1+sin(-ramp*pi/2))/2;% defines the raised cos fall-time shape
off=zeros(1,900);        % defines the off-time sample points

% concatenate the parts of the amplitude of the pulse into a single array
am = [rise on fall off];

% define an array which contains the the chirp waveform
fm=(chirp_dev/2)*([-ones(1,n) (-1:2/(length(on)-1):1) ones(1,n) ones(1,length(off))]);

% use an integral to translate from fm to pm
pm=(2*pi/sampclk)*cumsum(fm);
```

Appendix F: Pulse with linear fm chirp (continued)

```
%  plot the fm-samples and scale the plot
plot(fm);
axis ([0 length(fm) -10e6 10e6]);

% convert am and pm to i and q and scale amplitude
i = am.* cos(pm);
q = am.* sin(pm);

% define a composite iq matrix for download to the PSG using the
% PSG/ESG Download Assistant
IQData = [i + (j * q)];

% define a marker matrix and activate a marker to indicate the beginning of the
waveform
Markers = zeros(2,length(IQData)); %fill Marker array with zero ie. no markers set
Markers(1,1:10) = 1; %set Marker to first ten points of playback

% make a new connection to the PSG over the GPIB interface
io = agt_newconnection('gpib',0,19);

% verify that communication with the PSG has been established
[status, status_description,query_result] = agt_query(io,'*idn?');
if (status < 0) return; end

% preset the instrument
[status, status_description] = agt_sendcommand(io,':STATus:PRESet');

% set carrier frequency and power on the PSG using the PSG Download Assistant
[status, status_description] = agt_sendcommand(io, 'SOURce:FREQuency 1e9');
[status, status_description] = agt_sendcommand(io, 'POWer 0');

% put the ALC into manual control and set the IQ real time scaling
[status, status_description] = agt_sendcommand(io, 'POWer:ALC:STATe OFF');
[status, status_description] = agt_sendcommand(io, 'RADio:ARB:RSCaling 70.7');

% download the iq waveform the PSG baseband generator for playback
[status, status_description] = agt_waveformload(io, IQData, 'lfm', sampclk,
'play', 'no_normscale', Markers);

% Turn on modulation
[status, status_description ] = agt_sendcommand( io, 'OUTPut:MODulation:STATe ON');

% Turn on RF output power
[status, status_description ] = agt_sendcommand( io, 'OUTPut:STATe ON' );
```

Appendix G: Pulse with non-linear fm chirp

```
% Script file:  NLFM_Chirp.m
%
% Purpose:
%   To calculate and download an arbitrary waveform file that simulates a
%   pulsed signal with a non-linear fm chirp to the PSG vector signal generator.
%
%   Record of revisions:
%
%   Date          Programmer         Description of change
%   ====          ==========         =====================
%   4/15/2002     Randal Burnette    Initial version for 2002 AD Symposium
%   8/14/2002     John Hutmacher     Added comments and Download Assistant
%   9/4/2002      Randal Burnette    Added Preset, turned ALC off, and IQ Scaling
%   6/26/2003     Randal Burnette    Added Modulation ON
%
%   Define variables:
%
%   n             -- counting variable (no units)
%   ramp          -- ramp from -1 to almost +1; used to build sine waves
%   rise          -- raised cosine pulse rise-time definition (samples)
%   on            -- pulse on-time definition (samples)
%   fall          -- raised cosine pulse fall-time definition (samples)
%   off           -- pulse off-time definition (samples)
%   ontime        -- total number of points in the rise + on + fall
%   i             -- in-phase modulation signal (samples)
%   q             -- quadrature modulation signal (samples)
%   IQData        -- complex array containing both i and q waveform samples
%   Markers       -- array containing markers for Event Markers 1 and 2
%   am            -- amplitude envelope for the pulse, linear units
%   pm            -- phase of the pulse vs time in rads
%   fm            -- offset frequency from carrier vs time in Hz
%   sampclk       -- clock freq for the D/A converters in the IQ modulator
%   chirp_dev     -- total chirp frequency deviation in Hz

sampclk = 100e6;         % defines the ARB Sample Clock for playback
chirp_dev = 10e6;        % defines the total chirp deviation in Hz

n=4;                     % defines the number of points in the rise-time & fall-
time
ramp=-1:2/n:1-2/n;       % ramp from -1 to almost +1 over n points
rise=(1+sin(ramp*pi/2))/2; % defines the raised cos rise-time shape
on=ones(1,92);           % defines the on-time characteristics
fall=(1+sin(-ramp*pi/2))/2; % defines the raised cos fall-time shape
off=zeros(1,900);        % defines the off-time sample points

% concatenate the parts of the amplitude of the pulse into a single array
am = [rise on fall off];

% define an array which contains the the non-linearity of the chirp waveform
% the non-linearity is in the form of one cycle of a sine wave across the
% chirp.
nonlinear=.2*sin((pi)*(-1:2/(length(on)-1):1));
```

Appendix G: Pulse with non-linear fm chirp (continued)

```
% add the nonlinearity to the chirp and concatenate the other sections
fm=(chirp_dev/2)*([-ones(1,n) nonlinear+(-1:2/(length(on)-1):1) ones(1,n)
ones(1,length(off))]);

%  plot the fm-samples and scale the plot
plot(fm);
axis ([0 length(fm) -10e6 10e6]);

% use an integral to translate from fm to pm
pm=(2*pi/sampclk)*cumsum(fm);

% convert am and pm to i and q and scale amplitude
i = am.* cos(pm);
q = am.* sin(pm);

% define a composite iq matrix for download to the PSG using the
% PSG/ESG Download Assistant
IQData = [i + (j * q)];

% define a marker matrix and activate a marker to indicate the beginning of the
waveform
Markers = zeros(2,length(IQData)); %fill Marker array with zero ie. no markers set
Markers(1,1:10) = 1; %set Marker to first ten points of playback

% make a new connection to the PSG over the GPIB interface
io = agt_newconnection('gpib',0,19);

% verify that communication with the PSG has been established
[status, status_description,query_result] = agt_query(io,'*idn?');
if (status < 0) return; end

% preset the instrument
[status, status_description] = agt_sendcommand(io,':STATus:PRESet');

% set carrier frequency and power on the PSG using the PSG Download Assistant
[status, status_description] = agt_sendcommand(io, 'SOURce:FREQuency 1e9');
[status, status_description] = agt_sendcommand(io, 'POWer 0');

% put the ALC into manual control and set the IQ real time scaling
[status, status_description] = agt_sendcommand(io, 'POWer:ALC:STATe OFF');
[status, status_description] = agt_sendcommand(io, 'RADio:ARB:RSCaling 70.7');

% download the iq waveform the PSG baseband generator for playback
[status, status_description] = agt_waveformload(io, IQData, 'nlfm', sampclk,
'play', 'no_normscale', Markers);

% Turn on modulation
[status, status_description ] = agt_sendcommand( io, 'OUTPut:MODulation:STATe ON');

% Turn on RF output power
[status, status_description ] = agt_sendcommand( io, 'OUTPut:STATe ON' );
```

Online

The source code for the examples used in this paper along with Download Assistant for MATLAB 6.5 or later can be downloaded from the Agilent Web site **www.agilent.com/find/psg**.

Agilent Technologies' Test and Measurement Support, Services, and Assistance
Agilent Technologies aims to maximize the value you receive, while minimizing your risk and problems. We strive to ensure that you get the test and measurement capabilities you paid for and obtain the support you need. Our extensive support resources and services can help you choose the right Agilent products for your applications and apply them successfully. Every instrument and system we sell has a global warranty. Support is available for at least five years beyond the production life of the product. Two concepts underlie Agilent's overall support policy: "Our Promise" and "Your Advantage."

Our Promise
Our Promise means your Agilent test and measurement equipment will meet its advertised performance and functionality. When you are choosing new equipment, we will help you with product information, including realistic performance specifications and practical recommendations from experienced test engineers. When you use Agilent equipment, we can verify that it works properly, help with product operation, and provide basic measurement assistance for the use of specified capabilities, at no extra cost upon request. Many self-help tools are available.

Your Advantage
Your Advantage means that Agilent offers a wide range of additional expert test and measurement services, which you can purchase according to your unique technical and business needs. Solve problems efficiently and gain a competitive edge by contracting with us for calibration, extra-cost upgrades, out-of-warranty repairs, and onsite education and training, as well as design, system integration, project management, and other professional engineering services. Experienced Agilent engineers and technicians worldwide can help you maximize your productivity, optimize the return on investment of your Agilent instruments and systems, and obtain dependable measurement accuracy for the life of those products.

Agilent Email Updates

www.agilent.com/find/emailupdates
Get the latest information on the products and applications you select.

Agilent T&M Software and Connectivity
Agilent's Test and Measurement software and connectivity products, solutions and developer network allows you to take time out of connecting your instruments to your computer with tools based on PC standards, so you can focus on your tasks, not on your connections. Visit **www.agilent.com/find/connectivity** for more information.

By internet, phone, or fax, get assistance with all your test & measurement needs

Phone or Fax
United States:
(tel) 800 452 4844
Canada:
(tel) 877 894 4414
(fax) 905 282 6495
China:
(tel) 800 810 0189
(fax) 800 820 2816

Europe:
(tel) (31 20) 547 2323
(fax) (31 20) 547 2390
Japan:
(tel) (81) 426 56 7832
(fax) (81) 426 56 7840
Korea:
(tel) (82 2) 2004 5004
(fax) (82 2) 2004 5115

Latin America:
(tel) (305) 269 7500
(fax) (305) 269 7599
Taiwan:
(tel) 0800 047 866
(fax) 0800 286 331
Other Asia Pacific Countries:
(tel) (65) 6375 8100
(fax) (65) 6836 0252
Email: tm_asia@agilent.com

Online Assistance:
www.agilent.com/find/assist

Product specifications and descriptions in this document subject to change without notice.

Printed in USA, December 4, 2003
5988-9212EN

Agilent Technologies