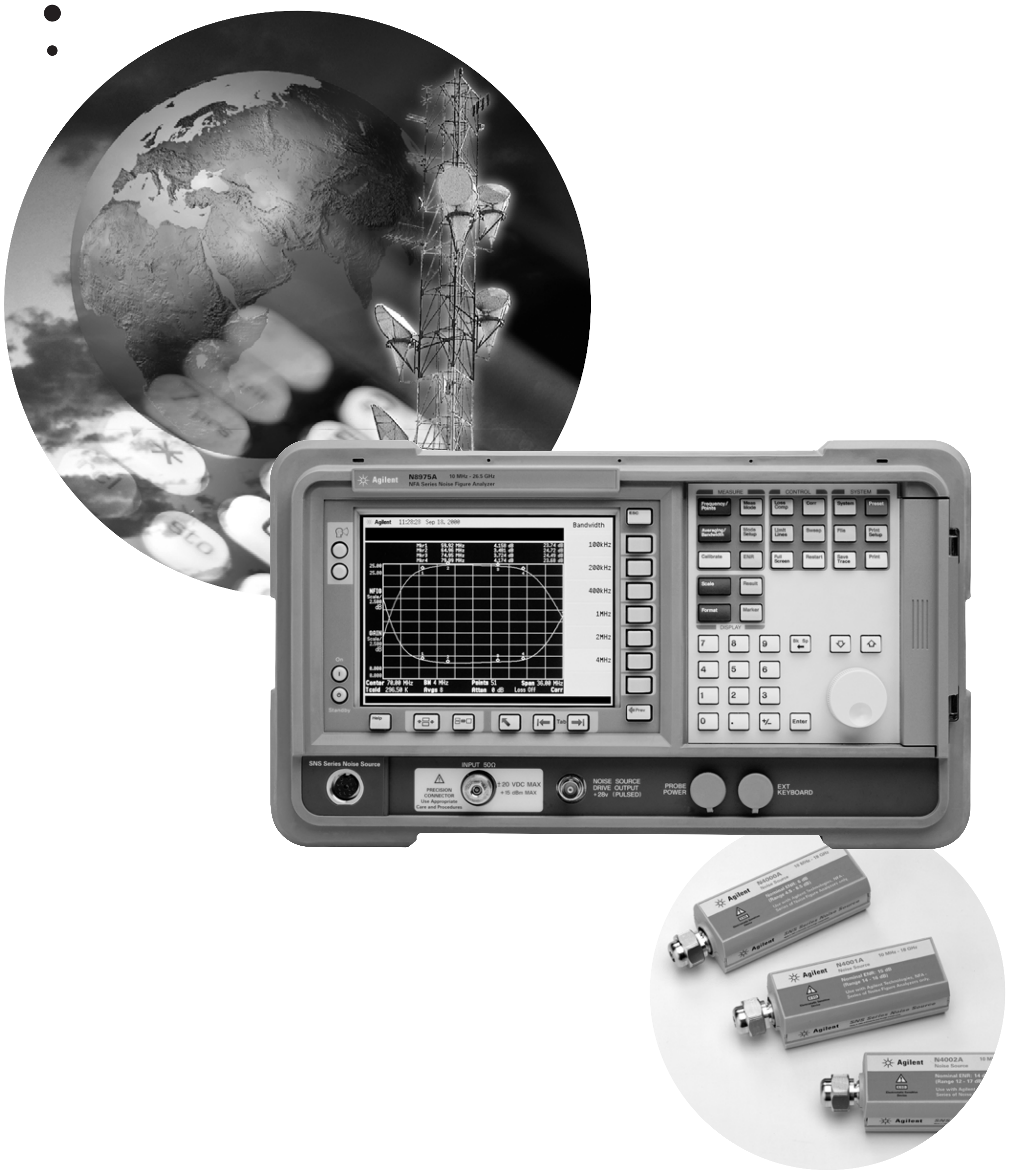
Agilent NFA Series Noise Figure Analyzer Programming Examples

Product Note



Agilent Technologies

Innovating the HP Way

Table of Contents

Introduction

This product note will demonstrate how to write software to enable the automated control of the new NFA Series noise figure analyzers using SCPI standard commands.

This product note is meant to compliment the information and procedures given in the NFA "User's Guide" and "Programmer's Guide" (literature numbers N8972-90001 and N9872-90002). These and other documents are shipped on CD ROM with the NFA and are available via the web at **www.agilent.com/find/nfa**

The examples in covered in this product note will work with all Agilent NFA-Series products. The narrowband noise figure measurement example will work on all models of NFA-series noise figure analyzers, except the N8972A, which has no narrow measurement bandwidth functionality. This product note is applicable to firmware revision A.00.01 and above.

Conventions used in this product note

Programming examples shown in this note are illustrated using pseudocode. GPIB indicates the IEE-488 interface bus for remote instrument control; the GPIB default address is 8 for the NFA.

Upper case letters in commands must be sent; lower case letters are optional. (Using the minimum number of characters minimizes GPIB command transmission time).

Example of two equivalent commands:

:SENSe:SWEep:POINts 15
[Only the upper case characters need to be sent]

:SENS:SWE:POIN 15
[The same command can be shortened to read as shown]

Assumptions

- Code will run on a computer connected to the test set via the GPIB interface.
- Commands described in this product note are applicable to both the N8972A and N8973A except where stated otherwise.
- Features described in this product note apply to NFA firmware revision A.00.01. and beyond.

Hardware requirements

- Controller, GPIB, GPIB cables: A PC or workstation with an IEEE 488.2 GPIB industry standard interface.
- NFA: An N8972A, N8973A, N8974A and N8975A.
- 346A: A noise source, whose choice will depend upon the frequency and noise figure of the DUT to be measured.
- DUTs: A mini circuits ZFL-1000LN low noise amplifier is used in both the basic noise figure and gain measurement example and also the narrowband measurement example.
- Filters: In the narrowband measurement section a mini circuits SBP-70 70 MHz bandpass filter is used.
- RF Cables: A range of Type-N to Type-N cables will be required in order to make some of the measurements.
- Adapters: The choice of adapters will be dependant upon the noise source connector, some adapters may be needed to connect the noise source to the DUT and then to the input of the NFA.

Agilent produces a special option for the NFA called the K10 demo kit. The K10 demo kit contains the DUT's used in the examples shown in this product note. The kit also contains a range of cables and connectors to enable most of the measurement scenarios presented in the product note to be completed. A noise source is not included in the kit and should be ordered separately.

To order the K-10 option contact your local Agilent sales office.

1. Connecting up the NFA via GPIB cable to the host computer

This section describes the layout of the GPIB connectors located on the NFA's rear panel.

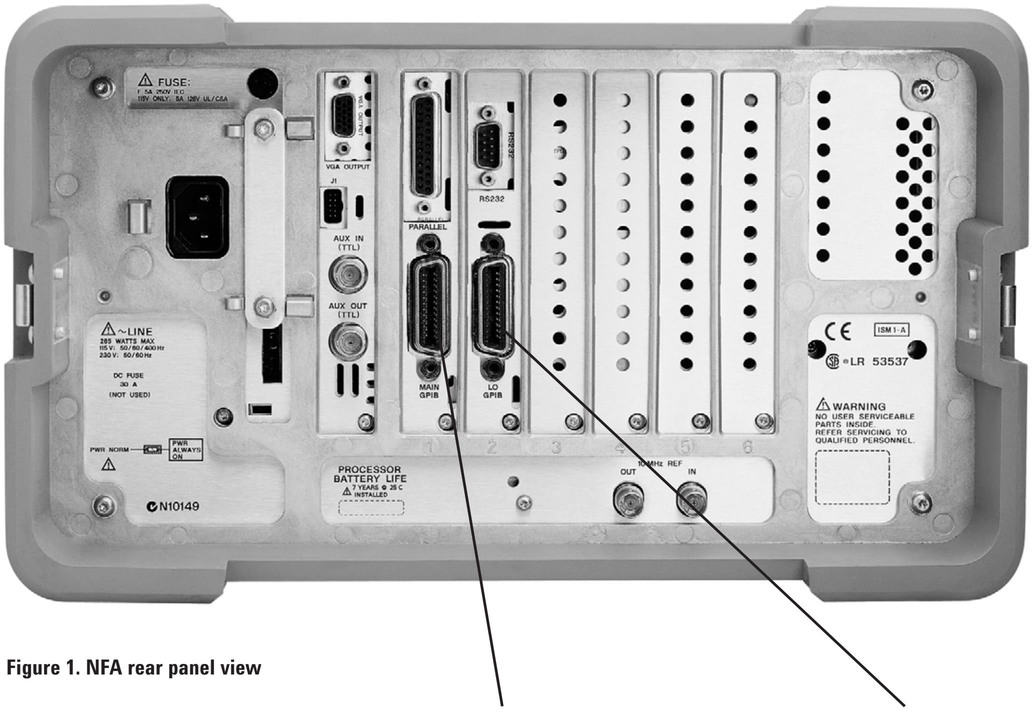


Figure 1. NFA rear panel view

1. Main GPIB connector to which the local host computer should be connected.
2. GPIB connector to which the local oscillator, if present, should be connected.

2. How to make basic noise figure and gain measurements on an amplifier

Description:

This section demonstrates how the NFA can make basic noise figure and gain measurements on an amplifier. Any generic type of amplifier, active filter or attenuator may be used as the device under test as long as its frequency range is appropriate and there is no frequency conversion within the device. This section also covers the Limit Lines feature of the NFA, where the DUT results are compared to a pre-loaded set of test parameters for quick and easy pass/fail testing.

Practical considerations:

The example is split into the following sections:

2.1 Loading and using the ENR file
2.2 Loading and using a limit line file
2.3 Set the measurement parameters
2.4 Perform a user calibration
2.5 Make the measurement
2.6 Retrieve the results

2.1 Loading and using an ENR file

The following programming example explains how to load an ENR table from the local host to the NFA. It also demonstrates how to store the table in the instruments file system and then how to retrieve it again. This example assumes that the ENR table "Source01.ENR" is available on the host computer that is to be loaded into the NFA.

The :MMEMORY:DATA *command*

The NFA uses the :MMEMORY:DATA command and query to transfer files between the local host and itself. The command transfers files from the local host to the NFA; the query transfers files in the opposite direction.

Files are loaded into the NFA using the :MMEMORY:DATA command, e.g.

```
:MMEMORY:DATA 'C:SOURCE01.ENR',#3252.....
```

It has two arguments–the file name and a data block.

Currently the file name must be fully qualified with no directory path e.g. C:SOURCE01.ENR

The data block is actually an IEEE488.2 definite length arbitrary data block

Files are retrieved from the NFA using the :MMEMORY:DATA? query, e.g.

```
:MMEMORY:DATA? 'C:SOURCE01.ENR'
```

It takes a single argument, file name, to be retrieved.

The query returns an IEEE488.2 definite length arbitrary data block

Definite length arbitrary data block

#223	this is the actual data
#	as the first character lets the NFA know that a data block is coming
2	lets the NFA know that the block uses two dig its for the data length
23	is the data length and gives the number of data bytes in the block

The remaining 23 bytes is the actual data

The arbitrary data block has more than two forms. The one described here is the definite length variety. The other form, the indefinite length block is not covered here. Using # as the first character of an argument alerts the NFA to the fact that an arbitrary data block is coming. Any non-zero digit identifies the block as a definite length block and gives the number of digits that make up the remainder of the block header, 2 in the example here. The remaining header digits give the number of data bytes in the block, 23 in this example. The remainder of the block is the actual data.

Building the :MMEMORY:DATA *command*

```
1 function BuildMmemData (string NfaFile,
  number length,string command)
2 string fileLengthString = ToString (length)
3 number headerLength = StringLength(fileLengthString)
4 string headerLengthString = ToString (headerLength)
5 string blockHeader = “#” + headerLengthString +
  fileLengthString
6 command = “:MMEMORY:DATA ‘” + NfaFile + “‘,” +
  blockHeader
7 end function
```

Comments to code for building the :MMEMORY:DATA *command*

Function BuildMmemData constructs the :MMEMORY:DATA command from the name of the file to be written to on the NFA.

Line 1 shows that the function has three arguments. The first is the name of the file to be written to on the NFA. The second is length of the source file on the local host. The third is a return parameter that holds the constructed command.

Line 2 converts the source file length to a string representation of the number e.g. ToString(123) gives “123”.

Line 3 determines the length of the source file length string so that it can generate the first digit of the block header.

Line 4 converts the header length to a string.

Line 5 builds the block header by concatenating the parts together.

Line 6 constructs the command by concatenating the SCPI header with the arguments.

Note that the data does not form part of the command at this time.

Transferring a file to the NFA:

```
1    function CopyFileToNFA (string fromName, string toFile)
2    string mmemData, byte
3    number length = FileLength(fromFile)
4    BuildMmemData (toFile, length, mmemData)
5    RemoteWrite (mmemData, false)
6    loop length times
7    GetByte (fromFile, byte)
8    if last byte then
9    RemoteWrite (byte, true)
10   else
11   RemoteWrite (byte, false)
12   endif
13   end loop
14   ReportErrors( )
15   end function
```

Transferring a file to the NFA example: code comments

Line 1 Function CopyFileToNFA copies the contents of fromFile on the local host to toFile on the NFA. Note that this is not actually ENR file specific and can be used to copy any file.

Line 2 declares string variables used to hold the :MMEMORY:DATA command and data bytes to be transferred.

Line 3 gets the length of the source file.

Line 4 constructs the command as described previously.

Line 5 sends the command to the NFA. At this point no data is sent. To stop the command from terminating the second argument to RemoteWrite is set to false. This stops the controller from asserting EOI, which marks the end of a transmission. See the Appendix section of this product note for further explanation.

Lines 6 to 13 copy the data a byte at a time. When sending the last byte the controller is told to terminate the transfer.

Line 14 checks for error (e.g. out of file system space).

Note that, in this example, it is not possible to use the SendCommand function because the command itself had to contain all the data to be transferred. SendCommand assumes that the whole command is in a single buffer. It is possible to create a buffer large enough for the whole file but this is unnecessary and not always advisable. Because SendCommand wasn’t used the ReportErrors function were called explicitly.

The following programming example explains how to load an ENR table from the local host to the NFA. It assumes an ENR table; “source01.enr” is present on the host computer that is to be loaded into the NFA.

Example pseudocode for transferring an ENR file into the NFA

```
1  CopyFileToNFA (“source01.enr”,”c:source01.enr”)
2  SendCommand (“:MMEMORY:LOAD:
        ENR MEASUREMENT,’c:source01.enr’”)
```

Transferring an ENR file to the NFA example: code comments

Line 1 uses function CopyFileToNFA, defined above, to load an ENR file into the device.

Line 2 makes the ENR table, in file c:source01.enr, the active ENR file.

Within the NFA, there is the capability to use two ENR tables, one for the calibration and another for the measurement. This would be useful in the case of making measurements at Microwave frequencies using a waveguide noise source. By enabling the use of both of the ENR tables, the instrument can be correctly calibrated with the lower frequency noise source and then the measurements can be made at the higher frequencies with the waveguide noise source. It is not possible to achieve accurate results calibrating with a waveguide connector at lower frequencies.

ENR files are text files with a predefined format. The following is an example of an ENR file. Lines starting with a ‘#’ are treated as comments and can be omitted.

```
# ENR Data File
# Created by N8973A NFA Series noise figure analyzer
# Serial Number 8L39240553 Firmware Revision A.00.00
# 14:42:35  Jun 7, 2000
# Format is: Frequency (Hz), ENR (dB)
[Filetype ENR]
[Version 1.0]
[Serialnumber 3318A05185]
[Model 346A]
10000000, 5.4500
100000000, 5.5300
1000000000, 5.2700
2000000000, 5.0800
3000000000, 4.9400
4000000000, 4.8600
5000000000, 4.8300
6000000000, 4.9100
7000000000, 4.9900
8000000000, 5.1300
9000000000, 5.2500
10000000000, 5.3200
11000000000, 5.2900
12000000000, 5.3100
13000000000, 5.3300
14000000000, 5.3000
15000000000, 5.3400
16000000000, 5.3500
17000000000, 5.3100
18000000000, 5.1600
```

2.2 Loading and using a limit line file

The following programming example explains how to load a limit line file table from the local host to the NFA. This example assumes that a limit line table called "amplev01.lim" is present on the host computer that is to be loaded into the NFA.

Example pseudocode for transferring a Limit line file into the NFA

```
1 CopyFileToNFA ("amplev01.lim","c:amplev01.lim"):
2 SendCommand (":MMEM:LOAD:
      LIMIT LLINE1,'c:amplev1.lim'")
```

Line 1 shows how to load the limit line file into the NFA.

Line 2 makes "amplev01.lim" the limit line 1 file. Within the NFA, there is the capability to use 4 limit line tables including the choice for an upper and lower limit line for both noise figure and gain measurements.

LIM files are text files with a predefined format. The following is an example of an LIM file. Lines starting with a '#' are treated as comments and can be omitted.

```
# Limit Line data file
# Created by N8973A NFA Series Noise Figure Analyzer
# Serial Number GB39490112 Firmware Revision A.00.01
# 14:54:53 Jun 28, 2000
# Format is: Frequency (Hz), Magnitude (unitless),
  connected (1 or 0)
[Filetype LIM]
[Version 1.0]
[Limittype UPPER]
10000000, 3.2000, 1
100000000, 3.2000, 1
600000000, 2.4000, 1
800000000, 3.2000, 1
1000000000, 3.2000, 1
1200000000, 3.2000, 1
```

Format is: Frequency (Hz), Magnitude (will depend on whether Log/linear units is selected), connected (1 or 0)

The connected (1 or 0) indicator signifies whether that measurement point is connected to the previous one or not. If a zero is placed in this column then the NFA will not test the section between the unconnected points, and if a one is placed in this column then the NFA will test the section between the previous point and that point.

2.3 Set the measurement parameters

This section will explore how to load a user defined upper limit line for noise figure, set functions such as frequency range, number of measurement points, number of averages and also determine the measurement bandwidth for the NFA.

In the following example the ENR file is set as source01.enr and the limit line file amplev01.lim to be the current .ENR and .LIM file

Example pseudocode for setting basic measurement system parameters:

```
1     function SetMeasParams ( )
2     SendCommand (“*RST”)
3     SendCommand (“*CLS”)
4     SendCommand (“:MMEM:LOAD:
         ENR MEASUREMENT,’c:source01.enr’”)
5     SendCommand (“:MMEM:LOAD:
         LIMIT LLINE1,’c:amplev1.lim’”)
6     SendCommand (“:SENSE:FREQUENCY:STOP 1.2 GHZ”)
7     SendCommand (“:SENSE:FREQUENCY:POINTS 21”)
8     SendCommand (“:SENSE:AVERAGE:STATE ON”)
9     SendCommand (“:SENSE:AVERAGE:COUNT 15”)
10    SendCommand (“:SENSE:BANDWIDTH 4000000”)
11    SendCommand (“:SENSE:CALCULATE:LLINE1:
         STATUS ON”)
12    SendCommand (“:INITIATE:CONTINUOUS:ALL OFF”)
13    end function
```

Example pseudocode for setting basic measurement system parameters: code comments

Line 1 opens the function.

Line 2 puts the NFA into a factory-preset state for parameters such as frequency range, measurement bandwidth and number of measurement points.

Line 3 clears the status byte by emptying the error cue and clearing all the bits in all the event registers.

Line 4 sets the active ENR file in the NFA.

Line 5 loads a limit line file as limit line 1

Line 6 sets the upper measurement frequency limit for the NFA at 1.2 GHz.

Line 7 sets the number of measurement points throughout the sweep.

Line 8 turns the “Point” averaging function on.

Line 9 sets the number of average points to 15.

Line 10 indicates to the NFA which measurement bandwidth to use, In this case 4 MHz.

Line 11 defines a limit line, against which the measured data can be compared.

Line 12 enables the operator to stop continuous measurement sweep mode.

Line 13 closes the function.

2.4 Perform a user calibration

This section requires the user to physically connect the noise source to the front of the NFA to enable the calibration procedure to be completed. The calibration procedure removes the measurement systems own noise figure, often called second stage noise contribution, and the displayed results are shown in a corrected form. The results for both corrected and uncorrected measurements can be retrieved from the NFA.

Connect up the NFA and the Noise source as shown in the diagram below.

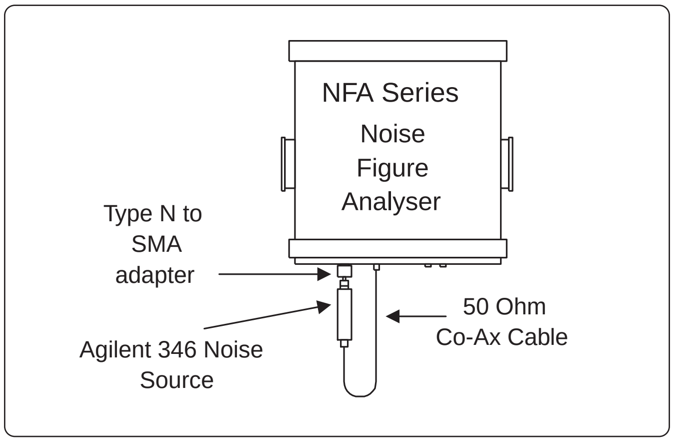


Figure 2. Calibration setup

Use a 50Ω BNC cable to connect the NFA to the 28V Noise source drive on the front of the NFA and an N type to SMA adapter, to connect the noise source to the input of the NFA.

Example pseudocode for calibrating the NFA:

```
1  function UserCalibration ( )
2  SendCommand (":SENSE:CORRECTION:COLLECT:
        ACQUIRE STANDARD")
3  SendCommand ("*WAI")
4  end function
```

Example pseudocode for calibrating the NFA: code comments

Line 1 opens the function.

Line 2 starts collection of user calibration data.

Line 3 forces the NFA to wait until it finishes calibrating before processing the next command.

Line 4 ends the function.

2.5 Making basic noise figure and gain measurements on an amplifier

After the calibration has been completed, follow the connection diagram below enable the noise figure measurement to be made. Ensure that the device under test is powered up before connection to the noise source and NFA. This pre-connection power up will ensure that no spikes will be presented to the input of the NFA.

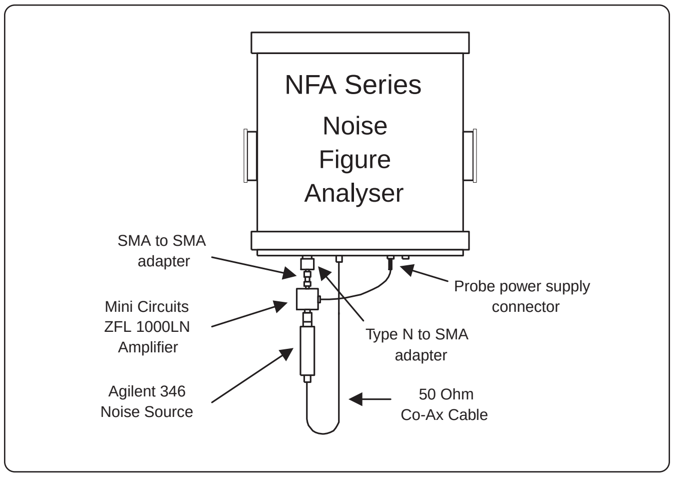


Figure 3. Basic noise figure and gain measurement connection diagram

Example pseudocode for making a basic noise figure and gain measurement on an amplifier:

```
1  function Measurement ( )
2  SendCommand ("INITIATE:IMMEDIATE")
3  SendCommand ("*WAI")
4  end function
```

Example pseudocode for making a basic noise figure and gain measurement using an amplifier: code comments.

Line 1 opens the function.

Line 2 the measurement has been selected and is waiting, this command causes the system to come out of idle and starts making the measurement.

Line 3 waits to continue.

Line 4 ends the function.

2.6 Retrieving some example results

The NFA provides an opportunity to retrieve multiple results from a single measurement

1. It is possible to retrieve uncorrected trace data for noise figure, Y-factor, hot power, cold power and effective temperature.

2. It is possible to retrieve corrected trace data for noise figure, gain, hot power, cold power and effective temperature.

3. The above can be retrieved with or without loss compensation. It is also possible to get peak, trough, delta and single amplitude values for the above.

Example pseudocode for retrieving some example results:

```
1 SendQuery (“FETCH:ARRAY:CORRECTED:
    NFIGURE?”, corrNfig)
2 SendQuery (“FETCH:ARRAY:CORRECTED:GAIN?”, corrGain)
3 SendQuery (“FETCH:ARRAY:UNCORRECTED:
    PHOT?”, uncorrPhot)
4 SendQuery (“TRACE:DATA:CORRECTED:AMPLITUDE:
    VALUE? NFIGURE,1.0GHZ”,ampl1GHz)
5 SendQuery (“TRACE:DATA:CORRECTED:AMPLITUDE:
    MAXIMUM? GAIN”, maxGain)
6 SendQuery (“TRACE:DATA:CORRECTED:AMPLITUDE:
    MINIMUM? NFIGURE”, MinNfig)
7 SendQuery (“TRACE:DATA:CORRECTED:
    AMPLITUDE:DELTA? GAIN,50MHZ,100MHZ”, deltaGain)
8 SendQuery (“CALCULATE:LLINE1:DATA?”,limitline)
9 SendQuery (“STATUS:QUESTIONABLE:INTEGRITY:
    CONDITION?”,status)
```

Example pseudocode for retrieving some example results: code comments

Line 1 retrieves the corrected noise figure sweep.

Line 2 retrieves the corrected sweep of gain measurements.

Line 3 retrieves the uncorrected sweep of Phot measurements.

Line 4 retrieves the value of corrected Noise figure at 1.0 GHz.

Line 5 retrieves the value of maximum gain from the results.

Line 6 retrieves the minimum noise figure value from the results.

Line 7 retrieves the amplitude difference between the Gain values at 50 MHz and 100 MHz.

Line 8 retrieves the list of frequency points and amplitude values set in the Limit line file.

Line 9 returns the value of the Questionable Integrity Status register. The returned value is the sum of all asserted bits in the register. If bit 7 is set it indicates that the result was outside the limit specified by limit line 1.

3. How to make narrowband measurements on filters

This section describes how to make narrowband measurements on a filter and amplifier combination, using and narrow measurement bandwidth and the frequency list function.

The only real differences in this setup from the previous basic noise figure and gain measurement are using the frequency list function and also a narrower measurement bandwidth.

This measurement example shall be split into the following sections:

3.1) Loading and using an ENR file
3.2) Creating and then loading a frequency list into the NFA
3.3) Set the measurement system parameters
3.4) User calibration
3.5) Measurement
3.6) Retrieving the results

3.1 Loading and using an ENR file

This programming example explains how to load an ENR table from the local host to the NFA. This example assumes that an ENR table called "Source01.ENR" is available on the host computer that is to be loaded into the NFA. The file can have any filename as long as it is no more than 8 characters long and is followed by the .ENR extension.

Example pseudocode for copying the ENR file into the NFA

```
1 CopyFileToNFA("source01.enr","c:source01.enr")
2 SendCommand(":MMEMORY:LOAD:
     ENR MEASUREMENT,'c:source01.enr'")
```

Example pseudocode for copying the ENR file into the NFA: code comments

Line 1 loads ENR file source01.enr, held on the local host, into the memory file system, denoted 'c:', of the NFA.

Line 2 makes the ENR table held in the file the active ENR table.

3.2 Creating and loading a frequency list into the NFA

Creating an example frequency list file: it is possible to create a file in this format as a text file with a .lst extension

```
# Frequency list data file
# Created by N8973A NFA Series Noise Figure Analyzer
# Serial Number GB40050100 Firmware Revision X.00.01
# 14:20:41  Jun 27, 2000
# Format is: Frequency (Hz)
[Filetype LST]
[Version 1.0]
54000000
60000000
70000000
78000000
84000000
```

In this Example at the five points listed indicate which to calibrate and then measure, 54 MHz, 60 MHz, 70 MHz, 78 MHz and 84 MHz.

Example pseudocode for copying the frequency list file into the NFA:

```
1 CopyFileToNFA ("test0000.lst","c:test0000.lst")
2 SendCommand (":MMEMORY:LOAD:LIST,'C:test0000.lst")
```

Example pseudocode for copying the frequency list file into the NFA: code comments

Line 1 copies the file "test0000.lst" into the NFA from the host computer.

Line 2 enables the NFA to use the frequency list function, with the list file test0000.lst .

3.3 Set the measurement parameters

In this section, functions such as frequency range, number of measurement points, number of averages and also the measurement bandwidth for the NFA will be set and examined. In this example a frequency list file will be loaded. The frequency list file will set only a number of specific points at which to make measurements.

Example pseudocode to set the parameters for the narrowband noise figure measurement:

```
1    function SetMeasParams ( )
2    SendCommand (“*RST”)
3    SendCommand (“*CLS”)
4    SendCommand (“:MMEM:LOAD:
        ENR MEASUREMENT,’c:source01.enr’”)
5    SendCommand (“:MMEM:LOAD:
        FREQUENCY,’c:test0000.lst”)
6    SendCommand (“:INITIATE:CONTINUOUS:ALL OFF”)
7    SendCommand (“:SENSE:FREQUENCY:MODE LIST”)
8    SendCommand (“:SENSE:AVERAGE:STATE ON”)
9    SendCommand (“:SENSE:AVERAGE:COUNT 15”)
10   SendCommand (“:SENSE:BANDWIDTH 400000”)
11   end function
```

Example pseudocode to set the parameters for the narrowband noise figure measurement: code comments

Line 1 opens the function.

Line 2 puts the NFA into a factory-preset condition.

Line 3 clears the status byte by emptying the error cue and clearing all the bits in all the event registers.

Line 4 sets the current ENR file to “source01.enr”.

Line 5 loads the frequency list file “test0000.lst into the NFA as the current frequency list file.

Line 6 gets the NFA ready to make one complete frequency sweep and then stop.

Line 7 sets the NFA to only calibrate and then measure at the frequencies specified in the list file “test0000.lst”.

Line 8 sets the NFA Point averaging function to on.

Line 9 sets the number of averaging points to 15.

Line 10 sets the NFA to have a measurement bandwidth of 400 KHz. Line 11 closes the function.

3.4 Perform a user calibration

This section requires the user to physically connect the noise source to the front of the NFA to enable the calibration procedure to be completed. The calibration procedure removes the measurement systems own noise figure, often called second stage noise contribution, and the displayed results are shown in a corrected form. The results for both corrected and uncorrected measurements can be retrieved from the NFA.

Connect up the NFA and the Noise source as shown in the diagram below.

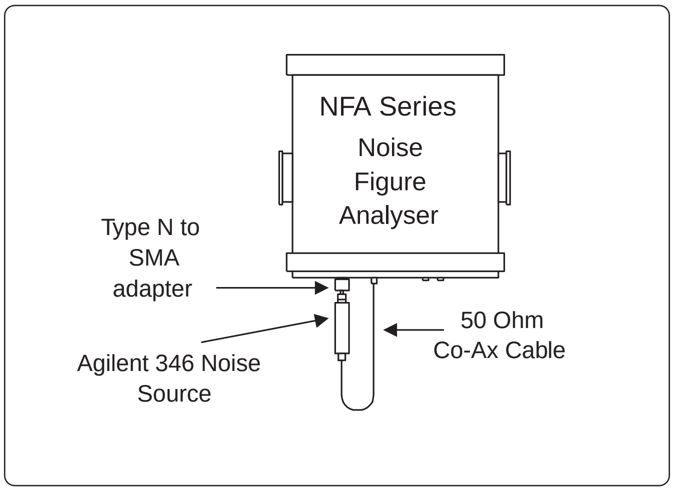


Figure 4. Calibration setup

Use a 50Ω BNC cable to connect the NFA to the 28V Noise source drive on the front of the NFA and an N type to SMA adapter, to connect the noise source to the input of the NFA.

Example pseudocode for calibrating the NFA:

```
1  function UserCalibration ( )
2  SendCommand (“:SENSE:CORRECTION:COLLECT:
       ACQUIRE STANDARD”)
3  SendCommand (“*WAI”)
4  end function
```

Example pseudocode for calibrating the NFA: code comments

Line 1 opens the function.

Line 2 starts collection of user calibration data.

Line 3 forces the NFA to wait until it finishes calibrating before processing the next command.

Line 4 ends the function.

3.5 Making basic noise figure and gain measurements on a filter

After the calibration is complete, follow the connection diagram below to enable the noise figure measurement to be made.

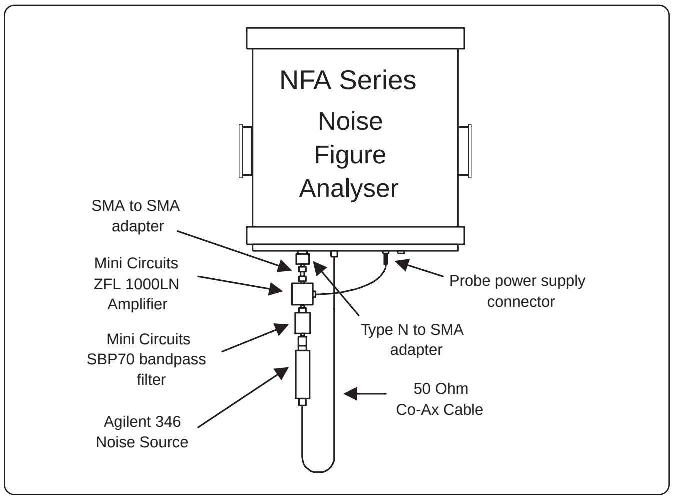


Figure 5. Basic noise figure and gain measurement connection diagram

Example pseudocode for making a noise figure measurement on a filter:

```
1 function Measurement ( )
2 SendCommand (“INITIATE:IMMEDIATE”)
3 SendCommand (“*WAI”)
4 end function
```

Example pseudocode for making a noise figure measurement on a filter: code comments

Line 1 opens the function.

Line 2 the measurement has been selected and is waiting, this command causes the system to come out of idle and starts making the measurement.

Line 3 waits to continue.

Line 4 ends the function.

3.6 Retrieving some example results

As mentioned previously, the NFA allows retrieval of multiple results after a single measurement. In this example the results will be retrieved for both noise figure and gain, specifically the maximum gain and minimum noise figure plus the delta between both noise figure and gain at both of the band edges of the filter.

Example pseudocode for retrieving some example results:

```
1 SendQuery (“FETCH:ARRAY:CORRECTED:
    NFIGURE?”, corrNfig)
2 SendQuery (“FETCH:ARRAY:CORRECTED:GAIN?”, corrGain)
3 SendQuery (“TRACE:DATA:CORRECTED:AMPLITUDE:
    VALUE? NFIGURE,70MHZ”, ampl1GHz)
4 SendQuery (“TRACE:DATA:CORRECTED:AMPLITUDE:
    MAXIMUM? GAIN”, maxGain)
5 SendQuery (“TRACE:DATA:CORRECTED:AMPLITUDE:
    MINIMUM? NFIGURE”, minNfig)
6 SendQuery (“TRACE:DATA:CORRECTED:AMPLITUDE:
    DELTA?  NFIGURE, 54MHZ, 60MHZ”, deltaNFIGURE)
7 SendQuery (“TRACE:DATA:CORRECTED:AMPLITUDE:
    DELTA? GAIN, 54MHZ, 60MHZ”, deltaGain)
8 SendQuery (“TRACE:DATA:CORRECTED:AMPLITUDE:
    DELTA? NFIGURE,78MHZ, 84MHZ”, deltaNFIGURE)
9 SendQuery (“TRACE:DATA:CORRECTED:
    AMPLITUDE:DELTA?  GAIN, 78MHZ, 84MHZ”, deltaGain)
```

Example pseudocode for retrieving some example results: code comments

Line 1 retrieves the corrected sweep noise figure measurements.

Line 2 retrieves the corrected sweep of gain measurements.

Line 3 retrieves the corrected value of Noise figure at 70 MHz (arbitrary value of frequency).

Line 4 retrieves the value of maximum gain from the results.

Line 5 retrieves the minimum noise figure value from the results.

Line 6 retrieves the amplitude difference between the noise figure values at 54 MHz and 60 MHz.

Line 7 retrieves the amplitude difference between the gain values at 54 MHz and 60 MHz.

Line 8 retrieves the amplitude difference. between the noise figure values at 78 MHz and 84 MHz.

Line 9 retrieves the amplitude difference between the gain values at 78 MHz and 84 MHz.

4. Advanced noise figure and gain measurements using mixers

4.1 Using a mixer as a DUT

Description:

This section explains how to set-up an NFA to control a local oscillator, with a mixer being used as the DUT. This example is based around the Mini Circuits ZFM-4212 mixer.

The example characterizes the mixer itself from 3.5 to 4.2 GHz, where the LO is varied and the NFA has a fixed IF of 40 MHz. The measurement will also be DSB so the gain will be 3 dB higher and noise figure 3 dB lower than actual. Use a fictitious input loss of -3 dB at 290 K before the DUT to compensate for this produce more accurate results.

Practical considerations:

When specifying local oscillators care must be taken. Factors such as phase noise, spectral purity and noise floor of the local oscillator may affect noise figure measurements. Filtering may therefore be required on some models of signal generators to enable accurate noise figure measurements to be made.

There are some other LO considerations, which the NFA must take into account

1. Frequency limits —by default 10 MHz to 26.5 GHz is assumed
2. Settling time—by default 100 msec is assumed
3. GPIB address—by default 19 is assumed
4. Remote commands—default to SCPI

In this example it is assumed that the default settings are appropriate.

This measurement example shall be split into the following sections:

4.1.1 Set the measurement system parameters
4.1.2 Perform a user calibration
4.1.3 Mixer as a DUT noise figure measurement
4.1.4 Retrieving some example results

4.1.1 Set the measurement system parameters

Example pseudocode for mixer as a DUT

This section shows the example pseudocode to set the following system parameters: Measurement mode, Loss compensation, Local Oscillator and Sweep.

This example assumes that an ENR table called "Source01.ENR" is present on the NFA (see example 2.1). The loss compensation function is used in this example to make the noise figure measurement more accurate. Often the noise figure is measured in two sidebands, especially without any filtering present. The result is that the measurement will be approximately 3 dB too high, therefore a fictitious loss of -3 dB is added. This will reduce the measured noise figure parameter by this approximate level. See application note 57-2 or the advanced noise figure measurement techniques section found in the NFA users guide (part number N8972-90001) for more details on this type of measurement.

```
function SetMeasParams( )
SendCommand ("*RST")
SendCommand ("*CLS")
SendCommand (":MMEM:LOAD:
   ENR MEASUREMENT,'c:source01.enr'")
SendCommand (":INITIATE:CONTINUOUS:ALL OFF")
SendCommand  (":SENSE:CONFIGURE:MODE:
   DUT DOWNCONVERTER")
SendCommand (":SENSE:CONFIGURE:MODE:DUT:
   LOSCILLATOR VARIABLE")
SendCommand (":SENSE:CONFIGURE:MODE:
   DOWNCONVERTER:IF:FREQUENCY 40 MHZ")
SendCommand (":SYSTEM:CONFIGURE:LOSCILLATOR:
   CONTROL:STATUS ON")
SendCommand (":SYSTEM:CONFIGURE:
   LOSCILLATOR:TYPE SCPI")
SendCommand (":SYSTEM:CONFIGURE:LOSCILLATOR:
   PARAMETER:POWER:LEVEL 7")
SendCommand (":SENSE:CORRECTION:LOSS:
   BEFORE:VALUE -3")
SendCommand (":SENSE:CORRECTION:LOSS:
   BEFORE:STATE ON")
SendCommand (":SENSE:CORRECTION:
   TEMPERATURE:BEFORE 290")
SendCommand (":SENSE:FREQUENCY:START 3.5 GHZ")
SendCommand (":SENSE:FREQUENCY:STOP 4.2 GHZ")
end function
```

Example pseudocode for mixer as a DUT: code comments

Line 1 opens the function.

Line 2 puts the NFA into a factory-preset condition.

Line 3 clears the status byte by emptying the error cue and clearing all the bits in all the event registers.

Line 4 sets the current ENR file to "source01.enr".

Line 5 sets the NFA to make one complete frequency sweep and then stop.

Line 6 sets the device under test to be a device whose operation involves frequency downconversion.

Line 7 sets the NFA to know that it is going to be sweeping a local oscillator, i.e. the local oscillator is variable.

Line 8 sets the NFA to use a fixed IF of 40 MHz.

Line 9 turns the NFA control over the local oscillator to on.

Line 10 indicates to the NFA that the local oscillator uses the SCPI command set.

Line 11 sets the power level of the local oscillator to 7 dBm; this is the manufacturers power level recommendations for the Mini Circuits ZfM-4212 mixer in the Agilent K10 demo kit.

Line 12 sets the NFA to have a loss compensation value of -3 dB's before the DUT, due to making a double sideband measurement. If a loss is added before the DUT then the system shall reduce the resulting measurement made by 3 dB's to compensate.

Line 13 sets the NFA loss compensation function to on.

Line 14 Lets the NFA know the temperature at which the loss occurs, in this case 290 k (room temperature).

Line 15 sets the NFA to start sweeping the local oscillator from 3.5 GHz.

Line 16 sets the NFA to stop sweeping the local oscillator at 4.2 GHz.

Line 17 closes the function.

4.1.2 Perform a user calibration

This section requires the user to physically connect the noise source to the front of the NFA to enable the completion of the calibration procedure. The calibration procedure removes the measurement system's own noise figure, often called second stage noise contribution, and the displayed results are shown in a corrected form. The results for both corrected and uncorrected measurements can be retrieved from the NFA.

Connect up the NFA and the Noise source as shown in the diagram.

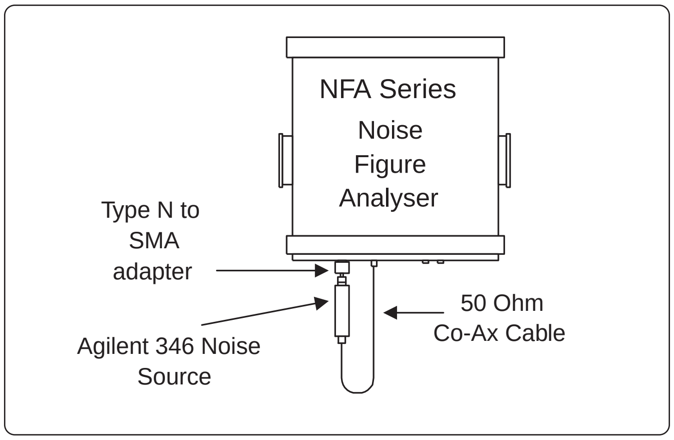


Figure 6. Calibration setup

Use a 50Ω BNC cable to connect the NFA to the 28V Noise source drive on the front of the NFA and an adapter to connect the noise source to the input of the NFA.

Example pseudocode for calibration:

```
1 function UserCalibration ( )
2 SendCommand (":SENSE:CORRECTION:COLLECT:
      ACQUIRE STANDARD")
3 SendCommand ("*WAI")
4 end function
```

Example pseudocode for calibrating the NFA: code comments

Line 1 opens the function.

Line 2 starts collection of user calibration data.

Line 3 waits to continue.

Line 4 ends the function.

4.1.3 Mixer as a DUT noise figure measurement

This section requires the user to physically connect the NFA, Noise source and signal generator as shown in the diagram below. Use a 50Ω BNC cable to connect the noise source to the 28V noise source drive on the front of the NFA and an adapter, (if applicable), to connect the noise source output to the input of the Mixer. The local oscillator should be connected to the Dedicated LO GPIB connector on the back of the NFA VIA a GPIB cable and the input to the mixer, from the LO, may also require an adapter.

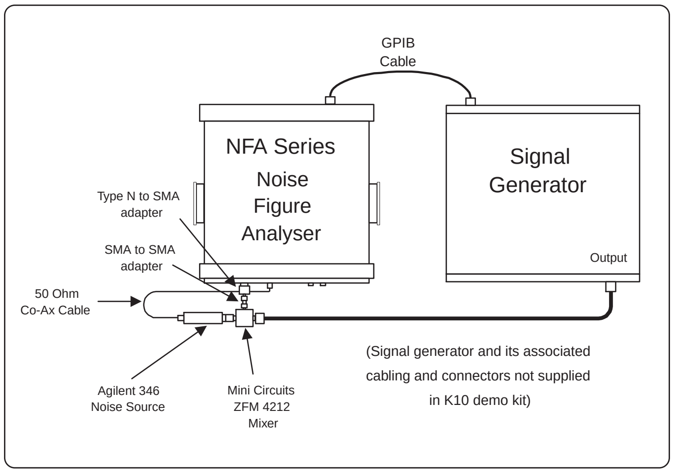

GPIB Cable
NFA Series Noise Figure Analyser
Signal Generator
Output
Type N to SMA adapter
SMA to SMA adapter
50 Ohm Co-Ax Cable
Agilent 346 Noise Source
Mini Circuits ZFM 4212 Mixer
(Signal generator and its associated cabling and connectors not supplied in K10 demo kit)

Figure 7. Post calibration connection diagram with mixer as the device under test

Example pseudocode for mixer as a DUT measurement:

```
1 function Measurement ( )
2 SendCommand (“INITIATE:IMMEDIATE”)
3 SendCommand (“*WAI”)
4 end function
```

Example pseudocode for mixer as a DUT measurement: code comments

Line 1 opens the function.

Line 2 the measurement has been selected and is waiting, this command causes the system to come out of idle and starts making the measurement.

Line 3 waits to continue.

Line 4 ends the function.

4.1.4 Retrieving some example results

Example pseudocode for retrieving the results from the NFA when using a mixer as the device under test:

The Pseudocode given here retrieves the following data: the whole corrected measurement sweep for noise figure and gain, the values over a complete sweep of uncorrected Phot measurements, the corrected amplitude value result for noise figure and gain at 1.0 GHz, the maximum gain throughout the sweep, the minimum noise figure throughout the sweep, the max peak to peak gain difference and the delta difference between gain measurements amplitudes at 50 MHz and 100 MHz.

```
1 SendQuery (“FETCH:ARRAY:CORRECTED:
    NFIGURE?”, corrNfig)
2 SendQuery (“FETCH:ARRAY:CORRECTED:
    GAIN?”, corrGain)
3 SendQuery (“FETCH:ARRAY:UNCORRECTED:
    PHOT?”, uncorrPhot)
4 SendQuery (“TRACE:DATA:CORRECTED:AMPLITUDE:
    VALUE? NFIGURE,1.0GHZ”, ampl1GHz)
5 SendQuery (“TRACE:DATA:CORRECTED:AMPLITUDE:
    MAXIMUM? GAIN”, maxGain)
6 SendQuery (“TRACE:DATA:CORRECTED:AMPLITUDE:
    MINIMUM? NFIGURE”, minNfig)
7 SendQuery (“TRACE:DATA:CORRECTED:AMPLITUDE:
    DELTA? GAIN, 50MHZ,100MHZ”, deltaGain)
```

Example pseudocode for retrieving the results from the NFA when using a mixer as the device under test: code comments

Line 1 retrieves the corrected sweep noise figure measurements.

Line 2 retrieves the corrected sweep of gain measurements.

Line 3 retrieves the uncorrected sweep of Phot measurements.

Line 4 retrieves the value of corrected Noise figure at 1.0 GHz.

Line 5 retrieves the value of maximum gain from the results.

Line 6 retrieves the minimum noise figure value from the results.

Line 7 retrieves the amplitude difference between the Gain values at 50 MHz and 100 MHz.

4.2 Using a mixer as part of the measurement system

This section can be split into 4 sub sections:

4.2.1 Set the measurement system parameters
4.2.2 User calibration when using a mixer as part of ther measurement system
4.2.3 Mixer as a DUT noise figure measurement
4.2.4 Retrieving some example results

4.2.1 Set the Measurement system parameters

Example pseudocode for using a mixer as part of the measurement system:

This section shows the example Pseudocode to set the following system parameters: Measurement mode, Loss compensation, Local Oscillator and Sweep.

This example assumes that an ENR table called “Source01.ENR” is present in the NFA

```
1    function SetMeasParams ( )
2    SendCommand (“*RST”)
3    SendCommand (“*CLS”)
4    SendCommand (“:MMEM:LOAD:
        ENR MEASUREMENT,’c:source01.enr’”)

5    SendCommand (“:INITIATE:CONTINUOUS:OFF”)
6    SendCommand (“:SYSTEM:CONFIGURE:
        LOSCILLATOR:CONTROL:STATUS ON”)
7    SendCommand (“:SYSTEM:CONFIGURE:
        LOSCILLATOR:TYPE SCPI”)
8    SendCommand (“:SYSTEM:CONFIGURE:LOSCILLATOR:
        PARAMETER:POWER:LEVEL 7”)
9    SendCommand (“:SYSTEM:CONFIGURE:LOSCILLATOR:
        PARAMETER:MIN:10MHZ”)
10   SendCommand (“:SYSTEM:CONFIGURE:LOSCILLATOR:
        PARAMETER:MAX:20GHZ”)
11   SendCommand (“:SENSE:CONFIGURE:MODE:
        DUT AMPLIFIER”)
12   SendCommand (“:SENSE:CONFIGURE:MODE:SYSTEM:
        DOWNCONVERTER ON”)
13   SendCommand (“:SENSE:CONFIGURE:MODE:SYSTEM:
        LOSCILLATOR:VARIABLE”)
14   SendCommand (“:SENSE:FEQUENCY:MODE:SWEEP”)
15   SendCommand (“:SENSE:CONFIGURE:MODE:SYSTEM:
        DOWNCONVERTER:IF:FREQUENCY 40 MHZ”)
16   SendCommand (“:SENSE:FREQUENCY:START 3.5 GHZ”)
17   SendCommand (“:SENSE:FREQUENCY:STOP 4.2 GHZ”)
18   SendCommand (“:SENSE:AVERAGE:STATE ON”)
19   SendCommand (“:SENSE:AVERAGE:COUNT 15”)
20   SendCommand (“:SENSE:BANDWIDTH 4000000”)
21   end function
```

Example pseudocode for using a mixer as part of the measurement system: code comments

Line 1 opens the function.

Line 2 puts the NFA into a factory-preset condition.

Line 3 clears the status byte by emptying the error cue and clearing all the bits in all the event registers.

Line 4 sets the current ENR file to “source01.enr”.

Line 5 sets the NFA to make one complete frequency sweep and then stop.

Line 6 turns the NFA control over the local oscillator to on.

Line 7 indicates to the NFA that the local oscillator uses the SCPI command set.

Line 8 sets the power level of the local oscillator to 7 dBm.

Line 9 sets the min frequency of the controlled local oscillator at 10 MHz.

Line 10 sets the maximum frequency of the controlled local oscillator at 20 GHz.

Line 11 sets the device under test to be an amplifier.

Line 12 sets the NFA to control the local oscillator as part of the frequency downconverting measurement system.

Line 13 sets the NFA to know that there is going to be a variable local oscillator within the measurement system.

Line 14 sets the NFA to be using the Sweep frequency mode.

Line 15 sets the NFA to use a fixed IF of 40 MHz.

Line 16 sets the NFA to start measuring Noise figure of the DUT at 3.5 GHz.

Line 17 sets the NFA to stop measuring Noise figure when it gets to 4.2GHz.

Line 18 sets the NFA to have the Point averaging function on.

Line 19 sets the number of average points at 15.

Line 20 sets the measurement bandwidth of the NFA at 4 MHz.

Line 21 closes the function.

4.2.2 User calibration when using a mixer as part of the measurement system.

Connect the system as shown in the following figure. The calibration procedure removes the measurement systems own noise figure, often called second stage noise contribution, which in this case includes the mixer and the local oscillator. The displayed results are shown in a corrected form. The results for both corrected and uncorrected measurements can be retrieved from the NFA.

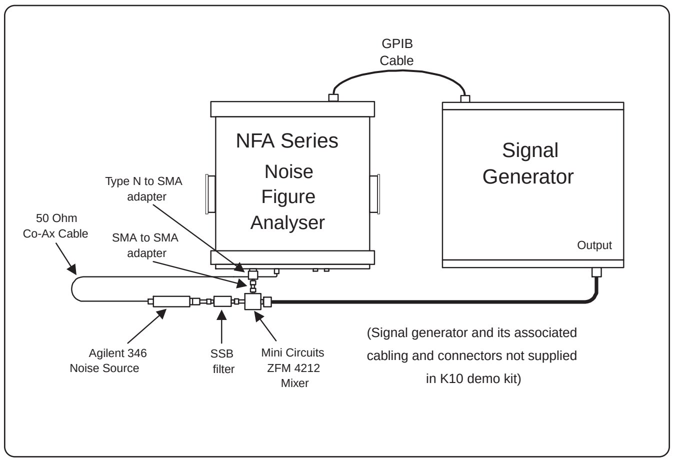

GPIB Cable
NFA Series Noise Figure Analyser
Signal Generator
Output
Type N to SMA adapter
50 Ohm Co-Ax Cable
SMA to SMA adapter
Agilent 346 Noise Source
SSB filter
Mini Circuits ZFM 4212 Mixer
(Signal generator and its associated cabling and connectors not supplied in K10 demo kit)

Figure 8. Calibration diagram with a mixer as part of the measurement system

The filter has been included in this example to make sure that there is only one sideband generated before being mixed into the capture range of the NFA. This can be dependant upon the DUT used and if there are two sidebands generated, then the final result could be out by a factor of 2x (3 dB too high).

Example pseudocode for calibrating the NFA:

```
1 function UserCalibration ( )
2 SendCommand (":SENSE:CORRECTION:COLLECT:
        ACQUIRE STANDARD")
3 SendCommand ("*WAI")
4 end function
```

Example pseudocode for calibrating the NFA: code comments

Line 1 opens the function.

Line 2 starts collection of user calibration data.

Line 3 waits to continue.

Line 4 ends the function.

4.2.3 Noise figure measurements with a mixer as part of the measurement system

This section demonstrates the connection diagram for a mixer to be used as part of the measurement system.

An SSB filter can be included in this measurement setup after the DUT to ensure that the noise figure measurement will be as accurate as possible. Without SSB filtering there may be a problem with the noise figure measurement being 3dB too high, due to the measurement of two sidebands, this is usually dependant upon the DUT operation. See the advanced measurement section in the NFA Users Guide (part number N8972A-90001) for further explanation.

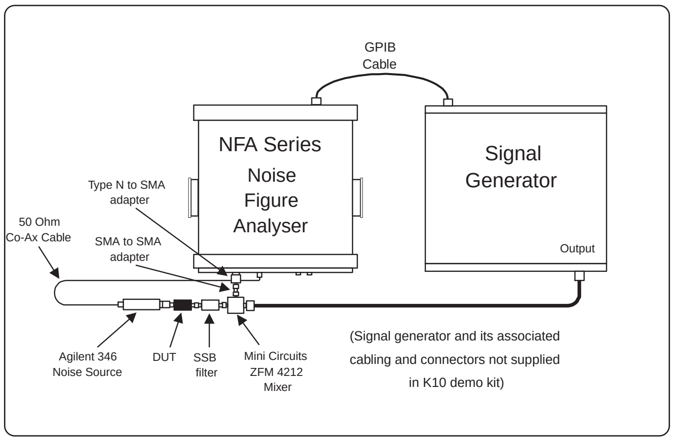

GPIB Cable
NFA Series Noise Figure Analyser
Signal Generator
Output
Type N to SMA adapter
50 Ohm Co-Ax Cable
SMA to SMA adapter
Agilent 346 Noise Source
DUT
SSB filter
Mini Circuits ZFM 4212 Mixer
(Signal generator and its associated cabling and connectors not supplied in K10 demo kit)

Figure 9. Measurement diagram with a mixer as part of the measurement system

Example pseudocode for making noise figure measurements using a mixer as part of the measurement system.

```
1 function UserCalibration ( )
2 SendCommand (":SENSE:CORRECTION:COLLECT:
        ACQUIRE STANDARD")
3 SendCommand ("*WAI")
4 end function
```

Example pseudocode for making noise figure measurements using a mixer as part of the measurement system: code comments

Line 1 opens the function.

Line 2 the measurement has been selected and is waiting, this command causes the system to come out of idle and starts making the measurement.

Line 3 waits to continue

Line 4 ends the function.

4.2.4 Retrieving some example results

Example pseudocode for retrieving the results from the NFA:

This section will examine how to retrieve the results for the entire corrected measurement sweep of noise figure and gain, the noise figure and gain specifically at 1.0 GHz, The maximum Gain throughout the sweep, the minimum noise figure and the delta between gain measurements at 50 MHz and 100 MHz.

```
1 SendQuery (“FETCH:ARRAY:CORRECTED:
    NFIGURE?”, corrNfig)
2 SendQuery (“FETCH:ARRAY:CORRECTED:
    GAIN?”, corrGain)
3 SendQuery (“TRACE:DATA:CORRECTED:AMPLITUDE:
    VALUE? NFIGURE,1.0GHZ”, ampl1GHz)
4 SendQuery (“TRACE:DATA:CORRECTED:AMPLITUDE:
    MAXIMUM? GAIN”, maxGain)
5 SendQuery (“TRACE:DATA:CORRECTED:AMPLITUDE:
    MINIMUM? NFIGURE”,minNfig)
6 SendQuery (“TRACE:DATA:CORRECTED:AMPLITUDE:
    DELTA? GAIN, 50MHZ, 100MHZ”, deltaGain)
```

Example pseudocode for retrieving the results from the NFA: code comments

Line 1 retrieves the corrected swept noise figure measurements.

Line 2 retrieves the corrected swept values of gain measurements.

Line 3 retrieves the value of corrected noise figure at 1.0 GHz.

Line 4 retrieves the value of maximum gain from the results.

Line 5 retrieves the minimum noise figure value from the results.

Line 6 retrieves the amplitude difference between the gain values at 50 MHz and 100 MHz.

5. Appendix

Minimal pseudocode library

RemoteWrite (string command, Boolean terminate)

RemoteWrite is a function that takes a string of bytes and sends them to the NFA. If terminate, the second argument, is true then when the last character is sent EOI is asserted marking the end of the message. Setting terminate to false allows long sequences of data (e.g. files) to be split into records that are loaded into the instrument one at a time. The transmission is terminated by setting terminate true when sending the last record only.

RemoteRead (string reply)

RemoteRead reads a string of bytes from the NFA. It also takes care of the GPIB protocol. This command will time out after a reasonable time. This ability to Time out is necessary to recover from query failures. When a query fails the NFA places an error message in the error queue but does not output any data to satisfy the query. If RemoteRead did not time out then the controlling program would hang.

GetErrorNumber (string errorMessage, number errorNumber)

GetErrorNumber takes an error message, as returned by the 'SYSTEM:ERROR:NEXT?' query, and extract the error number as a number.

GetErrorText (string errorMessage, string errorText)

GetErrorText takes an error message and extracts the error text.

PrintError (string command, number errorNumber, string errorText)

PrintError formats an error message on the error output.

FileLength (string fileName)

FileLength returns the size in bytes of the named file on the local host.

ToString (number n)

ToString converts a number to its string representation.

StringLength (string s)
StringLength returns the length of the given string.

GetByte (string fileName, string byte)

GetByte gets a byte from the named file on the local host and returns it in string argument byte.

Pseudocode for sending a command

```
1 function SendCommand (string command)
2 RemoteWrite (command, true)
3 ReportErrors (command)
4 end function
```

Pseudocode for sending a command code comments:

SendCommand is a function that is used to send a command string to the NFA.

Line 1 shows the function definition, which takes a string argument. A string in this pseudocode is simply a sequence of byte values of known length. The byte values can be presented and displayed as ASCII text.

Line 2 uses the pseudo code library call described earlier to send the command to the NFA. Note that SendCommand assumes that the whole command is passed to it because it terminates the message.

Line 3 calls a function that displays any errors associated with the execution of the command. ReportErrors is detailed on a later slide.

Pseudocode for sending a query

```
1 function SendQuery (string query, string response)
2  RemoteWrite (query, true)
3 RemoteRead (response)
4 ReportErrors (query)
5 end function
```

Pseudocode for sending a Query code comments:

Line 1 shows the function definition. It differs from the SendCommand only in that it takes a second argument, which is used to return the response to the query.

Line 2 is as for the previous example except that string is a SCPI query.

Line 3 reads the response from the NFA. This command must timeout if it does not receive a response within a reasonable time. Note that some of the NFA queries can take a very long time and so it is good to be able to tune the timeout to the query. I/O libraries such as SICL have this ability.

Line 4 reports any errors associated with the query.

The error cue

When the instrument detects an error condition, it places an error message in an error queue. The remote error queue is 30 entries deep.

Error messages have a signed error number followed by some error text in double quotes.

Negative error numbers are for predefined SCPI errors e.g. error -350,“Queue overflow” which is issued if an error occurs when the error queue is already full. Positive errors are instrument specific.

The query used to get the head of the error queue is “SYSTEM:ERROR:NEXT?”. It can only retrieve one error at a time.

The special error message +0,“No error” indicates that the error queue is empty. It is possible to query the error queue often, when it is empty the result is +0, “No error”.

A single command or query can generate more than one error message. For this reason it is best to drain the error queue after each command or query. If not, there is a danger of loosing track of what commands caused the particular errors.

Errors can occur that are not directly related to the last command issued. To determine if the command generated an error refer to status information. Status information will report if a different type of error has occurred. However, if the status information indicates there are different types of error in the error queue, it is impossible to determine which of the errors was caused by the last command unless it is obvious from the error itself.

It is important that the read routine can time out to avoid hanging the program.

Pseudocode for querying the error cue

```
1 function QueryError (number errorNumber,
  string errorText)
2 string errorMessage = ‘-999,”QueryError failure!”’
3 RemoteWrite (‘:SYSTEM:ERROR:NEXT?’, true)
4 RemoteRead (errorMessage)
5 GetErrorNumber (errorMessage,errorNumber)
6 GetErrorText (errorMessage,errorText)
7 end function
```

Querying the error cue code comments:

Function QueryError gets the error at the head of the error queue.

Line 1 shows that it takes two arguments used to return the error number and text to the enclosing scope.

Line 2 defines a string variable used to hold the error message read from the NFA. It is initialized to an error message that is not generated by the NFA but that is used to indicate that the program can’t access the NFA’s error queue.

Line 3 sends the query that causes the NFA to output the error message for the error at the head of the error queue. If there is no error then it outputs the special message ‘+0,“No error”’.

Line 4 gets the error message ouput by the NFA. If this command times out then string errorMessage will not be updated and will therefor contain the special case message ‘-999,” Error queue failure!”’.

Lines 5 and 6 fill in QueryError’s arguments by extracting the information from the error message.

There are as many schemes for handling errors in SCPI as there are programmers writing SCPI programs. This following is a simple example of something that can be done to handle errors in SCPI. It does not carry error information into the outer scope of the controlling program; however, it can easily be extended to perform this function.

Pseudocode for reporting errors from the error cue

```
1   function ReportErrors (string command)
2   number errorNumber
3   string errorText
4   loop
5   QueryError (errorNumber,errorText)
6   if errorNumber equals 0 exit loop
7   PrintError (command,errorNumber,errorText)
8   if errorNumber equals -999 exit loop
9   end loop
10  end function
```

Reporting errors from the error cue code comments:

ReportErrors reports any errors that are on the error queue, draining the queue in the process.

Line 1 shows that ReportErrors takes a single argument, the last command (or query) executed. As mentioned earlier, errors can be placed in the error queue that are not in direct response to a command or query but because of some other event within the NFA. In this example it is assumed that all errors are a result of the previous command.

Lines 2 and 3 declare local variables used to hold the error number and error text from the error message returned by the NFA.

Line 4 marks the start of a loop that extends to line 9.

Line 5—function QueryError gets the details of the next error from the error queue.

Line 6 exits the loop (and therefor the function) if the error number is zero (i.e. there was no error).

Line 7 prints the details of the error.

Line 8 is a special case test. It stops the code getting caught in an infinite loop caused when the error query itself fails.

Line 9 ends loop.

Agilent Technologies' Test and Measurement Support, Services, and Assistance

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

Our Promise

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

Your Advantage

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

For more assistance with your test and measurement needs or to find your local Agilent office go to:

www.agilent.com/find/assist

Agilent Technologies Literature Part Number: 5968-9498E

Revision: Release 1.0 (11/01/00)

Agilent Technologies
Innovating the HP Way