Radar Emitter Simulation Using Vector Signal Generators

Randal Burnette

Synergent Technologies, 5301 Buckeystown Pike, Suite #306, Frederick, Maryland, 21704 USA
Main Phone: (240) 215-0040

Abstract -This session is intended for engineers, engineering managers and technicians who are responsible for the design, manufacture and support of airborne or ground based radar systems, subsystems and components. Simulating radar emitters has been proven to be a process that is either expensive or time consuming or both. Unique signals must be generated to exercise the system under a multitude of real world conditions. Using off-the-shelf vector signal generators, it is now possible to create the complex waveforms needed to successfully complete this important element of radar system design verification. This session will illustrate how this can be accomplished.

I. Pulse Test Pattern Generation

With the advent of microwave signal generators and spectrum analyzers with vector capability, the engineer can now 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 paper is to help the design engineer generate and evaluate complex radar signals using standard microwave test equipment.

The approach taken is to show how to implement a pulse test pattern generator. The pulse test pattern generator produces a series of pulses that will stress one specific parameter in the receiver, such as the ability to correctly identify two closely spaced pulses. The pulse test pattern generator has the flexibility to simulate a static emitter, but lacks the memory depth or real time processing to simulate movement of the emitter over time. Experience has shown that a series of complex pulse patterns will enable the user to perform roughly 80% of the tests necessary to evaluate the performance of a system. The final 20% of the receiver testing is typically done on a test range using real emitters.

II. Agilent's E8267C PSG Vector Signal Generator

The general-purpose test equipment needed to evaluate a receiver is 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 0.5-18GHz. If the receiving system must process phase or frequency coded pulses, then the generator must be able to produce these signals.

The Agilent E8267C Vector Performance Signal Generator (VPSG) that covers 250KHz to 20GHz 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 new wide-bandwidth I/Q modulation with an internal arbitrary waveform generator providing 80MHz of modulation bandwidth. The generator offers an optional 1GHz analog IQ modulator with external inputs. When used with an external baseband source, the generator could produce the signal typical of synthetic aperture radars or imaging radars. This paper will focus on the use of the internal arbitrary waveform generator to produce complex radar signals.

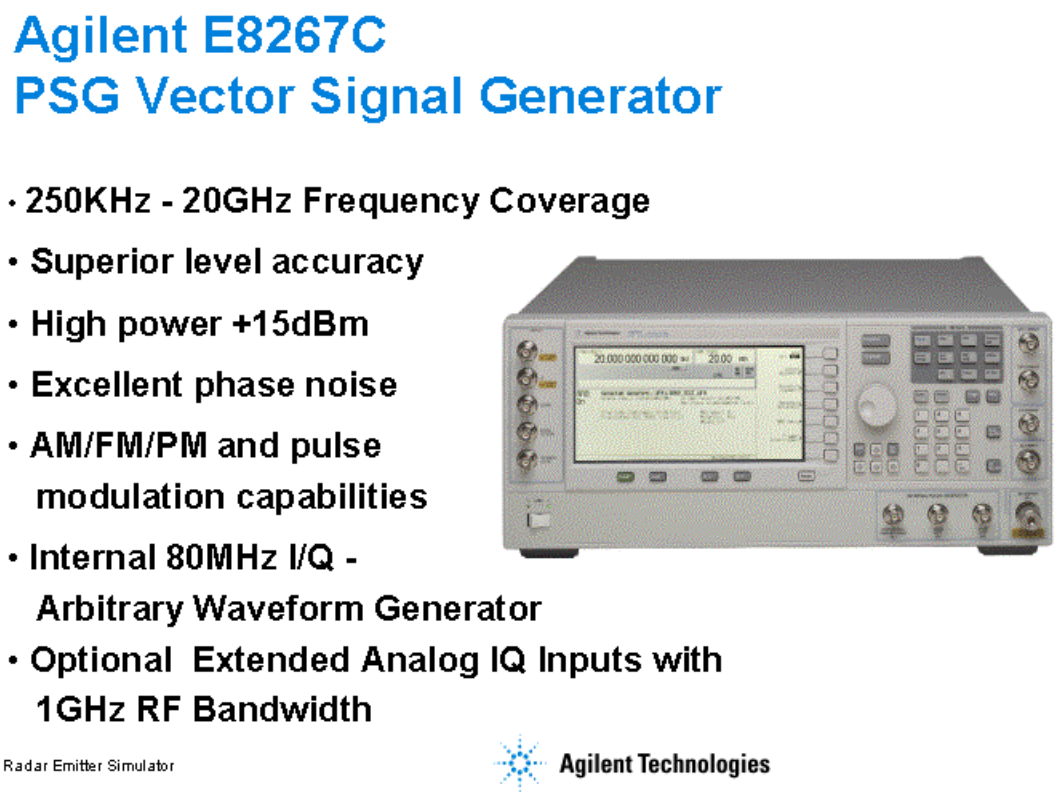

Figure #1

III. Agilent's 'Download Assistant' and MatLab® 6.5

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 VPSG. The problem to be dealt with is how to move the digital waveforms from MatLab into the arbitrary waveform generator inside the VPSG.

Agilent has developed 'Download Assistant' for MatLab® to enable users to easily download their IQ waveforms into the arbitrary waveform generator's memory. In addition to downloading waveforms, it allows the user to send any 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 the use of 'Download Assistant' with MatLab 6.5 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: www.agilent.com/find/psg.

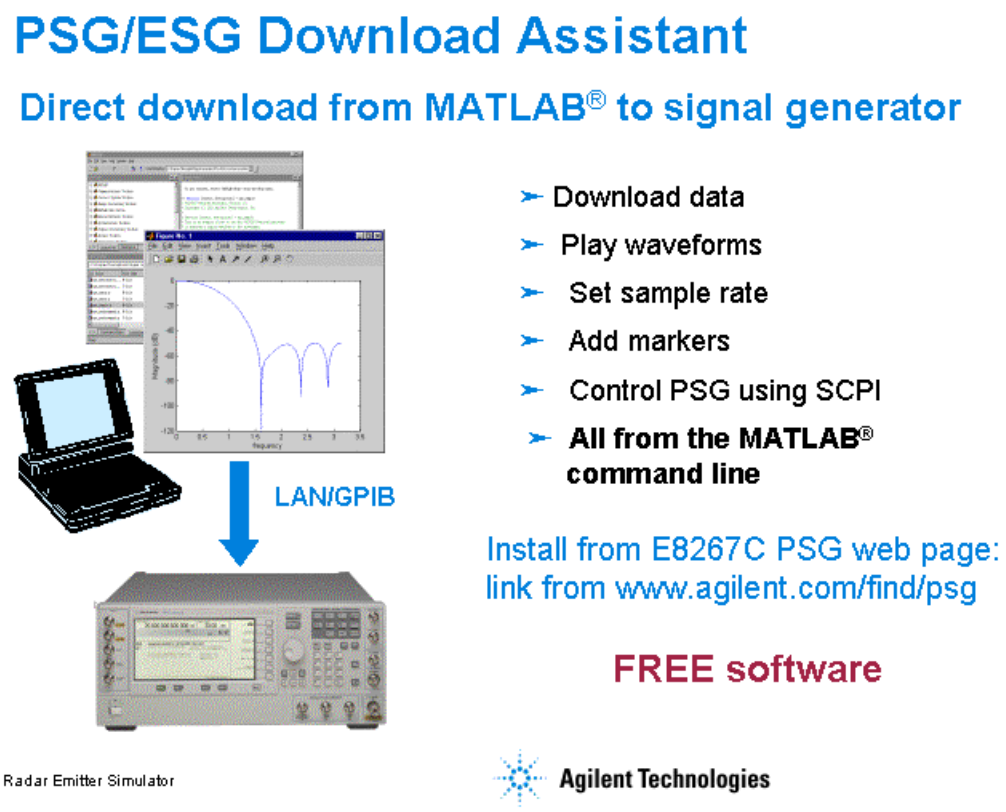

Figure #2

IV. Agilent's E4440A Performance Spectrum Analyzer

To capture and demodulate signals the E4440A Performance Spectrum Analyzer was used with the 89601 Vector Analysis Software. This configuration provided frequency coverage of 26.5GHz with up to 36MHz of analysis bandwidth. Using the internal digitizer provides 8MHz of analysis bandwidth and using an external 89611AVXI digitizer provides 36MHz of analysis bandwidth.

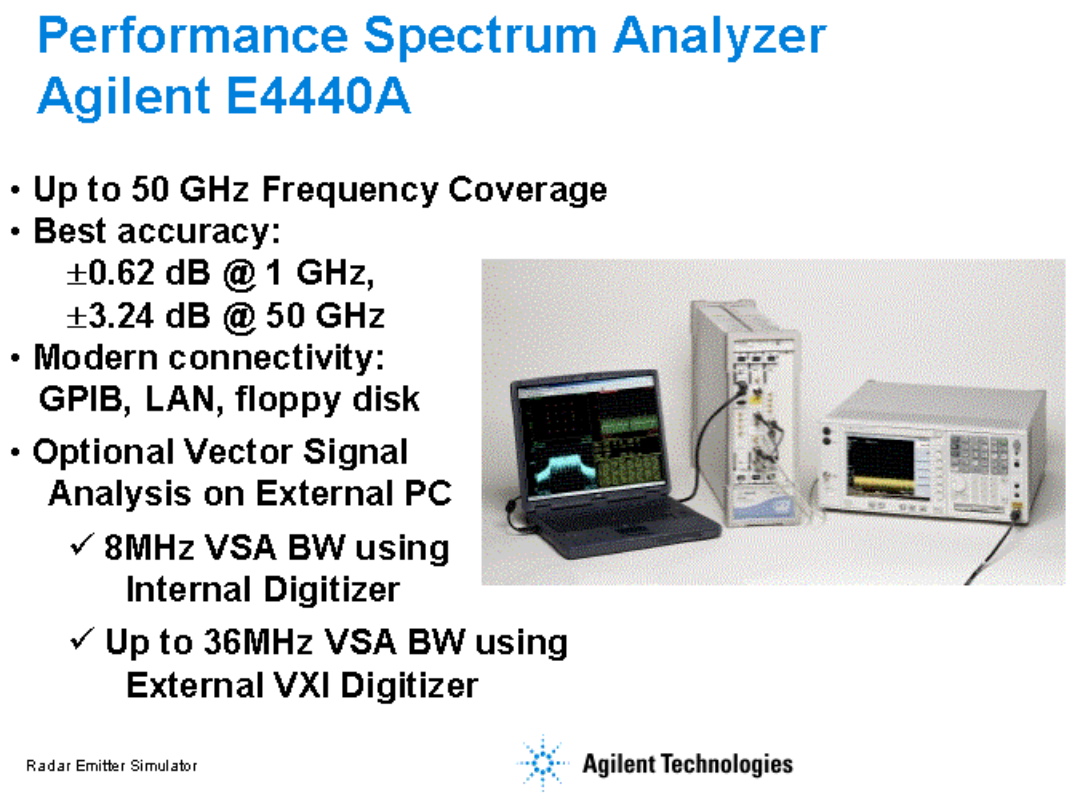

Figure #3

Generate a Simple Pulse

To adequately test elint receiver performance, a wide variety of test signals are needed. The user may wish 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.

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 #4 shows a plot of a typical pulsed waveform and Figure #5 reveals the code used to generate the waveform.

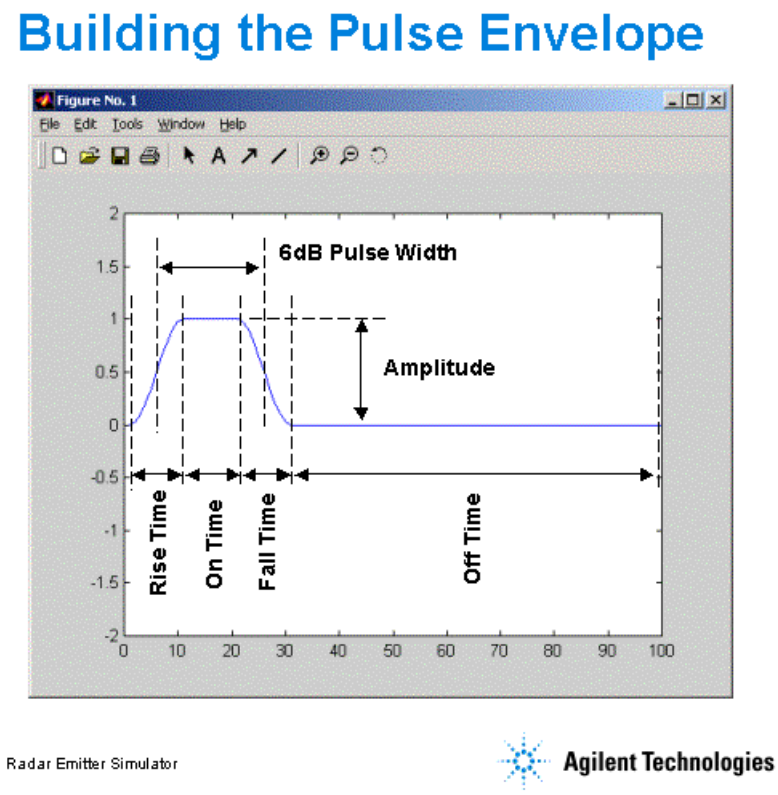

Figure #4

```
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=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)];
```

Figure #5

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 is built using the ***ones(1,10)*** and ***zeros(1,70)*** functions. In this case the ***ones*** command creates a 1X 10 array and fills it with 1's. Likewise, the ***zeros*** command creates a 1X70 element array and fills it with 0's. 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 sin 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 1 to the result and dividing by 2 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 #1 shows a plot of the final pulse.

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.

The variable ***smplclk=100e6*** is used to set the clock frequency for the arbitrary waveform generator to 100MHz. This allows time to be associated with each point in the waveform. Each point within the waveform will occupy 1/smplclk or 10nsec 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 10nsec*n or 100nsec where n describes the number of points in the arrays ***rise*** or ***fall***. While the 0-100% 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 10-90% rise-time is equal to .59 times the 0-100% rise-time. In this case, the 10-90% rise-time would equal 59nsec. 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 it's maximum value of 100MHz use four points or more in the rise-time waveform.

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 6dB 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 6dB 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 VPSG 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 1 and in the imaginary array to 0, if we set the output power of the signal generator to 0dBm, then the peak power of the pulse will equal 0dBm. 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 1 will reduce the output power. However there are several scaling factors that must be addressed.

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 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 good practice to turn off the IQ modulation and press the manual calibration softkey under the Power hardkey to update the calibration.

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. The purpose is to easily enable to user 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% may reduce the non-linear distortion produced by the IQ modulator. A value of 70% will reduce the output power by 3dB. The command in MatLab (which uses 'Download Assistant') to set the real time scaling value is:

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

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. The automatic calibration process within the signal generator uses a GPIB power meter to measure the power at the device under test input to generate the User Calibration array. When the signal generator is set to a new frequency, the processor within the VPSG will correct for the losses and provide the displayed power at the device under test input. For detailed information about using this feature, refer to the VPSG's User Manual.

Running Pulse.m

The file Pulse.m is a complete MatLab program to generate and download a simple pulse into the VPSG. A printout of the program can be found in Appendix A of this paper. The source code for all of the programming examples used in this paper can be downloaded for free at the Agilent web site: www.agilent.com/find/psg. Note that the pulse parameters were modified from the above example to produce a more realistic signal. The pulse is 1µsec wide with a pulse repetition interval of 10µsec. The reference level for the signal generator is set to 0dBm, but the peak pulse amplitude of the waveform is set to 0.707 producing an output power of –3dBm. Figure #6 provides a screen capture from the Vector Signal Analyzer to analyze the 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 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 Vector PSG is coherent in frequency and phase with the Vector Signal Analyzer.

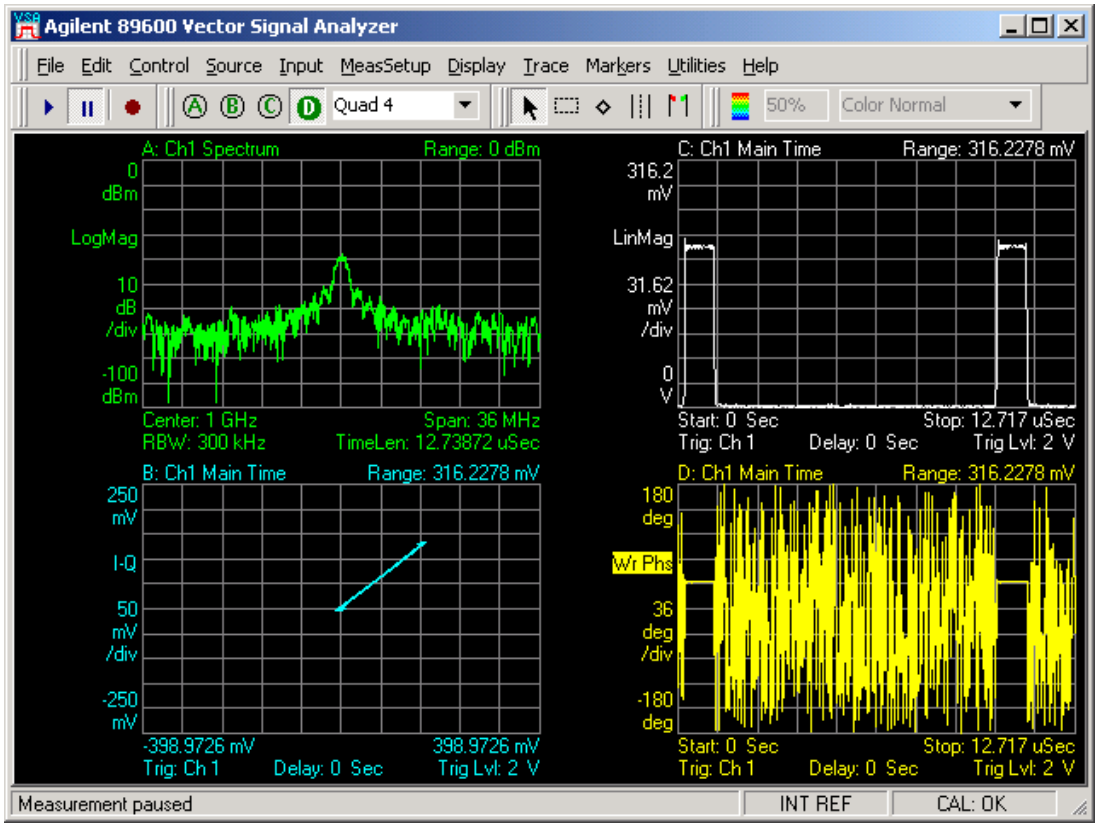

Figure #6

Generate 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. Refer to Figure #8 for plots of a pulse doublet.

Generating a pulse doublet using Pulse.m as a starting point is fairly straightforward. Refer to Figure #7. 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 we concatenate the new pulses with separation between them.

```
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 #7

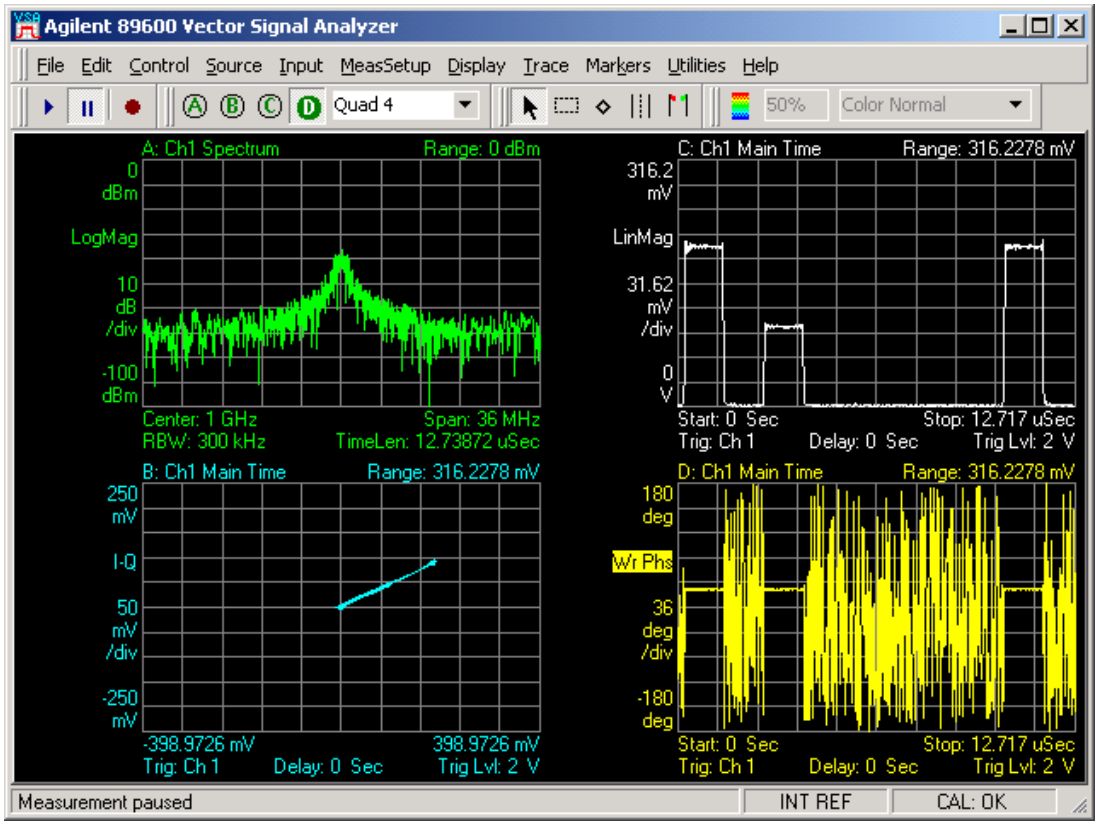

Figure #8

Generate 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 was 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 in Figure #10 the π/2 phase shift between pulse1 and pulse2 shown in both the IQ plot and the phase versus time plot.

```
% 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);
```

Figure #9

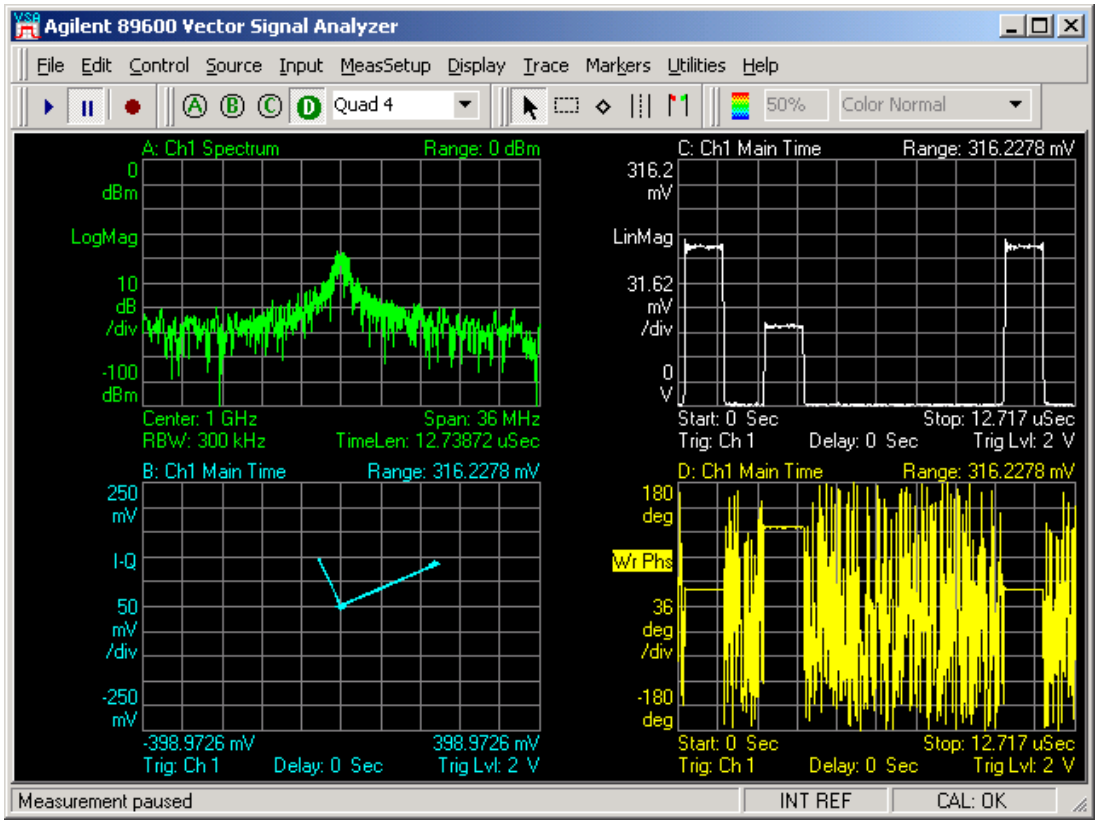

Figure #10

Create 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 #11 an array `fm` is produced that contains a constant offset frequency in Hz. 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*pi/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 math shown in Figure #11 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 32Mbytes of memory within the signal generator to play the waveforms. With 100MHz sample clock, the 32Mbytes of memory will allow the production of 32msec of unique signal.

```
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);
```

Figure #11

Now to discuss the two vector displays of the doppler signal. Figure #12 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 MatLab program doppler.m, the pulse width can be calculated to be 1μsec and the doppler shift set to 100KHz. The phase shift generated by a 100KHz doppler shift over 1μsec should be:

$$(100\text{KHz})*(360^{\circ}/\text{cycle})*(1\text{e-}6\text{sec}) = 36^{\circ}$$

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

In Figure #13 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 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.

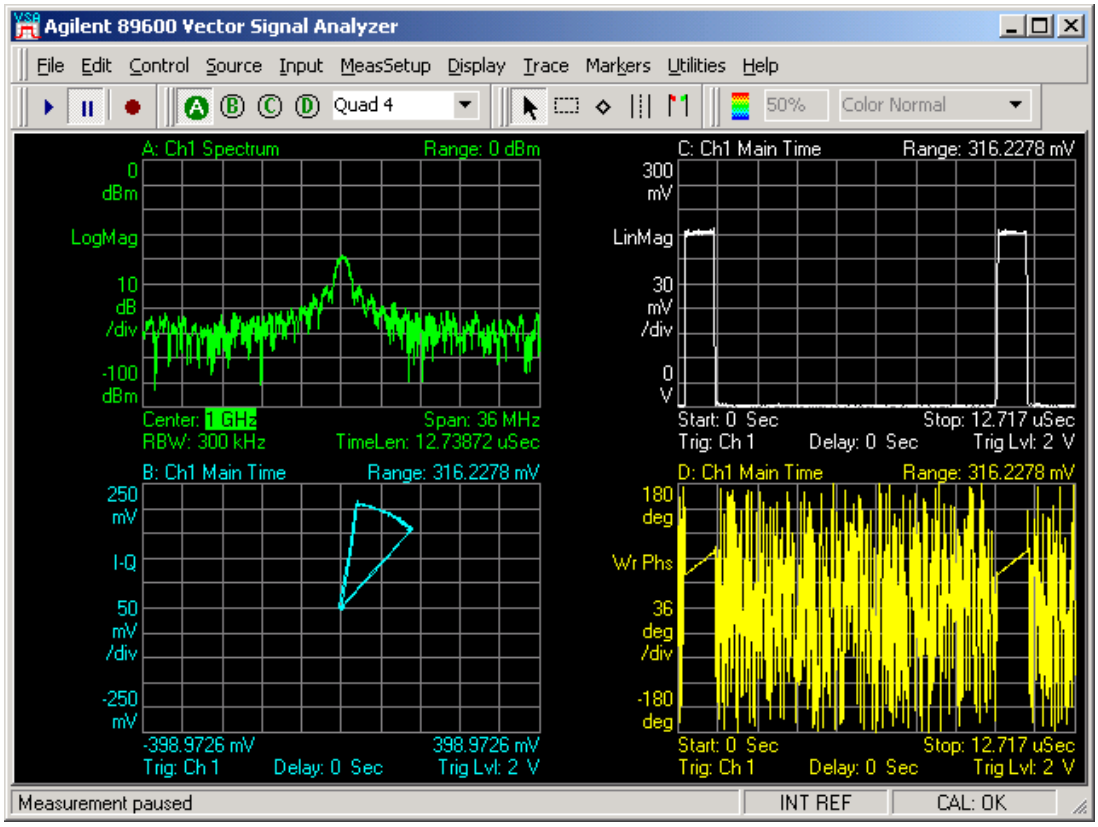

Figure #12

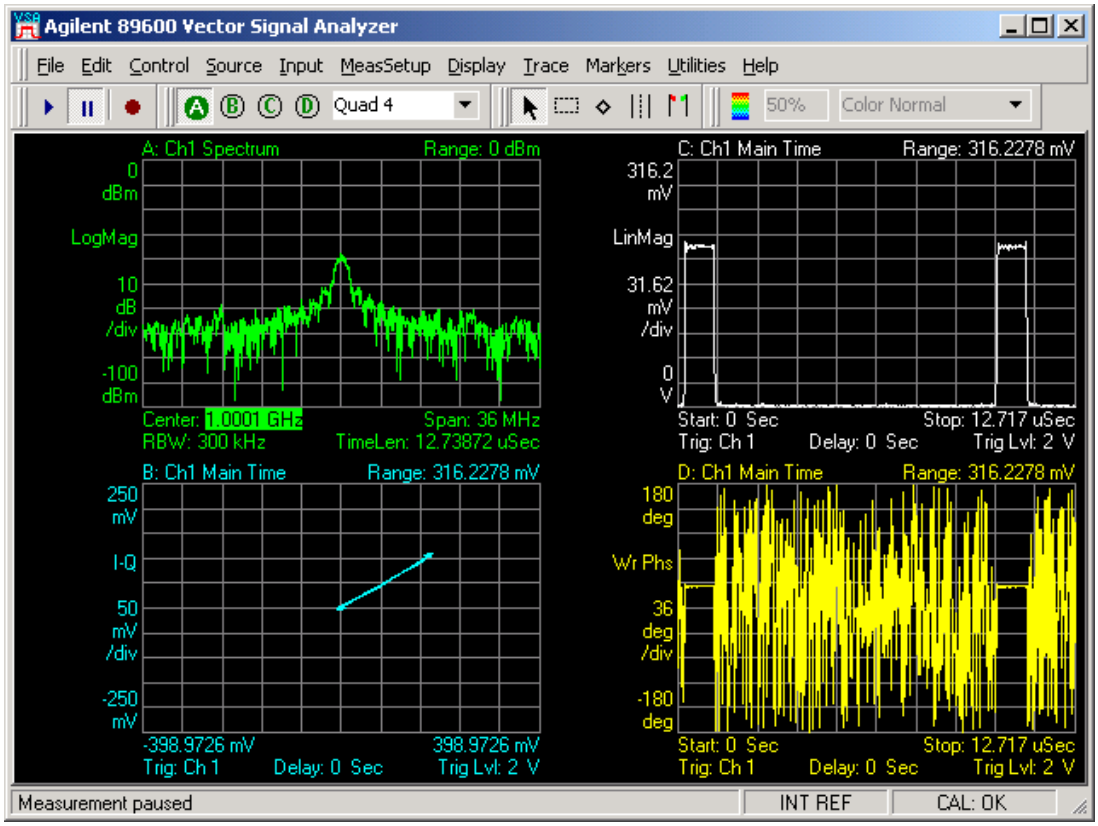

Figure #13

IV. 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.

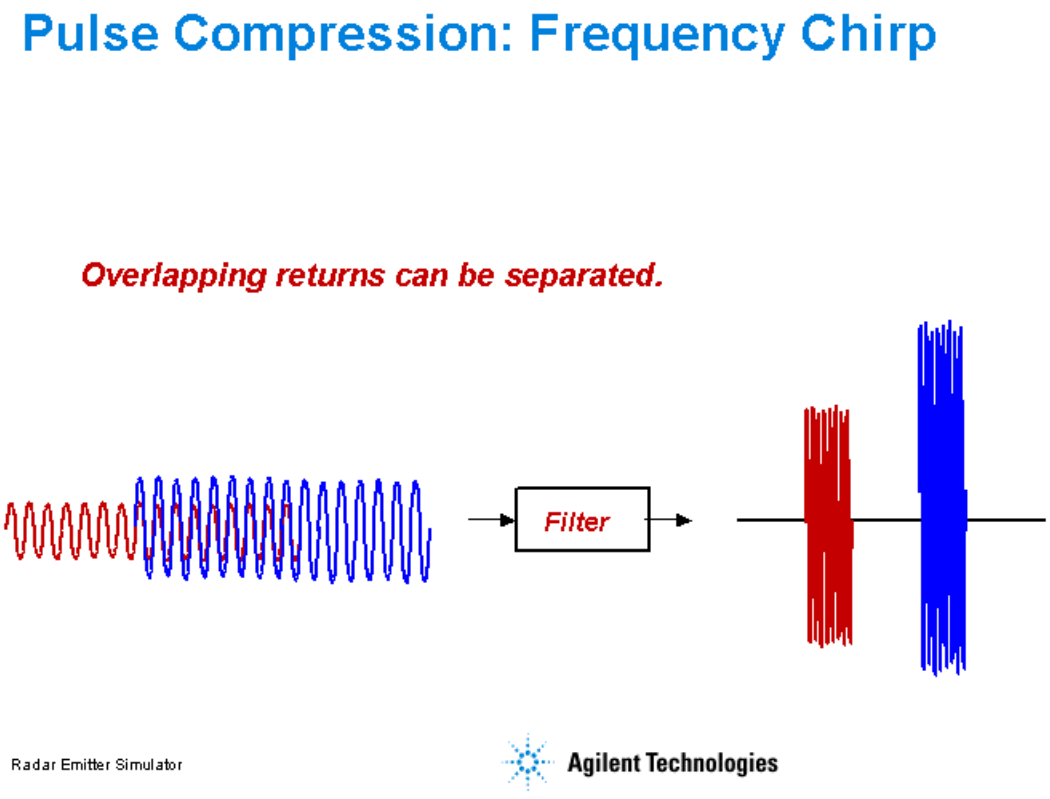

Figure #14

Generate 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 a 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.

```
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 #15

Figure #16 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.

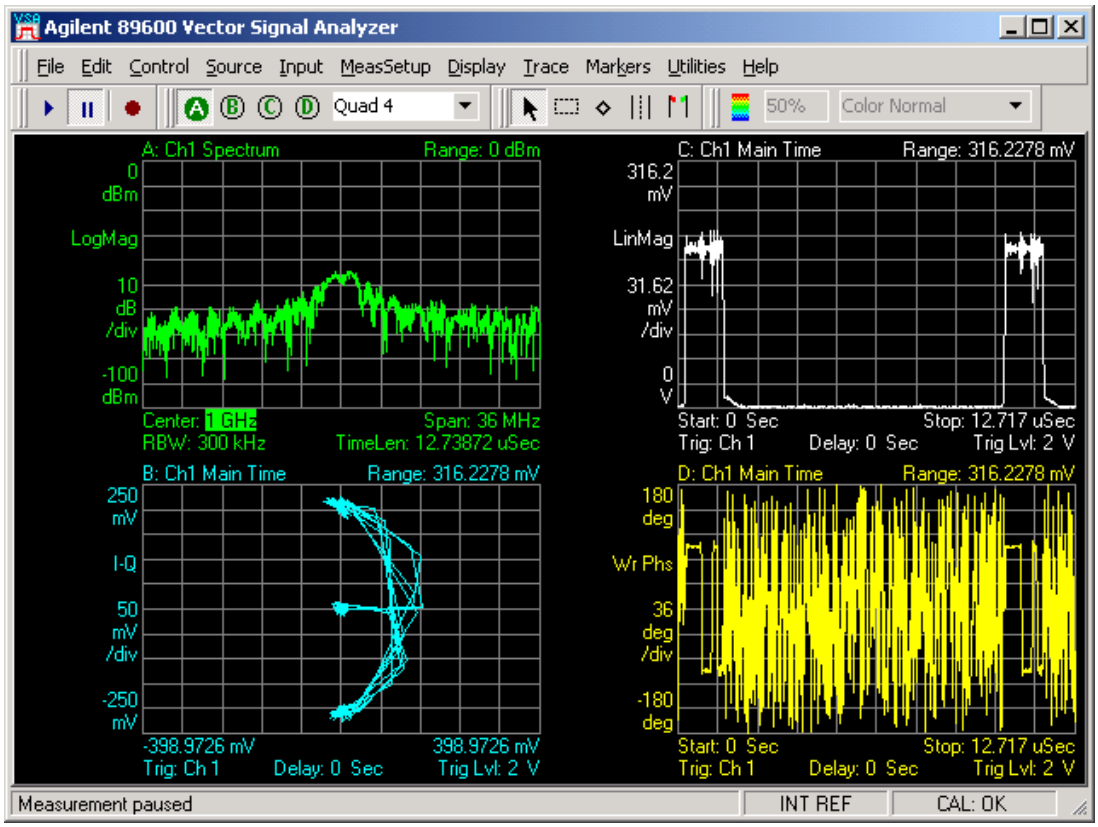

Figure #16

Generate a Linear FM Chirp (LFM_Chirp.m)

The final type of signal to demonstrate is a linear FM chirp. 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 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.

In Figure #17 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 2. 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 80MHz 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 #18 note the demodulated fm waveform in the lower right hand plot. The chirp is linear and the deviation is equal to the 10MHz defined in the program (+/- 5MHz). 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 10MHz 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.

```
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);
```

Figure #17

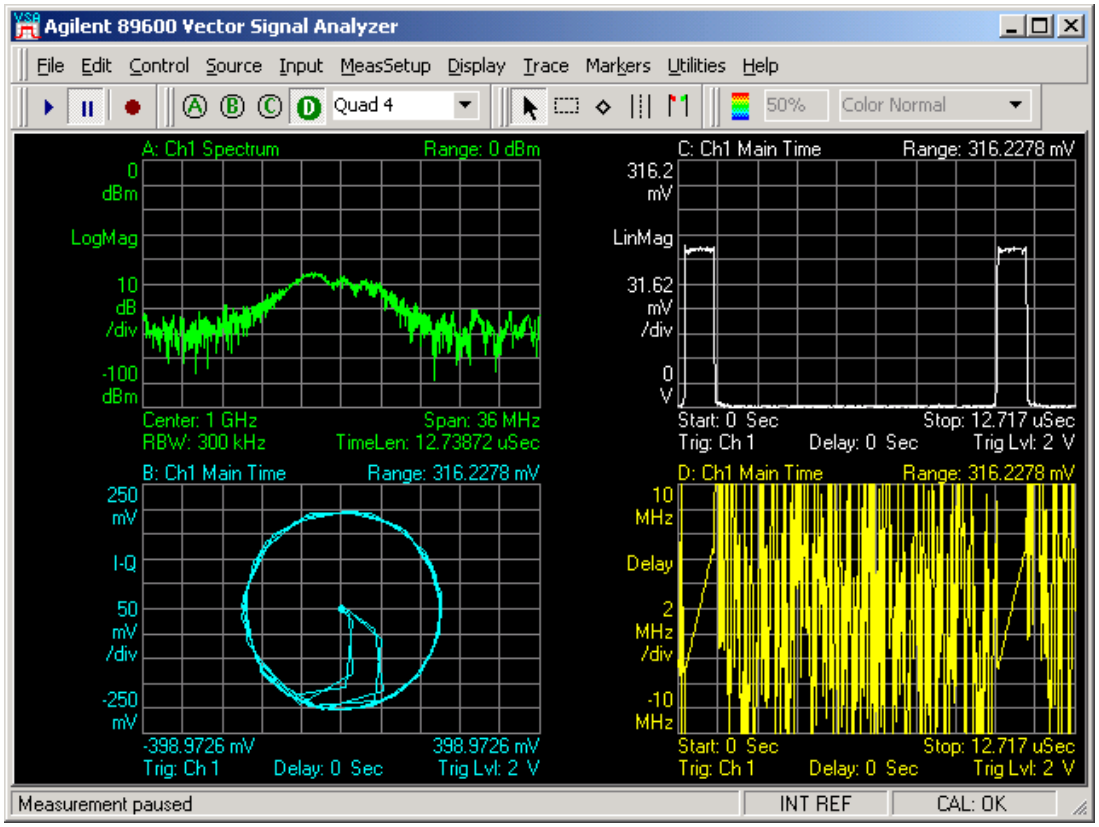

Figure #18

Generate 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.

```
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);
```

Figure #19

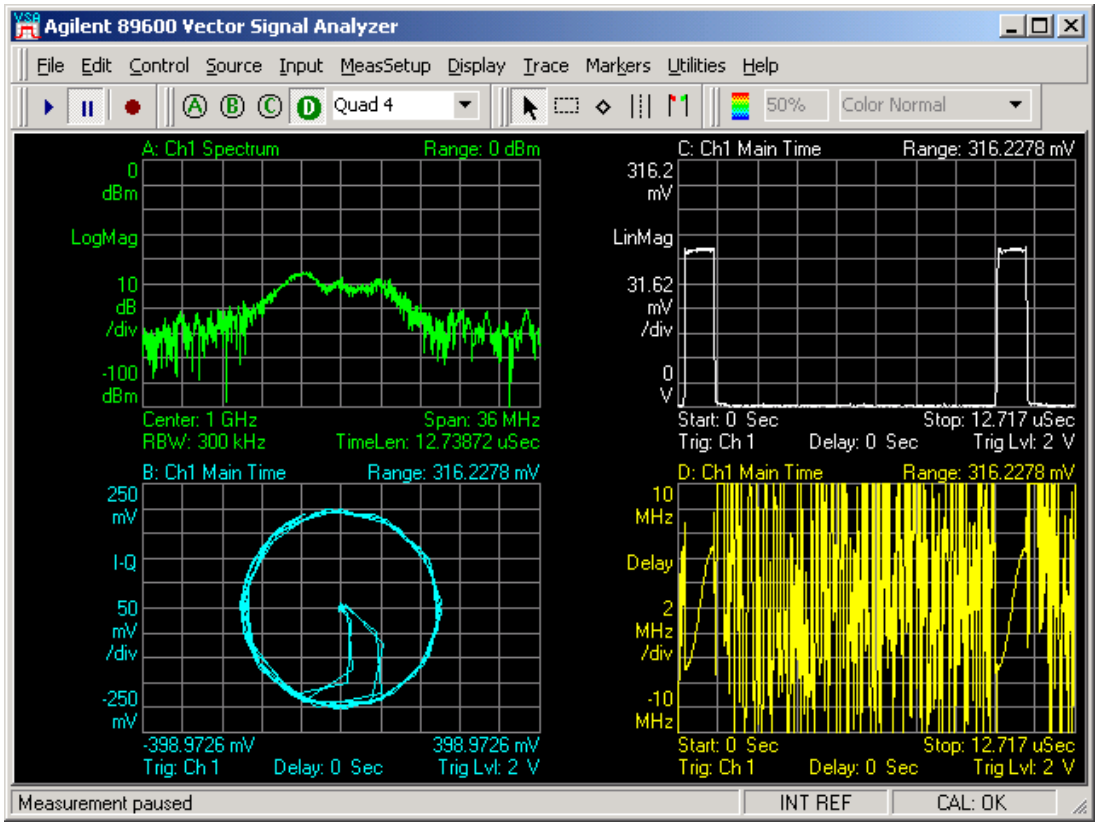

Figure #20

Agilent 'Pulse Builder' Signal Studio Software

For many common applications Agilent's 'Pulse Builder' Signal Studio Software eliminates the need to write custom software programs. Pulse Builder provides a Graphical User Interface that makes it easy to create complex pulsed signals without detailed knowledge of the math involved in the creation of the signals. Pulse Builder provides simple CW pulses with shaping and complex phase and frequency-encoded signals. The user can easily build pulses and pulse patterns to simulate complex emitters. Once created, the signals can be saved on the signal generator's internal hard disk drive and recalled later without an external pc or software.

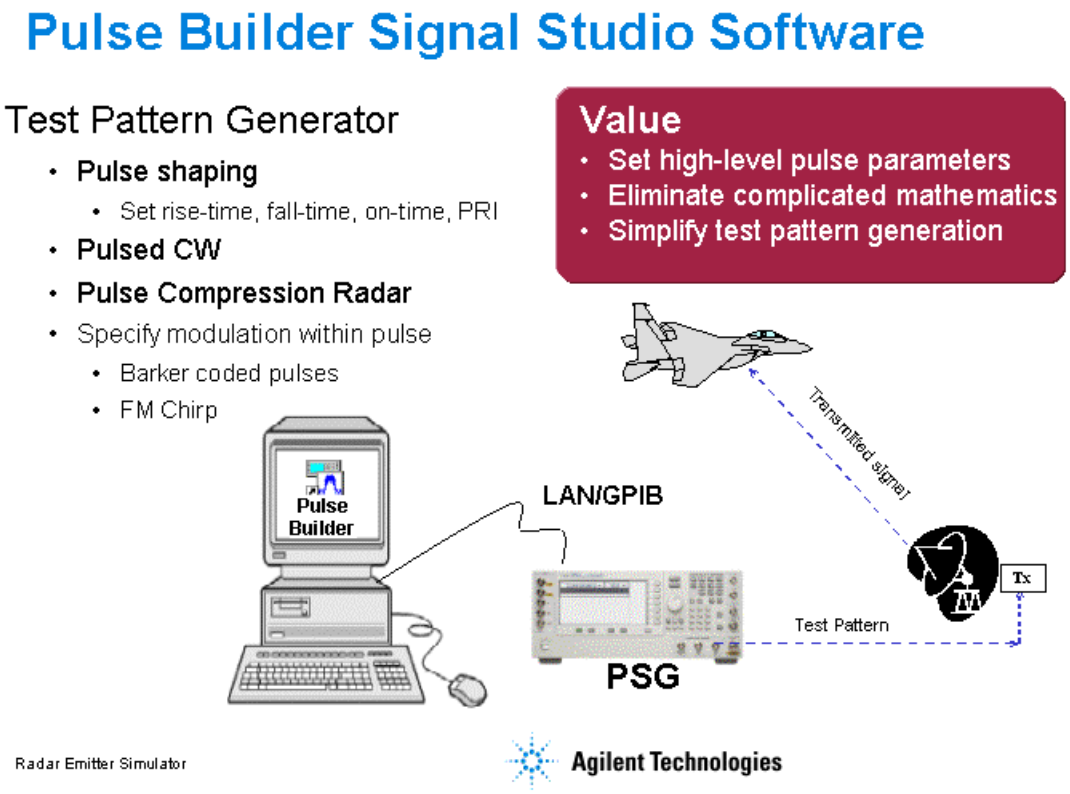

Figure #21

Multiple Emitter Simulation Solutions

There are many applications where several emitters are required to test the receiver and signal processing algorithms. Elint systems must be able to process multiple emitters simultaneously and provide correct responses. Typical tests include sensitivity tests and dynamic range tests. Sensitivity tests are performed by very accurately being able to control the low level power of an emitter. Dynamic range tests where a large amplitude emitter is close in frequency to a low level emitter. Many of the systems are required to be able to process overlapping pulses where the emitters are separated over a wide bandwidth. The emitter definition must be flexible and may change over time period of the simulation. This is often a very difficult test to instrument and perform.

Synergent Technologies ™ is working closely with Agilent to develop a multi-emitter pulse pattern generator based on the Vector PSG. The simulator will provide frequency coverage from 0.5 to 18GHz with 2-4-6-8 simultaneous emitters. The emitters will be completely independent and will produce complex pulse patterns containing wide bandwidth phase coded signals. The entire simulation process can be controlled from a flexible timeline editor. The timeline editor will enable the user to very accurately

control when an emitter becomes active or inactive and control the timing between multiple emitters. The characteristics of the system can be tailored to meet the needs of individual customers.

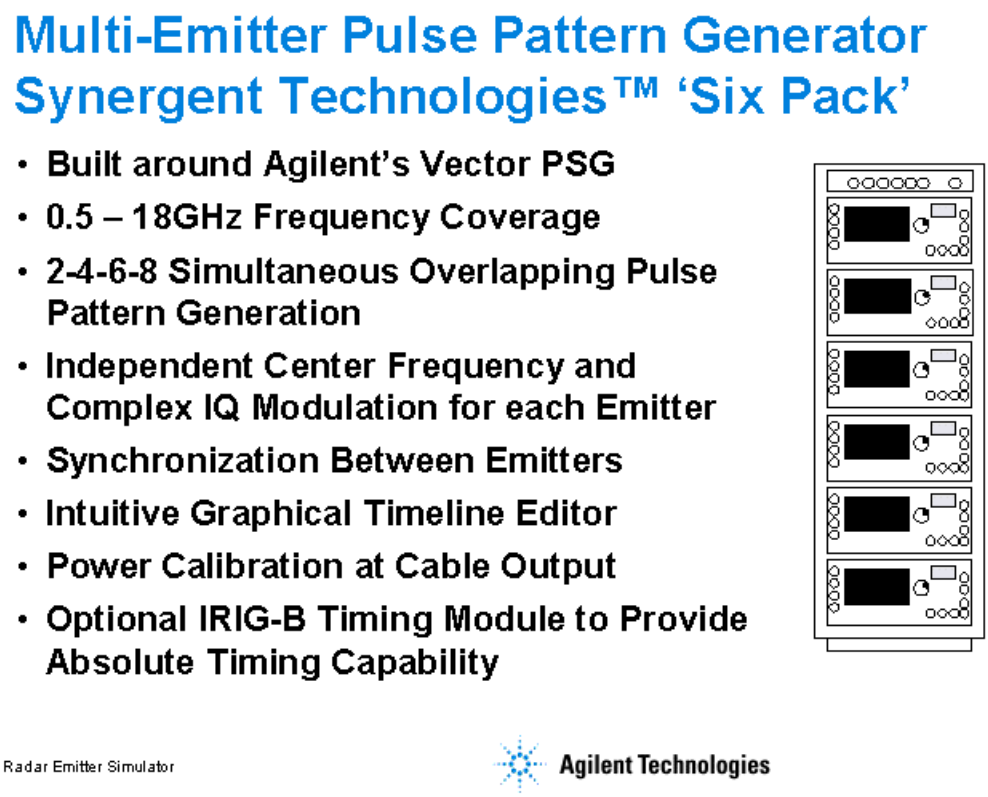

Figure #22

Summary

Printed copies of the programs used in this paper are provided in the appendices. The source code for the examples used in this paper along with 'Download Assistant' for MatLab 6.5 can be downloaded from the Agilent web site: www.agilent.com/find/psg.

APPENDEX 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
%
%   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
%   sampclk         -- clock freq for the D/A converters in the IQ modulator


sampclk = 100e6;                % ARB Sample Clock for playback

n=4;                            % number of pts 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 and scale it 3.01dB below the output amplitude
i = .707*[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));

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

% define a matrix and activate a marker for the beginning of the waveform
```

```
Markers = zeros(2,length(IQData));  %fill Marker array with zeros
Markers(1,1) = 1;                   %set Marker to first point 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,query_result] = agt_query(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 100');

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

APPENDEX 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
%
%   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

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
i = .707*[pulse1 separation 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));

% 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) = 1;                   %set Marker to first point of play back

% 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,query_result] = agt_query(io,':STATus:PRESet');

% set carrier frequency and power on the PSG using the PSG Downlaod Assistant
[status, status_description] = agt_sendcommand(io, 'SOURce:FREQuency
1000000000');
[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 100');

% 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, 'pulse', sampclk,
'play', 'no_normscale', Markers);

% 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
%
%   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];

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

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

% 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) = 1;                   %set Marker to first point of play back

% 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,query_result] = agt_query(io,':STATus:PRESet');

% set carrier frequency and power on the PSG using the PSG Download Assistant
[status, status_description] = agt_sendcommand(io, 'SOURce:FREQuency
1000000000');
[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 100');

% 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 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
%
%   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(i) -2 2])

% 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 and scale amplitude
i=.707*am.* cos(pm);
q=.707*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 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) = 1;                   %set Marker to first point of play back

% 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,query_result] = agt_query(io,':STATus:PRESet');

% set carrier frequency and power on the PSG using the PSG Downlaod Assistant
[status, status_description] = agt_sendcommand(io, 'SOURce:FREQuency
1000000000');
[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 100');

% 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 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
%
%
%    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);

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);

% 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) = 1;                   %set Marker to first point of play back

% 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,query_result] = agt_query(io,':STATus:PRESet');

% set carrier frequency and power on the PSG using the PSG Download Assistant
[status, status_description] = agt_sendcommand(io, 'SOURce:FREQuency
1000000000');
[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 100');

% 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 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, IQ Scaling
%                               and corrected fm to pm integration calc
%
%   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))]);

%  plot the fm-samples and scale the plot
```

```
%plot(fm);
%axis ([0 length(am) -2 2]);

% 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);

% 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) = 1;                   %set Marker to first point of play back

% 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,query_result] = agt_query(io,':STATus:PRESet');

% set carrier frequency and power on the PSG using the PSG Download Assistant
[status, status_description] = agt_sendcommand(io, 'SOURce:FREQuency
1000000000');
[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 100');

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

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

APPENDIX G: Pulse with Non-Linear FM Chirp

```
% Script file:  LFM_Chirp.m
%
% Purpose:
%   To calculate and download an arbitrary waveform file that simulates a
%   simple pulse 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  Added comments and Download Assistant
%   9/4/2002    Randal Burnette Added Preset, turned ALC off, and IQ Scaling
%
%   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));

% add the non-linearity 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 am-samples and scale the plot
%plot(fm);
%axis ([0 length(am) -2 2]);

% 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);

% 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) = 1;                   %set Marker to first point of play back

% 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,query_result] = agt_query(io,':STATus:PRESet');

% set carrier frequency and power on the PSG using the PSG Download Assistant
[status, status_description] = agt_sendcommand(io, 'SOURce:FREQuency
1000000000');
[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 100');

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

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