LA19-13-02
3 GHz Vector Network Analyser

—

Programming Manual
Issue 2.0, March 2007

Document history	
Issue and date	
1.0, July 06	Introduced
2.0, March 07	Added: Non-insertable DUT calibration mode and impedance conversion facility. Modified: SetKit, MeasCal, GetInfo Added: SetSysZo, ZConversion

LA Techniques Ltd

The Works, Station Road
Claygate, Surrey KT10 9DH
VAT no. GB 689 4720 79
Registered in England No: 3356289 Registered Office as above

Tel: 01372 466040
Fax: 01372 466688
E-mail: info@latechniques.com
Web site: www.latechniques.com

Blank Page

CONTENTS

1. Introduction

This manual describes the functionality of the DLL software (ActiveX library) available to support the LA19-13-02 Vector Network Analyser. The main intention of the software is to provide support for the control of the VNA from specialist test and measurement programs like Agilent's VEE and National Instrument's LabView. Errors accepted.

Labview and VEE are registered trademarks of National Instruments Inc., and Agilent Inc., respectively.

2. Availability and Installation

The library (VNAControl2.dll) has been shipped with the main user interface control program (VNA_UI2) starting from release version 1.4 (July 2006). Installation of the VNA_UI2 program will automatically install and register VNAControl2.dll on the host PC.

It is recommended that the user always first installs VNA_UI2 since this will install and register all the necessary libraries required. In the case where the user is looking to update the VNAControl2.dll, the following procedure may be followed.

To manually register the DLL file, use the following steps.

(i) Copy the file VNAControl2.dll to a directory of choice (e.g. ../system32)
(ii) From the Start > Run type command to bring up a DOS window
(iii) Navigate to the directory of choice
(iv) Register the VNAControl2 file by typing **Regsvr32 VNAControl2.dll**

3. RS232 Interface

Control of the LA19-13-02 is through its serial port. The settings are as follows.

Baud Rate:	115.2 kb/s
# of bits:	8
# of stop bits:	1
Parity:	off
MSComm InBuffer:	25000
MSComm OutBuffer:	25000
Valid ports:	1 to 16
Handshake	RTS / CTS

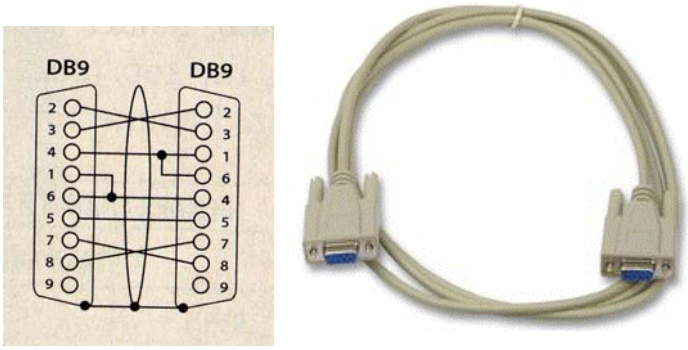


Fig. 2.1. RS232 Null Modem Cable and Connector wiring

Control using the USB port is also possible by using the optional USB to RS232 adaptor. This approach is recommended, particularly with older PCs where otherwise high speed data communications through the serial port may prove unreliable.

Whenever the USB adaptor is used, ensure that the allocated port lies in the range 1 to 16. This can be verified / modified as follows.

Control Panel > Performance and Maintenance
System > Hardware tab
Device Manager > Expand Ports

If the USB serial port number is greater than 16 (e.g. COM18) then proceed as follows:

Right click on USB Serial Port (select Properties)
Click on Port Settings tab
Click on Advanced
Select port number less than 17 from the box at the top ('Port number')
Click OK and exit

4. Command Summary

Section	Command	Description	Input	Output
6.6	AppCal	Function, apply the calibration	CalType	String: “OK” or “Error”
8.4	AppMemMath	Function, apply vector math to the current measurement	Para, Func	String: “OK” or “Error”
13.4	DLLVer	Function, query the current DLL version	None	String: <dll version>
6.9	DoP1dBCal	Function, perform a P1dB calibration	Loss1,Loss2,Ftest	String: “OK” or “Error”
6.10	DoAMPMCal	Function, perform an AM to PM calibration	Loss1,Loss2,Ftest	String: “OK” or “Error”
7.3	DoP1dBMeas	Function, measure P1dB	None	String: <P1dB value> or <error message>
7.4	DoAMPMMeas	Function, measure AM to PM	Ptest	String: <AM PM value> or <error message>
5.1	FND	Function, find instrument	None	Integer, serial port
9.3	FndPt	Function, read data (marker function)	Freq, Para, MeasType, Func	String: <freq, pk or min value, bandwidth, freq index> or “Error”
9.1	GetData	Function, read measured data	Para, MeasType, Pnt	String: <freq,value> or “Error”
10.1	GetInfo	Function, read instrument status	Para	String: <value>, “Error”
9.2	GetMem	Function, read memory data	Para, MeasType, Pnt	String:<freq, value> or “Error”
13.1	InitVar	Function, initialise all variables	None	String: “OK”
14.1	InsDiag	Function, perform diagnostic tests	None	None (displays window)
12.1	LoadCal	Function, load a calibration and status from disk	FileName or “?”	String: “OK” or “Error”
6.5	MeasCal	Function, measure a calibration standard	CalType, Standard	String: “OK” or “Error”
7.2	Measure	Function, perform a single sweep measurement	Para	String: “OK” or “Error”
5.2	ResetErr	Routine, resets communication error flag	None	None
11.3	SaveCal	Function, save the current calibration and status to disk	FileName or “?”	String: “OK” or “Error”
11.1	SaveKit	Function, calibration kit [Note: Thru data only saved if kit is on port 1]	FileName, Port	String: “OK” or “Error”
8.3	SaveToMem	Function, save measured data to memory	Para	String: “OK” or “Error”
6.8	SelectAMPM	Routine, measure AM to PM	None	None (displays window)
6.2	SelectCal	Routine, calibrate the instrument	“?” or FileName	String: “OK” or “Error” (displays window)
6.1	SelectKit	Routine, create, load or save a calibration kit	“?” or FileName, Port	String: “OK” or “Error” (displays window)
6.7	SelectP1dB	Routine, measure P1dB	None	None (displays window)
11.2	SelectSaveMeas	Routine, save measured data to disk	None	None (displays window)
13.2	SelectSigGen	Routine, set up the instrument as a synthesised signal generator	None	None (displays window)
8.1	SetEnhance	Function, set measurement enhancement	Para, k	String: “OK” or “Error”
6.3	SetFreqPlan	Function, set the frequency plan	Fstart, Fstep, I, P	String: “OK” or “Error”
6.2	SetKit	Function, create a calibration kit	Name, Ksex, Koff, CF0,CF1,CF3,Port, MData, FileName1, Tdata, FileName2	String: “OK” or “Error”
9.4	SetLimits	Function, set pass / fail limits	Segment, Para, FreqLow, FreqHigh, ValMin,ValMax>	String: “Error”, “OK” or data string

Command Summary (continued)

Section	Command	Description	Input	Output
13.3	SetSigGen	Function set up the instrument as a synthesised signal generator	Freq, P	String: “OK” or “Error”
8.2	SetRef	Function, set measurement reference plane	Para, k	String: “OK”, “Error” or “No cal”
6.11	SetRXQ	Function, set quiescent point of the receiver	None	String: “OK” or “Error”
9.5	TestLimits	Function, used to do a pass / fail test on the current measured data against set pass / fail limits	<Para>, <FreqLow>, <FreqHigh>,< MeasType >	String: “Error”, “Pass”, or string data with failed data
8.6	ZConversion	Function, converts a set of s-parameters from one base impedance to another	<ZoOld>,<ZoNew>, <S11r>,<S11i>,<S21r>, <S21i>,<S12r>,<S12i>, <S22r>,<S22i>	String: data string or “Error”
8.5	SetSysZo	Function, used to set the system impedance to a value other than 50Ω	<SysZo>, <ExMatch>	String: “OK” or “Error”

5. Communications

5.1 Discover instrument

Internal DLL declaration:
Public Function **FND**() As Integer

Typical usage:
This function is used to find the instrument connected to the PC.
<variable> = FND

Action:
Searches all available serial ports to find the instrument. If the instrument is found, factory data is read from the instrument’s EEPROM. This takes in the region of 12 seconds to complete. The instruments dc offset voltage is read at this time too. No parameter is passed in the call.

Parameter passed	Description	Value
None		

Returns:
Serial port number where the instrument is detected (1 to 16) or 0 if no instrument is found.

Return values	Type	Executed correctly	Error
<0 to 16>	Integer	<1 to 16>	<0>

5.2 Reset serial communications error flag

Internal DLL declaration:
Public Sub **ResetErr**()

Typical usage:
This routine is used to reset the error flag.

Action:
Resets the serial communications error flag. No parameters are passed in the call.

Parameter passed	Description	Value
None		

Returns:
Nothing

6. Calibration

Note! The dc offset voltage dcOff must be read before any calibration is done. Further, it is strongly advised that a call is made (GetInfo("dcOff")) every one or two minutes interval to ensure optimum performance.

6.1 Create kit (displays a dialogue form)

Internal DLL declaration:
Public Function **SelectKit**(ByVal FileName As String, ByVal port As Integer) as String

Typical usage:
This routine is used to create or load a calibration kit.
<variable> = SelectKit(<FileName>, <port>)

Action:
If the user passes a "?" instead of a file name, then this function displays a dialogue window which allows the user to select (load from disk) or create and save a calibration kit. On the other hand, if a valid full path file name is passed (of the calibration kit to be loaded) together with either "1" for port 1, or "2" for port 2, then the kit is loaded and applied without a dialogue form.

Parameter passed	Description	Comments
Filename, Port or "?"	Full path file name (of calibration kit), test port ('1' or '2') or "?" string	If a "?" is passed a dialogue form is displayed

Returns:

Return values	Type	Executed correctly	Error
"OK" or "Error"	String	"OK"	"Error"

6.2 Create kit

Internal DLL declaration:
Public Function **SetKit**(ByVal name As String, ByVal Ksex As String, ByVal Koff As Variant, ByVal CF0 As Variant, ByVal CF1 As Variant, ByVal CF2 As Variant, ByVal CF3 As Variant, ByVal port As Integer, ByVal MData As String, ByVal FileName1 As String, ByVal Tdata as String, ByVal FileName2 as String) As String

Typical usage:
<variable> = SetKit(<name>, <Ksex>, <Koff>,<CF0>, <CF1>,<CF 2>, <CF3>, <port>, <MData>, <FileName1>, <Tdata>, <FileName2>)

Action:
Creates a calibration kit using the parameters passed. See table below for the type and format of data required. Note that Thru data is only loaded if the kit is to be associated with Port 2 of the instrument.

Parameter passed	Description	Value	Unit
Name	Cal kit name		
Ksex	Cal kit connector sex	"m" or "f"	
Koff	Cal kit's reference plane		metre
CF0	Open standard capacitance coefficient		Farad
CF1	Open standard capacitance coefficient		Farad
CF2	Open standard capacitance coefficient		Farad
CF3	Open standard capacitance coefficient		
Port	Port (1 or 2) to which kit applies	1 or 2	
MData	Flag indicates if load data is available	"True" or "False"	
FileName1	Path and file name of Load data file		
TData	Flag indicates if Thru data is available	"True" or "False"	
FileName2	Path and file name of Thru data file		

Returns:

Return values	Type	Executed correctly	Error
"OK" or "Error"	String	"OK"	"Error"

Important: Through adaptor data is only loaded and saved when the kit is associated with **Port 1** of the instrument. A kit with Thru data may be loaded to Port 2 but the Thru data will be ignored, therefore, if the kit is subsequently re-saved the Thru data will be lost.

6.3 Set frequency plan

Internal DLL declaration:
Public Function **SetFreqPlan**(ByVal Fstart As Variant, ByVal Fstep As Variant, ByVal I As Variant, ByVal P As Variant) As String

Typical usage:
This routine is used to set the frequency plan of the instrument. Typically, the format used is as follows:

<variable> = SetFreqPlan(<start frequency>, <frequency step>, <# of points>,<power level>)

Action:
Sets the frequency sweep.

Parameter passed	Units	Min Value	Max Value	Step size
Start Frequency	MHz	3	3079.995	
Frequency Step	MHz	0.0001 (100Hz)	60.33	
Number of points (I)		51,101, 201, 401, 801		
Power Level	dBm	-20	0	1

The sweep is set by passing the start frequency and step size. The step size is calculated by taking the start frequency from the stop frequency and dividing by the (number of points – 1). The stop frequency must not be greater than 3080MHz

$F_{step} = (F_{stop} - F_{start})/(N_{points} - 1)$ $F_{stop} <= 3080$

Returns:

Return values	Type	Executed correctly	Error
"OK" or "Error"	String	"OK"	"Error"

Note: If a valid calibration exists, setting a new frequency plan (different from that used during calibration) will cause a new set of (interpolated) error terms to be generated so that a new calibration need not be carried out. Note that this process will delete any measurements saved to memory. Note further that in order to obtain the best instrument capability a new calibration should be performed.

6.4 Calibrate and set frequency plan (displays a dialogue form)

Internal DLL declaration:
Public Function **SelectCal**(ByVal FileName as string) as String

Typical usage:
This routine is used to set the frequency plan and calibrate the instrument.

Action:
If a "?" string is passed then this function displays a dialogue form which allows the user to set the frequency plan (start, stop and number of frequency points) and complete calibration of the instrument. If a valid full path file name holding a calibration is passed, then the calibration will be loaded without displaying a form.

Parameter passed	Description	Value
Filename or "?"	Full path file name (of calibration file) or "?"	If a "?" is passed a dialogue form is displayed

Returns:

Return values	Type	Executed correctly	Error
"OK" or "Error"	String	"OK"	"Error"

Note: If a valid calibration exists, setting a new frequency plan (different from that used during calibration) only will cause a new set of (interpolated) error terms to be generated so that a new calibration need not be carried out. Note that this process will delete any measurements saved to memory. Note further that in order to obtain the best instrument capability a new calibration should be performed whenever the sweep parameters change.

6.5 Calibrate (measure calibration standard)

Internal DLL declaration:
Public Function **MeasCal**(ByVal CalType As Variant, ByVal Standard As Variant) As String

Typical Usage:
<variable> = MeasCal(<CalType>, <Standard>)

Action:
Measures a calibration standard (used during the calibration procedure). The parameters passed are described below.

Parameter passed	Description	Value
CalType	Calibration type	"S11", "S21", "S11+S21", "All", "Alln"
Standard	Calibration standard	"Open", "Short", "Load", "Thru" or "Isolation"

Note! Calibration types are as follows:

S11: 1 port correction
S21: frequency response (with optional isolation) correction
S11+S21: 1 port correction, with frequency response and isolation correction and source match correction
All: Full 12-term correction for insertable device (2 cal kits needed)
Alln: Full 12-term correction for non-insertable device (only 1 cal kit)

Returns:

Return values	Type	Executed correctly	Not executed
"OK" or "Error"	String	"OK"	"Error"

6.6 Apply calibration

Internal DLL declaration:
Public Function **AppCal**(ByVal CalType As Variant) As String

Typical Usage:
<variable> = AppCal(<CalType >)

Action:
Applies the calibration (used after all calibration standards, i.e. short, open and load) have been measured). The parameter passed is described in the table below.

Parameter passed	Description	Value
CalType	Calibration type	"S11", "S21", "S11+S21", "All", "Alln"

Returns:

Return values	Type	Executed correctly	Not executed
"OK" or "Error"	String	"OK"	"Error"

Note! Only use with the MeasCal commands. Not after using SelectCal or LoadCal

6.7 P1dB Calibration and Measurement (display a dialogue form)

Internal DLL declaration:
Public Sub **SelectP1dB**()

Typical Usage:
This function is used to perform P1dB calibration and measurements. Typically implemented as follows:

Call SelectP1dB or
SelectP1dB

Action:
A dialogue form is displayed that allows the user to carry out interactively P1dB measurements. No parameters are passed.

Parameter passed	Description	Value
None		

Returns:
Nothing

6.8 AM-PM Calibration and Measurement (displays a dialogue form)

Internal DLL declaration:
Public Sub **SelectAMPM**()

Typical Usage:
Used to perform an AM to PM calibration. Typically implemented as follows:

Call SelectAMPM or
SelectAMPM

Action:
A dialogue form is displayed that allows the user to carry out interactively AM to PM measurements. No parameters are passed.

Parameter passed	Description	Value
None		

Returns:
Nothing

6.9 P1dB Calibration

Internal DLL declaration:
Public Function **DoP1dBCal**(ByVal Loss1 As Variant, ByVal Loss2 As Variant, ByVal Ftest As Variant) As String

Typical Usage:
Used to perform a P1dB calibration. Typically implement as follows:
<variable> = DoP1dBCal(<Loss1>, <Loss2>, <Ftest>)

Action:
Performs a P1dB calibration. This is used prior to carrying out a P1dB measurement. No dialogue form is displayed. The parameters passed are described in the table below.

Note! P_{1dB} Calibration should be carried out without the input or output attenuators in place.

Parameter passed	Description	Value	Unit
Loss1	Input attenuation / loss		dB
Loss2	Output attenuation / loss		dB
Ftest	Frequency at which the Cal is done	3 to 3080	MHz

Returns:

Return values	Type	Executed correctly	Not executed correctly
"OK" or "Error"	String	"OK"	"Error"

6.10 AM-PM Calibration

Internal DLL declaration:
Public Function **DoAMPMCal**(ByVal Loss1 As Variant, ByVal Loss2 As Variant, ByVal Ftest As Variant) As String

Typical Usage:
Used to perform an AM to PM calibration. Typically implemented as follows:
<variable> = DoAMPMCal(<Loss1>, <Loss2>, <Ftest>)

Action:
Performs an AM to PM calibration at the specified frequency. The parameters passed are described in the table below.

Note! AM-PM Calibration should be carried out without the input or output attenuators in place.

Parameter passed	Description	Value	Unit
Loss1	Input attenuation / loss		dB
Loss2	Output attenuation / loss		dB
Ftest	Frequency at which the Cal is done	3 to 3080	MHz

Returns:

Return values	Type	Executed correctly	Not executed correctly
"OK" or "Error"	String	"OK"	"Error"

6.11 Set quiescent point of Receiver

Internal DLL declaration:
Public Function **SetRXQ**() As String

Typical Usage:

<variable> = SetRXQ

Action:
This function optimises the quiescent point of the instrument's receiver. This function is not normally necessary unless the ultimate capability of the instrument is required. If used, it must be called immediately before performing a measurement sweep. No parameters are passed.

Parameter passed	Description	Value
None		

Returns:

Return values	Type	Executed correctly	Not executed correctly
"OK" or "Error"	String	"OK"	"Error"

7. Measurements

Note! The dc offset voltage (dcOff) must be read before any calibration or measurement is done. It is strongly advised that a call is made (GetInfo("dcOff")) at every one or two minutes interval to ensure optimum performance.

7.1 Measure one sweep (S11, S21, S11+S21, or 'All' using current calibration)

Internal DLL declaration:
Public Function **Measure**(ByVal Para As Variant) As String

Typical Usage:
Executes a measurement sweep. It is typically called as follows:

<variable> = Measure(<para>)

Action:
Performs a single measurement sweep using the current calibration. The parameter passed is described in the table below.

Parameter passed	Description	Value
Para	Measurements to be made	"S11", "S21", "S11+S21", "All"

Note! If the current calibration is not for the parameter passed, then an error will be returned. For example, if "S21" is passed, then the current calibration must be either an S21, S11+S21 or 'All' calibration. Similarly, if "S12" is passed, then the current calibration must be an 'All' or 'Alln' calibration.

Returns:

Return values	Type	Executed correctly	Not executed correctly
"OK" or "Error"	String	"OK"	"Error"

7.2 Measure P1dB

Internal DLL declaration:
Public Function **DoP1dBMeas**() As String

Typical Usage:
Used to perform a P1dB measurement.

<variable> = DoP1dBMeas

Action:
Performs a P1dB measurement. Before this function is used, a P1dB calibration must be carried out at which time the losses at the input and output of the device under test can be set. No parameters are passed.

Parameter passed	Description	Value
None		

Returns:

Return values	Type	Executed correctly	Not executed correctly
P1dB value, "No cal" or "Error"	String	<P1dB Value>	"Error" or "No cal"

7.3 Measure AM-PM

Internal DLL declaration:
Public Function **DoAMPMMeas**(ByVal Ptest As Variant) As String

Typical Usage:
Used to carry out an AM to PM coefficient test.

<variable> = DoAMPMMeas(<Ptest>)

Action:
Performs a AM to PM measurement. Before using this function, an AM to PM calibration must be done at which time the losses at the input and output of the device under test can be set. A single parameter is passed as described in the table below.

Parameter passed	Description	Value	Unit
Ptest	Power level at which measurement reported	-20 to 0	dBm

Returns:

Return values	Type	Unit	Executed correctly	Not executed correctly
AM to PM value, "No cal" or "Error"	String	deg/dB	<AM to PM value>	"Error" or "No cal"

8. Signal Processing

8.1 Set enhancement parameters

Internal DLL declaration:
Public Function **SetEnhance**(ByVal Para As Variant, ByVal k As Variant) As String

Typical Usage:
Used to set measurement enhancements.
<variable> = SetEnhance(<Para>, <k>)

Action:
This function is used to set the various measurement enhancements available. Two parameters are passed ("Para" and "k") as described in the table below.

Note! Frequency Sweep (SetFreqPlan or SelectCal, etc) must be set before this command is used otherwise an error is generated. The set value applies at the next sweep.

Para	Description	k	Units
"Aver"	Sets number of averages	1 to 255	
"Smoo"	Sets the smoothing	0 to 10	%
"Dwel"	Sets dwell time for each measurement	0.5 or 2.5	ms
"Powr"	Sets power level	-20 to 0	dBm (1dB steps)
"Dem1"	Sets de-embedding network for port 1	File path and name	
"Dem2"	Sets de-embedding network for port 1	File path and name	
"ApplyDem"	Turns de-embedding on or off	"ON" or "OFF"	
"TDTime"	Sets the start time and stop time for time domain measurements	*See following table	ns
"TDP"		*See following table	

Para	Argument k	Examples	Notes
"TDTime"	<start time>, <stop time> in ns	"-5, 166" "0, 50"	Start time = -5 ns (min) Stop time = 332 ns (max) Maximum span = 166 ns
"TDP"	<dc termination type>, <resistance value (if applicable)>, <window type>, <window order>	"auto,rect" "res,50,kaiserb,6"	'Res' value only needed if termination type is 'Res'. 'window order' only needed if 'window type' is 'kaiserb' **Possible terminations:** "auto", "short", "open", or "res". **Possible window types:** "rect", "kaiserb", or "raisedcos"

Returns:

Return values	Type	Executed correctly	Not executed correctly
"OK" or "Error"	String	"OK"	"Error"

8.2 Set Reference Plane

Internal DLL declaration:
Public Function **SetRef**(ByVal Para As Variant, ByVal k As Variant) As String

Typical Usage:
This function is used to set the measurement reference.
<variable> = SetRef(<Para>, <k>)

Action:
Used to set the reference plane of subsequent measurements. A reference plane for each of the four s-parameters can be specified. Two parameters are passed as described in the table below.

Para	Description	k	Units
"S11", "S21", "S12" or "S22"	Measurement parameter to which the reference plane will apply	<value in the range -10 to +10>, "auto" or "?"	metres

The value of k passed to the function must lie within the range -10 to +10 metres. If k is assigned "auto" then the reference plane will automatically be assigned as follows.

Measurement Parameter	'Automatic' reference plane value assigned
S11 or S22	Distance to a short or an open circuit
S21 or S12	Distance required to produce a mean phase of 0^o

Note! Set the reference plane to 0 before using the 'auto' facility.

Returns:

Return values	Type	Executed correctly	Not executed correctly
"OK", "No Cal" or "Error"	String	"OK"	"Error", "No cal" or "Error"

Note! The reference plane is reset to zero after calibration. Therefore, if a value other than zero is required, ensure that it is set after calibration

8.3 Save measurement to memory

Internal DLL declaration:
Public Function **SaveToMem**(ByVal Para As Variant) As String

Typical Usage:
Used to stored measured data to memory.
<variable> = SaveToMem(<Para>)

Action:
This function is used to store measured values to memory. A single parameter is passed as shown in the table below.

Para	Description
"S11" or "S21"	Forward parameters
"S22" or "S12"	Reverse parameters

Returns:

Return values	Type	Executed correctly	Not executed correctly
"OK" or "Error"	String	"OK"	"Error"

8.4 Apply memory Math

Internal DLL declaration:
Public Function **AppMemMath**(ByVal Para As Variant, ByVal Func As Variant) As String

Typical Usage:
Used to apply vector maths to current measured data.
<variable> = AppMemMath(<Para>, <Func>)

Action:
This function is used to apply vector maths to the current measurement only. The parameters passed are described in the table below.

Parameter passed	Description	Values	Comments
Para	Measurement parameter	"S11", "S21", "S12" or "S22"	S22 and S12 are reverse measurements that require the mode to be set to 'reverse'
Func	Vector maths to apply	"/", "+", "-", "Restore"	Division, addition and subtraction. "Restore" restores the measurement data to that before any math was applied.

Note! If this function needs to be called a second time before carrying out another measurement, use the 'Restore' facility to restore the measurement data.

Returns:

Return values	Type	Executed correctly	Not executed correctly
"OK" or "Error"	String	"OK"	"Error"

8.5 Set System Impedance

Internal DLL declaration:
Public Function **SetSysZo**(ByVal SysZo As Integer, ByVal ExMatch As Variant) As String

Action:
This function is used to set the system impedance Zo. The parameters passed as described in the table below.

Parameter passed	Description	Values	Comments
SysZo	System impednace	10 to 200	Only real impedance values
ExMatch	Flag indicating if external impedance matching networks are being used	-	If True, it is assumed external components are performing the impedance conversion. If False, the software will perform mathematical conversion from 50Ω to the requested value.

Note: If external impedance matching networks are being used, then a suitable calibration kit should be used to calibrate. For example, to measure a 75Ω device, 75Ω to 50Ω matching pads could be used and calibration carried out using a 75Ω calibration kit. If no impedance matching networks are used, it is assumed that calibration is carried out using a 50Ω calibration kit. The software then mathematically converts the measurements to 75Ω. Please refer to the users' manual for further information. When applied, **data stored in memory is unaffected.**

Returns:

Return values	Type	Executed correctly	Not executed correctly
"OK" or "Error"	String	"OK"	"Error"

8.6 Impedance Conversion Utility

Internal DLL declaration:
Public Function **ZConversion** (ByVal ZoOld As Variant, ByVal ZoNew As Variant, ByVal A11R As Variant, A11I As Variant, A21R As Double, A21I As Variant, _
A12R As Variant, A12I As Variant, A22R As Double, A22I As Variant) As String

Action:
This function is used to convert normalised S-parameters to a different impedance. The parameters passed are described in the table below.

Parameter passed	Description	Values	Comments
Zold	Base impedance of s-parameters to be converted	10 to 200	Only real impedance values
ZoNew	Base impedance of converted s-parameters	10 to 200	Only real impedance values

A11R, A11I	Real and imaginary parts of S11	-	Normalised values
A21R, A21I	Real and imaginary parts of S21	-	Normalised values
A12R, A12I	Real and imaginary parts of S12	-	Normalised values
A22R, A22I	Real and imaginary parts of S22	-	Normalised values

Note: The function requires that a full set of s-parameters is passed, i.e. a total of 8 terms representing the real and imaginary parts of each parameter. If a full set is not available, then missing parameters must be given default values. Possible values for any parameter not available are 10^{-6}, j0.0. Please be aware that all s-parameters are interrelated so in some circumstances this assumption may not yield best results.

Returns:

Return values	Type	Executed correctly	Not executed correctly
<string value> or "Error"	String	"S11real, S11imag, S21real, S21imag, S12real, S12imag, S22real, S22imag"	"Error"

9. Get Processed Data

9.1 Get data

Internal DLL declaration:
Public Function **GetData**(ByVal Para As Variant, ByVal MeasType As Variant, ByVal Pnt As Variant) As String

Typical Usage:
Used to get measured data (one frequency point per call) in a specified format.
<variable> = GetData(<Para>, <MeasType>, <Pnt>)

Action:
This function is used to get measured data in a specified format. The parameters required to be passed are shown in the table below.

Parameter passed	Description	Values	Comments
Para	Measurement parameter	"S11", "S21", "S12" or "S22"	S22 and S12 are reverse measurements that require the mode to be set to 'reverse'
MeasType	Required data format	"real", "imag", "logmag", "phase", "swr", "gd", "td"	The available formats are: real, imaginary, log magnitude, phase, standing wave ratio, group delay, time domain
Pnt	Frequency index	<frequency index>	A value between 1 and the number of sweep points.

Returns:

Return values	Type	Executed correctly	Not executed correctly
<string value> or "Error"	String	"<frequency (Hz)>, <parameter value>"	"Error"

9.2 Get memory

Internal DLL declaration:
Public Function **GetMem**(ByVal Para As Variant, ByVal MeasType As Variant, ByVal Pnt As Variant) As String

Typical Usage:
Used to get memory data (one frequency point per call) in a specified format.

<variable> = GetMem(<Para>, <MeasType>, <Pnt>)

Action:
This function is used to get measured data in a specified format. The parameters required to be passed are shown in the table below.

Parameter passed	Description	Values	Comments
Para	Measurement parameter	“S11”, “S21”, “S12” or “S22”	S22 and S12 are reverse measurements that require the mode to be set to ‘reverse’
MeasType	Required data format	“real”, “imag”, “logmag”, “phase”, “swr”, “gd”, “td”	The available formats are: real, imaginary, log magnitude, phase, standing wave ratio, group delay, time domain
Pnt	Frequency index	<frequency index>	A value between 1 and the number of sweep points.

Returns:

Return values	Type	Executed correctly	Not executed correctly
<string value> or “Error”	String	“<frequency (Hz)>, <parameter value>”	“Error”

Note! Smoothing is not applied to returned memory data.

9.3 Find data point

Internal DLL declaration:
Public Function **FndPt**(ByVal Freq As Variant, ByVal Para As Variant, ByVal MeasType As Variant, ByVal Func As Variant) As String

Typical Usage:
Gets data in a specified format. Includes facility for finding the peak or minimum value and 3 or 6 dB bandwidth.

<variable> = FndPt(<freq>, <Para>, <MeasType>, <Func>)

Action:
This function is used to get measured data in a specified format. It can be considered as a marker function. It is similar to the ‘GetData’ except that instead of a frequency index, a frequency value (in MHz) is passed. In addition, a parameter can be specified to find the peak or minimum value as well as the 3 or 6 dB bandwidth points. Note that in both cases this will return the measurement points nearest the requested points. The parameters required to be passed are shown in the table below.

Parameter passed	Description	Values	Comments
Freq	Frequency in MHz	<frequency>	A value between 3 and 3080 (MHz).
Para	Measurement parameter	“S11”, “S21”, “S12” or “S22”	S22 and S12 are reverse measurements that require the mode to be set to ‘reverse’
MeasType	Required data format	“real”, “imag”, “logmag”, “phase”, “swr”, “gd”, “td”	The available formats are: real, imaginary, log magnitude, phase, standing wave ratio, group delay, time domain
Func	‘Marker’ function required	“normal”, “pk”, “min”, “-3dB”, “-6dB”, “+3dB”, “+6dB”	‘Normal’ returns the value at the specified frequency. ‘pk’ and ‘min’ find the peak and minimum values, respectively. The other values specified a bandwidth, such as “-3dB” refers to the -3 dB bandwidth.

Returns:

Return values	Type	Executed correctly	Not executed correctly
<string value> or "Error"	String	"<frequency (Hz)>, < (pk, min, etc) value>, <bandwidth value>, <frequency index>"	"Error"

Note! Frequency index is an integer representing the point in the sweep.

9.4 Set Pass / Fail Limits

Internal DLL declaration:
Public Function **SetLimits**(ByVal Segment As Integer, ByVal Para As String, ByVal FreqLow As Variant, ByVal FreqHigh As Variant, ByVal Min As Variant, ByVal Max As Variant) As String

Typical Usage:
Sets the minimum and maximum pass / fail limits. Up to 8 segments may be specified.

<variable> = SetLimits(, <Para>, <FreqLow>, <FreqHigh>,
<ValMin>,<ValMax>)

Action:
This function is used to set segments of the pass / fail limits. It is possible to specify up to 8 segments for each measurement parameter (S11, S21, S12 or S22). The parameters required to be passed are described in the table below.

Parameter passed	Description	Values	Comments
Segment	Segment identifier	1 to 8	
Para	Measurement parameter	"S11", "S21", "S12" or "S22"	S22 and S12 are reverse measurements that require the mode to be set to 'reverse'
FreqLow	Start frequency of segment in MHz	3 to 3080 or "?" or "Reset"	The start frequency of the segment must lie within the calibration frequency band. It must be less than the stop frequency. If FreqLow is set to a string = "?" then the set values of the segment will be returned. Similarly, if it is set to a string = "Reset" the segment is initialised.
FreqHigh	Stop frequency of segment in MHz	3 to 3080	The stop frequency of the segment must lie within the calibration frequency band. It must be greater than the start frequency.
ValMin	Minimum value limit of segment	-1E9 to 1E9	
ValMax	Maximum value limit of segment	-1E9 to 1E9	

Returns:

Return values	Type	Executed correctly	Not executed correctly
<string value> or "Error"	String	" OK" or if query, "<start freq (Hz)>, < stop freq (Hz) >, <min value>, <max value>"	"Error"

9.5 Pass / Fail Measurement

Internal DLL declaration:
Public Function **TestLimits**(ByVal Para As String, ByVal FreqLow As Variant, ByVal FreqHigh As Variant, ByVal MeasType As Variant) As String

Typical Usage:
Compares current measurement against the set pass / fail limits. All 8 segments are checked.

<variable> = TestLimits(<Para>, <FreqLow>, <FreqHigh>,< MeasType >)

Action:
This function is used to test against the pass / fail limits. The parameter (S11, S21, S12 or S22) needs to be specified together with type of measurement. The parameters required to be passed are described in the table below.

Parameter passed	Description	Values	Comments
Para	Measurement parameter	"S11", "S21", "S12" or "S22"	S22 and S12 are reverse measurements that require the mode to be set to 'reverse'
FreqLow	Start frequency of test band in MHz	3 to 3080 or 0	The start frequency of the test band in MHz. If set to 0, then the current calibration's entire band is used to perform the pass / fail test.
FreqHigh	Stop frequency of test band in MHz	3 to 3080	The stop frequency of the test band in MHz.
MeasType	Required data format	"real", "imag", "logmag", "phase", "swr", "gd", "td"	The available formats are: real, imaginary, log magnitude, phase, standing wave ratio, group delay, time domain

Returns:

Return values	Type	Executed correctly	Not executed correctly
<string value> or "Error"	String	" Pass" or if test fail, "<fail freq (Hz)>, < "Max (or Min) limit =" ValMin (or ValMax)>, <"Actual =">, <Measured value>"	"Error"

10. Get Info

10.1 Get instrument / cal information

Internal DLL declaration:
Public Function **GetInfo**(ByVal Para As Variant) As String

Typical Usage:
This function is used to get data related to the instrument status. A single parameter is passed to the function as described in the table below.

<variable> = GetInfo(<Para >)

Para	Description	Valid return
'"AverSet"	Number of averages set	1 to 255
"SmooSet"	Smoothing set	0 to 10 (%)
"DwelSet"	Dwell time set	0.9 or 9 (ms)
"CalType"	Calibration type set	"S11+S21", "S11", "S21", "12 Term"
"CalPower"	Signal power used during calibration	-20 to 0 (dBm)
"CalAver"	Averages used in calibration	1 to 255
"CalKit1" or "Clakit2"	Cal kit name loaded for port 1 or port 2	<cal kit name used>
"KitSex1" or "KitSex2"	Port 1 or port 2 cal kit sex	"Male" or "Female"
"KitC1" or "KitC2"	Calkit cap coefficients for port 1 kit or port 2 kit	<coefficient 0,coefficient 1, coefficient 2, coefficient 3>
"KitR1" or "KitR2"	Port 1 or port 2 cal kt reference plane	Reference plane of kit in meters and seconds
"KitL1" or "KitL2"	Port 1 or port 2 cal kit load reflection data flag	Returns 1 if data available or 0 if no data available (considered perfect load)
"KitD1" or "KitD2"	Port 1 or port 2 cal kit load reflection data	Comma separated string as follows. <# points, frequency (MHz), real part, imaginary part,…>
"Kit1T"	Port 1 cal kit Thru data flag	Returns 1 if data available or 0 if no data available
"Kit1TD"	Port 1 cal kit Thru data (s-parameters)	Comma separated string as follows. <# points, frequency (MHz), S11r, S11i, S21r, S21i,S12r,S12i,S22r,S22i
"Dem1"	De-embedding network for port 1 loaded or not flag	Returns "Loaded" or "Not loaded" if network for port 1 has been loaded or not.
"Dem2"	De-embedding network (port 2) loaded or not	Returns "Loaded" or "Not loaded" to indicate if de-embedding network for port 2 has been loaded
"ApplyDem"	Flag indicating if de-embedding is on or off	Returns "On" or "Off" to indicate if de-embedding is applied or not
"dcOff"	dc Offset (reads internal dc offset)	Returns "OK" if offset within factory limits or "Error". The dcOff must be called periodically to maintain measurement accuracy.
"FreqPlan"	Queries the Frequency plan set	Returns a comma separated string with the current frequency plan. Frequency values returned are in Hz. <start freq>, <stop freq>, <step freq>, <# of sweep points>, <test level>.

Para	Description	Valid return
“SysZo”	Queries the system impedance set	Returns a string <SysZo>
"ExMatch"	Queries if external impedance matching	Returns a string, <1> if networks used otherwise <0>
"CFreqPlan”	Queries the Frequency plan used for calibration	Returns a comma separated string with the frequency plan used. Frequency values returned are in Hz. <start freq>, <stop freq>, <step freq>, <# of sweep points>, <test level>.
“Interpol”	Queries if calibration being used is interpolated from another	Returns “Yes” or “No”
"Model"	Instrument model	String. Typically “LA19-13-02”
"SN"	Serial number	String. Typically a four digit number.
"Temp"	Instrument temperature	String. <Temp deg C> or “0” if an error is encountered.
"Stat"	Instrument status	String: “R”: Ready, “S”: Ready from power up, “E”: Error
"ErrFlag"	Read serial comms Error flag	String. Typically “Error” or “OK”
"TDP"	Time domain parameters	Returns a comma separated string with the following data. “<Start Time>, <Stop Time>, <Step Time>, <Termination type>,<Res value if applicable>, <Window type>, <Window order (if applicable)>”

Note! The dc offset voltage dcOff must be read before any calibration is done. Further, it is strongly advised that a call is made (GetInfo(“dcOff”)) every one or two minutes interval to ensure optimum performance.

Returns:
According to the table above or “Error” if an error has been encountered.

11. Data Storage

11.1 Save cal kit

Internal DLL declaration:
Public Function **SaveKit**(ByVal FileName As Variant, ByVal port As Integer) As String

Typical Usage:
This function is used to save the calibration kit on disk. The parameter passed is described in the table below.

<variable> = SaveKit (<FileName>, <port>)

Parameter passed	Description	Values	Comments
FileName Port	File name Port to which kit applies	<path + file name> '1' or '2'	The file name must include the full path name

Returns:

Return values	Type	Executed correctly	Not executed correctly
"OK" or "File Error"	String	"OK"	"File Error"

Important: Through adaptor data is only loaded and saved when the kit is associated with **Port 1** of the instrument. A kit with Thru data may be loaded to Port 2 but the Thru data will be ignored, therefore, if the kit is subsequently re-saved the Thru data will be lost.

11.2 Save measurement

Internal DLL declaration:
Public Sub **SelectSaveMeas**()

Typical Usage:
This routine is called to save measured data. When called, a window is displayed that allows the user to select the data and format to be saved as well as the destination file.

No parameters are passed or returned.

11.3 Save status and calibration

Note! Calibration and status files saved using the remote control DLL are not compatible with those saved using the direct control user interface software.

Internal DLL declaration:
Public Function **SaveCal**(ByVal FileName As Variant) As String

Typical Usage:
This function is used to save the current calibration and status. A single parameter is passed as described in the table below.

<variable> = SaveCal(<FileName>)

Parameter passed	Description	Values	Comments
FileName	File name or "?"	<path + file name> or <"?">	The file name must include the full path name. If a "?" is passed, a window is displayed to allow the user to interactively save the calibration and status.

Returns:

Return values	Type	Executed correctly	Not executed correctly
"OK" or "Error writing file"	String	"OK"	"Error writing file"

12. Data Retrieval

12.1 Load status and calibration

Internal DLL declaration:
Public Function **LoadCal**(ByVal FileName As Variant) As String

Typical Usage:
This function is used to load a calibration and status file. A single parameter is passed as described in the table below.

<variable> = LoadCal(<FileName>)

Parameter passed	Description	Values	Comments
FileName	File name or "?"	<path + file name> or <"?">	The file name must include the full path name. If a "?" is passed, a window is displayed to allow the user to interactively load a calibration and status file.

Returns:

Return values	Type	Executed correctly	Not executed correctly
"OK" or "Error reading file"	String	"OK"	"Error reading file"

13. Miscellaneous

13.1 Initialise all variables

Internal DLL declaration:
Public Function **InitVar**() as String

> **Warning!** This subroutine resets the dc offset voltage. Therefore, the dc offset must be read again (GetInfo("dcOff")) before any calibration or measurement is carried out.

Typical Usage:
This function is called to initialise all internal variables.

<variable> = InitVar

Returns:

Return values	Type	Executed correctly	Not executed correctly
"OK"	String	"OK"	Not Applicable

13.2 Set to signal generator mode (display a dialogue form)

Internal DLL declaration:
Public Sub **SelectSigGen**()

Typical Usage:
This routine is called to display a window that allows the user to set the instrument up as synthesised signal generator.

Call SelectSigGen
SelectSigGen

Returns:
No parameters are passed or returned.

13.3 Set to signal generator mode

Internal DLL declaration:
Public Function **SetSigGen**(ByVal Freq As Variant, ByVal P As Variant) As String

Typical Usage:
This function is used to set up the instrument as synthesised signal generator. The parameters passed are described in the table below.

<variable> = SetSigGen(<Freq>, <P>)

Parameter passed	Description	Values	Comments
Freq	Frequency in MHz	<frequency>	Must be within the range 3 to 3080
P	Power in dBm	<power>	Must be within the range -20 to 0

Returns:

Return values	Type	Executed correctly	Not executed correctly
"OK" or "Error"	String	"OK"	"Error"

13.4 Get DLL program version

Internal DLL declaration:
Public Function **DLLVer**() As String

Typical Usage:
This function is used to query the DLL program version. No parameters are passed.

<variable> = DLLVer

Returns:

Return values	Type	Executed correctly	Comments
<version>	String	<version>	Example: "V1.0 June 2006"

14. Diagnostics

14.1 Run diagnostics tests (display form)

Internal DLL declaration:
Public Function **InsDiag**() As String

Typical Usage:
This function is used to display a window which allows the user to carry out instrument diagnostics tests. No parameters are passed.

<variable> = InsDiag

Returns:

Return values	Type	Executed correctly	Not executed (aborted)
<string>	String	"Tests completed"	"Tests not performed"

15. Examples

The diagram below shows the steps to carry out a measurement. It is simplified and incorporates the minimum number of steps to perform a measurement.

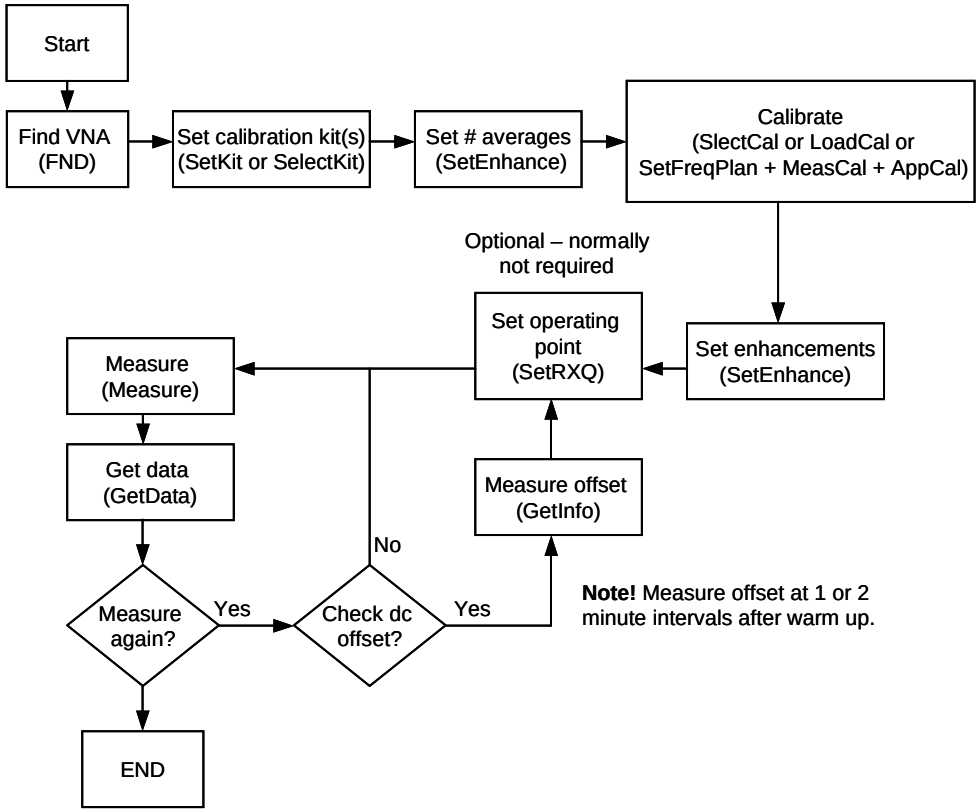


Figure 15.1: Simple measurement flow chart

In the following section some examples of code that make use of the ActiveX DLL are provided. These illustrate the basic approach that can used to control the LA19-13-02 VNA.

15.1 Visual Basic 6

A simple VB6 example (DLL_Test) is described below that makes use of the DLL library to carry out a calibration, perform a measurement sweep, get measured data and display data of a single measurement point.

Note! The dc offset voltage dcOff must be read before any calibration is done. Further, it is strongly advised that a call is made (GetInfo("dcOff")) every one or two minutes interval to ensure optimum performance.

15.1.1 Program Form

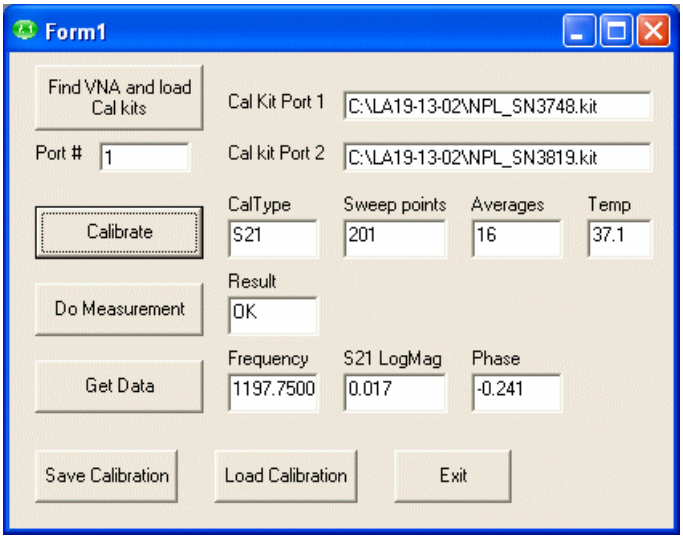


Figure 15.2: Form of simple example described

15.1.2 Loading the DLL library

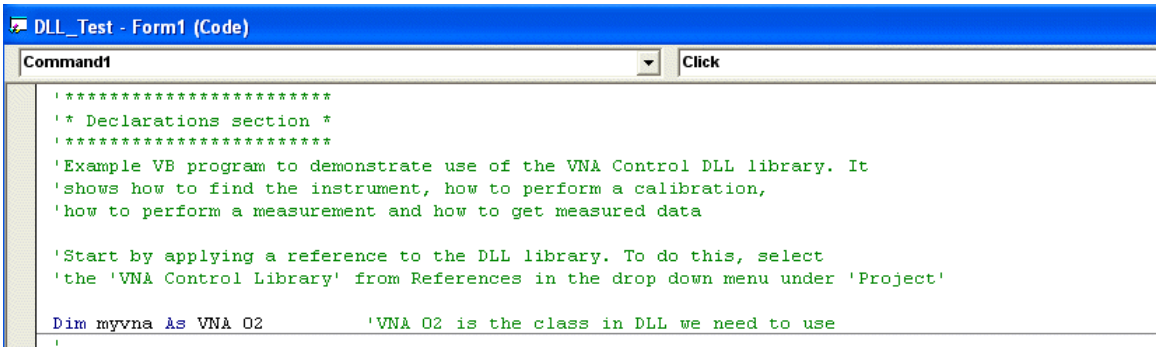
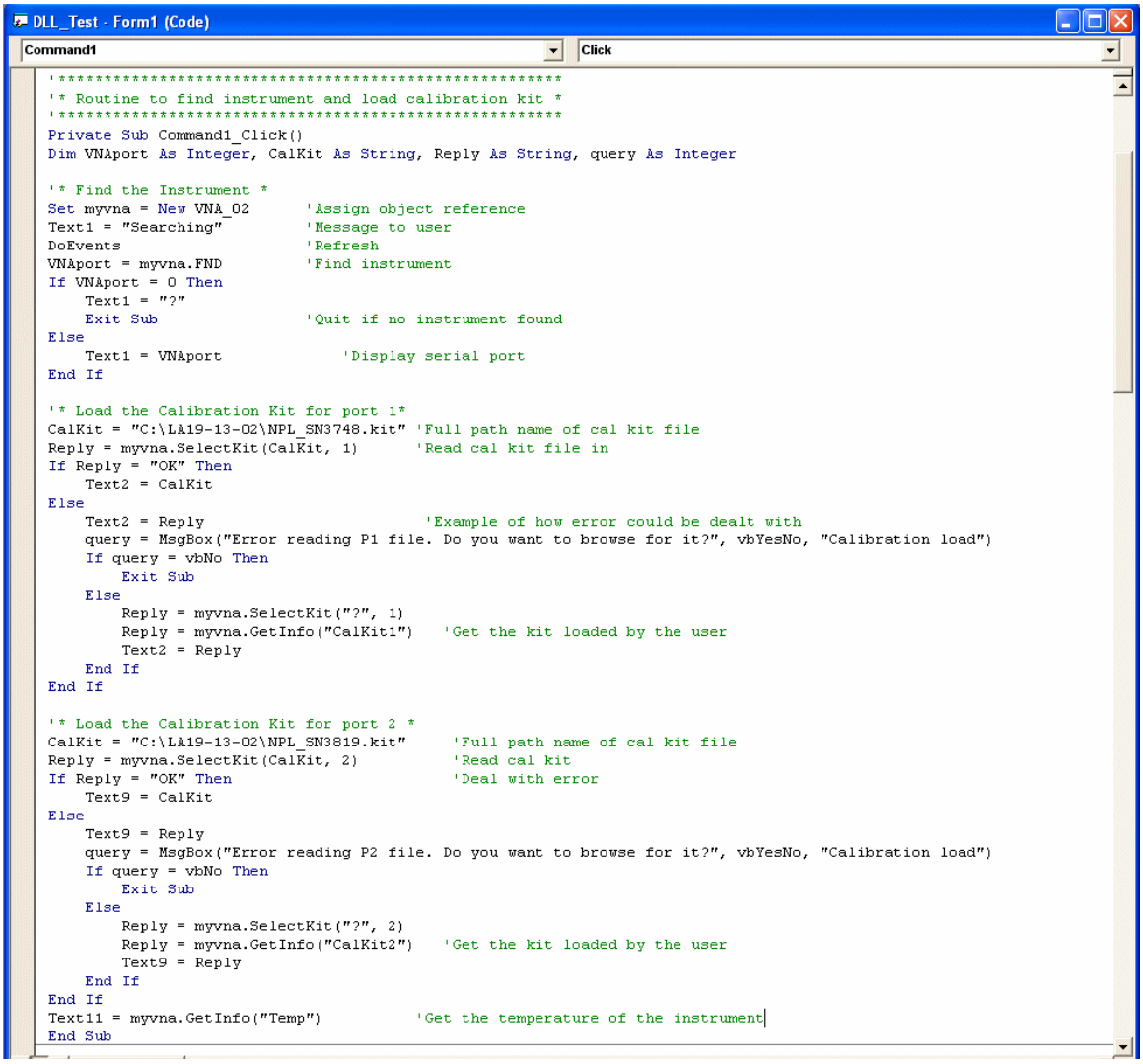


Figure 15.3: Setting the reference to the DLL library (VNA_02)

15.1.3 Discovering the instrument and loading a calibration kit

The code shown below finds the instrument and returns the serial port to which it is connected. It then proceeds to load calibration kit files. Note that the calibration kit file names must include the full path. The code also gets the temperature if the instrument.




```
'*******************************************************
'* Routine to find instrument and load calibration kit *
'*******************************************************
Private Sub Command1_Click()
Dim VNAport As Integer, CalKit As String, Reply As String, query As Integer

'* Find the Instrument *
Set myvna = New VNA_02      'Assign object reference
Text1 = "Searching"         'Message to user
DoEvents                    'Refresh
VNAport = myvna.FND         'Find instrument
If VNAport = 0 Then
    Text1 = "?"
    Exit Sub                'Quit if no instrument found
Else
    Text1 = VNAport             'Display serial port
End If

'* Load the Calibration Kit for port 1*
CalKit = "C:\LA19-13-02\NPL_SN3748.kit" 'Full path name of cal kit file
Reply = myvna.SelectKit(CalKit, 1)      'Read cal kit file in
If Reply = "OK" Then
    Text2 = CalKit
Else
    Text2 = Reply                       'Example of how error could be dealt with
    query = MsgBox("Error reading P1 file. Do you want to browse for it?", vbYesNo, "Calibration load")
    If query = vbNo Then
        Exit Sub
    Else
        Reply = myvna.SelectKit("?", 1)
        Reply = myvna.GetInfo("CalKit1")    'Get the kit loaded by the user
        Text2 = Reply
    End If
End If

'* Load the Calibration Kit for port 2 *
CalKit = "C:\LA19-13-02\NPL_SN3819.kit"     'Full path name of cal kit file
Reply = myvna.SelectKit(CalKit, 2)          'Read cal kit
If Reply = "OK" Then                        'Deal with error
    Text9 = CalKit
Else
    Text9 = Reply
    query = MsgBox("Error reading P2 file. Do you want to browse for it?", vbYesNo, "Calibration load")
    If query = vbNo Then
        Exit Sub
    Else
        Reply = myvna.SelectKit("?", 2)
        Reply = myvna.GetInfo("CalKit2")    'Get the kit loaded by the user
        Text9 = Reply
    End If
End If
Text11 = myvna.GetInfo("Temp")          'Get the temperature of the instrument
End Sub
```

Figure 15.4: Example of code to discover the instrument and load calibration kits

15.1.4 Performing a calibration

The code below can be used to calibrate the VNA. Note that the function used (SelectCal) is called with a "?" string argument. This causes a window to be displayed that allows the user to perform a calibration. If instead, a valid full path calibration name of a valid calibration file is passed, then the function will load the file instead. Note that calibration files saved using the main user interface software are not compatible with this DLL function.

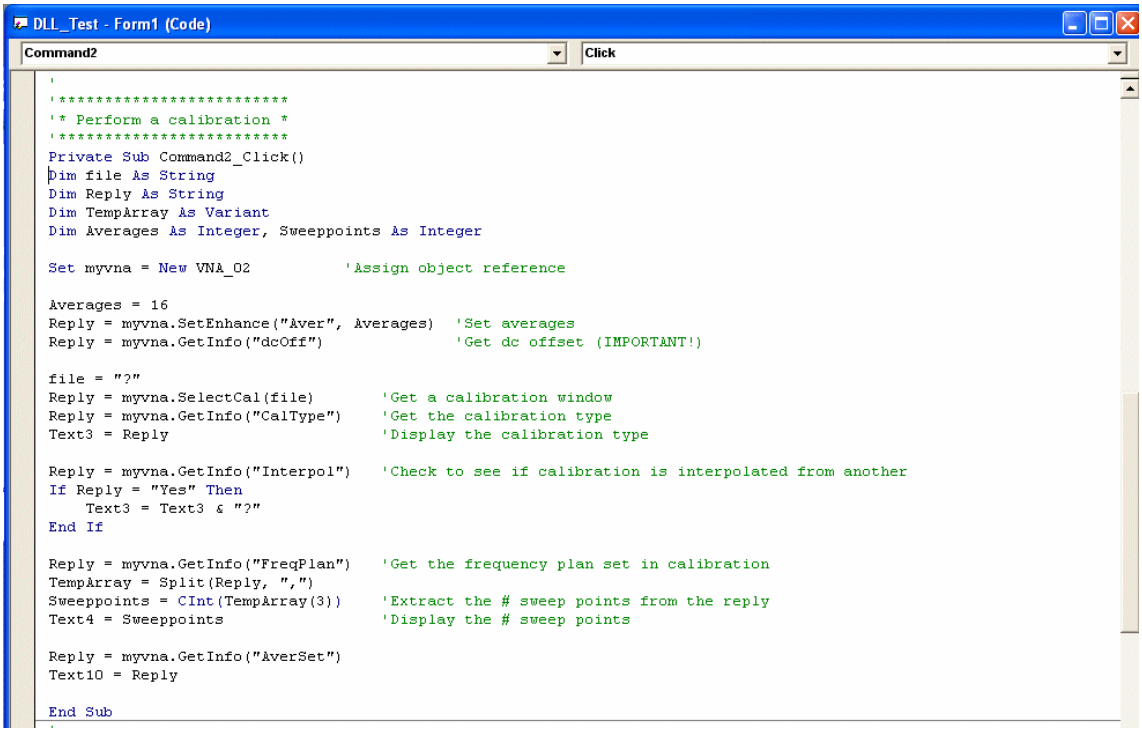


Figure 15.5: Example of code that may be used to perform a calibration

The GetInfo function is used to get information about the calibration performed. In the example shown in Fig. 15.5, passing an argument of "FreqPlan" returns the frequency plan, including the number of sweep points as a comma separated string. The various parameters can be extracted using the VB 'split' command as shown. Note that the GetInfo("Interpol") is used to check if the calibration being used is interpolated from another. That is, if the user changed the frequency sweep parameters after performing a calibration.

15.1.5 Performing a measurement

The code shown below can be used to perform a single sweep measurement. The 'Measure' function returns either an "OK" or "Error" string. In the example, this is displayed in text box Text8.

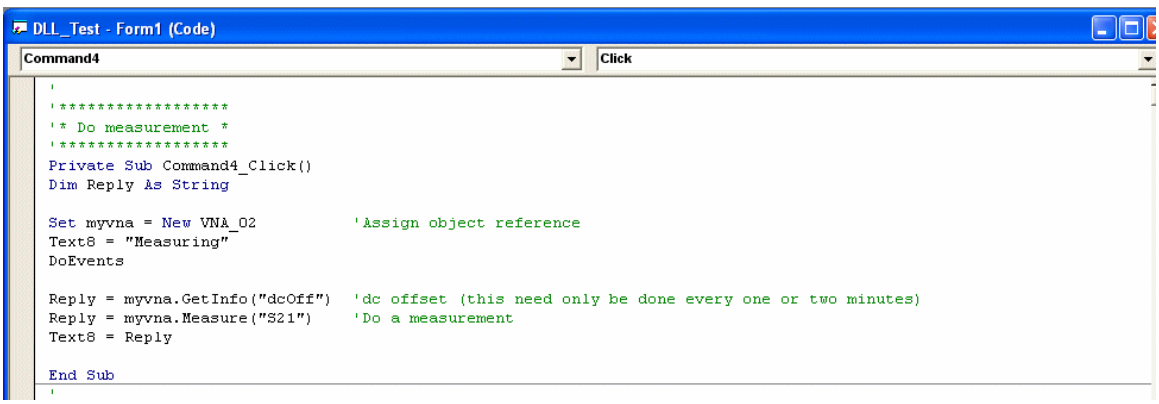


Figure 15.6: Example of code that may be used to perform an S21 measurement

15.1.6 Getting measured data

In the example below, the function FndPt is used to get the measured value at a specified frequency. The function returns a comma separated string. In the example shown, the first term in the string is the nearest frequency to that requested and the second term is the parameter (S21) value in log magnitude form. The second use of the function FndPt gets the phase of S21.

Figure 15.7: Example of code that may be used to get measured S21 data

15.2 Agilent's VEE

Three simple Agilent Vee examples are given that show the typical usage of ActiveX components. The first is a calibration followed by an S-parameter measurement, the second is a calibration followed by a Time Domain measurement, and the third shows how to put the VNA into signal generator mode and communicate with an Agilent spectrum analyser to display the signal. The first example is covered in some detail while the other two are described very briefly, the files being best studied in Agilent Vee.

15.2.1 S-Parameter Example (VNASparmExample.vee)

The diagram below shows the overall view of the Vee program.

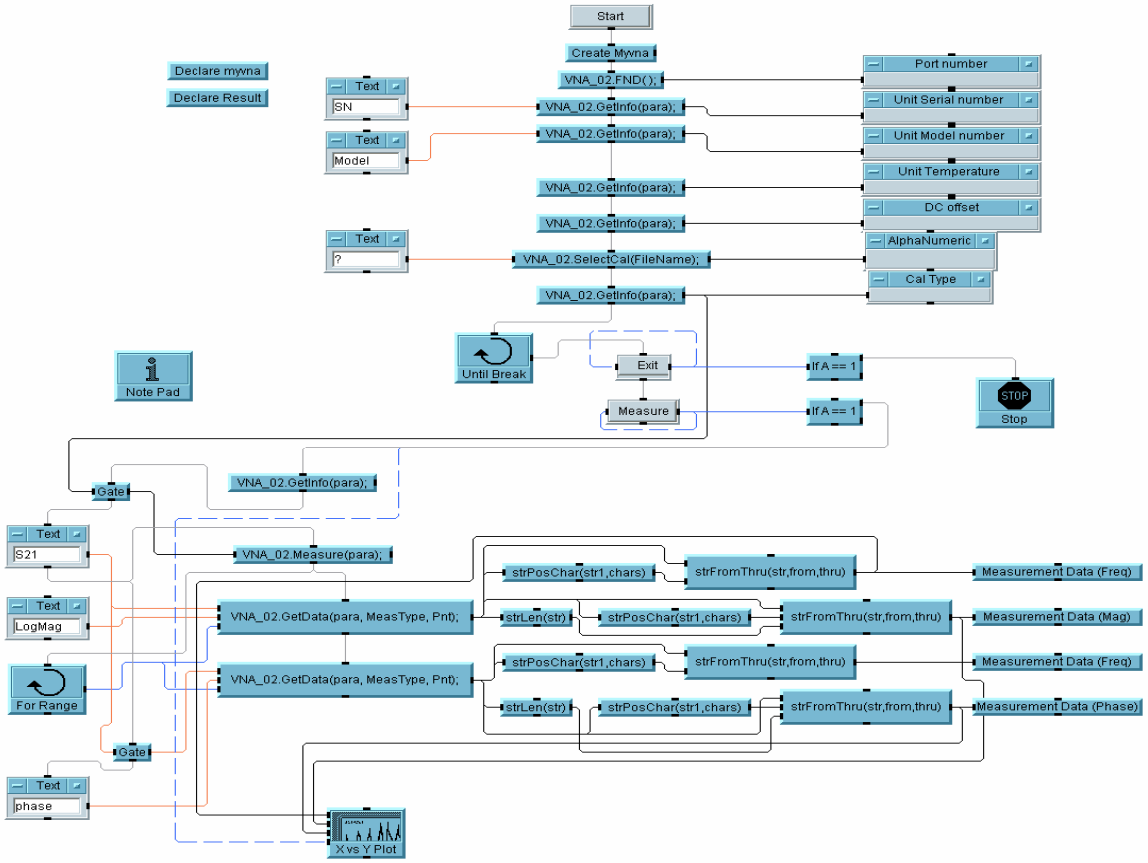


Figure 15.8: Overall diagram of S parameter measurement program

1. If it has not already been done during installation, the first thing that must be done is to register the VNA Control DLL on the PC as described in Section 2 of this document. (The VNA Control DLL is normally registered automatically during installation of the VNA software. It should only need registering manually if an update has been issued.)

2. Once the VNA Control DLL has been registered, Start Agilent Vee

3. From the tool bar select **Devices ⇒ ActiveX automation references**, and tick the box for the **VNA Control Library**.

4. Declare the Variable name with which the VNA will be addressed by selecting a Declare variable block. From the toolbar select **Data ⇒ Variable ⇒ Declare Variable.**

5. Place the block in the main window and fill in the boxes.
   In this case we have chosen the name **myvna**
   The scope is set to Local to Context
   Type is set to Variant
   Number of Dimensions is set to 0

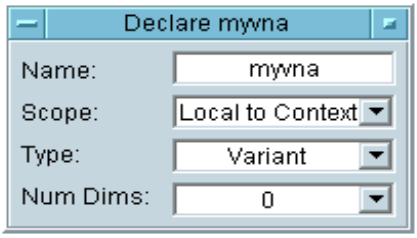


Figure 15.9: Declare VNA variable

6. In a similar fashion a Variable to store the Result is declared

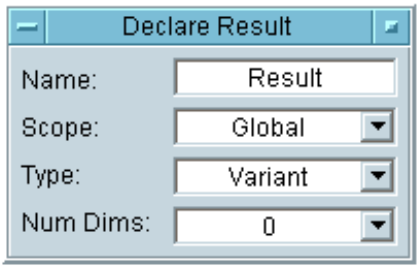


Figure 15.10: Declare Result variable

7. Next the variable **myvna** must be assigned to the VNA with a CreateObject statement. From the toolbar select **Device ⇒ Formula** and place it in the main window.
   Delete the input terminal by right clicking on the input port and select **Delete Terminal ⇒ Input**, select A and click **OK**
   Delete the default formula and type:
   set myvna = CreateObject("VNAControl2.VNA_02");

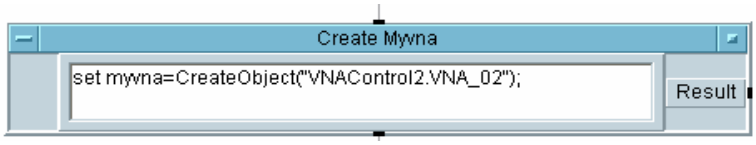


Figure 15.11: Create VNA Object

8. Now the VNA can be controlled through the VNA function library using ActiveX controls.
   From the toolbar select **Device ⇒ Object browser.**
   Under "Type" select ActiveX Objects
   Under "Library" scroll down and select VNAControl
   Under "Members" select the function you wish to call, in this case FND,
   Click **Create call** and place the block on the main window
   Inside the block replace VNA_02 with myvna so the call now says
   myvna.FND();

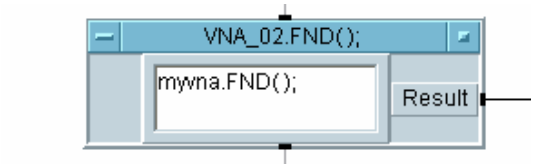


Figure 15.12: Find VNA function

9. Add an Alphanumeric box to show the result by selecting from the toolbar **Display ⇒ Alphanumeric** and place the block in the main window.
   Connect the result output of the FND function to the Alphanumeric

10. If this simple program is run the number of the Com port to which the VNA is attached will be returned in the alphanumeric block.

Further functions can be added to determine the state of the instrument or to tell it to carry out measurement functions and read back the results.

11. Repeat the process in step 7 to make a call using the **GetInfo** function. Remember to change VNA_02 to read **myvna.**
    Add an alphanumeric block to display the result.
    Add a text constant block, from the toolbar select **Data ⇒ constant ⇒text**, and enter the parameter you wish to discover from the instrument, in this case SN which will return the instrument's serial number.
    Similarly the **GetInfo** function can be repeated to determine the Model number.

Parameters can also be passed directly to the **GetInfo** function by typing the parameter within the brackets of the function. If the parameter is a string parameter then the parameter must be enclosed between inverted commas. For example to find the temperature of the receiver create a **GetInfo** function call, delete the input terminal and replace the word **Para** inside the brackets with **"Temp"**. (Remember to replace VNA_02 with **myvna** in the function call). The function should read: **myvna.GetInfo("Temp");** This works in exactly the same way as if Temp had been passed from a text block.

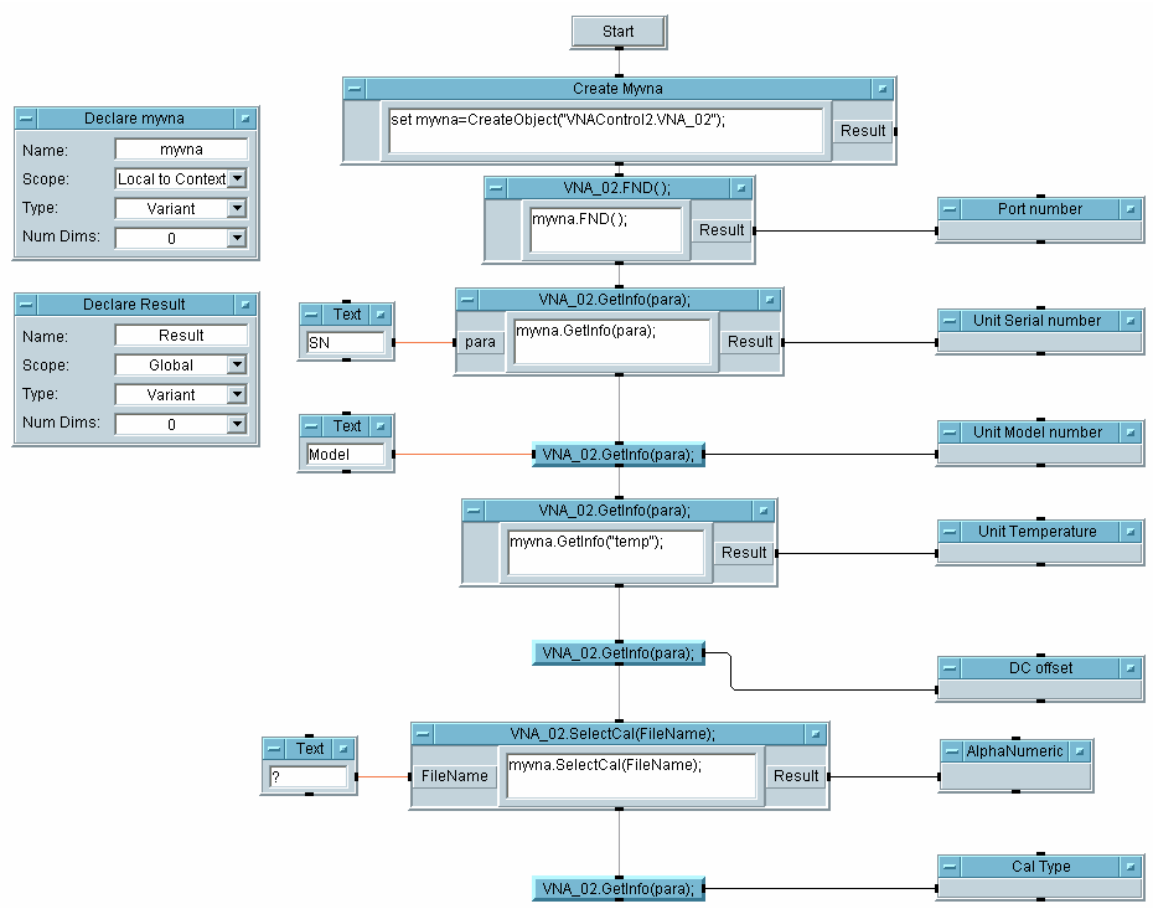


Figure 15.13: Diagram showing typical Vee function calls

12. Before making a measurement it is necessary to check the DC offset and calibrate the instrument. Also during repeated measurements a DC offset check should be carried out every few minutes.

13. Create a call to check the DC offset. From the toolbar select **Device ⇒ Object browser.**
    Select **GetInfo** and click on **create call**. Place the block in the main program. Delete the data input terminal and replace Para in the function with **"dcoff"**. Place an alphanumeric block for the result.

14. Create a call to do a calibration. From the toolbar select **Device ⇒ Object browser** and select **SelectCal**.

15. Create a call to determine what type of calibration was performed as this information will be used to determine what measurements are possible at a later stage. From the toolbar select **Device ⇒ Object browser.** Select **GetInfo** and click on **create call**. Place the block in the main program. Delete the data input terminal and replace Para in the function with **"CalType"**. Place an alphanumeric block for the result.

Now we are ready to set up the measurement procedure. This sets the program in a continuous loop so that measurements are made every time a button is pressed. To exit the loop a second Exit button must be pressed.

16. From the toolbar select **Flow ⇒ Repeat ⇒ Until Break**. Place the block in the program.

17. Create two Control buttons **Data ⇒ Toggle control ⇒ Button**
    These buttons will be used to make a measurement or to exit. Name one "Exit" and the other "Measure" by typing the names in the title section in the properties box.
    Add a reset terminal to each button by right clicking the button, **⇒ Add terminal ⇒Control input**, select **Reset** and click OK

18. Create two conditional blocks. Select **Flow ⇒ Conditional ⇒ If A= =B**.
    Delete the B input terminal and set the condition to A= =1

19. From the Flow menu create a stop block. **Flow ⇒ Stop**

20. Connect the blocks as shown below

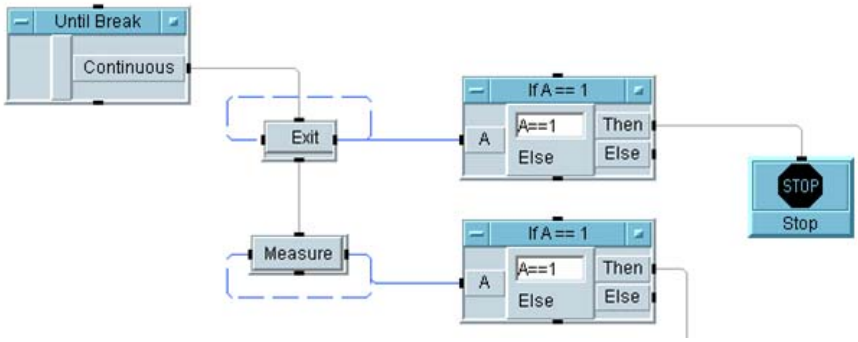


Figure 15.14: Setting up loop for measurements

21. Create a **Gate** block to control the data flow. Gate blocks are used to ensure the data is updated for each measurement. Select **Flow ⇒ Gate** and place the block in the main window

22. Create a call to check the dc offset, which should be done before each measurement.

23. Create calls to make a measurement and get the data for magnitude and phase.

24. Create a **For Range** and set it to loop from 1 to 101 in steps of 1 to match the number of points used in the calibration.

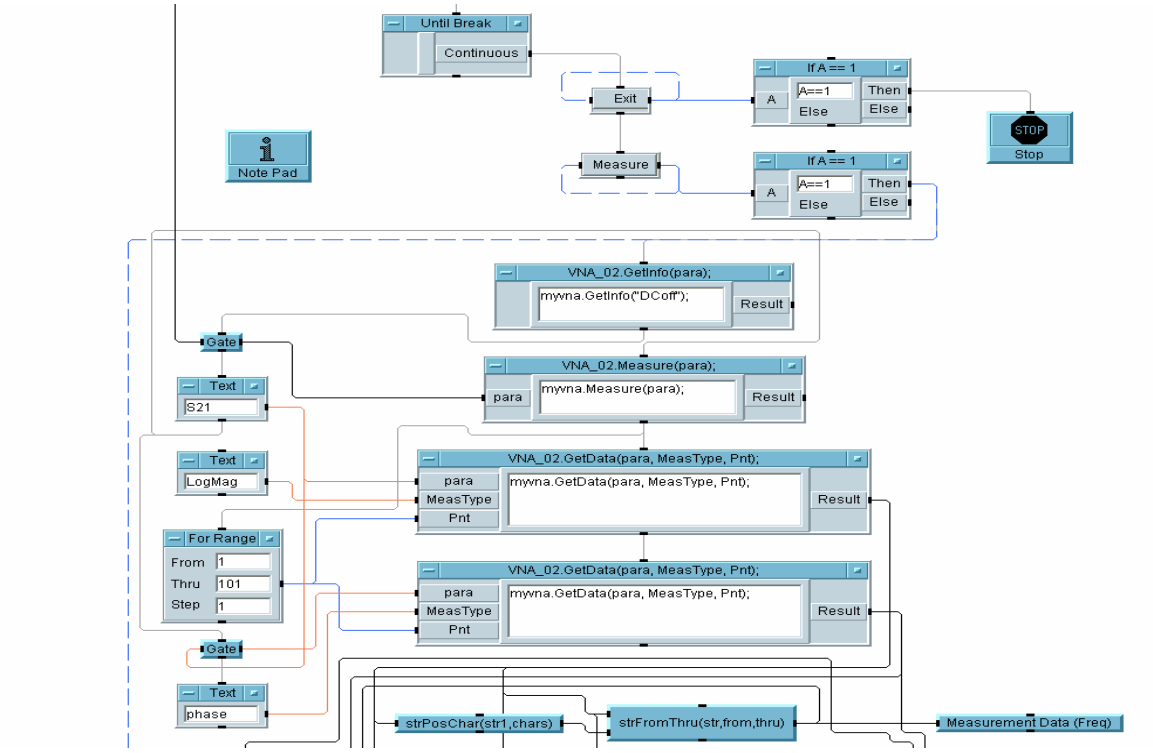


Figure 15.15: Completing the measurement loop

25. Once the measurement blocks have been placed and wired the results are manipulated by various built-in Vee String commands before being displayed on an x,y Plot

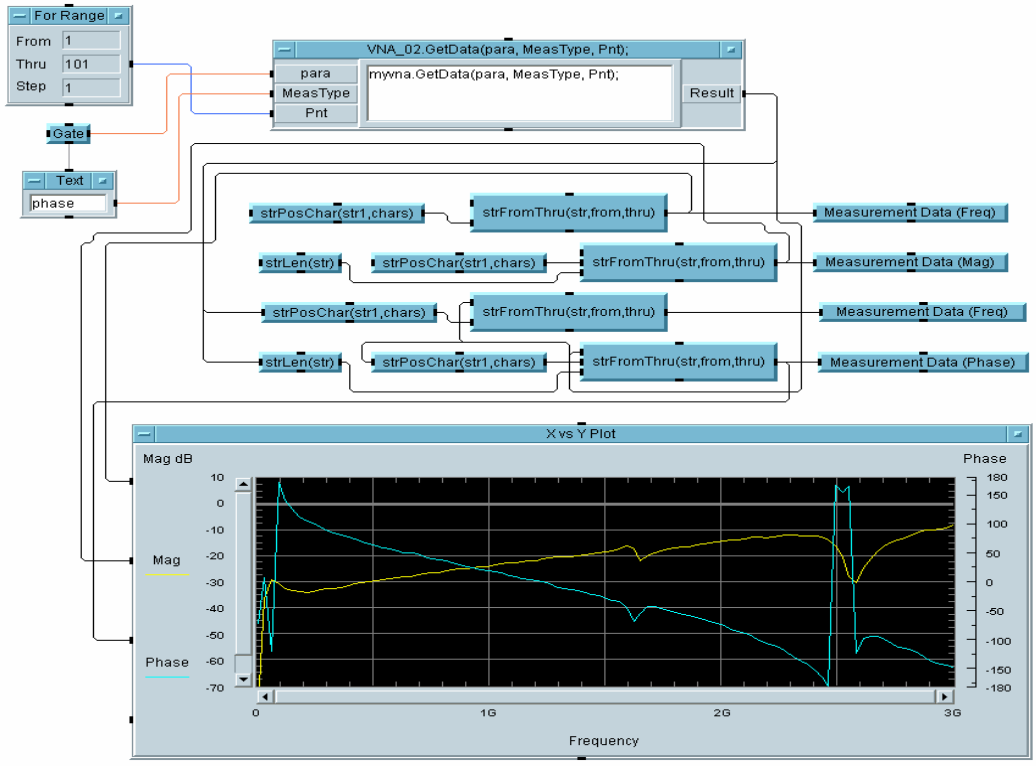


Figure 15.16: Processing and displaying the results

26. Finally the relevant control buttons and displays can be put into a panel to produce a user-friendly interface, and notes can be added with a Note Pad block.

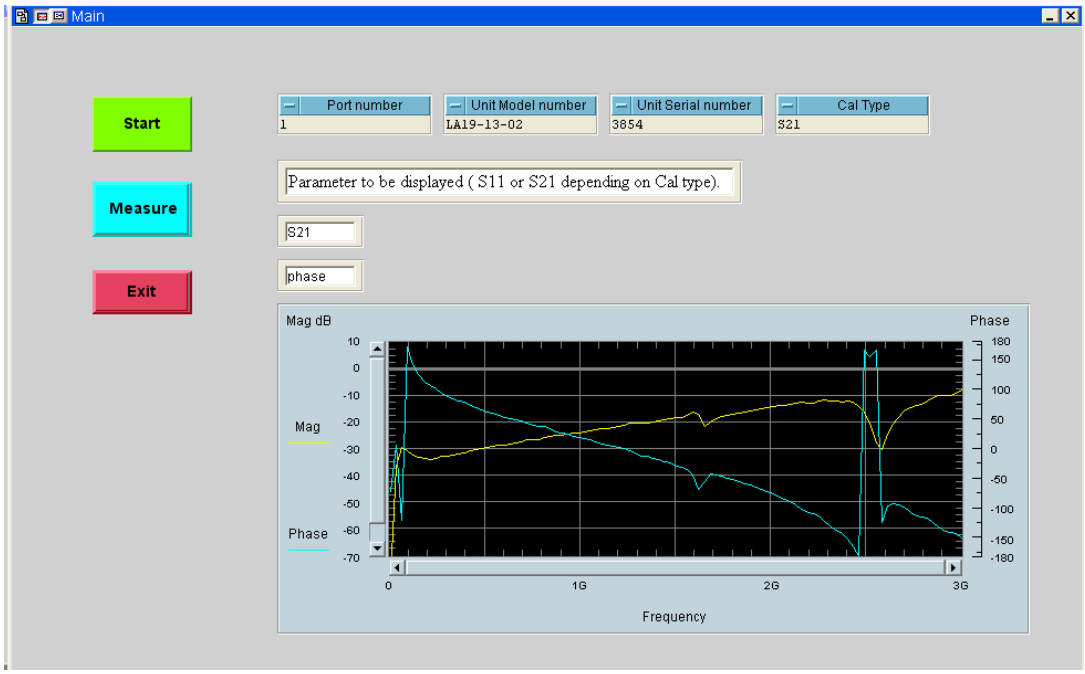


Figure 15.17: Placing the controls in a Panel to create a user interface

15.2.2 Time Domain example (VNATimeDomainExample.vee)

In a similar fashion a program can be assembled that makes a time domain calibration then carries out a time domain measurement.

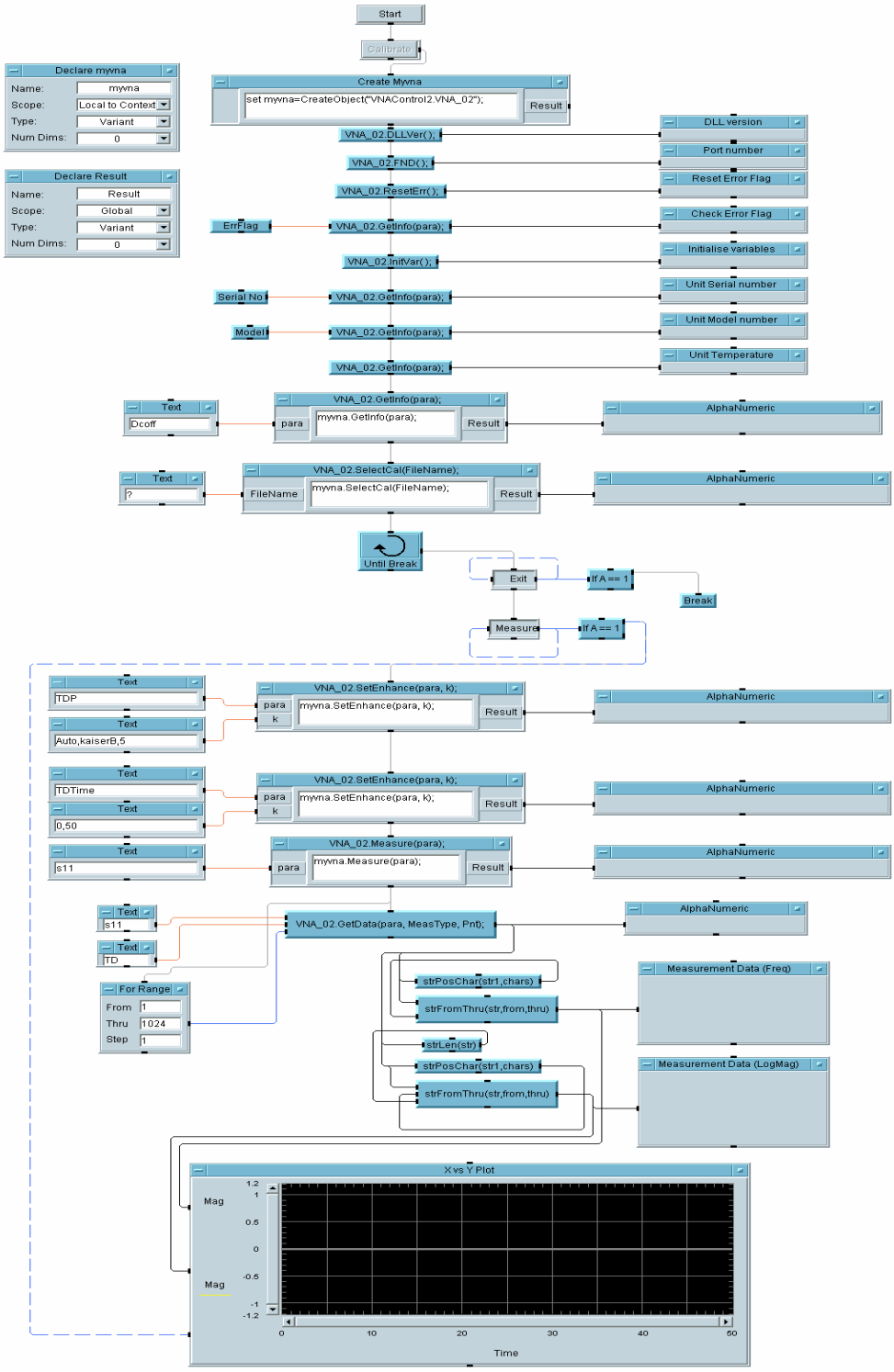


Figure 15.18: Overall diagram of Time Domain Example

The relevant buttons can be selected from the main program and placed on a panel to produce a simple user interface.

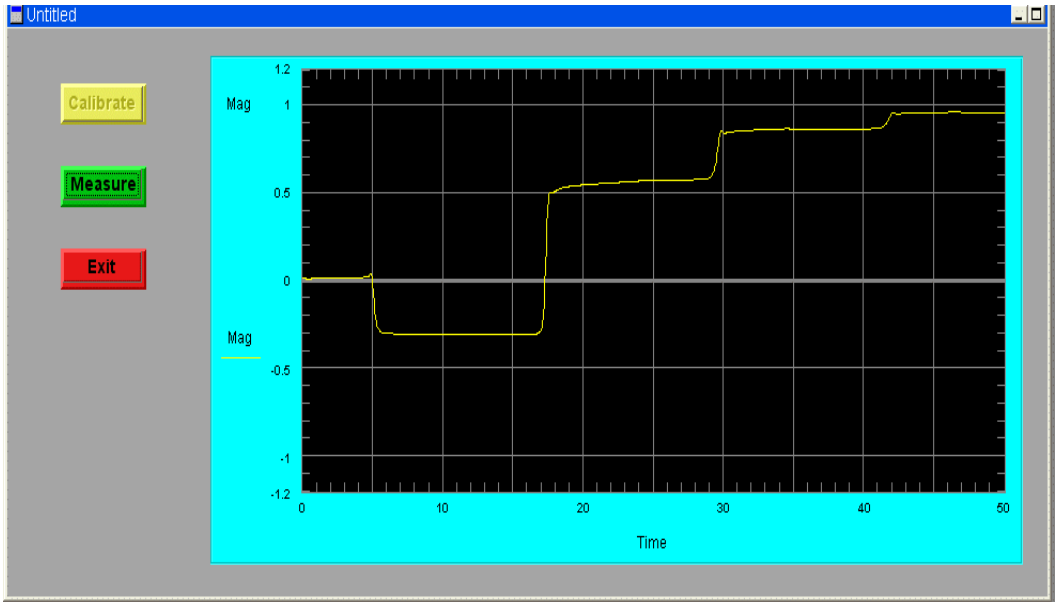


Figure 15.19: User interface for Time domain example

15.2.3 Signal generator example (VNASigGenExample.vee)

The signal generator example describes how to use the VNA in the Signal Generator mode.
The program gives an example of how to:

- Set the frequency an amplitude using the interactive form,
- Set the frequency and amplitude using text entry
- Communicate with an Agilent E4407B spectrum analyser
- Program the VNA to do a frequency sweep
- Program the VNA to do an amplitude sweep

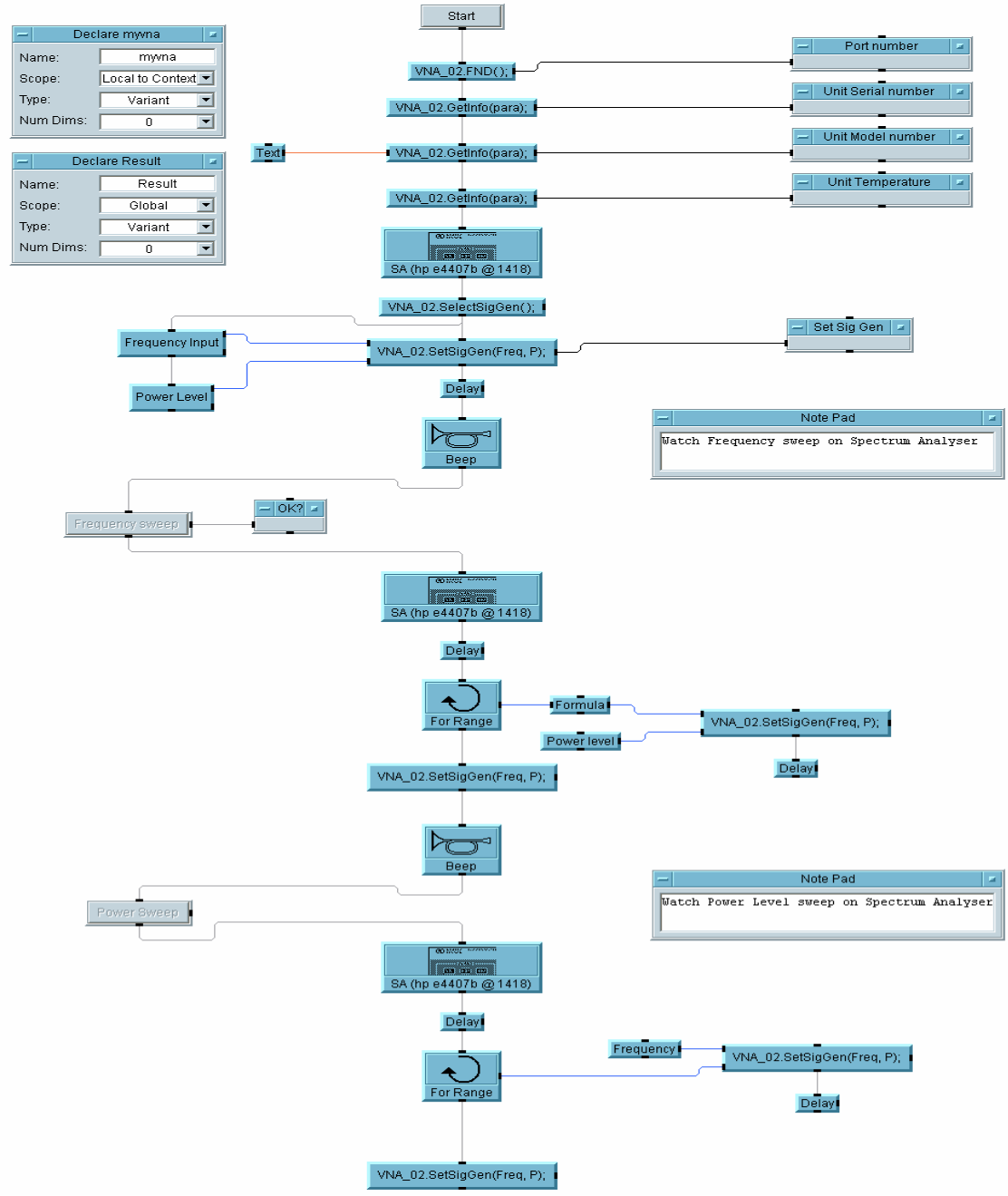


Figure 15.20: Overall diagram of Signal Generator example

The relevant controls can be selected from the main program and placed in a panel to make a user interface.

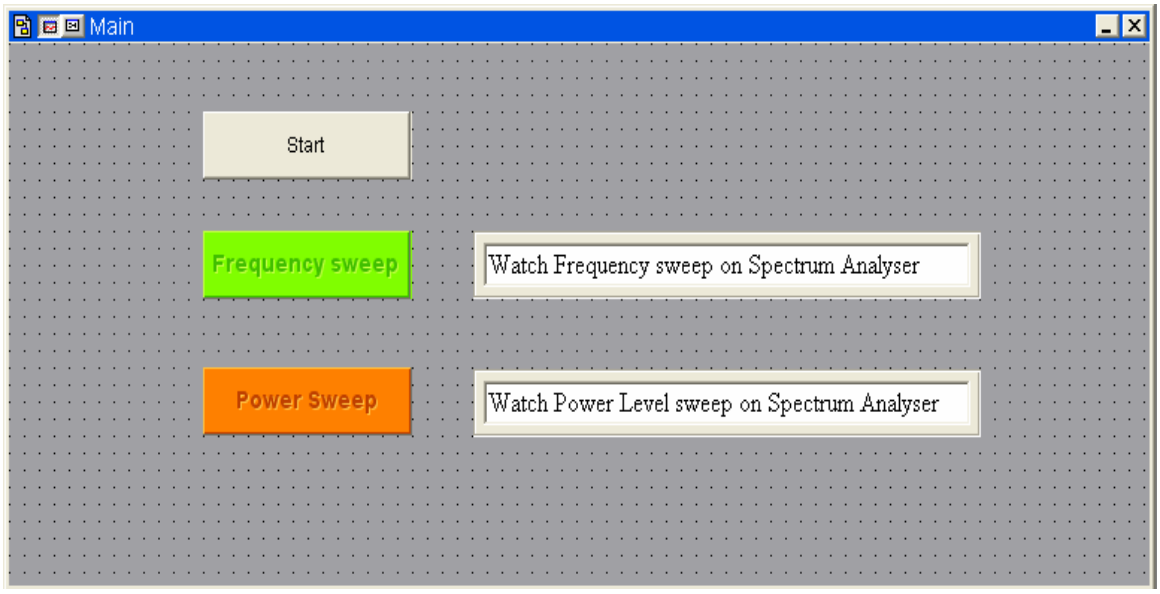


Figure 15.21: Diagram of Signal Generator example user interface

15.3 National Instruments LabVIEW

A simple example of how to get started and communicate with the instrument is provided. The functions work in the same way as in Visual basic or Agilent Vee, it is only the way one sets up the calls that differs so it may be useful to look at the Vee examples even if LabView is being used.

15.3.1 LabVIEW example

This simple example shows how to set up ActiveX calls to control the instrument or to receive information from the instrument.

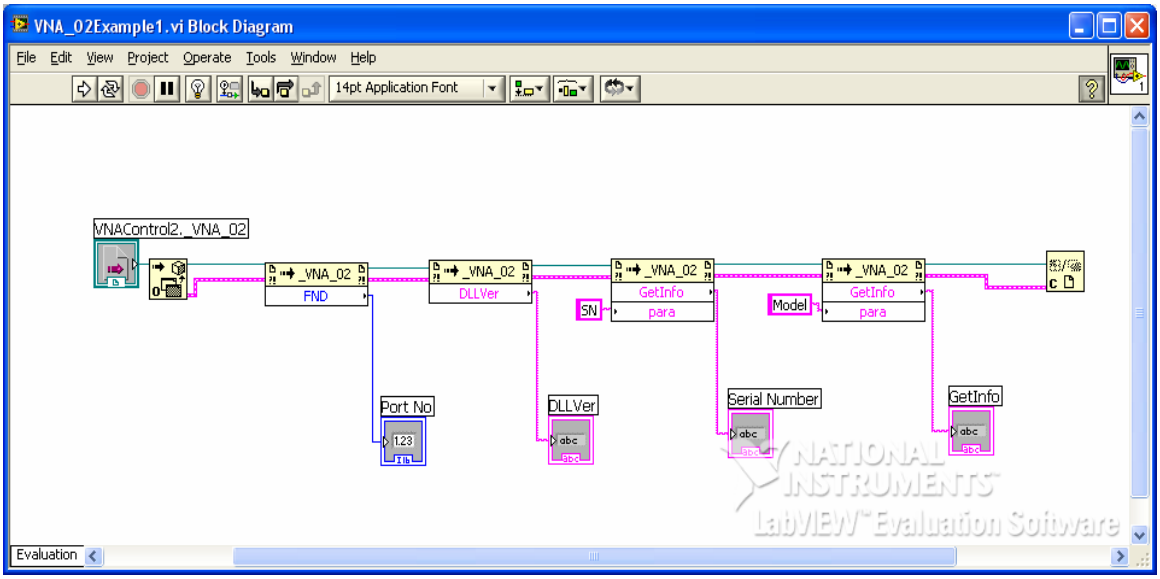


Figure 15.22: LabView Block diagram

1. Start LabView and open a new VI

2. If the Block diagram window is not visible, make it visible by selecting **Window ⇒ Block diagram**

3. Right click in the block window and select **Connectivity ⇒ ActiveX ⇒ Automation Open.** Place it in the block window. Right click on the Automation Open block and select **ActiveX Class.** In the Type library select **VNA2 Control library Version xx.x** where xx.x is the version number. Click on **VNA_02(VNAControl2.VNA_02)** and click OK. This generates an Automation Refnum

4. Right Click on the Open Automation block and select **ActiveX Palette ⇒ Invoke Node.** Place it in the block window. Right click on the block and select **Select Class ⇒ ActiveX ⇒ VNAControl2._VNA_02**.
   Left click on **Method** and choose the function you wish to use. In this case FND.

5. Wire the Refnum to the reference input.

6. Right click on the output of FND and select **Create ⇒ Indicator**. An indicator block of the correct type will be placed on the port.

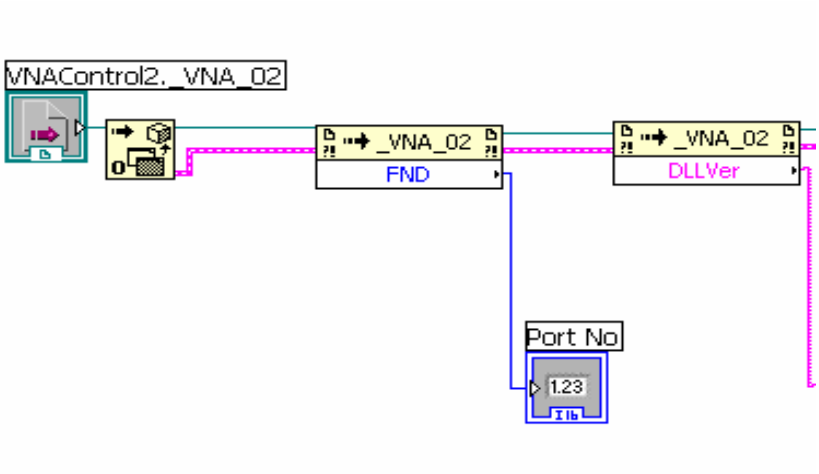


Figure 15.23: View of initial blocks

7. In a similar fashion place a DLLVer call, add an indicator block and an Indicator block on the output.

8. Add a GetInfo call. Right click on Para and select **Create ⇒ Constant**, a string constant box will be generated. Type SN into the box.

9. Right click on GetInfo and select **Create ⇒ Indicator** and place the result block. Rename the block Serial No

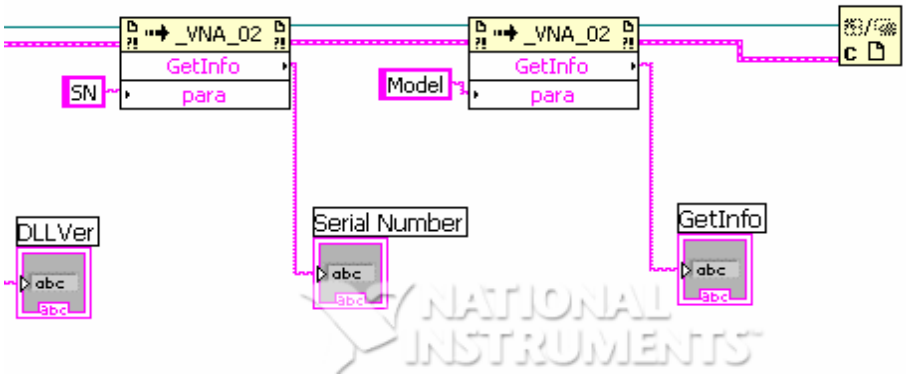


Figure 15.24: View of final blocks

10. Add another GetInfo block and this time type Model in the Para input box and rename the output box Model.

11. Finally when all the function blocks have been placed a **Close Reference** block must be added. Right click in the block window and select Connectivity ⇒ **ActiveX⇒ Close Reference.** Wire the Refnum to the reference port of the **Close Reference**. Also wire the error out's to the error in's if required.

12. Move to the Front Panel window and arrange the Indicator blocks to suit.

13. Run the program. After a short wait the Port number, DLL version, model number and serial number should be returned in the appropriate box.

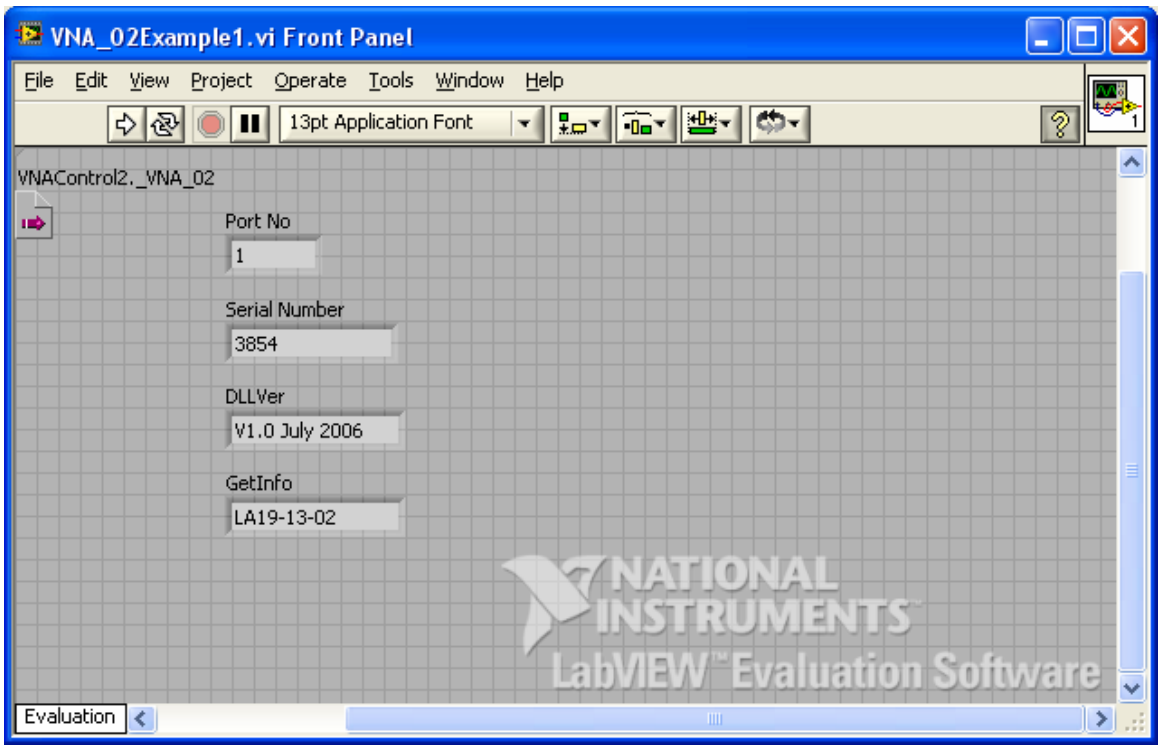


Figure 15.25: View of Front Panel with results