Purpose

The main purpose of this document is to guide the reader through the necessary steps to successfully connect and retrieve data from certain laboratory equipment using MATLAB Instrument Control Toolbox (ICT). Secondary, a discussion of each function is presented to give the reader insight into the interfacing functions. Consequently, the document is subdivided into two main parts, function description and examples.

Conventions

For clarity, the following conventions are used in this document.

Bold – denotes a button (or soft-button) name
Italics – denotes the title of a program or device
`Text` – denotes a command, parameter, argument, or output

Host PC Communication with Instrument

Generally speaking, the instruments communicate with the PC using standard IEEE 488.2, also known as General Purpose Interface Bus (GPIB). GPIB was first developed by Hewlett Packard (under the name HP-IB) in the 1960s. While other technologies exist to interface with modern measurement equipment, GPIB offers a mature interface platform. Moreover, the standard is extendable up to 15 devices, each device with a unique address identifier.

GPIB Direct Connection

To communicate via GPIB, the laboratory computers use an *Agilent 82357A USB-to-GPIB Interface*. Since the computers do not use a GPIB controller, an I/O software package, *Agilent IO Control*, must also be installed to allow communication via the *82357A* adapter. The following figure shows the *USB-to-GPIB Interface*.

Figure 1: Agilent 82357A USB-to-GPIB Interface

Physically connecting the *82357A* is straightforward. The parallel end connects to the GPIB port on the back of the test equipment. The USB Type-A connects to an available USB port on the host PC. After powering on the respective laboratory equipment, the *READY* light will illuminate green indicating the adapter is correctly installed.

USB Direct Connection

Note, it is only possible to directly connect the *Agilent MSO6032A* oscilloscope via a Type-A to Type-B USB cord. Again, *Agilent IO Control*, is used to interface with measurement device. The following figure shows the USB Type-A to Type-B cord. The Type-B connector attaches to the rear of the oscilloscope and the Type-A connector attaches to the PC.

Figure 2: Type-A to Type-B USB Cord

Device Address Specification

As previously eluded to, each connected device will have a unique address identifier. As will be shown later, the MATLAB ICT interface software is dependent on the device address for proper communication. The *Agilent IO Control* software is used as a "wrapper" for communication with the host PC and the end device test equipment. With that said, the software has an extension program, *Agilent Connection Expert*, that provides pertinent addressing information to the end user. The user can access the program by double clicking the system tray icon highlighted in Figure 3.

Figure 3: Agilent IO Control System Tray Icon

The following discussion details the process for determining the device's GPIB address.

After opening *Agilent Connection Expert*, the user can easily obtain the GPIB address associated with the test equipment of interest. Referencing the figure below, the red dotted box outline shows a typical instrument report for the various instrument I/O ports on the host PC. Note, other I/O ports are also listed, such as LAN and COM ports. The I/O ports of interest will always be listed in the form $USB/GPIB\ (GPIBX)$. Where X denotes the N^{th} *82357A* adapter attached to the host PC. Note: The program retains previous connected instruments, thus, this represents the N^{th} unique USB/GPIB setup.

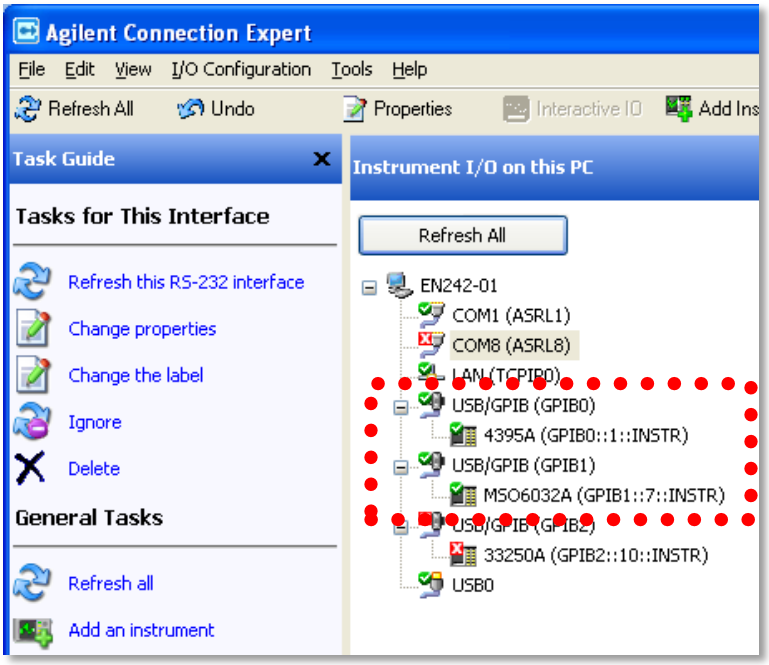

Figure 4: Agilent Connection Expert Screenshot

Using Figure 4 as an example, the *Agilent 4395A Network/Spectrum* analyzer is connected to $GPIB0::1::INSTR$. Whereas, the *Agilent MSO6032A* is connected to $GPIB1::7::INSTR$. It is important to note that the address can be changed by the end test equipment. Therefore, it is pertinent to obtain the GPIB address of the test equipment of interest using the above as reference. It is recommended to record the respective GPIB address for reference when using MATLAB ICT.

The following discussion details the process for determining the device's USB address when connecting the oscilloscope via direct USB connection. The procedure for capturing the device address is naturally similar to the above procedure. The following figure highlights the address specification for the oscilloscope.

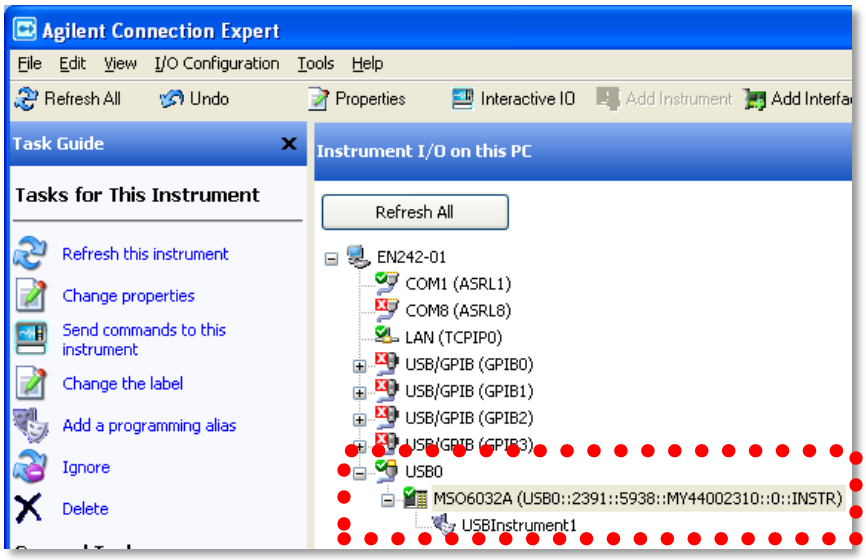

Figure 5: Agilent Connection Expert Screenshot

Using Figure 4 as an example, the *Agilent MSO6032A Mixed Signal Oscilloscope* is connected to $USB0 :: 2391 :: 5938 :: MY44002310 :: 0 :: INSTR$.

Note, in the event the device is not registered in *Agilent Connection Expert*, ensure the device is set to GPIB and/or USB control. Note, the Agilent 4395A does not support USB, thus, one must check the GPIB adapter and connection to the PC. On the MSO6032A, the user can select I/O control settings by pressing the **Utility** button, then select the **I/O** soft-button, followed by **Control**.

MATLAB ICT Communication with Instrument

The MATLAB ICT offers multiple ways to communicate with end devices using the host PC. One method is to use the Test & Measurement Tool (T&M Tool), part of the Instrument Control Toolbox. One can access the T&M Tool by typing `tmtool` in the command window. One particular disadvantage to using the T&M Tool is that one must explicitly link IVI-COM drivers (used to facilitate communication) to the respective laboratory equipment. Thus, a custom `.m` file was written to aid in opening communication with the *Agilent 4395A* or *MSO6032A*.

Opening/Closing a Communication Port using `portOpen.m`

Before attempting to control and obtain data from the test equipment, a communication channel must first be created. Once the device communication channel is no longer needed, it should be properly cleared. With that said, the file `portOpen.m` is used to create and open a communication channel between MATLAB and the test equipment. The file `portClose.m` is used to close the channel.

`portOpen.m`

The following gives a description of `portOpen.m`. As with most MATLAB functions, the help description can be accessed by typing `help portOpen` in the command window.

```
io = portOpen(GPIB_addr, inst_type)
Description: Creates a VISA port connection to the instrument (inst_type)
             connected to the respective GPIB address (GPIB_addr).
             Reference example below .
Input:
    GPIB_addr -> String indicating GPIB address of equipment
                 (retrieved from Agilent IO)
    inst_type -> String indicating instrument type as defined below
        1) 'analyzer' connects to Agilent 4395A Network/Spectrum/Impedance
Analyzer
        2) 'scope' connects to Agilent MSO6032A oscilloscope
        3) 'func_gen' connects to Agilent 33120A Function Generator
Return: io -> Interface object
Example:
    scope = portOpen('GPIB1::7::INSTR', 'scope')
    analyzer = portOpen('GPIB0::1::INSTR','analyzer')
```

Referencing the above example and Figure 4, one can open a communication port with the *Agilent 4395A* by typing the following in the MATLAB command window:

```
>> analyzer = portOpen('GPIB0::1::INSTR','analyzer')
```

If a semicolon is not used (output not suppressed), a successful connection will result in the following output. However, if a semicolon is used, one can verify the status by typing `analyzer` (or the given name during the call to `portOpen`) in the command window.

```
VISA-GPIB Object Using AGILENT Adaptor: VISA-GPIB0-1

Communication Address
```

```
   BoardIndex:         0
   PrimaryAddress:     1
   SecondaryAddress:   0

Communication State
   Status:             open
   RecordStatus:       off

Read/Write State
   TransferStatus:     idle
   BytesAvailable:     0
   ValuesReceived:     0
   ValuesSent:         0
```

The above output provides useful information when troubleshooting communication problems. First, the communication state can easily be verified as `'open'` or `'closed'` under the Communication State heading. Of other interest is the information provided under the Communication Address heading. In particular, the BoardIndex and PrimaryAddress values. Assuming a GPIB address as shown below,

$$GPIBX :: Y :: INSTR \tag{1}$$

The `BoardIndex` value should read X, and the `PrimaryAddress` value should read Y. If the values differ from expected values, it is recommended to verify the address using *Agilent Connection Expert*.

When the device port is successfully created and opened, the analyzer is available as an object in the MATLAB workspace. The following figure shows the resultant object created after calling `portOpen`.

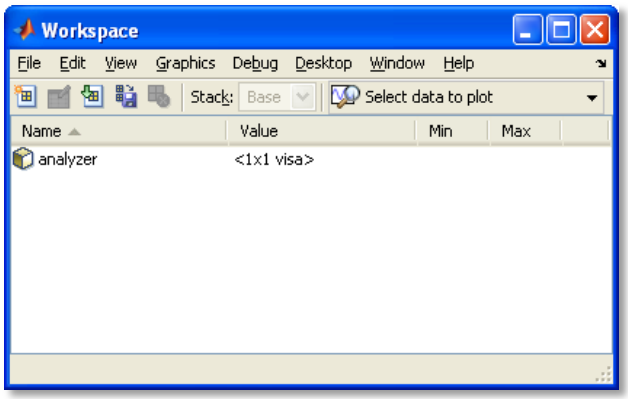

Figure 6: MATLAB Workspace Screenshot Showing Object Instance

It is important to not delete the object associated with the test equipment. Doing such can cause erratic behavior when using the interface functions. It is recommended to delete the device using the function `portClose`, described in a later section.

For completeness, the following example demonstrates connecting MATLAB ICT with the *Agilent MSO6032A*. Again referencing Figure 4, the oscilloscope was found to be on the address $GPIB1 :: 7 :: INSTR$. From the `portOpen` help description, the string `'scope'` is passed for the `inst_type` parameter. Therefore, the scope is opened by typing the following in the command window:

```
>> MSO6032A = portOpen('GPIB1::7::INSTR','scope')
```

The above illustrates the end user freedom in creating the equipment object instance name. In the above example, the object is called `MSO6032A`.

The following gives the output after calling `portOpen` to create the communication channel with the oscilloscope.

```
Instrument Device Object Using Driver : Agilent546XX.mdd

Instrument Information
    Type:               IAgilent546XX
    Manufacturer:       Manufacturer
    Model:              Model

Driver Information
    DriverType:         MATLAB IVI
    DriverName:         Agilent546XX.mdd
    DriverVersion:      1.0

 Communication State
    Status:             open
```

At this point, it is important to distinguish how MATLAB ICT communicates with the *Agilent 4395A* and *MSO6032A*. In short, MATLAB ICT considers the *4395A* a device object; while, the *MSO6032A* is considered an instrument object. A device object is device unique, whereas, an instrument object usually represents a family of instrument types. Thus, one will find the interface functions for the *4395A* written at a "low-level" compared to the *MSO6032A*. Although transparent to the user, MATLAB ICT uses an intermediate driver to communicate with the oscilloscope. Specifically, the driver `Agilent546XX.mdd`, available from Agilent, is used to ease communication with the *MSO6032A*.

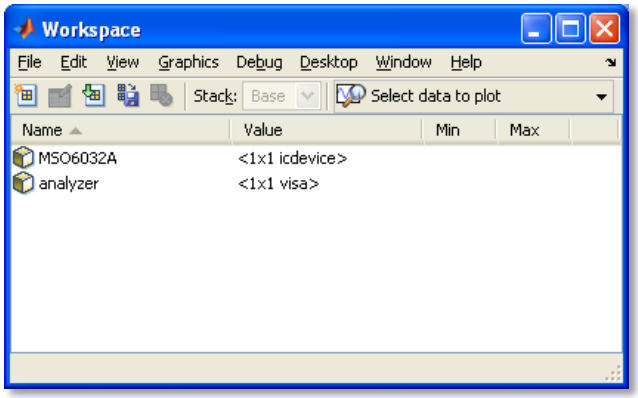

Figure 7: MATLAB Workspace Screen Shot Showing Analyzer and Scope Objects

At the time of this writing, `portOpen` crudely checks to ensure the device can be connected. This is accomplished by checking the passed GPIB address with connected instruments.

portClose.m

The main purpose of the portClose function is to assist the user in deleting the object from the MATLAB workspace. First, the communication channel must be closed. Next, the device object can be deleted from memory. The help description is listed below.

```
device = portClose(device, inst_type)
Description: Closes communication port and deletes object passed.
Input:
    device -> object to closeting GPIB address of equipment
                (retrieved from Agilent device)
    inst_type -> String indicating instrument type as defined below
      1) 'analyzer' clears connection to Agilent 4395A
                    Network/Spectrum/Impedance Analyzer
      2) 'scope' clears connection to Agilent MSO6032A oscilloscope
      3) 'func_gen' clears connection to Agilent 33120A Function Generator
Return: N/A
```

The user must pass the name of the device and the type of instrument. The parameter `inst_type` is the same as the parameter used in `portOpen`.

General Instrument Control

When sending a command or attempting to retrieve data, the communication protocol dictates GPIB have priority for input. Consequently, the user will not be able to change input control while the device is communicating. For the *Agilent MSO6032A*, this device lock is released once the command/query completes. However, the *Agilent 4395A* will enter a remote control instrument state. This is indicated by an amber light just above the `Local` key on the front panel. In order to make necessary changes, the user must press the. This releases the analyzer for manual control.

Data Capture

Given the nature of this independent study, focus was directed towards developing interfacing functions to capture data. Two main functions support data capture. The first, `getDataAnalyzer.m` is used to capture both network and spectrum analysis plots from the *Agilent 4395A*. The second, `getAnalogWaveformScope.m` is used to capture the waveforms on Channel 1 and/or Channel 2 from the *Agilent MSO6032A*.

Capturing Network/Spectrum Analysis from Analyzer

As previously mentioned, `getDataAnalyzer.m,` supports both Network and Spectrum Analyzer modes. Two parameters are passed, and the output argument is optional. It is important to note captured data points reflect the analyzer's main screen. Consequently, the user must specify the appropriate start and stop frequencies (if not using one of the support functions). Furthermore, the data is retrieved unscaled, regardless of the `scale/div` setting. In Network Analyzer mode, the frequency sweep can either be linear or logarithmic. For logarithmic sweeps, the returned x-axis data is logarithmically spaced using the MATLAB `logspace` function.

The help description is shown below for `getDataAnalyzer.m`.

```
[data, units] = getDataAnalyzer(device, channel)
Description: Retrieves axis data from analyzer.  The 4395A does not
             return x axis data, thus it must be inferred from the
             frequency sweep.  Function also retrieves unit base from
             device.  The channel variable is optional, if not passed,
             the current channel will be selected.
Input: device -> interface object
       channel -> the channel to retrieve data from: (optional)
                    1 - Channel 1
                    2 - Channel 2
Return: data -> structure containing x and y axis values and axis unit
```

getDataAnalyzer.m requires two parameters to be passed. The first parameter is the analyzer object created using the portOpen.m function. The second parameter is the channel to download. This parameter is optional. If the user does not pass a second parameter, the active channel is selected for download.

The function, by default, returns a structure containing myriad information about the capture. The following elements are returned to the user for further analysis.

```
.x -> waveform data for x axis
.y -> waveform data for y axis
.xunit -> units of x axis
.yunit -> units of y axis
.centfreq -> center frequency
.startfreq -> start frequency
.endfreq -> end frequency
```

To access each structure element, the user types the variable name followed by a period, then the element name. For example, assuming the output variable was called data, to access the x axis information, the user types the following at the command window,

```
>> data.x
```

A second option is to call the function with no output arguments. Doing such automatically plots the data for the user. Examples of the plots are shown in the section *Examples Capturing with the Agilent 4395A*.

Capturing Analog Waveforms from Oscilloscope

The getAnalogWaveformScope.m function operates similar to the getDataAnalyzer.m function. Two parameters are passed, and the output argument is optional. The help description for getAnalogWaveformScope.m. is shown below.

```
analog_wf = getAnalogWaveformScope(device, channel)
Description: Retrieves the desired analog waveform from the oscilloscope.
             If user does not specify output arguements, the function
             automatically plots the results.

Input:
```

```
    device -> interface object for oscilloscope
    channel -> respective channel to capture
               1 -> channel one
               2 -> channel two

Return: analog_wf -> structure containg various data for the analog
                     waveform
```

getAnalogWaveformScope.m requires two parameters to be passed. The first parameter is the scope object created using the portOpen.m function. The second parameter is the channel to download. The function, by default, returns a structure containing myriad information about the capture. The following elements are returned to the user for further analysis.

```
.data -> waveform data
.initial -> intial x value (functions sets to x_initial = 0sec
.increment -> delta x for each capture point in waveform
.time -> time vector for plotting
.freq -> frequency measurement in Hz
.period -> period measurement in sec
.voltrms -> rms voltage in volts
.voltpp -> peak-to-peak voltage in volts
.voltmax -> maximum peak voltage in volts
.voltmin -> minimum peak voltage in volts
.x1pos -> position of x1 cursor in sec (returns 0 if disabled)
.x2pos -> position of x2 cursor in sec (" ")
.y1pos -> position of y1 cursor in volts (" ")
.y2pos -> position of y2 cursor in volts (" ")
.deltax -> x2 - x1
.deltay -> y2 – y1
```

To access each structure element, the user types the variable name followed by a period, then the element name. For example, assuming the output variable was called analog_wf, to access the waveform data points, the user types the following at the command window,

```
>> analog_wf.data
```

A second option is to call the function with no output arguments. Doing such automatically plots the data for the user. Examples of the plots are shown in the section *Examples Capturing with the Agilent MSO6032A*.

Support Functions

Several support functions exist for the *Agilent 4395A*. The following table summarizes the support functions. Each function is listed in specified Appendix.

Table 1: Agilent 4395A Support Function Description

Function	Description
markerControl	Control operation of marker
avgerageFunc	Control averaging function
measureNetworkType	Selects the measurement mode
channelSelect	Selects a channel

Each function is discussed in further detail in the following paragraphs. It is recommended the user experiment with the support functions to gain a full understanding of the functions.

markerControl.m

The purpose of this function is to control the marker position. The help description is listed below.

```
data = markerControl(device, action)
Description: Controls how the marker is displayed on the analyzer.
             Allowable operations are turning marker off, track peak value,
             move peak value to center of display, or move marker to
             peak.  Returns the value of marker in a structure with unit.
Input: device -> Interface object to initialize
       action -> 'track' track peak value
                 'off' turns off the marker search
                 'center' move peak to center of display
                 'peak' move marker to peak value (default)
Return: data -> value of the marker position
```

If the user passes `track`, then the marker tracks the peak value. By passing `center`, the peak value is moved to the center of the display and the marker subsequently moved to the center peak. Passing the parameter, `peak`, moves the marker to the current peak value. The marker subsystem can be turned off by passing the `off` parameter. The value of the marker is returned to the MATLAB workspace for later reference. This can be especially useful when annotating plots.

averageFunc.m

The purpose of this function is to control the analyzer averaging subsystem. The help description is listed below.

```
data = averageFunc(device, action, setAvgValue)
Description: Controls the averaging function of the network analyzer.
             Allowable operations are turning on/off, restarting, and
             setting the averaging factor.  When setting the averaging
             factor, one must ensure enough sweeps occur for the
             averaging factor to complete averaging (count)
Input: device -> analyzer
       action -> 'on' turns on averaging factor (default 16)
```

```
                    'off' turns off averaging factor
                    'restart' restarts the averaging factor count
                    'set' sets the averaging factor
                    'query' or '?' queries the object for the current
                                   averaging factor
          setAvgValue -> optional when not setting averaging factor, when setting
                         factor, this parameter must be passed
   Return: data -> returns query value in query mode, value when
                   setting the averaging factor, value after reset, 0 otherwise
```

If the user passes the value `on`, then the averaging subsystem is turned on. Likewise, the averaging subsystem can be turned off by passing `off`. The averaging factor can be restarted by passing the parameter value `restart`. The user can optionally set the averaging by passing an additional value `setAvgValue`.

measureNetworkType.m

The purpose of this function is to select the common measurement modes while in Network Analyzer. The help description is listed below.

```
  Description: Selects the measurement measType for the passed interface
               object as defined by the 'measType' parameter
  Input: device -> Interface object to initialize
         input -> Input to measure, allowable values:
                  A/R: 'a/r', 'A/R', 'AR'
                  B/R: 'b/r', 'B/R', 'BR'
                  A: 'a', 'A'
                  B: 'b', 'B'
                  R: 'r', 'R'
  Return: N/A
```

As shown above, the user can select between 5 different measurement modes. The function ensures the user does not attempt to pass an erroneous value for `measType`.

channelSelect.m

The last support function developed was a function designed to select a different channel. Moreover, the function also supports a query mode. Therefore, this function naturally leads to a comparison operator. The help description is listed below.

```
   data = chanSel(device, channel, query)
   Description: Default mode of operation selects the channel indicated
                by the 'channel' parameter.  Optionally can query for
                current channel, thus enabling comparision operations.
                Query mode enabled by passing third arguement, normally 'y'
                or 'Y' but not necessary.
   Input: device -> interface object
          channel -> the channel to select
          query -> optional, pass arguement to query instead of selecting
                   channel
   Return: channel -> equals passed channel in selection mode, 0 otherwise
           query -> equals 0 in selection mode, returns channel number in
```

```
query mode
```

Depending on the mode, the function returns a structure with two elements. The first element is `channel` and the second element is `query`.

Examples Capturing with the Agilent MSO6032A

The following set of examples illustrates the functionality of `getAnalogWaveformScope.m`. Several different types of waveforms are captured and analyzed.

The following table gives a short description of each example.

Table 2: Summary of Oscilloscope Examples

Ex	Description
1	$56kHz$ Sinusoid, $1V_{pp}$
2	$1MHz$ Square Wave, $500mV_{pp}$, 50% Duty
3	$19.5kHz$ Ramp, $1.5V_{pp}$, 65% Symmetry
4	$3Hz$ Cardiac Rhythm, $2V_{pp}$

It is important to note that the following examples illustrate the use of the auto-plot feature of the `getAnalogWaveformScope.m` function. As such, the function data is returned to the MATLAB base stack as the variable `ans`. However, this data is rewritten each time the function is called. In order to preserve the values, the user must call the function with an output argument specified.

In the section *Further Examples* the user is guided through plotting captured waveforms without the auto-plot. Moreover, the examples shown in the aforementioned section illustrate how to combine the plot data into a single descriptive plot.

The following examples assume the oscilloscope has been properly instantiated via `portOpen`. For the following examples, the oscilloscope is called `MSO6032A.`

Note: An *Agilent 33250A 80MHz Function/Arbitrary Waveform Generator* was used to produce all the example waveforms, unless otherwise specified.

56kHz Sinusoid

From the above table, the first example is a $56kHz$ sinusoidal waveform. This is the most elementary waveform of analysis. The peak-to-peak voltage was set $1V_{pp}$.

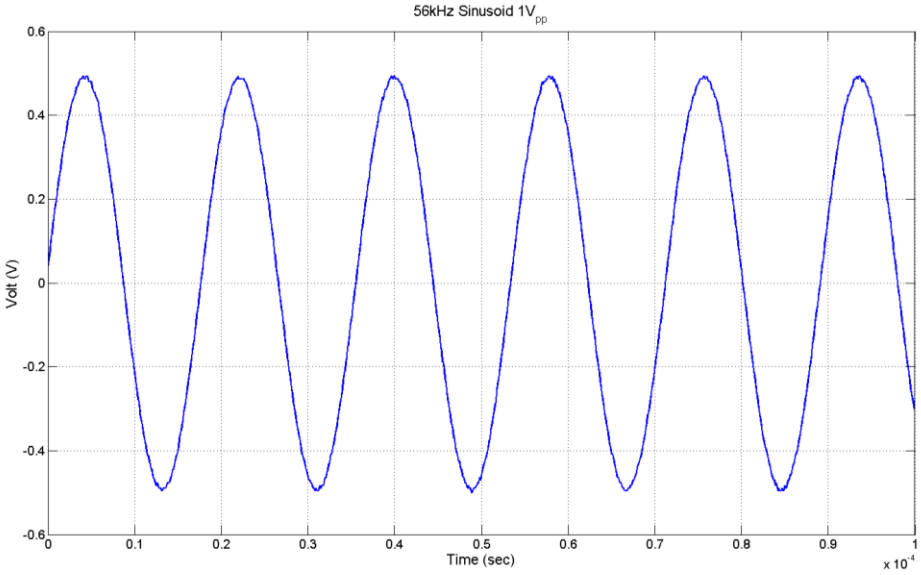

Figure 8: Oscilloscope Capture Example #1 (56kHz Sinusoid 1 $\boldsymbol{V_{pp}}$)

The following shows the data returned by the function. The x-axis data points are stored in the structure element `data`. The element `increment` gives the spacing between the points in the element array `time`. The following table gives a short comparison between expected values and observed values.

```
data: [1x1000 double]
     initial: 0
increment: 1.0000e-007
time: [1x1000 double]
  rise: 5.0000e-006
  fall: 5.1000e-006
     freq: 56180
 period: 1.7900e-005
   voltrms: 0.3482
    voltpp: 1.0063
   voltmax: 0.5050
   voltmin: -0.5013
```

Table 3: Capture Characteristics for $\boldsymbol{56kHz}$ Sinusoid

Parameter	Expected	Observed	% Diff
f	$56kHz$	$56.18kHz$	-0.321
V_{pp}	$1V$	$1.006V$	$-.60$
V_{max}	$+0.5V$	$+0.505V$	-1.00
V_{min}	$-0.5V$	$-0.5013V$	-0.26

1MHz Squarewave, 50% Duty Cycle

The next example illustrates the ability of the function to capture higher frequency waveforms. Moreover, the following example shows the ability to capture waveforms is not limited to near ideal sinusoids.

The frequency chosen for this experiment was $f_0 = 1MHz$. While a relatively slow signal, it is much greater than the previous example. Consequently, the returned time axis spans a smaller interval than the previous example. Therefore, if the user spans far out, resulting in a large number of time points, the instrument may return an error since the buffer size was exceeded.

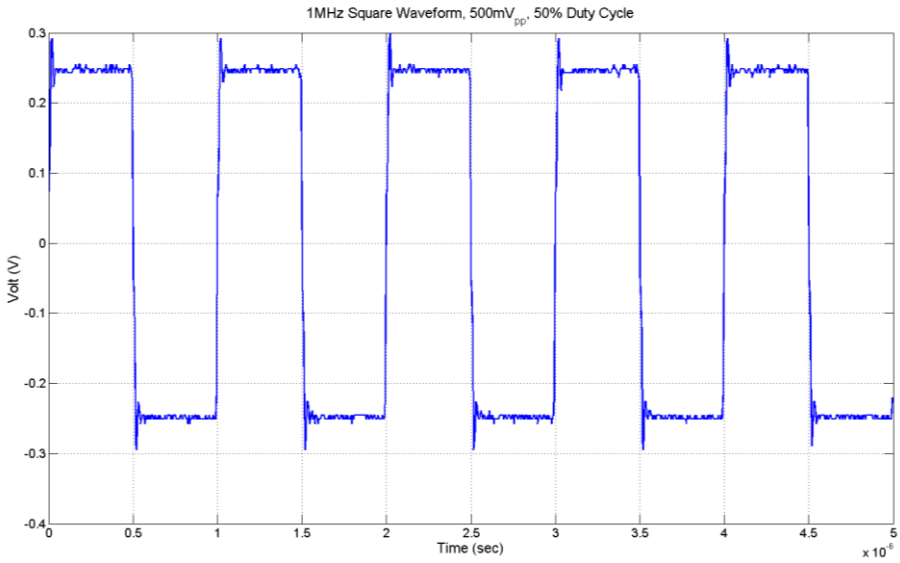

Figure 9: Oscilloscope Capture Example #2 ($\mathbf{1}\boldsymbol{MHz}$ Square Wave, $\mathbf{500}\boldsymbol{mV_{pp}}$, $\mathbf{50}\%$ Duty Cycle)

As with the previous example, the function returns a structure with numerous elements describing the plotted waveform (the results are returned to the MATLAB base stack as ans).

```
    data: [1x1000 double]
         initial: 0
   increment: 5.0000e-009
    time: [1x1000 double]
      rise: 1.5000e-008
      fall: 1.5000e-008
         freq: 1000000
     period: 1.0000e-006
      voltrms: 0.2450
       voltpp: 0.6000
      voltmax: 0.2988
      voltmin: -0.3013
```

Table 4: Capture Characteristics for $\mathbf{1}\boldsymbol{MHz}$ Squarewave

Parameter	Expected	Observed	% Diff
f	$1MHz$	$1MHz$	0.0
V_{pp}	$0.5V$	$0.6V$	-20.0
V_{max}	$+0.25V$	$+0.299V$	-19.6
V_{min}	$-0.25V$	$-0.301V$	-20.4

The above results give insight into the behavior of the function generator. Clearly, the output oscillates around the settling value of $\pm 0.25V$. The function generator does not allow the user to specify rise/fall times; therefore, no comparison could be made.

19.5kHz Ramp Waveform, 65% Symmetry

The next example extends the previous example such that the rise/fall times can be compared since the symmetry can be chosen by the user. The following figure shows a ramp waveform with 65% symmetry.

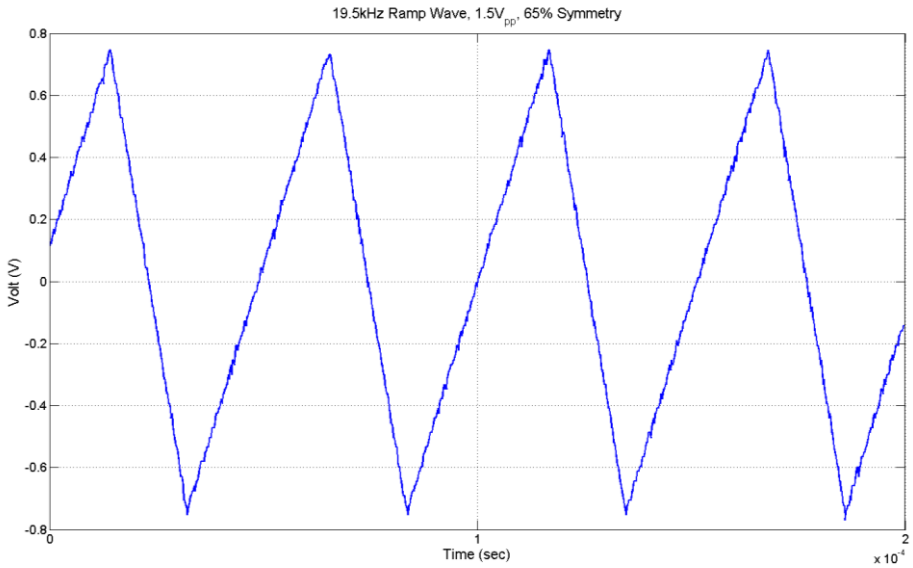

Figure 10: Oscilloscope Capture Example #3 ($\mathbf{19.5}\boldsymbol{kHz}$ Ramp Wave, $\mathbf{1.5}\boldsymbol{V_{pp}}$, $\mathbf{65}\%$ Symmetry)

As previously mentioned, using the ramp functions, the rise/fall times can be compared to expectations. The Agilent MSO6032 defines rise/fall time as 80% of expected value. Thus, the rise/fall times are with respect to $\pm 600\text{mV}$.

With symmetry of 65%, it follows that the positive slope takes $.65 \cdot T_{period}$, and the negative slope portion takes $.35 \cdot T_{period}$. The period is the reciprocal of the frequency, thus, $T_{period} = \frac{1}{f} = 51.28\mu s$. After substituting, the positive slope takes $33.33\mu s$ and the negative slope takes $17.948\mu s$. To calculate the rise/fall times with respect 80% of $\pm V_{max}$, one simply multiplies the previous results by $0.8 \cdot T_{slope}$. The following table summarizes the results.

```
data: [1x1000 double]
     initial: 0
increment: 2.0000e-007
time: [1x1000 double]
  rise: 2.6000e-005
  fall: 1.4000e-005
     freq: 19530
 period: 5.1200e-005
   voltrms: 0.4297
    voltpp: 1.5310
   voltmax: 0.7630
   voltmin: -0.7690
```

Table 5: Capture Characteristics for $\mathbf{19.5}\boldsymbol{kHz}$ Ramp Waveform

Parameter	Expected	Observed	% Diff
f	$19.5kHz$	$19.53kHz$	-0.153
V_{pp}	$1.5V$	$1.53V$	-2.00
V_{max}	$+0.75V$	$+0.763V$	-1.73
V_{min}	$-0.75V$	$-0.769V$	-2.53
t_{rise}	$26.66\mu s$	$26.0\mu s$	2.48
t_{fall}	$14.36\mu s$	$14.0\mu s$	2.51

3Hz Cardiac Rhythm

The next example is shown strictly to reinforce the functionality of `getAnalogWaveformScope.m`. The following waveform is a stored waveform included with the Agilent 33250A Function Generator. A frequency of $3Hz$ was chosen to represent an elevated heart rate. In particular, the ability of the function to return a reasonable frequency is verified.

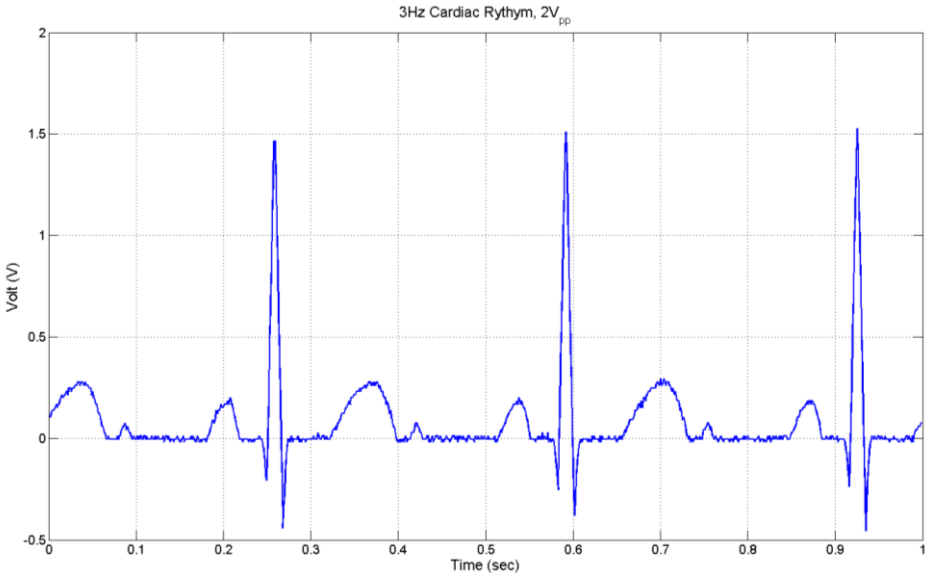

Figure 11: Oscilloscope Capture Example #4 (**$3Hz$** Cardiac Rhythm, **$2V_{pp}$**)

The above figure illustrates a common cardiac rhythm at a frequency of $f_{rate} = 3Hz$. The following summary shows the returned data represented by the above figure.

```
    data: [1x1000 double]
         initial: 0
   increment: 1.0000e-003
    time: [1x1000 double]
        rise: 0.0060
        fall: 0.0060
        freq: 3.0030
      period: 0.3330
     voltrms: 0.2222
      voltpp: 2.0310
     voltmax: 1.5590
     voltmin: -0.4720
```

Table 6: Capture Characteristics for **$3Hz$** Cardiac Waveform

Parameter	Expected	Observed	% Diff
f	$3Hz$	$3.003Hz$	0.1
V_{pp}	$2V$	$2.031V$	-1.55
V_{max}	$+1.5V$	$+1.559V$	-3.93
V_{min}	$-0.5V$	$-0.472V$	5.6

As expected, the returned is nearly identical to the expected value. As with the other examples, due to the function generator, the expected $V_{pp}, V_{max}, \& V_{min}$ deviate from ideal expectations.

Examples Capturing with the Agilent 4395A

The following set of examples illustrates the functionality of `getDataAnalyzer.m`. Several different types of spectral plots are obtained and analyzed.

The following table gives a short description of each example.

Table 7: Summary of Spectrum Analyzer Examples

Ex	Description
1	$1MHz$ Sinusoid, $1V_{pp}$
2	$80MHz$ Square Wave, $1V_{pp}$, 50% Duty

It is important to note that the following examples illustrate the use of the auto-plot feature of the `getDataAnalyzer.m` function. As such, the function data is returned to the MATLAB base stack as the variable `ans`. However, this data is rewritten each time the function is called. In order to preserve the values, the user must call the function with an output argument specified.

In the section *Further Examples* the user is guided through plotting captured spectral plots without the auto-plot. Moreover, the examples shown in the aforementioned section illustrate how to combine the plot data into a single descriptive plot.

The following examples assume the analyzer has been properly instantiated via `portOpen`. For the purposes of the following examples, the analyzer is called `A4395A`. When capturing data from the analyzer, the frequency span will remain unaltered. Consequently, the plotted data will exactly resemble the analyzer display. Moreover, the maximum number of data points, 801, are always captured.

Note: An *Agilent 33250A 80MHz Function/Arbitrary Waveform Generator* was used to produce all the example waveforms, unless otherwise specified.

1MHz Sinusoid

The following example illustrates the basic functionality of the `getDataAnalyzer.m` function. The user must ensure the Agilent 4395A is in Spectrum Analysis mode. This example also illustrates how the data capture function captures the current display.

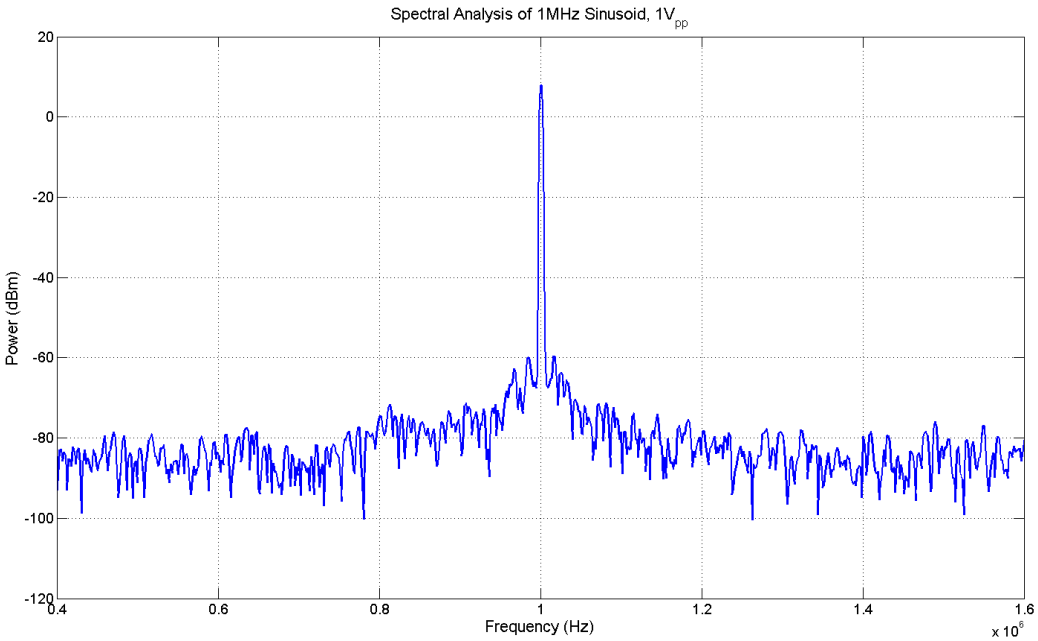

Figure 12: Spectral Plot for $\mathbf{1}\boldsymbol{MHz}$ Sinusoid, $\mathbf{1}\boldsymbol{V_{pp}}$

Referencing the above figure, if the frequency span was larger, one would see the harmonics present. However, one can easily obtain the above plot by using the `markerControl.m` and passing the `center` argument.

The following shows the function's output. As previously mentioned, the maximum number of returned points is 801. This is the maximum allowed by the device. The end user could easily replicate the above plot using the returned data.

```
retstr: [1x12015 char]
retdata: [801x1 double]
actualnumpoints: 801
anpc: 1
numdatapoints: 801
mode: 'SA'
centfreq: 1000000
freqspan: 1.4503e+006
startfreq: 2.7487e+005
endfreq: 1.7251e+006
delta: 1.8128e+003
sweep_type: 'LINF'
x: [801x1 double]
xunit: 'Hz'
y: [801x1 double]
yunit: 'dBm'
```

5MHz Square Wave, $1V_{pp}$, 50% Duty Cycle

The last example is a $5MHz$ Square Wave. The duty cycle is 50% with $1V_{pp}$. The objective of this example is to show the how the frequency span effects the captured data. The particular span used for this example is approximately $400MHz$.

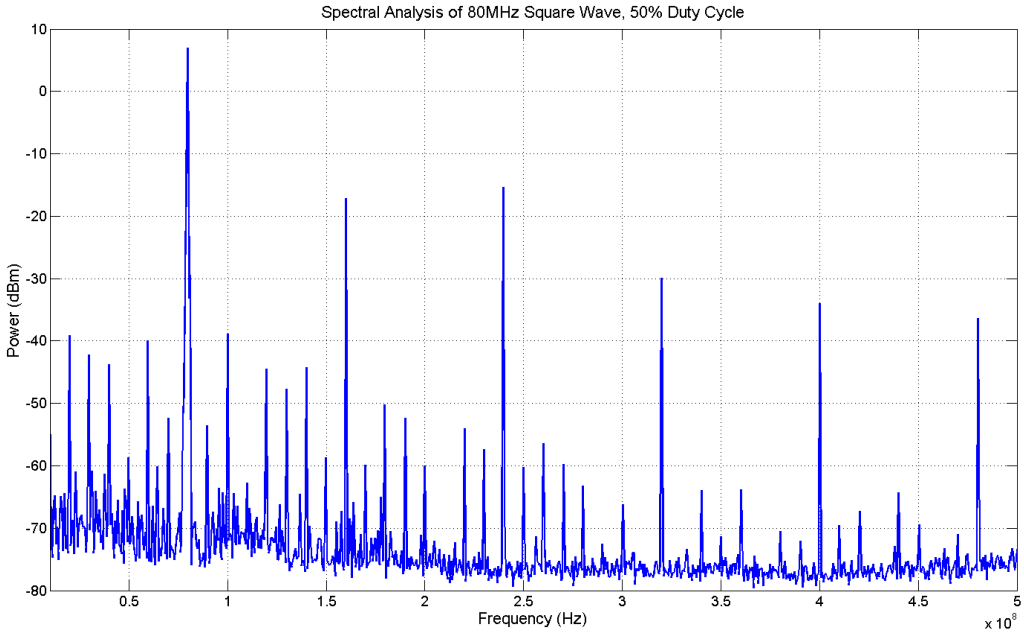

Figure 13: Spectral Plot for $\mathbf{80}\boldsymbol{MHz}$ Square Wave, $\mathbf{50}\%$ Duty Cycle

Using the following defin of a square wave, the fourier transform can be analyzed.

$$x_{square}(t) = \frac{4}{\pi}\sum_{k=1}^{\infty}\frac{\sin\big((2k-1)2\pi ft\big)}{2k-1} \tag{3}$$

The above equation shows the strong relationship to odd harmonics. The above plot shows this relationship in the frequency domain. Note the third harmonic has greater power than the second harmonic.

The below shows the function's output. The user could obtain a plot similar to the above using the returned data.

```
retstr: [1x12015 char]
retdata: [801x1 double]
actualnumpoints: 801
anpc: 1
numdatapoints: 801
mode: 'SA'
centfreq: 255000000
freqspan: 490000000
startfreq: 10000000
endfreq: 500000000
delta: 612500
```

```
sweep_type: 'LINF'
x: [801x1 double]
xunit: 'Hz'
y: [801x1 double]
yunit: 'dBm`
```

Further Examples

The following two examples illustrate how the two test equipment can be jointly used to obtain informative plots and aid in further analysis as required. The first example is a $50kHz$ pulse train, with width, $\tau = 1\mu s$. The second example is a simple RC low-pass filter. The characteristics of the filter are verified through the capture data and visually confirmed by plotting the results.

The following examples go through each step required in obtaining the respective data. However, it is assumed the oscilloscope and analyzer device objects have been created and the port communication is currently OPEN. For the following examples, the oscilloscope is called MSO6032A and the Spectrum Analyzer is called A4395A.

50kHz Pulse Train, $\tau = 1\mu s$

The first example in this section illustrates how the two test equipment can be jointly used to obtain informative plots as shown in Figure 14. Unlike the previous examples, by specifying an output argument, the captured data is not automatically plotted. Consequently, the data is stored to the MATLAB workspace. For the proceeding example, the time domain (oscilloscope) variable is `scope` and the frequency domain (analyzer) variable is `spectrum`.
To obtain the oscilloscope capture, the following command is entered,

```
scope = getAnalogWaveformScope(MSO6032A, 1);
```

If the output is not suppressed, the structure element values will be displayed to the command window as such,

```
data: [1x1000 double]
initial: 0
increment: 1.0000e-007
time: [1x1000 double]
rise: 3.0000e-007
fall: 3.0000e-007
freq: 50000
period: 2.0000e-005
voltrms: 0.2221
voltpp: 1.1880
voltmax: 1.0910
voltmin: -0.1120
x1pos: 0
x2pos: 0
y1pos: 0
y2pos: 0
deltax: 0
deltay: 0
```

To obtain the spectrum capture, the following command is entered,

```
spectrum = getDataAnalyzer(A4395A, 1);
```

Likewise, the structure element values for `spectrum` are as follows,

```
centfreq: 5000000
freqspan: 10000000
startfreq: 0
endfreq: 10000000
delta: 12500
sweep_type: 'LINF'
x: [801x1 double]
xunit: 'Hz'
y: [801x1 double]
yunit: 'dBm'
```

Next, to obtain a plot similar to Figure 14, the user can enter the following commands to plot the returned data,

```
>> subplot(211)
>> plot(scope.time,scope.data,'LineWidth',2)
>> set(gca, 'FontSize', 14);
>> grid on;
>> xlabel('Time (sec)', 'FontSize', 16)
>> ylabel('Volt (V)', 'FontSize', 16)
>> subplot(212)
>> plot(spectrum.x,spectrum.y, 'LineWidth', 2)
>> set(gca, 'FontSize', 14)
>> grid on
>> xlabel('Frequency (Hz)','FontSize',16)
>> ylabel('Power (dBm)','FontSize',16)
```

The top subplot shows the time domain waveform. The bottom subplot shows the frequency domain waveform.

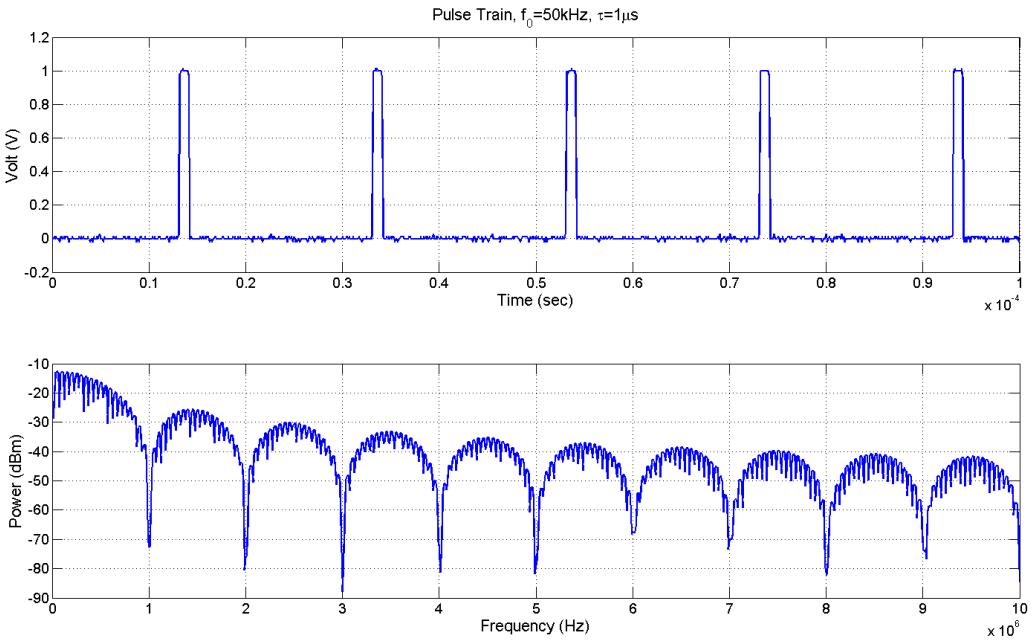

Figure 14: Pulse Train, $\boldsymbol{f_0 = 50kHz, \tau = 1\mu s}$

The most interesting thing to note in the above figure is the spectral nulls at $M * 1MHz$, where M is an integer. The pulse width of the above pulse train is $\tau = 1\mu s$, therefore, $f_\tau = 1MHz$. The significance of the spectral nulls follows from the following Fourier transform pair,

$$\mathcal{F}^{-1}\left\{\prod\left(\frac{t}{\tau}\right)\right\} = \tau \text{sinc}(\pi f) \tag{2}$$

It follows that the $\text{sinc}(\cdot)$ function is zero at $f_{null} = M * 1\text{MHz}$, consequently, the nulls are observed as shown in Figure 14.

Change Log

2/22/2011 – Added information regarding USB direct cable connection for Agilent MSO6032A Mixed Signal Oscilloscope. Added document conventions.