AUTOMATION OF RANDOM TELEGRAPH NOISE AND FLICKER NOISE CHARACTERIZATION SETUP

A Project Report

submitted by

SAPTAK ACHARJEE

in partial fulfilment of the requirements

for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS
MAY 2019

THESIS CERTIFICATE

This is to certify that the thesis titled ***Automation of Random Telegraph Noise and Flicker Noise Characterization Setup****, submitted by* ***Saptak Acharjee****, to the Indian Institute of Technology, Madras, for the award of the degree of* ***Master of Technology****, is a bona fide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.*

Dr. Deleep R Nair
Research Guide
Associate Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai
Date: May 2019

ACKNOWLEDGEMENT

The present project work is submitted in fulfilment of the requirements for the degree of Master of Technology of Indian Institute of Technology, Madras (IITM). I express my deepest gratitude to my supervisor Dr. Deleep R Nair of Indian Institute of Technology, Madras for his inestimable support, encouragement, profound knowledge, largely helpful conversations and also for providing me a systematic way for the completion of my project works. I am also extremely grateful to the Pavan Ch, PhD student of IITM, for his continuous help and support throughout the project. Last but not the least, this work would not have been possible without support of my fellow lab mates, especially Vivek Oza, MS Student, IITM.

Saptak Acharjee
MTech Student
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai
Date: May 2019

ABSTRACT

MOS devices in the nanometer regime suffer from major reliability issues due to high electric field phenomenon and process variability. Threshold voltage shift and mobility degradation are the results from these reliability mechanisms. Such intrinsic device parameter fluctuation affects the device performance which results unexpected behaviour of MOS transistors in IC chips. RTN (Random Telegraph Noise) and Flicker noise measurements are two methods in low frequency domain to test device reliability for different dimensions of MOS devices. To study RTN and flicker noise properly in MOS devices, a large number of measurement results (number of measured points in the range of lakhs) are needed for each of the MOS device in different gate and drain voltages with different drain current sampling times and same thing needs to repeat for all devices in a wafer. The results of this whole process come with such a huge volume of measured output data that it becomes very tedious cumbersome and time consuming to handle it manually. Therefore, instead of manually handling, RTN and flicker noise measurement instruments can be remotely interfaced in a desktop computer and automated its control using various programming software. This project work is mainly based on the developing codes for various programs to automate the RTN and flicker noise measurement instruments in the way that all measurements, data acquisition and analyse of these data become easy, efficient and very low time consuming. Here automation of instruments has been done using Visual C++ (for RTN measurement) and LabView (for flicker noise measurement) programming software.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

LFN	Low Frequency Noise
RTN	Random Telegraph Noise
WGFMU	Waveform Generator/Fast Measurement Unit
SMU	Source/Monitor Unit
RSU	Remote-sense and Switching Unit
ALWG	Arbitrary Linear Waveform Generation
PG	Pulse Generation
VCO	Voltage-Controlled Oscillator
SRAM	Static Random Access Memory
MOS	Metal-Oxide Semiconductor
CMOS	Complementary Metal-Oxide Semiconductor
FFT	Fast Fourier Transform
PSD	Power Spectral Density
VC++	Visual C++
IDE	Integrated development Environment
DUT	Device Under Test

Chapter 1

Introduction

Due to aggressive down scaling in device dimensions for improving speed and functionality, MOS devices in the nanometer regime suffer from major reliability issues due to high electric field phenomenon and process variability. These reliability mechanisms cause the MOS transistor parameter drifts, namely threshold voltage shift and mobility degradation. Process variations were originally considered in die-to-die variations. For nanoscale transistors, intra die variations are posing the major design challenge as technology node scales. The intrinsic device parameter fluctuations that result from high electric field phenomenon and process uncertainties have substantially affected the device characteristic and degraded its performance. Performance degradation of MOS transistors results unexpected behaviour in IC chips [1, 3]. So it becomes very important to test MOS devices reliability in this nanometer scaling for both single and multiple charge perturbation in the gate oxide. Many methods exist to test device reliability. Low Frequency Noise (LFN) [1, 2] measurements are one of those methods. Noise spectrum of large and small dimension MOS devices show different behaviour in low frequency domain. In low frequency, noise spectrum of large area devices show exact *1/f* behaviour which is called Flicker Noise [2] but small area devices show completely different behaviour from flicker noise spectrum. The low frequency noise behaviour of small area devices is commonly referred to as Random Telegraph Noise (RTN) [2 to 4].

1.1 Low Frequency Noise (LFN)

Low Frequency Noise (LFN) is a serious performance limiter in mixed signal CMOS circuits such as RF mixers, voltage-controlled oscillators (VCOs), and time-to-digital A/D converters. An important design subject for IC designers is how to minimize the noise contribution in the circuit. So accurate and timely LFN models for semiconductor devices are needed. However, performing meaningful LFN measurements for model verification is always a rather difficult task. Because of an extremely low level of the current fluctuations that have to be measured, great care must be taken to keep the background noise (BN) of the measurement system as low as possible. A lot of effort has been devoted to developing a measurement system characterized by a sufficiently low level [1].

Mainly, there are three different behaviours of the noise in the low frequency domain of MOS devices. The power spectrum density (PSD) plot of Figure 1-1-1 shows two of these behaviours. At higher frequencies, the figure shows the white noise and at lower frequencies, it shows the flicker noise. The flicker noise is characterized by a $1/f$ or pink spectrum behaviour. Therefore, it is commonly named as $1/f$ noise or pink noise. In contrast, the white noise is characterized by its frequency independent behaviour [2]. In the large devices 1/*f* in noise spectrum is noticed nicely due to the capture and release large number of carriers from the channel to the device defects or traps.

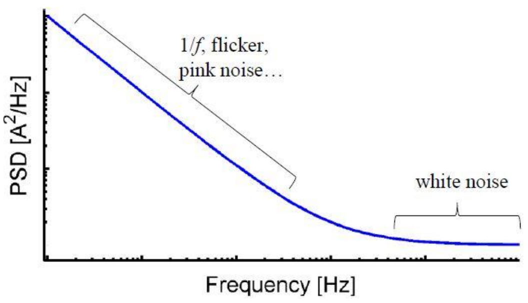


Figure 1-1-1: The noise power spectral density of an arbitrary MOS transistor [2]

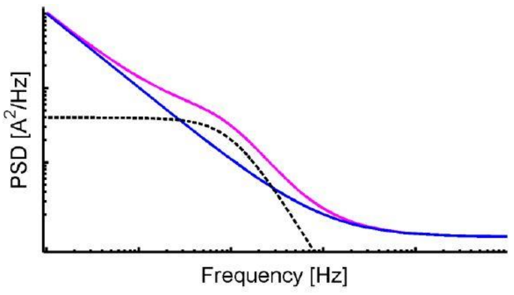


Figure 1-1-2: Example of a bump present in the LFN spectrum [2]

Therefore, flicker noise of a MOS device can be measured and studied directly from the PSD vs. frequency graph.

In addition to the flicker and white noise, it is also very common to see bumps on the noise spectra, as shown in Figure 1-1-2. These bumps are the result of the prominence of the effect of individual defects in the device. This effect adds Lorentzian-like spectrum to the already existing noise PSD. This low frequency noise behaviour is commonly referred to as Random Telegraph Noise (RTN) and it is often found on smaller devices due to the capture and release very few numbers of carriers (one or two) from the channel to the device defects or traps.

For flicker noise measurement, B2962A (Low noise power source) [9, 10], SR570 (low noise current pre amplifier) [11] and SR780 (Spectrum analyser) [12] instruments are used. B2962A is interfaced to the computer through USB A to USB B cable and other two instruments are interfaced using RS232 to USB serial converter. NI LabView programming, which is a laboratory virtual instrumentation electronic workbench, has been used to automate the control of these instruments. The reasons of using LabView instead of Visual C++ programming for flicker noise measurement automation is that LabView is a GUI programming software where plotting of output directly coming from the instrument and saving of measured data are done simultaneously and also no need to remember large number of command lists of all instruments (three instruments) of flicker noise measurement setup. Different type of controls of different instruments in flicker noise measurement setup like selection of voltage output channel, voltage level with its range, current compliance range, input offset current, input offset bias, different gain mode of amplifier, filter type selection, output display selection of instrument, change of output plot scale, resolution of FFT measurements are done using a single LabView front panel control. Also different measurements like time domain, FFT and PSD have been done with appropriate units by controlling the front panel of the LabView program.

1.2 Random Telegraph Noise (RTN)

Random telegraph noise (RTN) represents one of the most important reliability issues for modern deca-nanometer MOS technologies, attracting many research efforts devoted to the understanding of its statistical features. In particular, the statistical distribution of the RTN amplitude has been widely investigated, due to its importance in determining the RTN impact on the operation of digital devices, such as Flash memories and SRAM cells [3, 4].

RTN is characterized by the fluctuation of the drain current between two fixed levels with stochastic low- and high-level times, resembling a random telegraph signal (RTS), as shown in Figure A-2-1.These fluctuations are mainly caused by a defect in the gate dielectric or at the dielectric-silicon interface. These defects, or traps, are known to capture and release carriers very few numbers of carriers (one or two) from the channel. Therefore, the two current levels, between which the current fluctuates, represent the state of the trap (empty or occupied). In the case of an acceptor like trap (negatively charged when occupied by a carrier and neutral when not occupied) the current of an n-channel device decreases when the trap captures a carrier and returns to its highest level when the carrier is released back to the channel [2].

In the frequency domain, the RTN associated with a single trap is represented by a Lorentzian function. In the log-log domain, as shown in Figure A-2-2, the Lorentzian is characterized by a plateau region and by a $1/f^2$ region [2].

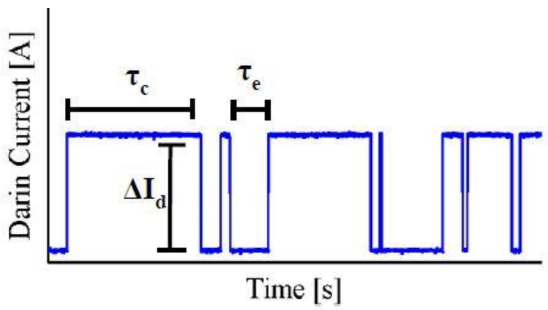


Figure 1-2-1: Illustration of the RTS given by the impact of a single defect in the oxide [2]

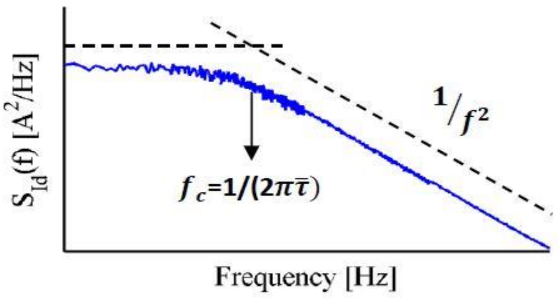


Figure 1-2-2: Illustration of RTN spectrum in log-log domain [2]

The power spectrum density (PSD) of the random telegraph noise, given by a single defect, is calculated as

$$S_{Id}(f) = \frac{4\Delta I_d^2}{(\bar{\tau}_c+\bar{\tau}_e)[\left(\frac{1}{\bar{\tau}_c}+\frac{1}{\bar{\tau}_e}\right)^2+(2\pi f)^2]} \tag{1.2}$$

Where $\bar{\tau}_c$ and $\bar{\tau}_e$ are the mean of the emissions and capture times respectively, and I_d^2 is the amplitude of the drain current fluctuation.

As discussed in previous two sections, to study RTN and flicker noise properly sampling measurement needs to be done for sufficiently long period of time to establish the probability distribution function of the switching times for each of the MOS device in different gate and drain voltages with different drain current sampling times or FFT resolution and same thing needs to repeat for all devices in a wafer. Also to look into the process variability same number of measured output result comes from each of different devices in a wafer. Even it needs to be done for different wafers also. So, the results of this whole process comes with such a huge volume of measured output data (number of measured points in the range of lakhs) that it becomes very tedious, cumbersome and very time consuming to handle these manually. Therefore, instead of manually handling RTN and flicker measurement instruments, these are remotely interfaced in a desktop computer and automated the control of these instruments using various programming software. In this project work Visual C++ and LabView programming software are used to automate RTN and flicker noise measurement instruments respectively. As a result of this automation of instruments, RTN and flicker noise measurement, measured data acquisition and analyse these data become easier, efficient and very low time consuming.

For RTN measurement, Keysight’s B1500 Semiconductor device analyser [5] integrated with B1530A waveform generator/fast measurement unit (WGFMU) [6, 7] is used. The B1500A is remotely interfaced to a desktop PC through GPIB cable. Microsoft Visual C++ programming has been used to automate RTN measurement. Programming for both IV and sampling measurements, WGFMU measurement module of B1500 instrument is used. Drain current sampling measurements using WGFMU module have been done for both fixed DC and pulsed biases. Sampling measurement programs are made for single sampling interval. But these programs can be extended for getting multiple sampling interval measurement outputs in a single program run. All the possible extension of these RTN measurement programs to do measurements more quickly and efficiently are discussed in discussion section of this report.

1.3 Thesis Overview

The details of RTN and flicker noise with its measurement techniques and related functions or commands that can be used to control the measurement instruments remotely are pretty long and an extensive literature. There are many research articles on LFN (RTN and flicker noise) measurement in different ways but there are very few papers which have done the RTN and flicker noise measurement instruments automation using Visual C/C++ and LabView programs. I wish to cover in this short review of all instrument setup, measurement modules, necessary functions or commands which are used to control the instruments through a PC. In **chapter 1** background and introduction of this work are illustrated. Measurement instruments and automation setups are explained in **chapter 2**. All necessary hardware arrangements for RTN measurement using B1500 instrument like automated measurement setup, modules and different operation modes for developing Microsoft Visual Studio C++ code to automate RTN measurement are discussed in section 2.1. In sub sections 2.1.1.1 and 2.1.1.2 WGFMU instrument library functions and available parameter values are discussed respectively. All necessary hardware arrangements for flicker measurement like automated measurement setups, different communication cables and converter to interface instruments to the computer are described at the section 2.2. Remote control mode and command lists which are used to develop LabView program in this work of the instruments B2962A, SR570 and SR780 are described in the sub sections 2.2.1, 2.2.2 and 2.2.3 respectively. In **chapter 3** the details of VC++ programming for automation of both I-V and sampling measurements of RTN measurement are described. All necessary software aspects, including library files of WGFMU module before run the VC++ program are clearly directed in section 3.1. Sections 3.2 and 3.3 are dedicated to the all I-V and RTN measurement VC++ programming and the outputs. In **chapter 4** the details of programming codes using LabView of flicker noise measurement instruments are described with corresponding test code demonstrations and outputs. All necessary software and drivers to automate all flicker noise measurement instruments before run the LabView programs are clearly directed in section 4.1. Sections 4.2, 4.3 and 4.4 are dedicated to the LabView programs with useful controls of B2962A, SR570 and SR780 instruments respectively. LabView program outputs are shown in section 4.5. I have discussed about the program outputs and the possible extension of the programs to do large number of measurements at a time in **discussion** section of **chapter 5**. The challenges of this work are discussed briefly in **challenges** section and also I have concluded in **conclusion** section with a summary and a discussion of the future direction in this field.

Chapter 2

Measurement Instruments and Automation Setup

2.1 RTN Measurement Instruments and Setup

Random telegraph noise (RTN) has become an increasing concern for reliability testing of MOS devices in nano scaled technologies affecting critical circuit parameters, such as delay and noise margin. RTN is attributed for the random capturing and emitting of individual charge carriers in the dielectric gate defects/traps from the channel for the small area devices. This is a characteristic of the RTN caused by the effect of individual defects, each of which adds a Lorentzian to the noise spectrum [Appendix A.2].

B1500A Semiconductor Device Analyzer

Agilent B1500A Semiconductor Device Analyzer has been used for RTN measurement which is the new generation one box solution for the semiconductor device DC/AC parametric measurement and analysis application [5]. It supports several plug-in modules like GNDU, HRSMU, MFCMU, SPGU, WGFMU. These measurement modules are supported by the EasyEXPERT software which is the system software of the B1500A. However the WGFMU is not supported by the EasyEXPERT Classic Test operation.

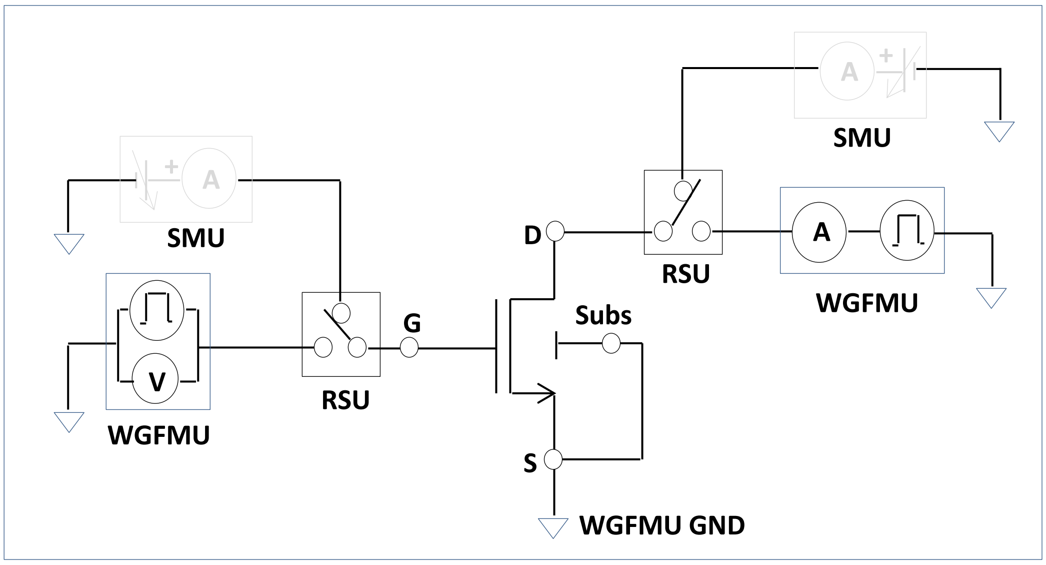


Figure 2-1: RTN measurement setup

As shown in the setup above, WGFMU and SMU as measurement module for DC and sampling measurements are used. Here, the emphasis only on WGFMU coding and to automate its control for both I-V and sampling measurements. Connection to SMU is not needed for this whole work. RSU is used to switch between the WGFMU and SMU without having to change any cabling. In addition to that RSU is designed to be mounted on the wafer prober close to the DUT to optimize measurement performance.

Remote control mode

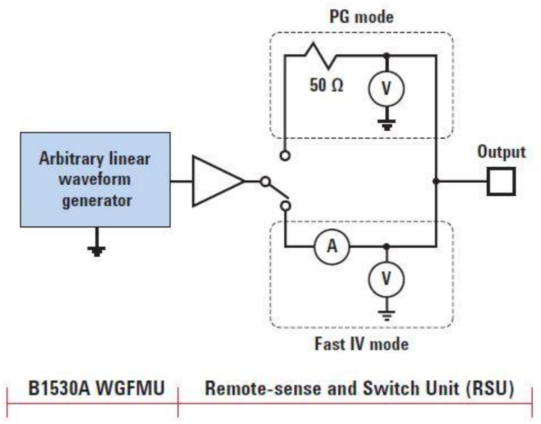
The B1500A instrument can be controlled in USB or GPIB or LAN remote control mode from an external computer. The EasyEXPERT is the GUI based measurement control and analysis software runs on the Microsoft Windows XP Professional. In this work B1500A and its measurement module (WGFMU) are controlled through GPIB to USB connector and automated using Microsoft visual C++ programming software. Details of measurement module control and programming are illustrated in next sections.

2.1.1 Waveform Generator/Fast Measurement Unit (WGFMU)

This section introduces Keysight B1530A waveform generator/fast measurement unit (WGFMU) [6] which is a measurement module for the Keysight B1500A Semiconductor Device Analyzer and Keysight B1531A remote-sense and switch unit (RSU). It is the first self-contained module which is offering the combination of arbitrary linear waveform generation (ALWG) with synchronized fast I-V measurement. It also enables accurate high-speed IV characterization. The B1530A WGFMU provides with powerful AC and DC waveform generation and high-speed. Since the B1530A WGFMU is a module, we can easily add this high-speed measurement capability to our existing B1500A. This reduces the overall cost of test if you already own a B1500A.

Different mode of operation

There are two available modes of operation in each B1530A WGFMU channel. These are Fast IV (current and voltage measurement) mode and PG mode (pulse generator). The Fast IV and PG modes can run independently on each available B1530A WGFMU channel. The WGFMU has the big advantage of being able to measure currently directly, without the need for an external resistor and the complex cabling and custom circuitry required when using the pulse generator and oscilloscope method. The channel also can be the DC voltage source. The simplified measurement circuit diagram is shown figure below.


PG mode
50 Ω
V
Arbitrary linear waveform generator
Output
A
V
Fast IV mode
B1530A WGFMU
Remote-sense and Switch Unit (RSU)
Fig. 2-1-1: Simplified B1530A WGFMU schematic

Fast IV mode

In Fast IV mode the WGFMU channels can create arbitrary waveforms via the ALWG function and can measure both current and voltage. In this mode WGFMU can make current measurements with 2nA resolution and with sampling speeds as fast as 5 ns, providing measurement capabilities not offered by existing solutions.

PG mode

WGFMU channels can create narrower pulses with the ALWG function than they can in Fast IV mode in PG mode and they can measure voltage. Also, in this mode the WGFMU channels have a 50 Ω output impedance to prevent reflection-induced waveform degradations.

Output voltage range of different operation mode

Table 2-1-1

Operation mode	Voltage output range	Voltage	Setting resolution
PG	3V fixed range	-3V to +3V	96 µV
	5V fixed range	-5V to +5V	160 µV
Fast IV	3V fixed range	-3V to +3V	96 µV
	5V fixed range	-5V to +5V	160 µV
	-10V fixed range	-10V to 0V	160 µV
	+10V fixed range	0V to +10V	160 µV
DC	3V fixed range	-3V to +3V	96 µV
	5V fixed range	-5V to +5V	160 µV
	-10V fixed range	-10V to 0V	160 µV
	+10V fixed range	0V to +10V	160 µV

Table 2-1-1: WGFMU output voltage value

The WGFMU can be controlled by the programs which use the Instrument Library furnished with the B1530A. The main emphasis of this work is to control WGFMU remotely using visual C++ programming for drain current sampling measurements. In the next sections WGFMU instrument library functions and parameters values which are used to automate it using visual C++ programming are discussed.

2.1.1.1 WGFMU Instrument library functions

Function name depends on the programming environment. In this paper I'm using Microsoft Visual C++ IDE for WGFMU codding. So for Microsoft Visual C++ .NET (or Visual Basic .NET, Visual Basic 6.0, or VBA) it's written as: WGFMU_functionName (ex. WGFMU_openSession)

In the table below some WGFMU instrument functions are listed. These functions have been used for DC measurement and drain current sampling measurements in this report. There are many other such functions also present to control WGFMU. The detailed syntax of these functions with all function parameters is in B1530 User Guide [6].

Table 2-1-1-1

Group	Function	Description
Common Initialize	WGFMU_openSession	Opens the communication session with the B1500A by using the WGFMU
	WGFMU_closeSession	Closes the communication session with the B1500A by using the WGFMU
	WGFMU_initialize	Resets all WGFMU channels.
	WGFMU_setTimeout	Sets timeout of the present session.
Common Error and Warning	WGFMU_getErrorSize	This function returns the length of the next error string
	WGFMU_getError	This function reads one error string
	WGFMU_openLogFile	Opens a file used to log errors and warnings
	WGFMU_closeLogFile	Closes a file used to log errors and warnings
Common Setup	WGFMU_setOperationMode	Sets the operation mode of the specified WGFMU channel, Fast IV, PG, DC operation mode
	WGFMU_setMeasureMode	Sets or reads the measurement mode, voltage or current measurement mode
	WGFMU_getMeasureMode	
	WGFMU_setForceVoltageRange	Sets the voltage output range of the specified source channel
	WGFMU_setMeasureCurrentRange	Sets or reads the current measurement range of the specified measurement channel.
	WGFMU_getMeasureCurrentRange	

Common Measurement	WGFMU_connect	Enables or disables the specified WGFMU channel and the RSU connected to the WGFMU
	WGFMU_disconnect	
WGFMU Initialize	WGFMU_clear	Clears the instrument library's software setup information such as all pattern and sequence information.
WGFMU Setup Pattern	WGFMU_createPattern	Creates a waveform pattern
	WGFMU_addVector	Specifies scalar data (time and voltage) and adds it to the specified waveform pattern or replaces the scalar previously defined in the specified waveform pattern with the scalar specified by this function.
	WGFMU_addVectors	
WGFMU Setup Event	WGFMU_setMeasureEvent	Defines a measurement event which is a sampling measurement performed by the WGFMU channel while it outputs a waveform pattern
WGFMU Setup Sequence	WGFMU_addSequence	Adds sequence data (pattern and count) to the source output sequence defined in the specified WGFMU channel
	WGFMU_addSequences	
WGFMU Setup check Sequence	WGFMU_getInterpolatedForceValue	Reads the voltage value applied by the specified WGFMU channel at the specified time
WGFMU Measurement	WGFMU_execute	Runs the sequencer of all enabled WGFMU channels
	WGFMU_waitUntilCompleted	Waits until all connected WGFMU channels are in the ready to read data status
WGFMU Data retrieve Measurement value	WGFMU_getMeasureValueSize	Reads the measurement data (time and voltage or current) for the measurement point defined in the sequences set to the specified WGFMU channel
	WGFMU_getMeasureValue	
WGFMU Export setup data	WGFMU_exportAscii	Creates a setup summary report and saves it as a csv (comma separated values) file
DC Measurement	WGFMU_dcforceVoltage	Starts DC voltage output immediately by using the specified WGFMU channel
	WGFMU_dcmeasureValue	Starts a voltage or current measurement immediately by using the specified WGFMU channel and returns the measurement value (voltage or current)

Table 2-1-1-1: WGFMU Instrument Library Functions

2.1.1.2 WGFMU available parameter values

Table below shows the available parameter values (constants) for the specific functions. As in this report Microsoft Visual C++ IDE is used for WGFMU codding. These parameters are only for Microsoft Visual C++ .NET (or Visual Basic .NET, Visual Basic 6.0, or VBA) programming environment.

Table 2-1-1-2

WGFMU Functions	**Mode**	**Description**
WGFMU_setOperationMode And WGFMU_getOperationMode	WGFMU_OPERATION_MODE_DC	DC mode. DC voltage output and voltage or current measurement (VFVM or VFIM). The functions of the DC Measurement group are available in this mode only.
	WGFMU_OPERATION_MODE_FASTIV	Fast IV mode. ALWG voltage output and voltage or current measurement (VFVM or VFIM).
	WGFMU_OPERATION_MODE_PG	PG mode. ALWG voltage output and voltage measurement (VFVM). The output voltage will be divided by the internal 50Ω resistor and the load impedance. Faster than the Fast IV mode.
WGFMU_setForceVoltageRange And WGFMU_getForceVoltageRange	WGFMU_FORCE_VOLTAGE_RANGE_AUTO	Auto range [a], default setting
	WGFMU_FORCE_VOLTAGE_RANGE_3V	3V fixed range [a] (-3 V to +3 V)
	WGFMU_FORCE_VOLTAGE_RANGE_5V	5V fixed range [a] (-5 V to +5 V)
	WGFMU_FORCE_VOLTAGE_RANGE_10V_NEGATIVE	-10V fixed range [a] (-10 V to 0 V)

	WGFMU_FORCE_VOLTAGE_RANGE_10V_POSITIVE	10V fixed range [a] (+10 V to 0 V)
WGFMU_setMeasureMode And WGFMU_getMeasureMode	WGFMU_MEASURE_MODE_VOLTAGE	Voltage measurement mode [a], default Setting .Changing the mode to this mode does not change the current measurement range setting.
	WGFMU_MEASURE_MODE_CURRENT	Current measurement mode [b] Changing the mode to this mode changes the voltage measurement range to the 5 V range.
WGFMU_setMeasureCurrentRange WGFMU_getMeasureCurrentRange And WGFMU_setRangeEvent	WGFMU_MEASURE_CURRENT_RANGE_1UA	1 µA controlled range (±1 µA)
	WGFMU_MEASURE_CURRENT_RANGE_10UA	10 µA controlled range (±10 µA)
	WGFMU_MEASURE_CURRENT_RANGE_100UA	100µA controlled range (±100 µA)
	WGFMU_MEASURE_CURRENT_RANGE_1MA	1 mA controlled range (±1 mA)
	WGFMU_MEASURE_CURRENT_RANGE_10MA	10 mA controlled range (±10mA),default setting
WGFMU_setMeasureEvent	WGFMU_MEASURE_EVENT_DATA_AVERAGED	Averaging data output mode. Only the averaging result data will be returned and the number of returned data will be points.

Table 2-1-1-2: Parameter values for the specified WGFMU functions

a. These are available for the Fast IV, PG, and DC operation mode.

b. These are available only for the Fast IV and DC operation mode. Not available for the PG mode.

2.1.1.3 WGFMU Program Execution Flow

WGFMU control program has been executed by using the program execution flow [6] shown in the below table.

The WGFMU online session is started by the WGFMU_openSession function and is ended by the WGFMU_closeSession function. This means that the functions for the step 1 to 3 can be used in the offline condition which the WGFMU is not connected.

Table 2-1-1-3

Step	Action	Function
	Starts error and warning logging (Optional)	WGFMU_openLogFile
	Clears instrument library (Optional)	WGFMU_clear
1	Creates pattern data	WGFMU_createPattern
		WGFMU_addVector
2	Defines several events	WGFMU_setMeasureEvent
3	Creates WGFMU channel output and measurement control data	WGFMU_addSequence
		WGFMU_addSequences
4	Opens session	WGFMU_openSession
	Initializes WGFMU channels (Optional)	WGFMU_initialize
5	Sets measurement condition	WGFMU_setOperationMode
		WGFMU_setMeasureMode
		WGFMU_setMeasureCurrentRange
		WGFMU_setMeasureVoltageRange
		WGFMU_getMeasureValueSize
		WGFMU_getMeasueValue
6	Enables WGFMU channels	WGFMU_connect
7	Starts output and measurement	WGFMU_execute
8	Disables WGFMU channels	WGFMU_disconnect
9	Closes session	WGFMU_closeSession
	Stops error and warning logging (Optional)	WGFMU_closeLogFile

Table 2-1-1-3: WGFMU Program execution flow

2.2 Flicker Noise measurement Instruments and setup

Flicker noise measurement is another reliability testing method like RTN in low frequency domain. The flicker noise is characterized by a *1/f* or a pink spectrum behaviour. Therefore, it is commonly named as *1/f* noise or pink noise. In device physics point of view the main difference between RTN and flicker noise is that RTN is attributed due to the random capturing and emitting of individual charge carriers in individual gate dielectric traps, but flicker noise (*1/f* behaviour) is related with the large number of charge perturbation in the gate oxide which is considered as collective behaviour of individual charge perturbation. An important design subject for IC designers is how to minimize the noise contribution in the circuit; thus, accurate and timely flicker noise models for semiconductor devices are needed [2].

In this section hardware requirement for LFN measurement is discussed. LFN measurement setups, instruments, necessary command to control the instruments are discussed in this section. LFN measurement setup and all necessary instruments are shown in below block diagram.

Measurement Setup

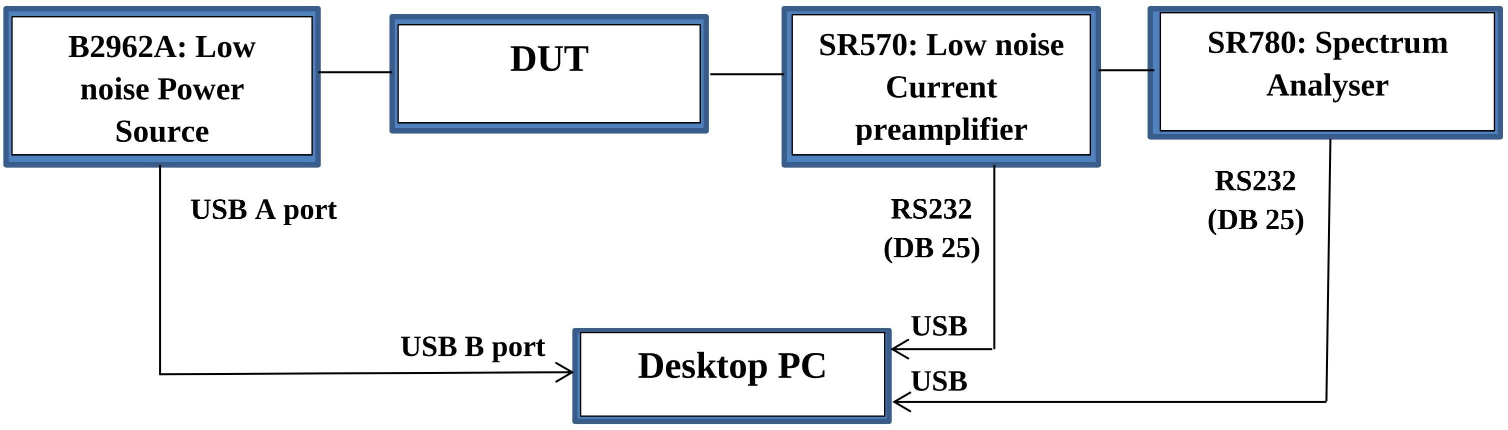


Figure 2-2: Flicker Noise Measurement Setup and Instruments

- B2962A can be interfaced to PC through GPIB to USB connector or USB A to USB B connector
- SR570 can be interfaced only through RS232 (DB25) connector
- SR780 can be interfaced to PC through RS232 (DB 25) to USB connector, GPIB to USB connector or LAN Connector

As GPIB to USB is costly, RS232 to USB connector has been used for interfacing SR780 and USB A to USB B connector has been used for interfacing B2962A instrument.

2.2.1 B2962A: Low noise power source

Keysight B2962A is 6½ digit low noise power source. It supports several functions, such as arbitrary waveform generation, pulse output, sweep output, trace buffer, math expressions, and graph plot. Hence, B2962A can be used as a DC (constant) voltage/current source, sweep voltage/current source, pulse generator, arbitrary waveform generator and multimeter. Device under test (DUT) can be connected to its source/measure terminals. For the all details about the instrument and the connection procedure of DUT refer B2962A user guide manual [9].

2.2.1.1 Remote Control mode of B2962A

B2962A supports GPIB, LAN, and USB interfaces and can be controlled in remote control mode using these interfaces from an external computer. All three interfaces are live at power-on. As discussed in the section 1.2.1 GPIB to USB connection is costly, this report is only based on the automation of B2962A using USB A to USB B remote interface to a PC. We can interface it by installing Keysight IO Libraries Suite and connecting the USB device port located on the back of the instrument to the USB port on the computer. LabView programing software is used to automate the instrument. If the computer is not automatically recognising to control it, we have to follow some steps before programming which are illustrated in the chapter 4. Sometimes it’s necessary to put the instrument’s VISA address manually during interfacing, in that case instrument’s VISA address can be viewed from the front panel by pressing the More function key, then the I/O > USB softkeys. Commands to control the instrument are listed in the next section.

2.2.1.2 Command list

Table 2-2-1-2 shows the command list with syntax which are used to execute LabView program of B2962A low noise power source instrument. For details of command list with syntax refer the reference [10]. Following commands are listed in same sequence as LabView program is executed.

Table 2-2-1-2

Type	Command	description
Confirming the Firmware Revision	*IDN	Instrument's (mainframe) identification and firmware revision are read by this command.
Resetting to the Initial Settings	*RST	The initial settings are applied by this command.
Setting the Power Frequency	:SYST:LFR	Power line frequency is set by this command.
Enabling or Disabling the Over Voltage/Current Protection	:OUTP:PROT :OUTP:PROT:ON :OUTP:PROT:OFF	Over voltage/current protection is set by this command. It can be on or off.
Setting the Limit/Compliance Value	:SENS:CURR/VOLT:PROT	Limit/compliance is set by this command.
Pulse output	:SOUR:FUNC PULS	Pulse output is set by this
DC output	:SOUR:FUNC DC	DC output is set by this command
Setting the Source Output Mode	:SOUR:FUNC:MODE CURR/VOLT	Source output mode is set by this command
Applying the DC Voltage/Current	:SOUR:FUNC:TRIG:CONT	DC current/voltage is immediately applied when we want to apply DC current/voltage output timing using a trigger
Setting the Output Range	:SOUR:VOLT:RANG:AUTO	Output range is set and auto range operation is enable or disable by this command
Setting the Output Range with lower limit	:SOUR:VOLT:RANG:AUTO:LLIM	The lower limit for the auto range operation is set by this command
Enabling or disabling the Source Output	:OUTP ON/OFF	Command Source output is enabled by this command
Reading an Error Message	:SYST:ERR?	Error message is read one by one by using this command.
Query for over current or voltage protection tripped or not	:SENS:CURR/VOLT:PROT:TRIP?	Indicates the over current or voltage protection is tripped or not
Performing Spot Measurement	:MEAS?	Spot measurement is performed by this command

Table 2-2-1-2: B2962A instrument command list

2.2.2 SR570: Low noise current preamplifier

The SR570 is a low-noise current preamplifier, providing a voltage output proportional to the input current. The sensitivity range of the instrument is from 1 mA/V down to 1 pA/V [11].

Specifications

The DC voltage at the input can be set as a virtual null or biased from -5V to +5V. An input offset current from 1pA to 1 mA may also be introduced with uncelebrated offset facility. User can choose between low noise, high bandwidth, and low drift settings, and can invert the output relative to the input. Two configurable R-C filters are provided to selectively condition signals in the frequency range from DC to 1 MHz.

Connectivity

The SR570 normally operates with a fully floating ground with the amplifier ground isolated from the chassis and the AC power supply. Input blanking, output toggling and listen-only RS-232 interface lines are provided for remote instrument control. These lines are optically isolated to reduce signal interference. Digital noise is eliminated by shutting down the processor clock when not executing a front-panel button press or an RS-232 command.

Battery

Internal sealed lead-acid batteries provide up to 15 hours of line-independent operation. Rear panel banana jacks provide access to the internal regulated power supplies (or batteries) for use as a voltage source.

RS232 Interface

The RS-232 interface allows listen-only communication with the SR570 at 9600 baud. All functions of the instrument (except power on) can be set via the RS-232 interface. The interface is configured as listen-only, 9600 baud DCE, 8 data bits, no parity, 2 stop bits. TheRS-232 interface electronics are opto-isolated from the amplifier circuitry to provide maximum noise immunity.

LED indicators

The ERROR LED on the front panel will light if the SR570 receives an unknown or improperly worded command. The LED will remain lit until a proper command is received and after getting the proper command corresponding light will lit up which indicates that the instrument is working properly.

2.2.2.1 Remote control mode of SR570

The SR570 can be remotely interfaced only using a standard DB-25 RS-232 connector on the rear panel and using this interface all functions of the instrument except the power button can be controlled. The interface is configured as listen-only, 9600 baud DCE, 8 data bits, no parity, 2 stop bits, and is optically isolated to prevent any noise or grounding problems. Data are sent to the instrument on RS232 connector pins 2 and 3, which are shorted together. The data flow control pins (5, 6, 8, 20) are shorted to each other. The ground pins (1 & 7) are connected to each other but optically isolated from the amplifier circuit ground and the chassis ground. Data in and out on the connector are tied together, echoing data back to the sender. Hardware handshaking lines CTS, DSR, and CD are tied to DTR.

LabView programing software is used to automate the instrument through the RS232 to USB serial converter. Computer automatically recognises the instrument after installing some necessary drivers and software before the programming which are illustrated in the chapter 4. All RS-232 commands consist of four letter codes followed, in most cases, by an integer value (n). Commands must end with a carriage return and line feed <CR> <LF>. Commands to control the instrument are listed in the next section.

2.2.2.2 Abridged RS232 Command list

Table 2-2-2-2 shows the command list which are used to execute LabView program of SR570 low noise current pre amplifier instrument. For details of commands with detailed syntax refer the reference [11]. Following commands are listed in same sequence as LabView program of the instrument is executed. Value of n is also given in the above table for most of the functions and the functions which have a range of n values. Details of n values are in the reference [11].

Table 2-2-2-2

Type	Command	Description
Sensitivity control commands	SENS n	Sets the sensitivity of the amplifier. n ranges from 0 (1 pA/V) to 27 (1 mA/V).
	SUCM n	Sets the sensitivity cal mode. 0 = cal, 1 = uncal.

	SUCV n	Sets the uncalibrated sensitivity vernier. [0 ≤ n ≤ 100] (percent of full scale).
Input Offset Current control commands	IOON n	Turns the input offset current on (n=1) or off (n=0).
	IOLV n	Sets the calibrated input offset current level. n ranges from 0 (1 pA) to 29 (5 mA).
	IOSN n	Sets the input offset current sign. 0 = neg, 1 = pos.
	IOUC n	Sets the input offset cal mode. 0 = cal, 1 = uncal.
	IOUV n	Sets the uncalibrated input offset vernier. [-1000 ≤ n ≤ +1000] (0 - ±100.0% of full scale).
Filter control commands	FLTT n	Sets the filter type. 0=6 HP, 1=12 HP, 2=6 BP, 3=6 LP, 4=12 LP, and 5=none.
	LFRQ n	Sets the value of the lowpass filter 3dB point. n ranges from 0 (0.03Hz) to 15 (1 MHz).
	HFRQ n	Sets the value of the highpass filter 3dB point. n ranges from 0 (0.03Hz) to 11 (10 kHz).
Bias Voltage control commands	BSON n	Turns the bias voltage on (n=1) or off (n=0).
	BSLV n	Sets the bias voltage level in the range. [-5000 ≤ n ≤ +5000] (-5.000 V to +5.000 V).
Other commands	GNMD n	Sets the gain mode of the amplifier. 0=low noise, 1=high bw, 2=low drift.
	INVT n	Sets the signal invert sense. 0=non-inverted, 1=inverted.
	BLNK n	Blanks the front end output of the amplifier. 0=no blank, 1=blank.
	*RST	Resets the amplifier to the default settings.

Table 2-2-2-2: SR570 instrument command list

2.2.3 SR780: Dynamic Signal Analyzer

The SR780 Dynamic Signal Analyzer offers high performance and low cost in a single instrument. It offers 102.4 kHz FFTs with 90 dB dynamic range, swept-sine measurements, ANSI standard octave analysis, waterfall displays, and transient capture for less than half the cost of other similarly equipped analyzers [12].

Spectrum Analysis

The SR780 delivers true two-channel, 102.4kHz FFT performance. Its fast 32-bit floating-point DSP processor gives the SR780 a 102.4kHz real-time rate with both channels selected. Two precision 16-bit ADCs provide a 90 dB dynamic range in FFT mode. It has selectable 100 to 800 line analysis which optimizes time and frequency resolution and also with the zooming capability in on any portion of the 102.4 kHz ranges with a frequency span down to 191 mHz. Two displays of the instrument function independently. Separate frequency spans, starting frequencies, number of FFT lines, or averaging modes for each display can be chosen in it. Sampling rate can also be varied. There are two sampling rates in this instrument (256 kHz or 262 kHz).

2.2.3.1 Remotely Control mode of SR780

The SR780 Network Signal Analyzer may be remotely programmed via either the RS232 or GPIB (IEEE-488) interfaces. Any computer supporting one of these interfaces may be used to program the SR780. Both interfaces are receiving at all times, however, the SR780 will send responses only to the Output Interface specified in the [System] <Remote> menu. All responses are directed to the interface selected by <Output To> in the [System] <Remote> menu, regardless of which interface received the query. The SR780 is configured as a DCE (transmit on pin 3, receive on pin 2) device and supports CTS/DTR hardware handshaking. The CTS signal (pin 5) is an output indicating that the SR780 is ready, while the DTR signal (pin 20) is an input that is used to control the SR780's data transmission. If desired, the handshake pins may be ignored and a simple 3 wire interface (pins 2,3 and 7) may be used. The RS232 interface Baud Rate, Word Length, and Parity must be set in the [System] <Remote> menu. In all three places like the SR780 instrument, LabView program front panel and in the serial port connection of PC 9600 baud rate, 8 data bits, no parity, 1 stop bits are used for better connectivity and data transfer. To assist in programming, the SR780 has 4 interface indicators which are displayed at the top of the screen. The RS232/GPIB indicator shows ‘RS232’ if the interface responses are directed to the RS232 serial port and ‘GPIB’ and the address if the interface responses are directed to the GPIB port. The common indicator flashes ‘RS232’ when there is activity on the RS232 interface and ‘GPIB’ when there is activity on the GPIB interface. It’s necessary to use the OUTX command at the beginning of every program to direct the SR780 responses to the correct interface.

As discussed in the section 1.2.1, GPIB to USB connection is costly this report is only based on the automation of SR780 using DB-25 RS232 to USB serial converter remote interface to a PC. LabView programing software is used to automate the instrument through the serial converter. Computer automatically recognises the instrument after installing some necessary drivers and software before the programming which are illustrated in the chapter 3. The terminator must be a linefeed <lf> or carriage return <cr> on RS232. Commands to control the instrument are listed in the next section

.

2.2.3.2 Command list

Table 2-2-3-2 shows the command list which are used to execute LabView program of SR780 Spectrum Analyzer instrument and for details of commands with detailed syntax refer the reference [12]. Following commands are listed in same sequence as LabView program of the instrument is executed. The detail syntax of command can be clarified from the SR780 spectrum analyser user guide [13].

Table 2-2-3-2

Type	Command	Description
Interface commands	*RST	The *RST command resets the SR780 to its default configurations. The communications setup is not changed.
	*IDN	The *IDN ? query returns the SR780's device identification string. This string is in the format "Stanford_Research_Systems,SR780, s/n00001, ver007".
System Commands	OUTX	The OUTX command sets (queries) the Output Interface. A parameter selects GPIB or RS232 .
Display Setup command	MGRP	The MGRP command sets (queries) the Measurement Group.
	MEAS	The MEAS command sets (queries) the Measurement. The parameter selects the measurement from the list specified in the next section. Only those measurements available in the current Measurement Group are allowed.

	VIEW	The VIEW command sets (queries) the view of display. The parameter selects the view from the list specified in the next section.
Source Command	SRCO	The SRCO command sets (queries) the Source On or Off.
	STYP	The STYP command sets (queries) the Source Type.
Sine Source Command	S1FR	The S1FR command sets (queries) the Frequency of Sine Tone 1.
	S1AM	The S1AM command sets (queries) the Amplitude of Sine Tone 1.
	SOFF	The SOFF command sets (queries) the Offset of the Sine Source.
	S2FR	The S2FR command sets (queries) the Frequency of Sine Tone 2.
	S2AM	The S2AM command sets (queries) the Amplitude of Sine Tone 2.
Noise Source Command	NAMP	The NAMP command sets (queries) the Noise Amplitude.
	NTYP	The NTYP command sets (queries) the Noise Type.
	NBUR	The NBUR command sets (queries) the Noise Burst Percentage.
	NPER	The NPER command sets (queries) the Noise Burst Source Period.
	CSRC	The CSRC command sets (queries) the Source Display.
Data Transfer Command	DSPN	This command queries the length of display. A parameter d selects Display A (0) or Display B (1).
	DSPY	This command command queries the data in display. A parameter d selects Display A (0) or Display B (1).
	DSPW	This command queries the data in waterfall display. A parameter d selects Display A (0) or Display B (1). This command is not valid if display does not have waterfall storage on.
	DSPB	This command returns the data in display d in binary format. A parameter d selects Display A (0) or Display B (1). This command is only available with the GPIB interface.
	DSWB	This command returns the data in waterfall display in binary format. This command is only available with the GPIB interface. A parameter d selects Display A (0) or Display B (1). This command is not valid if display d does not have waterfall storage on.

Table 2-2-3-2: SR780 instrument command list

Chapter 3

Programming for RTN Measurement

3.1 Software arrangement before programming

Before starting the program, it’s necessary to install necessary instrument libraries and also add include file & additional library path to the programming software. If we fail to do these properly, instrument will not be detected or not be remotely interfaced using PC or if it’s interfaced error will come during the program execution. Following are the steps to avoid interfacing error and compiler error during program execution.

Installing B1530 Instrument Libraries

1. Install the GPIB (IEEE 488) interface to a computer to be an instrument controller.
2. Install the Keysight IO Library Suite (15.0 or later).
3. Install the programming software. In this report Visual C++ 2017 is the programming software (Windows 10 operating system).
4. Install the Keysight B1530A WGFMU Instrument Library.

Work with Keysight Connection Expert and EasyEXPERT

1. Open Keysight Connection Expert and set the GPIB address as B1500’s GPIB if it’s not detected automatically.
2. Leave the Start EasyEXPERT button on the B1500A’s screen display or minimize (but terminate easy expert in the PC before run any measurement program. See caution below).

- **Caution:**

  It is very important to terminate the EasyEXPERT software in PC before run any Visual C++ program written in this report. If any Visual C++ program runs simultaneous with EasyEXPERT software in the PC, the DUT may damage. EasyEXPERT will run only in the B1500A instrument during all measurements using Visual C++ programming.

Add additional include files and library path

In this report B1500 programs are coded in Visual C++ 2017, that’s why adding additional include file and library path is only directly relevant to Visual C++ 2017 programming software.

1. Select console application for new project in VC++ and save the project solution file in document folder of PC.

2. Define additional include file search path(C/C++ → General):
   <user path>\Agilent\B1530A\include which stores the wgfmu.h file.
   <user path>\VISA\ WinNT \include which stores the VISA related include files.

3. Additional library search path:
   <user path>\Agilent\B1530A\lib which stores the wgfmu.lib file.
   <user path>\VISA\WinNT\ktvisa\lib\msc which stores the VISA related library files for Microsoft Visual C++ (only for SMU programs)

4. Additional project link library:
   wgfmu.lib
   visa32.lib (only for SMU programs)

<user path> indicates the folder where the software is installed.

3.2 Visual C++ programming for I-V and Sampling measurements

For correct RTN measurement of MOS devices both I-V and drain current sampling measurements are needed. In this section VC++ program code for I-V and drain current (I_d) sampling measurements are presented. In I-V measurement Visual C++ program for both I_d vs V_g with V_d as a parameter and I_d vs V_d with V_g as a parameter are developed in a single program. Similarly, in sampling measurement Visual C++ program for dc measurement (for getting an operating point of the DUT), drain current sampling measurement for fixed bias (in both gate and drain terminals) and pulsed bias (in gate terminal only) are developed in a single program. Here a NMOS DUT is used. Source terminal of the DUT is always grounded. These programs are only for WGFMU module control automation. Details of all used WGFMU instrument library functions and parameters in the program codes are in the sections 2.1.1.1 and 2.1.1.2.

I-V measurement program

```
#include "pch.h"
#include <stdio.h>
#include <stdlib.h>
#include "wgfmu.h"
#include <visa.h>
#include <iostream>

using namespace std;


// Checks the error being returned from the WGFMU

void checkError(int ret)
{
        if (ret < WGFMU_NO_ERROR) {
                throw ret;
        }
```

```
}
int checkError2(int ret)
{
        if (ret < WGFMU_NO_ERROR) {
                int size;
                WGFMU_getErrorSize(&size);
                char* msg = new char[size + 1];
                WGFMU_getError(msg, &size);
                fprintf(stderr, "%s", msg);
                delete[] msg;
        }
        return ret;
}

static const int VISA_ERROR_OFFSET = WGFMU_ERROR_CODE_MIN - 1;

void checkError3(int ret) //29
{
        if (ret < WGFMU_NO_ERROR && ret >= WGFMU_ERROR_CODE_MIN || ret <
                VISA_ERROR_OFFSET)
        {
                throw ret;
        }
}

// Saves the file from the WGFMU

void writeResults(int channelId, const char* Output1)
{
        //use_CRT_SECURE_NO_WARNINGS;
#pragma warning (disable : 4996)
        FILE* fp = fopen(Output1, "w");
        if (fp != 0) {
                int measuredSize, totalSize;
```

```
		WGFMU_getMeasureValueSize(channelId, &measuredSize, &totalSize);
		for (int i = 0; i < measuredSize; i++) {
			double time, value;


			WGFMU_getMeasureValue(channelId, i, &time, &value);
			fprintf(fp, "%.9lf, %.9lf\n", time, value);
		}
		fclose(fp);
	}
}

// Saves the file from the WGFMU with a row offset

void writeResults2(int channelId, int offset, int size, const char* Output2)
{
	FILE* fp = fopen(Output2, "w");
	if (fp != 0) {
		int measuredSize, totalSize;
		WGFMU_getMeasureValueSize(channelId, &measuredSize, &totalSize);
		for (int i = offset; i < offset + size; i++) {
			double time, value;
			WGFMU_getMeasureValue(channelId, i, &time, &value);
			fprintf(fp, "%.9lf, %.9lf\n", time, value);
		}
		fclose(fp);
	}
}

// Saves the file from the WGFMU with a row offset for each channel

void writeResults3(int channelId1, int channelId2, int offset, int size, const char* fileName)
{
	FILE* fp = fopen(fileName, "w");
```

```
        if (fp != 0) {
                int measuredSize, totalSize;
                fprintf(fp, "Vg,Vd,Id\n");
                WGFMU_getMeasureValueSize(channelId2, &measuredSize, &totalSize);
                for (int i = offset; i < offset + size; i++) {
                        double time, value, voltage1, voltage2;;
                        WGFMU_getMeasureValue(channelId2, i, &time, &value);
                        WGFMU_getInterpolatedForceValue(channelId1, time, &voltage1);
                        WGFMU_getInterpolatedForceValue(channelId2, time, &voltage2);
                        fprintf(fp, "%.9lf, %.9lf,%.9lf\n", voltage1, voltage2, value);
                }
                fclose(fp);
        }
}

// Global Variables

int gateChannel = 101;
int drainChannel = 102;
int polarity = 1;        //For PMOS polarity=-1

// Function for Id vs Vg measurement

void Id_vs_Vg_measurement(double vgMin, double vgMax, double vgStep, double vdMin,
double vdMax, double vdStep)
{
        double vgRiseTime = 100e-9;
        double vgStepLength = 500e-9;
        double vgStepDelay = 200e-9;
        int numberOfVgSteps = (double)((vgMax - vgMin) / vgStep) + 1;
        double vdRiseTime = 100e-9;
        double vdStepLength = (vgRiseTime + vgStepLength) * numberOfVgSteps;
        int numberOfVdSteps = (double)((vdMax - vdMin) / vdStep) + 1;
        WGFMU_openLogFile("C:Id_vs_Vg_errorlog.log");
```

```
// OFFLINE

WGFMU_clear();

// Gate Channel Pattern and Sequence

double vg = vgMin;
WGFMU_createPattern("Vg", vg * polarity);
for (int i = 0; i < numberOfVgSteps; i++) {
        vg = vgMin + vgStep * i;
        WGFMU_addVector("Vg", vgRiseTime, vg * polarity);
        WGFMU_addVector("Vg", vgStepLength, vg * polarity);
}
WGFMU_addSequence(gateChannel, "Vg", numberOfVdSteps);

// Drain Channel Pattern and Sequence

double vd = vdMin;
WGFMU_createPattern("Vd", vd);
for (int i = 0; i < numberOfVdSteps; i++) {
        vd = vdMin + vdStep * i;
        WGFMU_addVector("Vd", vdRiseTime, vd * polarity);
        WGFMU_addVector("Vd", vdStepLength, vd * polarity);
        WGFMU_setMeasureEvent("Vd", "Id", (vdRiseTime + vdStepLength) * i +
                vgRiseTime + vgStepDelay, numberOfVgSteps, vgRiseTime +
                vgStepLength, vgStepLength - vgStepDelay * 2,
                WGFMU_MEASURE_EVENT_DATA_AVERAGED);
}
WGFMU_addSequence(drainChannel, "Vd", 1);
WGFMU_exportAscii("C:waveform_Id_vs_Vg.csv");

// ONLINE

WGFMU_openSession("GPIB0::17::INSTR");
```

```
        WGFMU_initialize();
        WGFMU_setOperationMode(gateChannel,
                WGFMU_OPERATION_MODE_FASTIV);
        WGFMU_setOperationMode(drainChannel,
                WGFMU_OPERATION_MODE_FASTIV);
        WGFMU_setForceVoltageRange(gateChannel,
                WGFMU_FORCE_VOLTAGE_RANGE_AUTO);
        WGFMU_setForceVoltageRange(drainChannel,
                WGFMU_FORCE_VOLTAGE_RANGE_AUTO);
        WGFMU_setMeasureMode(drainChannel,
                WGFMU_MEASURE_MODE_CURRENT);
        WGFMU_setMeasureCurrentRange(drainChannel,
                WGFMU_MEASURE_CURRENT_RANGE_1MA);
        WGFMU_connect(gateChannel);
        WGFMU_connect(drainChannel);
        WGFMU_execute();
        WGFMU_waitUntilCompleted();

        // Save output files
        for (int i = 0; i < numberOfVdSteps; i++) {
                vd = vdMin + vdStep * i;
                char fileName[1024];
                sprintf(fileName, "C:Id-Vg@Vd=%.3fV.csv", (double)vd);
                writeResults3(gateChannel, drainChannel, numberOfVgSteps * i,
                        numberOfVgSteps, fileName);
        }
        cout << "Id vs Vg measurement output is sucessfully saved " << endl;
        WGFMU_initialize();
        WGFMU_closeSession();
        WGFMU_closeLogFile();

}
```

```
// Function for Id vs Vd measurement

void Id_vs_Vd_measurement(double vgMin, double vgMax, double vgStep, double vdMin,
double vdMax, double vdStep)
{
        double vdRiseTime = 100e-9;
        double vdStepLength = 500e-9;
        double vdStepDelay = 100e-9;
        int numberOfVdSteps = (double)((vdMax - vdMin) / vdStep) + 1;
        double vgRiseTime = 100e-9;
        double vgStepLength = (vdRiseTime + vdStepLength)*numberOfVdSteps;
        int numberOfVgSteps = (double)((vgMax - vgMin) / vgStep) + 1;
        double Idsamplinginterval = 500e-9;
        WGFMU_openLogFile("C:IVerrorlog.log");

        // OFFLINE

        WGFMU_clear();

        // Gate Channel Pattern and Sequence
        double vg = vgMin;
        WGFMU_createPattern("Vg", vg * polarity);
        for (int i = 0; i < numberOfVgSteps; i++) {
                vg = vgMin + vgStep * i;
                WGFMU_addVector("Vg", vgRiseTime, vg * polarity);
                WGFMU_addVector("Vg", vgStepLength, vg * polarity);
        }
        WGFMU_addSequence(gateChannel, "Vg", 1);

        // Drain Channel Pattern and Sequence
        double vd = vdMin;
        WGFMU_createPattern("Vd", vd);
```

```
for (int i = 0; i < numberOfVdSteps; i++) {
        vd = vdMin + vdStep * i;
        WGFMU_addVector("Vd", vdRiseTime, vd * polarity);
        WGFMU_addVector("Vd", vdStepLength, vd * polarity);
        WGFMU_setMeasureEvent("Vd", "Id", (vdRiseTime + vdStepLength)*i, 1,
                Idsamplinginterval, vdStepLength - vdStepDelay * 2,
                WGFMU_MEASURE_EVENT_DATA_AVERAGED);
}
WGFMU_addSequence(drainChannel, "Vd", numberOfVgSteps);
WGFMU_exportAscii("C:Id_vs_Vd_waveform.csv");

// ONLINE

WGFMU_openSession("GPIB0::17::INSTR");
WGFMU_initialize();
WGFMU_setOperationMode(gateChannel,
        WGFMU_OPERATION_MODE_FASTIV);
WGFMU_setOperationMode(drainChannel,
        WGFMU_OPERATION_MODE_FASTIV);
WGFMU_setForceVoltageRange(gateChannel,
        WGFMU_FORCE_VOLTAGE_RANGE_AUTO);
WGFMU_setForceVoltageRange(drainChannel,
        WGFMU_FORCE_VOLTAGE_RANGE_AUTO);
WGFMU_setMeasureMode(drainChannel,
        WGFMU_MEASURE_MODE_CURRENT);

WGFMU_setMeasureCurrentRange(drainChannel,
        WGFMU_MEASURE_CURRENT_RANGE_1MA);
WGFMU_connect(gateChannel);
WGFMU_connect(drainChannel);
WGFMU_execute();
WGFMU_waitUntilCompleted();
```

```
        // Save output files

        for (int i = 0; i < numberOfVgSteps; i++) {
                vg = vgMin + vgStep * i;
                char fileName[1024];
                sprintf(fileName, "C:Id-Vd@Vg=%.3fV.csv", (double)vg);
                writeResults3(gateChannel, drainChannel, numberOfVdSteps * i,
                        numberOfVdSteps, fileName);
        }
        cout << "Id vs Vd measurement output is sucessfully saved " << endl;

        WGFMU_initialize();
        WGFMU_closeSession();
        WGFMU_closeLogFile();
}

// Starts the session with the WGFMU and begins the console application.

int main()
{
        //I-V measurement
        //Example for Id-Vg measurement function call
        //Id_vs_Vg_measurement(0,1, 0.01, 0.05, 0.05, 0.01 );
        //(Minimum Vg in volt, Maximum Vg in volt, Vg step in volt, Minimum Vd in volt,
          Maximum Vd in volt, Vd step in volt)

        // Example for Id-Vd measurement function call
        //Id_vs_Vd_measurement(0.5,0.5, 0.1, 0.05,1, 0.01 );
        //(Minimum Vg in volt, Maximum Vg in volt, Vg step in volt, Minimum Vd in volt,
          Maximum Vd in volt, Vd step in volt)
}
```

Sampling measurement program

```
include "pch.h"
#include <stdio.h>
#include <stdlib.h>
#include "wgfmu.h"
#include <visa.h>
#include <iostream>
#include <fstream>

using namespace std;

//Global variables and Constants

int gatechannel = 101;
int drainchannel = 102;
int polarity = 1;                          // For PMOS polarity = -1;
const char* address = "GPIB0::17::INSTR";

// Checks the error being returned from the WGFMU

void checkError(int ret)
{
        if (ret < WGFMU_NO_ERROR) {
                throw ret;
        }
}

int checkError2(int ret)
{
        if (ret < WGFMU_NO_ERROR) {
                int size;
                WGFMU_getErrorSize(&size);
                char* msg = new char[size + 1];
                WGFMU_getError(msg, &size);
                fprintf(stderr, "%s", msg);
                delete[] msg;
        }
        return ret;
}
static const int VISA_ERROR_OFFSET = WGFMU_ERROR_CODE_MIN - 1;

void checkError3(int ret)
{
        if (ret < WGFMU_NO_ERROR && ret >= WGFMU_ERROR_CODE_MIN || ret <
                VISA_ERROR_OFFSET)
        {
                throw ret;
        }
}
// Saves the files containing results of RTN measurements

void writeResults(int channelId, const char* Output1)
{
#pragma warning (disable : 4996)
        FILE* fp = fopen(Output1, "w");
        if (fp != 0) {
                int measuredSize, totalSize;
                WGFMU_getMeasureValueSize(channelId, &measuredSize, &totalSize);
                for (int i = 0; i < measuredSize; i++) {
                        double time, value;
                        WGFMU_getMeasureValue(channelId, i, &time, &value);
                        fprintf(fp, "%.9lf, %.9lf\n", time, value);
                }
                fclose(fp);
        }
}
```

```
// Saves the file from the WGFMU with a row offset

void writeResults2(int channelId1, int channelId2, int offset, int size, const char* Output2)
{
	FILE* fp = fopen(Output2, "w");
	if (fp != 0) {
		int measuredSize, totalSize;
		fprintf(fp, "Time,Vg,Vd,Id\n");
		WGFMU_getMeasureValueSize(channelId2, &measuredSize, &totalSize);
		for (int i = offset; i < offset + size; i++) {
			double time, voltage1, voltage2, value;
			WGFMU_getMeasureValue(channelId2, i, &time, &value);
			WGFMU_getInterpolatedForceValue(channelId1, time, &voltage1);
			WGFMU_getInterpolatedForceValue(channelId2, time, &voltage2);
			fprintf(fp, "%.9lf, %.9lf,%.9lf, %.9lf\n", time, voltage1, voltage2,
value);
		}
		fclose(fp);
	}
}

// Saves the file from the WGFMU with a row offset for each channel

void writeResults3(int channelId1, int channelId2, int offset, int size, const char* fileName)
{
	FILE* fp = fopen(fileName, "w");
	if (fp != 0) {
		int measuredSize, totalSize;
		WGFMU_getMeasureValueSize(channelId2, &measuredSize, &totalSize);
		for (int i = offset; i < offset + size; i++) {
			double time, value, voltage;
			WGFMU_getMeasureValue(channelId2, i, &time, &value);
			WGFMU_getInterpolatedForceValue(channelId1, time, &voltage);
			fprintf(fp, "%.9lf, %.9lf\n", voltage, value);
```

```
                }
                fclose(fp);
        }
}

//Function defined for DC Measurement

void DCMasurement(double gatevol, double drainvol)
{

        double mVal;
        const char* fname = "C:DC_errlog.log";
        WGFMU_openLogFile(fname);
        WGFMU_openSession(address);
        WGFMU_clear();
        WGFMU_initialize();
        WGFMU_setOperationMode(gatechannel, WGFMU_OPERATION_MODE_DC);
        WGFMU_setOperationMode(drainchannel, WGFMU_OPERATION_MODE_DC);
        WGFMU_setForceVoltageRange(gatechannel,
                WGFMU_FORCE_VOLTAGE_RANGE_AUTO);
        WGFMU_setForceVoltageRange(drainchannel,
                WGFMU_FORCE_VOLTAGE_RANGE_AUTO);
        WGFMU_setMeasureMode(drainchannel,
                WGFMU_MEASURE_MODE_CURRENT);
        WGFMU_setMeasureCurrentRange(drainchannel,
                WGFMU_MEASURE_CURRENT_RANGE_100UA);
        WGFMU_connect(gatechannel);
        WGFMU_connect(drainchannel);
        WGFMU_dcforceVoltage(gatechannel, gatevol*polarity);
        WGFMU_dcforceVoltage(drainchannel, drainvol*polarity);
        WGFMU_dcmeasureValue(drainchannel, &mVal);
        WGFMU_execute();
        WGFMU_waitUntilCompleted();
        // Save output file

        fstream file;
        file.open("DC Measurement.txt", ios::out);
        file << "Output Current=" << mVal << endl;
        file.close();
        cout << "DC Measurement output is sucessfully saved " << endl;

        WGFMU_closeSession();
        WGFMU_closeLogFile();
}

//Function for sampling measurement with fixed bias

void Samplingwithfixedbias(double vgInterval, double vg, int numberOfVgMeasurement, int
numberofVgSignal,
        double vdInterval, double vd, int numberOfVdMeasurement, int numberofVdSignal,
        double Idinterval, int numberofIdSampling)
{
        WGFMU_openLogFile("C:Fixedbias_Samp_errlog.log");

        // OFFLINE
        WGFMU_clear();

        // Gate Channel Pattern and Sequence
        WGFMU_createPattern("Vg", vg * polarity);
        for (int i = 0; i < numberOfVgMeasurement; i++) {
                WGFMU_addVector("Vg", vgInterval, vg * polarity);
        }
        WGFMU_addSequence(gatechannel, "Vg", numberofVgSignal);

        // Drain Channel Pattern and Sequence
        WGFMU_createPattern("Vd", vd * polarity);
```

```
for (int i = 0; i < numberOfVdMeasurement; i++) {
        WGFMU_addVector("Vd", vdInterval, vd * polarity);
}

WGFMU_setMeasureEvent("Vd", "Id", 0, numberofIdSampling, Idinterval, 0,
        WGFMU_MEASURE_EVENT_DATA_AVERAGED);
WGFMU_addSequence(drainchannel, "Vd", numberofVdSignal);
WGFMU_exportAscii("C:Fixedbias_samp_waveform.csv");

// ONLINE

WGFMU_openSession(address);
WGFMU_initialize();
WGFMU_setOperationMode(gatechannel,
        WGFMU_OPERATION_MODE_FASTIV);
WGFMU_setOperationMode(drainchannel,
        WGFMU_OPERATION_MODE_FASTIV);
WGFMU_setForceVoltageRange(gatechannel,
        WGFMU_FORCE_VOLTAGE_RANGE_AUTO);
WGFMU_setForceVoltageRange(drainchannel,
        WGFMU_FORCE_VOLTAGE_RANGE_AUTO);
WGFMU_setMeasureMode(drainchannel,
        WGFMU_MEASURE_MODE_CURRENT);
WGFMU_setMeasureCurrentRange(drainchannel,
        WGFMU_MEASURE_CURRENT_RANGE_100UA);
WGFMU_connect(gatechannel);
WGFMU_connect(drainchannel);
WGFMU_execute();
WGFMU_waitUntilCompleted();

// Save output files

char fileName[1024];
```

```
        sprintf(fileName, "C:FB_samp@St=%fs,Vg=%.2fV&Vd=%.2fV.csv",
                (double)Idinterval, (double)vg, (double)vd);
        writeResults2(gatechannel,drainchannel, 0, numberofIdSampling, fileName);
        cout << "Fixed bias sampling measurement output is sucessfully saved " << endl;

        WGFMU_initialize();
        WGFMU_closeSession();
        WGFMU_closeLogFile();
}

//Function for sampling measurement with pulsed bias

void Samplingwithpulsedbias(double groundbiaslength, double vgRiseTime, double
pulselength, double vgFallTime, double vg, int numberofVgSignal, double vdInterval, double
vd, int numberOfVdMeasurement, int numberofVdSignal, double Idinterval, int
numberofIdSampling)
{
        WGFMU_openLogFile("C:Pulsedbias_Samp_errlog.log");

        // OFFLINE

        WGFMU_clear();

        // Gate Channel Pattern and Sequence

        WGFMU_createPattern("Vg", 0);
        WGFMU_addVector("Vg", 0.00025, 0);
        WGFMU_addVector("Vg", 0.0001, vg * polarity);
        WGFMU_addVector("Vg", 0.001, vg * polarity);
        WGFMU_addVector("Vg", 0.0001, 0);
        WGFMU_addVector("Vg", 0.00025, 0);
        WGFMU_addSequence(gatechannel, "Vg", numberofVgSignal);
```

```
// Drain Channel Pattern and Sequence

WGFMU_createPattern("Vd", vd * polarity);
for (int i = 0; i < numberOfVdMeasurement; i++) {
        WGFMU_addVector("Vd", vdInterval, vd * polarity);
}
WGFMU_setMeasureEvent("Vd", "Id", 0, numberofIdSampling, Idinterval, 0,
        WGFMU_MEASURE_EVENT_DATA_AVERAGED);
WGFMU_addSequence(drainchannel, "Vd", numberofVdSignal);
WGFMU_exportAscii("Pulsedbias_samp_waveform.csv");

// ONLINE

WGFMU_openSession("GPIB0::17::INSTR");
WGFMU_initialize();
WGFMU_setOperationMode(gatechannel,
        WGFMU_OPERATION_MODE_FASTIV);
WGFMU_setOperationMode(drainchannel,
        WGFMU_OPERATION_MODE_FASTIV);
WGFMU_setForceVoltageRange(gatechannel,
        WGFMU_FORCE_VOLTAGE_RANGE_AUTO);
WGFMU_setForceVoltageRange(drainchannel,
        WGFMU_FORCE_VOLTAGE_RANGE_AUTO);
WGFMU_setMeasureMode(drainchannel,
        WGFMU_MEASURE_MODE_CURRENT);
WGFMU_setMeasureCurrentRange(drainchannel,
        WGFMU_MEASURE_CURRENT_RANGE_100UA);
WGFMU_connect(gatechannel);
WGFMU_connect(drainchannel);
WGFMU_execute();
WGFMU_waitUntilCompleted();
```

```
        // Save output files

        char fileName[1024];
        sprintf(fileName, "C:PB_samp@St=%fs & Vd=%.2fV.csv",
                (double)Idinterval,(double)vd);
        writeResults2(gatechannel,drainchannel, 0, numberofIdSampling, fileName);
        cout << "Pulsed bias sampling measurement output is sucessfully saved " << endl;

        WGFMU_initialize();
        WGFMU_closeSession();
        WGFMU_closeLogFile();
}

// Starts the session with the WGFMU and begins the console application.

int main()
{
//Example of DC Measurement function call with
        //DCMasurement(0.05,0.05);
        //(Gate voltage in volt , drain voltage in vol)
//Example of samplimng measurent with fixed bias function call
        //Samplingwithfixedbias(0.00001, 0.5,100,1,0.00001, 0.05,100,1,0.00001,100);
        //(Vg sampling interval,Vg in volt, No. of Vg measurement, Number of Vg signal,
      Vg sampling interval,
        //Vg,No. of Vg measurement, Number of Vg signal Id sampling interval, Number of
      Id sampling)
//Example of samplimng measurent with single frequency pulsed bias function call
        //Samplingwithpulsedbias(0.00025, 0.0001, 0.001, 0.0001, 0.4,1,0.000005,
      0.05,340,1,0.000005,340);
        //(Vg ground bias duration, Vg Risetime, Vg pulse duration, Vg Fall time, Vg in
      volt, Number of Vg signal, Vd sampling interval, Vd in volt ,No. of Vd
      measurement, Number of Vd signal,Id sampling interval, Number of Id sampling)
}
```

3.3 Visual C++ programming outputs

I-V measurement outputs

Output window:

I_d vs V_g or I_d vs V_d measurement output is successfully saved

After plotting output data:

A. I_d vs V_g

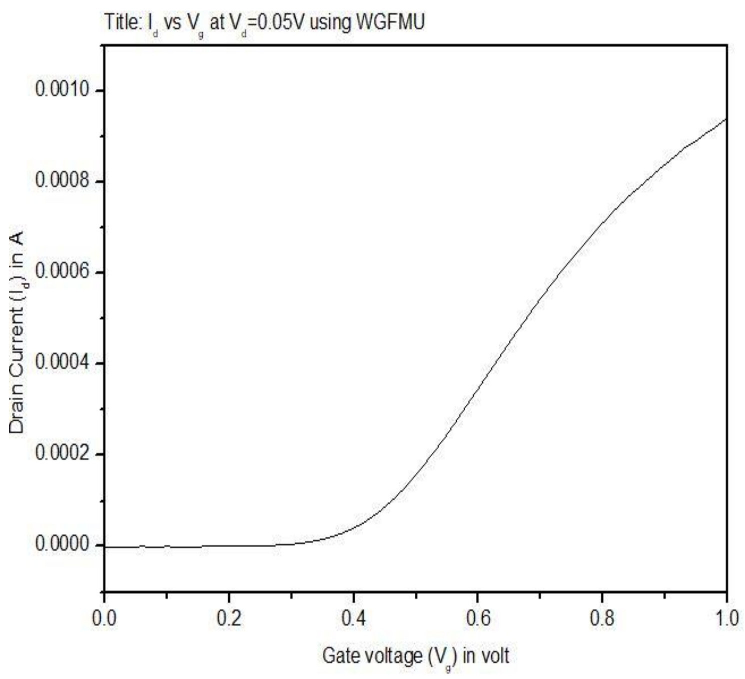


Figure 3-3-1: I_d vs V_g graph at V_d=0.05V using WGFMU

B. I_d vs V_d

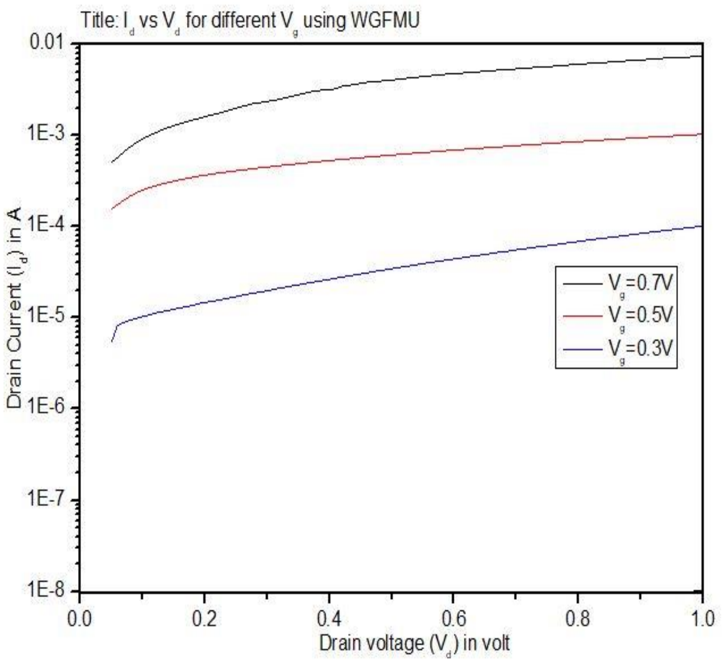


Figure 3-3-2: I_d vs V_d graph for three different values of V_g using WGFMU

Sampling measurement outputs

A. DC Measurement output:

Output window:

DC Measurement output is successfully saved

Text File:

Output Current= 0.000159414

(Here V_g=0.5V and V_d=0.05V)

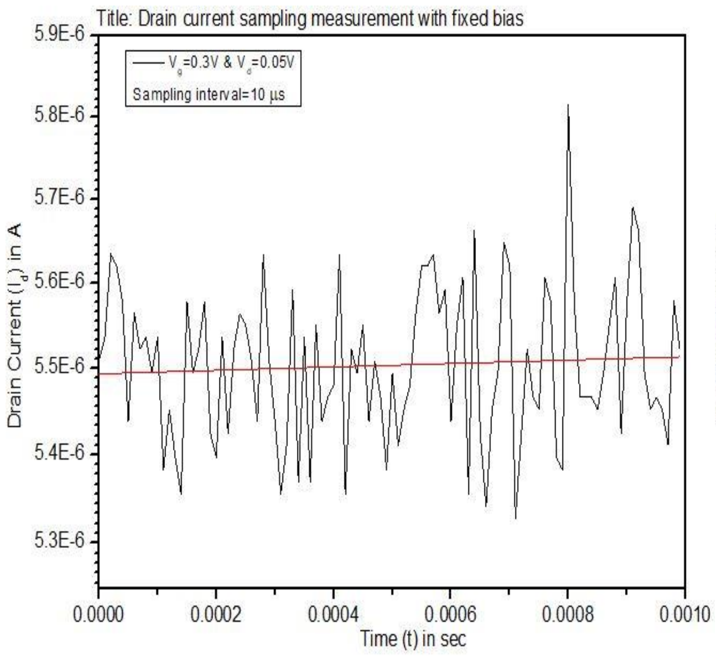
B. Fixed bias sampling Measurement output:

Output window:

Fixed bias sampling measurement output is successfully saved

After plotting output data:

I_d sampling measurement with different sampling intervals keeping V_g and V_d at some fixed values are shown in figure 3-3-3 and figure 3-3-4:



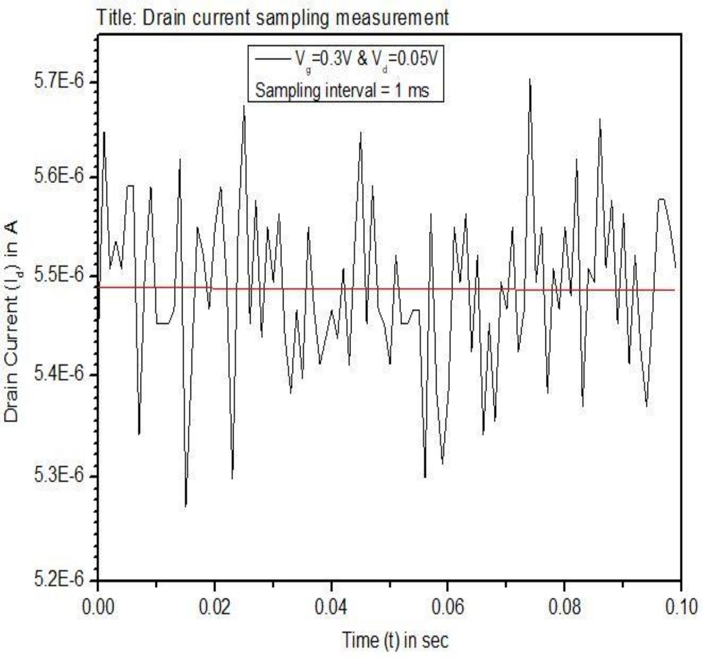
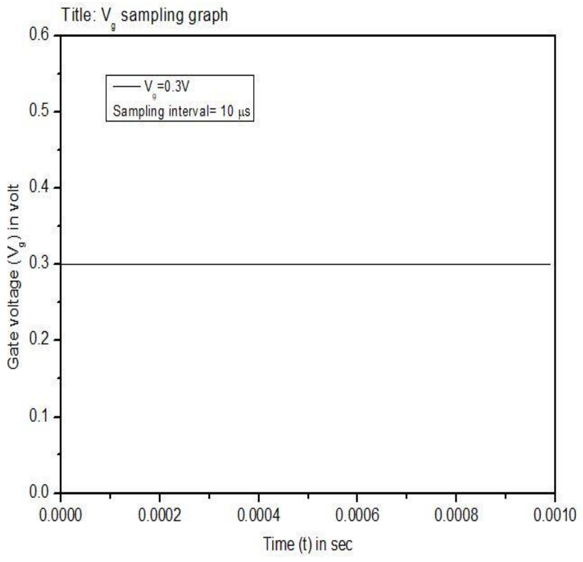


Figure 3-3-3 : Fixed bias I_d sampling measurement with sampling interval=10 μs

Figure 3-3-4: Fixed bias I_d sampling measurement with sampling interval=1 ms

Sampling of fixed V_g and V_d are shown in below figures:



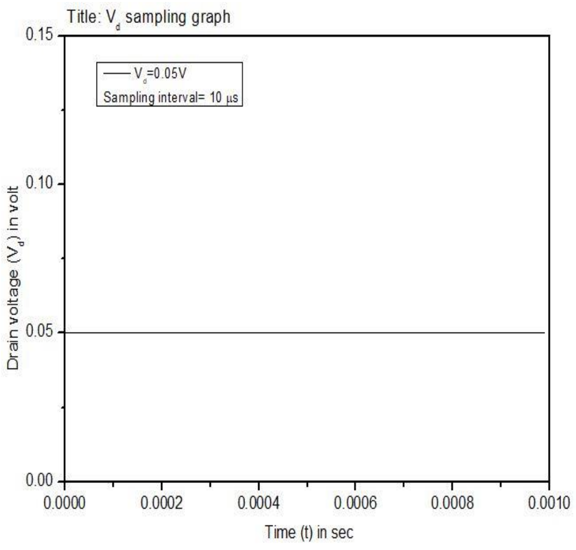


Figure 3-3-5: V_g sampling graph for fixed bias with sampling interval=10 μs

Figure 3-3-6: V_d sampling graph for fixed bias with sampling interval=10 μs

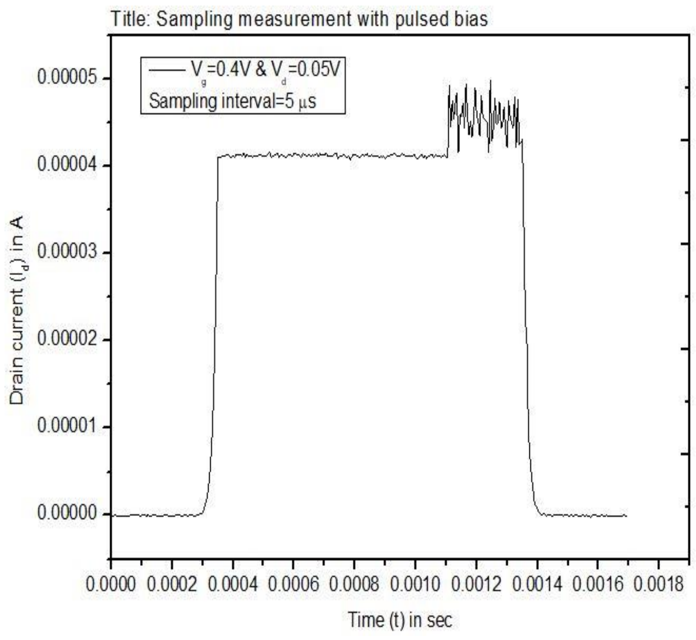
C. Pulsed bias sampling Measurement output:

Output window:

Pulsed bias sampling measurement output is successfully saved

After plotting output data:

I_d sampling measurement with different sampling intervals pulsed V_g and fixed V_d at some value are shown in figure 3-3-7 and figure 3-3-8:



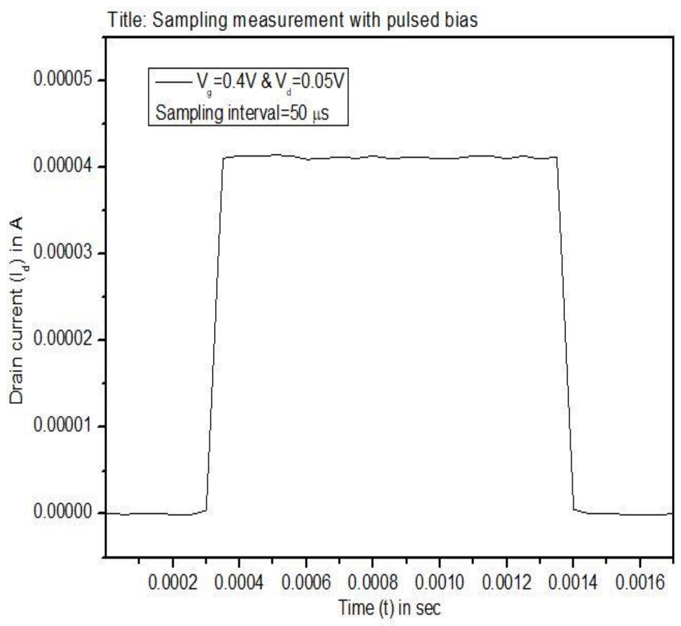
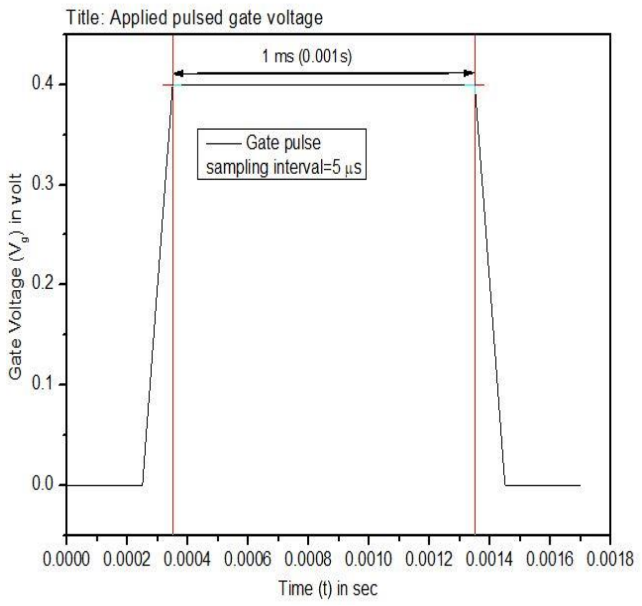


Figure 3-3-7 : Pulsed bias I_d sampling measurement with sampling interval=5µs

Figure 3-3-8: Pulsed bias I_d sampling measurement with sampling interval =50µs

Sampling of pulsed V_g and fixed V_d are shown in below figures:



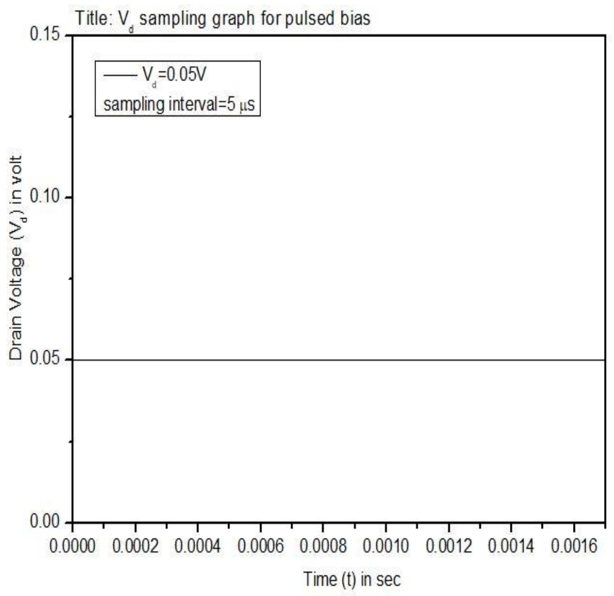


Figure 3-3-9: Gate pulse sampling graph with sampling interval=5µs

Figure 3-3-10: V_d sampling graph for pulsed bias with sampling interval=5µs

Chapter 4

Programming for Flicker Noise Measurement

4.1 Software arrangement before programming

Before starting the LabView programs, it's necessary to install necessary drivers and software to interface instruments and run LabView programs without any issue. If we fail to do these properly, instrument will not be detected or not be remotely interfaced using PC or if it's interfaced error will come during the program execution. Following are the steps to avoid interfacing error for all the instruments.

Drivers and software

1. NI VISA driver
2. NI Serial driver
3. LabView instrument driver of all instruments that will interface using LabView program
   (Tools ⟶ Instrumentation ⟶ Find instrument drivers ⟶ Instrument manufacturer name and instrument name ⟶ Search ⟶ Install)

Other necessary software

1. NI max (it installs with the NI VISA driver)
   (Measurement and automation explorer of NI MAX is used to identify the device and interfaces which are connected to the computer and the installed software)
2. Keysight IO library suite (optional)
3. Quick IV measurement software
   (This software is necessary for B2962A instrument. If the instrument is not recognised in Keysight IO library suite or NI MAX after connecting the instrument with the PC through USB A to USB B connector, it is necessary to start the instrument first time by giving appropriate values in address, model no. , channel no. and name of the instrument and identify it. After that it is automatically recognised in Keysight connection expert)

4.2 B2962A LabView programming

Only dc output voltage of B2962A instrument is required to apply low noise power at the DUT for flicker noise measurement. LabView program is developed as per this requirement shown in figure 4-2-1 below. Details of all commands that are used to develop B2962A LabView program are listed in section 2.2.1.2.

Program Block diagram

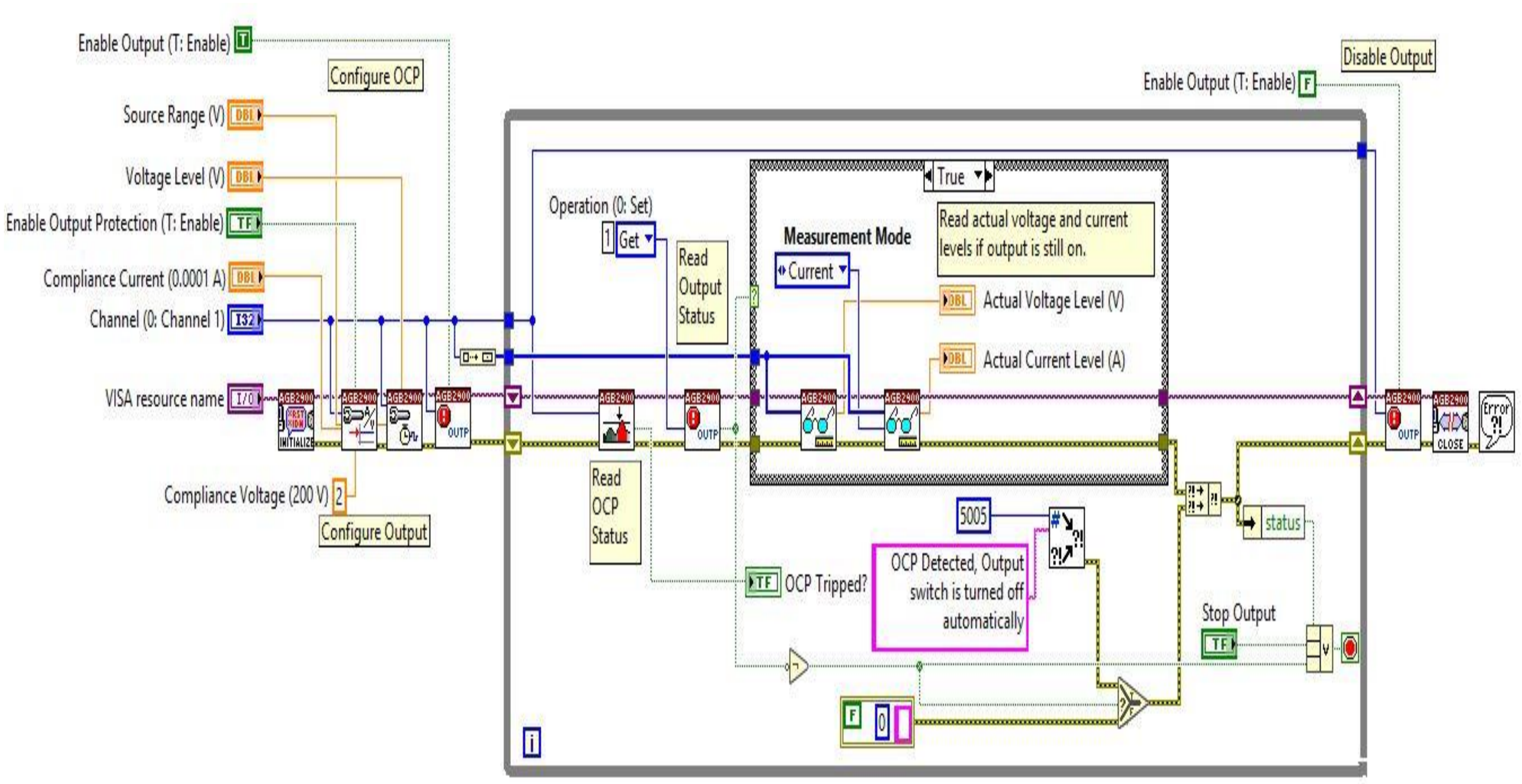


Figure 4-2-1: B2962A LabView program block diagram

4.2.1 Program description

A. First block: Establishes communication with instrument and operationally perform reset with the initial settings. *IDN, *RST and :SYST:LANG? commands are used in this VI.

B. Second block: It enable or disable over current/voltage protection and set compliance range. :OUTP:PROT and :SENS:CURR/VOLT:PROT: commands are used in this block.

C. Third block: It creates pulse output, dc output, set the source mode, apply dc voltage or current, set output auto range and the range of output lower limit. SOUR:FUNC:MODE CURR/VOLT and SOUR:VOLT:RANG:AUTO:LLIM commands are used in this VI.

D. Fourth block and Sixth block: These VIs enable or disable the source output and read the error message. Error block is inside this block which reads to error message. :OUTP, :OUTP:STAT: commands are used in these Vis.

E. Fifth block: This VI specifies the tripping of over current or voltage and returns the output. :SENS:CURR/VOLT:PROT:TRIP?: command is used here.

G. Seventh and Eighth block: It executes the spot measurement and returns the measurement data. :MEAS ?: command is used in this VI.

H. Ninth block: Performs error query before terminating software connection the instrument.

4.3 SR570 LabView programming

For flicker noise measurement, SR570 low noise current preamplifier's LabView program is developed with some specific controls as per requirements shown in figure 4-3-1. Details of all commands that are used to develop SR570 LabView program are listed in section 2.2.2.2.

Program Block diagram

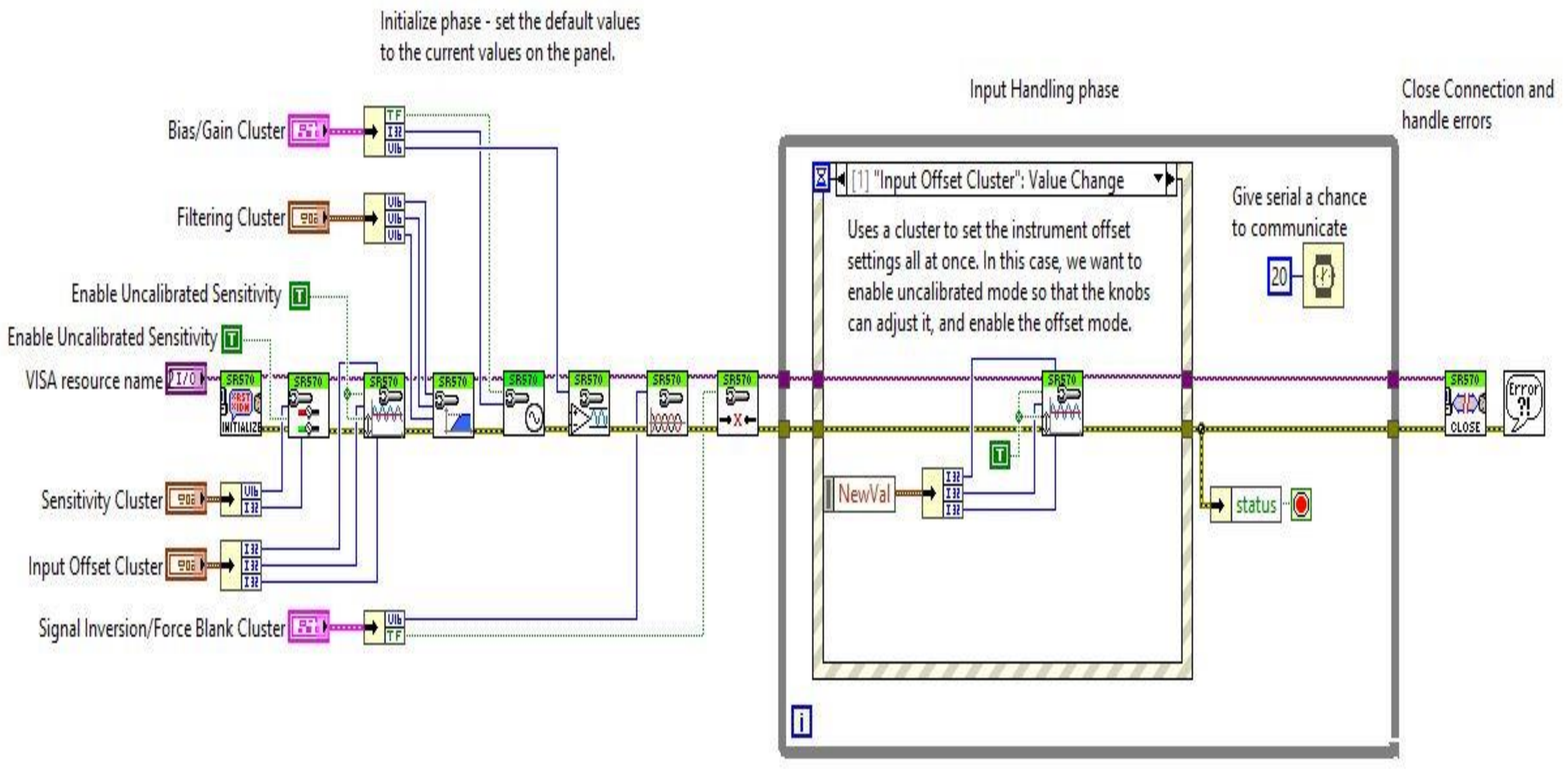

Initialize phase - set the default values to the current values on the panel.
Input Handling phase
Close Connection and handle errors
Bias/Gain Cluster
Filtering Cluster
Enable Uncalibrated Sensitivity
Enable Uncalibrated Sensitivity
VISA resource name
Sensitivity Cluster
Input Offset Cluster
Signal Inversion/Force Blank Cluster
[1] "Input Offset Cluster": Value Change
Uses a cluster to set the instrument offset settings all at once. In this case, we want to enable uncalibrated mode so that the knobs can adjust it, and enable the offset mode.
Give serial a chance to communicate
20
NewVal
status
SR570
INITIALIZE
CLOSE
Error ?!

Figure 4-3-1: SR570 labView program block diagram

4.3.1 Program description

A. First block: Establishes communication with instrument and operationally perform reset with the initial settings. Here specified baud rate, flow control, parity, data bits and stop bits are 9600, none, none, 8 and 2 respectively which should be same for the instrument, LabView programing and the computer. *RST command is used here.

B. Second block: It configures current to voltage ratio (sensitivity) between the input and the output. SENS, SUCM and SUCV commands are used here.

C. Third block and Ninth block: It controls current amplitude offset applied to the input current. IOON, IOSN, IOLV, IOUC and IOUV commands are used in this VI.

D. Fourth block: It configures the filter settings. These include lowpass, highpass and bandpass filter setting. FLTT, HFRQ and LFRQ commands are used.

E. Fifth block: Configures the offset bias voltage in the input. BSON and BSCV commands are used in this VI.

F. Sixth block: This VI sets the gain mode of the amplifier. GNMD command is used here

G. Seventh block: Inverts the output signal. INVT command is used in this VI.

H. Eighth block: It blanks the front end output of the amplifier if enabled. BLNK command is used here.

I. Tenth block: Performs error query before terminating software connection the instrument.

4.4 SR780 LabView programming

In flicker noise measurement, this instrument is used to collect the voltage signal which is coming as the output of SR570 after amplification and analyse it in time domain, FFT, PSD (power spectral density) in appropriate measurement scale, unit with collecting the measurement data. LabView program is developed as per these requirements shown in figure 4-4-1 below. Details of all commands that are used to develop SR780 LabView program are listed in section 2.2.3.2.

Program Block diagram

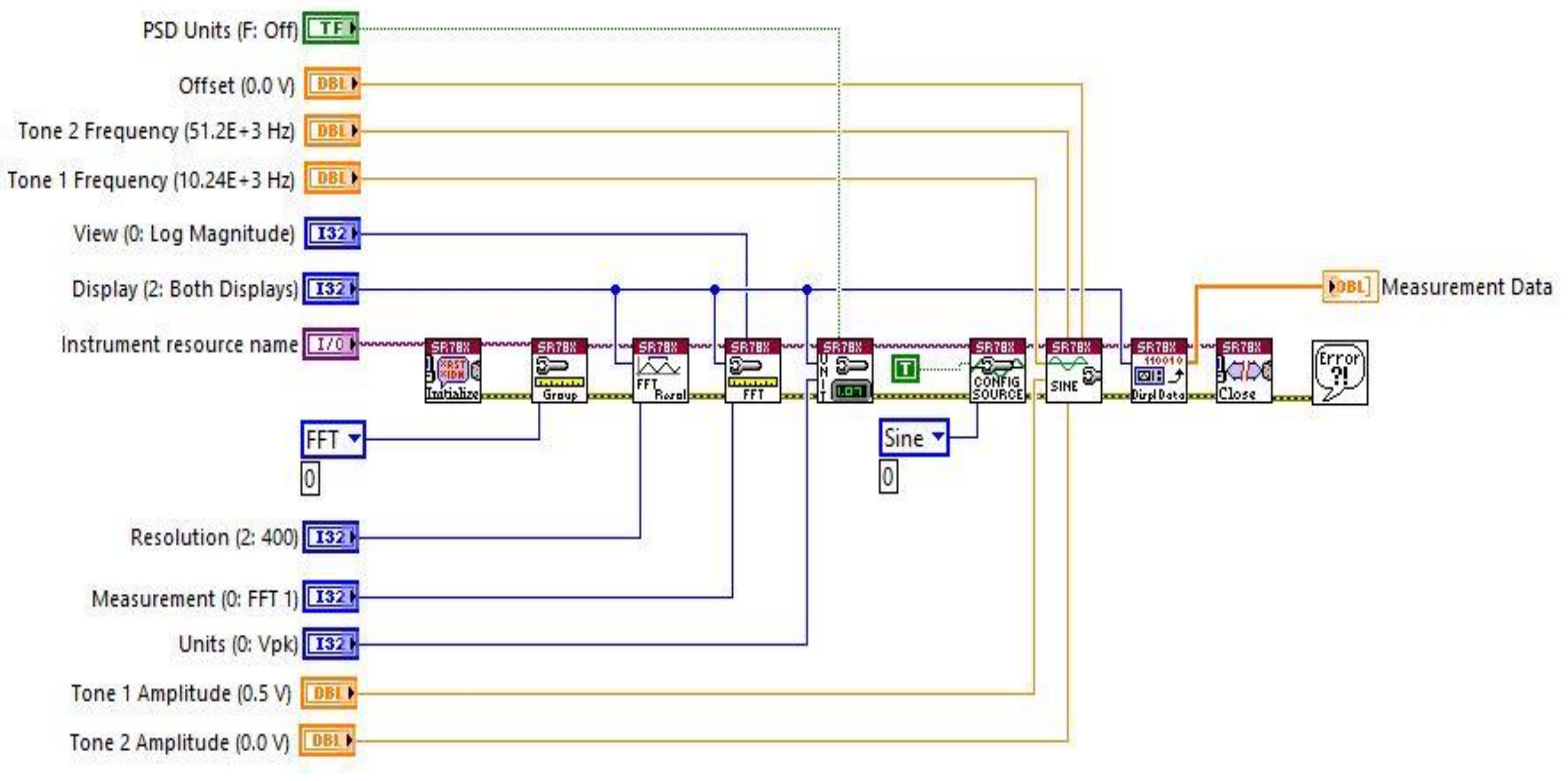


Figure 4-4-1: SR570 LabView program block diagram

4.4.1 Program description

A. First block: This VI passes the addressing information in the instrument descriptor to the Instr Open VI and returns the instrument ID. It initializes the instrument and reset it with the default settings. It also returns the SR780's device identification string and sets (queries) the output interface (RS232 or GPIB). Only the remote interface settings are not changed (all stored data are lost). Here specified baud rate, flow control, parity, data bits and stop bits are 9600, none, none, 8 and 1 respectively which should be same for the instrument, LabView programing and in the port settings of the computer.

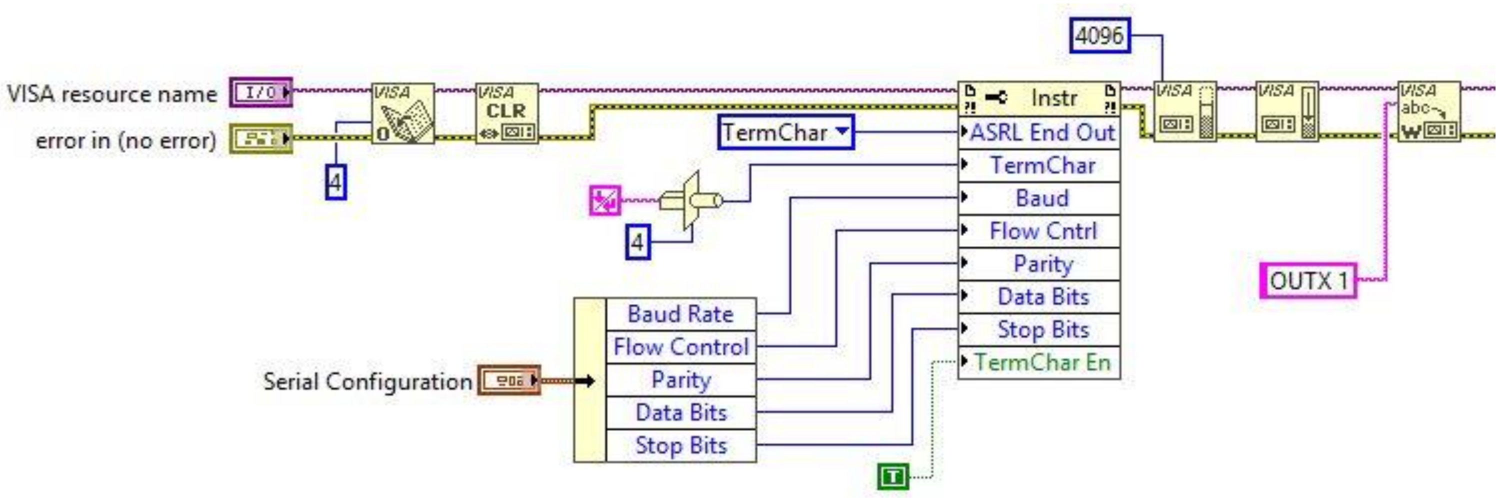


Figure 4-4-1-1: Serial communication cluster

The above cluster structures (block diagram of the initialization block) are important to specify for any LabView program which is interfaced using serial communication in computer. OUTX 1 is to set the output interface as RS232. *IDN and *RST commands are used hare.

B. Second block: This VI configures the Measurement Group for both displays. MGRP command is used in this VI.

C. Third block: This VI configures the FFT resolution. It is valid only when the Measurement Group is FFT or Correlation. FLIN command is used in this block.

D. Fourth block: Configures the FFT or time domain measurement for FFT measurement group. MEAS and VIEW commands are used in this block.

E. Fifth block: This VI configures the display units for selected display. UNIT and PSDU commands are used in this VI.

F. Sixth block: This VI configures the source. SRCO and STYP commands are used in this VI.

G. Seventh block: It configures the sine source. This VI is valid only when the Source type is Sine. S1FR, S1AM, SOFF, S2FR and S2AM commands are used here.

H. Eighth block: This VI returns all the data in selected display. The returned data depends upon the display View and Units. DSPY? command is used for RS232 communication interface and DSPN?, DSPB? commands are used for GPIB communication interface.

I. Ninth block: This VI closes the I/O interface with the instrument.

I. Ninth block: Indicate whether an error occurred. If an error occurred, this VI returns a description of the error and optionally displays a dialog box.

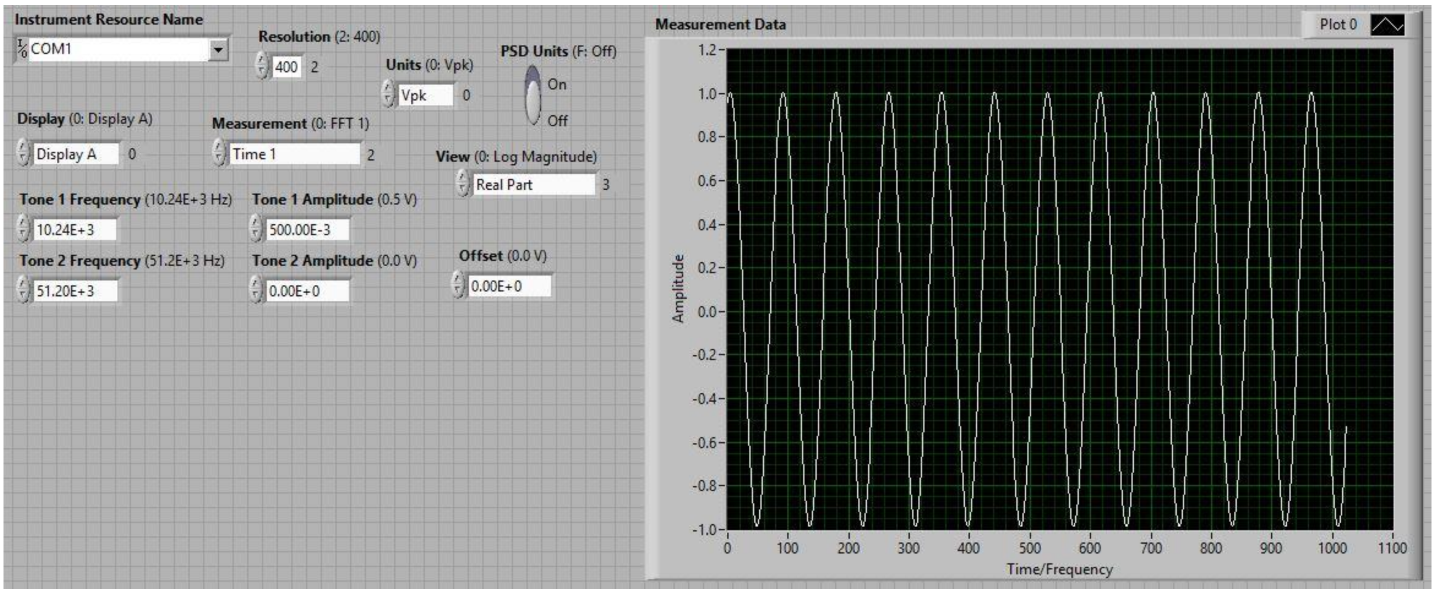
4.5 SR780 LabView programming outputs

A. Sinusoidal signal

<u>Time domain</u>: (400 FFT lines)

Y axis unit: Volt (pk to pk)

Frequency of the signal= 10.24 kHz



Instrument Resource Name
COM1

Resolution (2: 400)
400 2

Units (0: Vpk)
Vpk 0

PSD Units (F: Off)
On
Off

Display (0: Display A)
Display A 0

Measurement (0: FFT 1)
Time 1 2

View (0: Log Magnitude)
Real Part 3

Tone 1 Frequency (10.24E+3 Hz)
10.24E+3

Tone 1 Amplitude (0.5 V)
500.00E-3

Tone 2 Frequency (51.2E+3 Hz)
51.20E+3

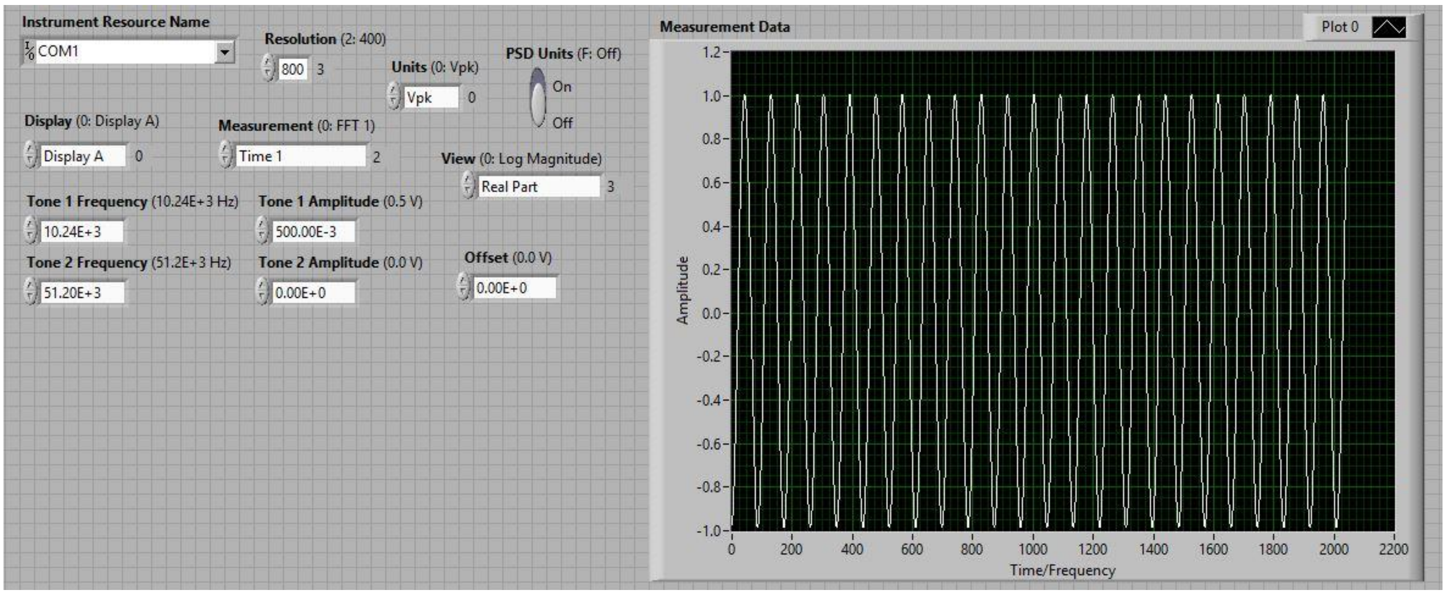
Tone 2 Amplitude (0.0 V)
0.00E+0

Offset (0.0 V)
0.00E+0

Figure 4-5-1: Sinusoidal signal in time domain with 400 FFT lines

<u>Time domain</u>: (800 FFT lines)

Y axis unit: Volt (pk to pk)



Instrument Resource Name
COM1

Resolution (2: 400)
800 3

Units (0: Vpk)
Vpk 0

PSD Units (F: Off)
On
Off

Display (0: Display A)
Display A 0

Measurement (0: FFT 1)
Time 1 2

View (0: Log Magnitude)
Real Part 3

Tone 1 Frequency (10.24E+3 Hz)
10.24E+3

Tone 1 Amplitude (0.5 V)
500.00E-3

Tone 2 Frequency (51.2E+3 Hz)
51.20E+3

Tone 2 Amplitude (0.0 V)
0.00E+0

Offset (0.0 V)
0.00E+0

Figure 4-5-2: Sinusoidal signal in time domain with 800 FFT lines

FFT: (400 FFT lines)

Y axis unit: Volt (pk to pk)

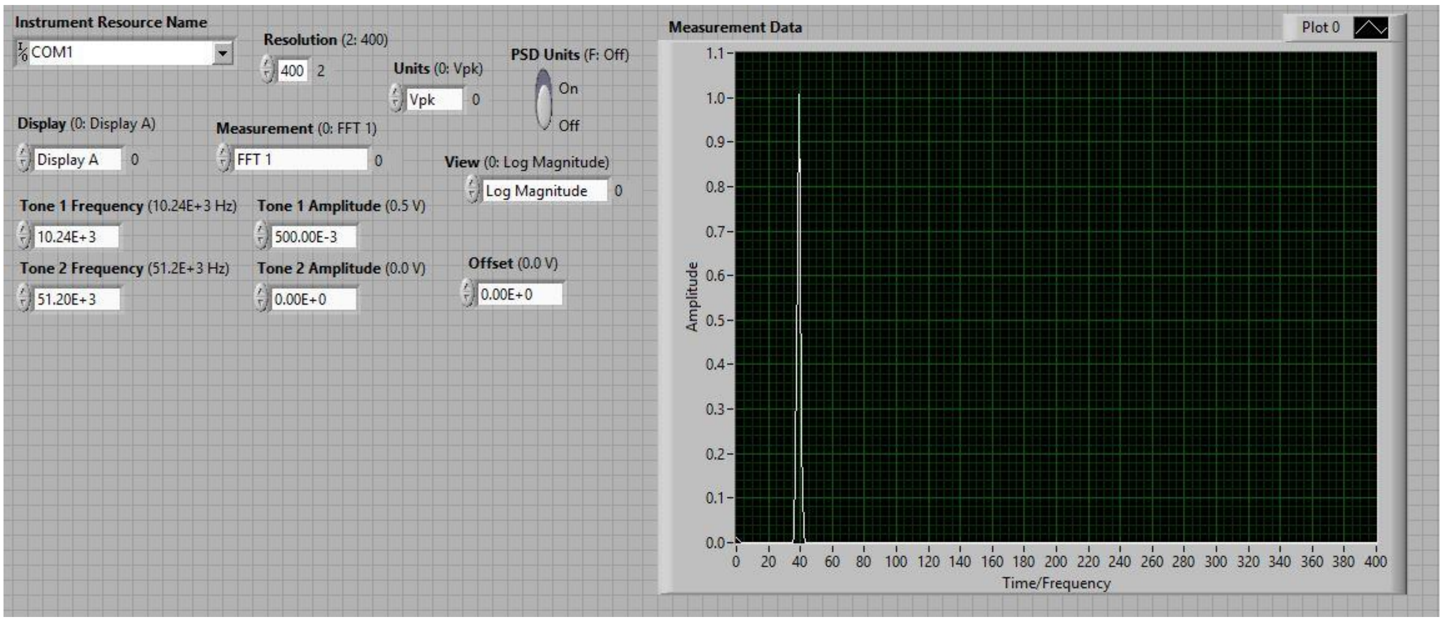

Instrument Resource Name: COM1
Resolution (2: 400): 400 2
Units (0: Vpk): Vpk 0
PSD Units (F: Off): On / Off
Display (0: Display A): Display A 0
Measurement (0: FFT 1): FFT 1 0
View (0: Log Magnitude): Log Magnitude 0
Tone 1 Frequency (10.24E+3 Hz): 10.24E+3
Tone 1 Amplitude (0.5 V): 500.00E-3
Tone 2 Frequency (51.2E+3 Hz): 51.20E+3
Tone 2 Amplitude (0.0 V): 0.00E+0
Offset (0.0 V): 0.00E+0

Figure 4-5-3: Sinusoidal signal FFT with 400 FFT lines

FFT: (800 FFT lines)

Y axis unit: Volt (pk to pk)

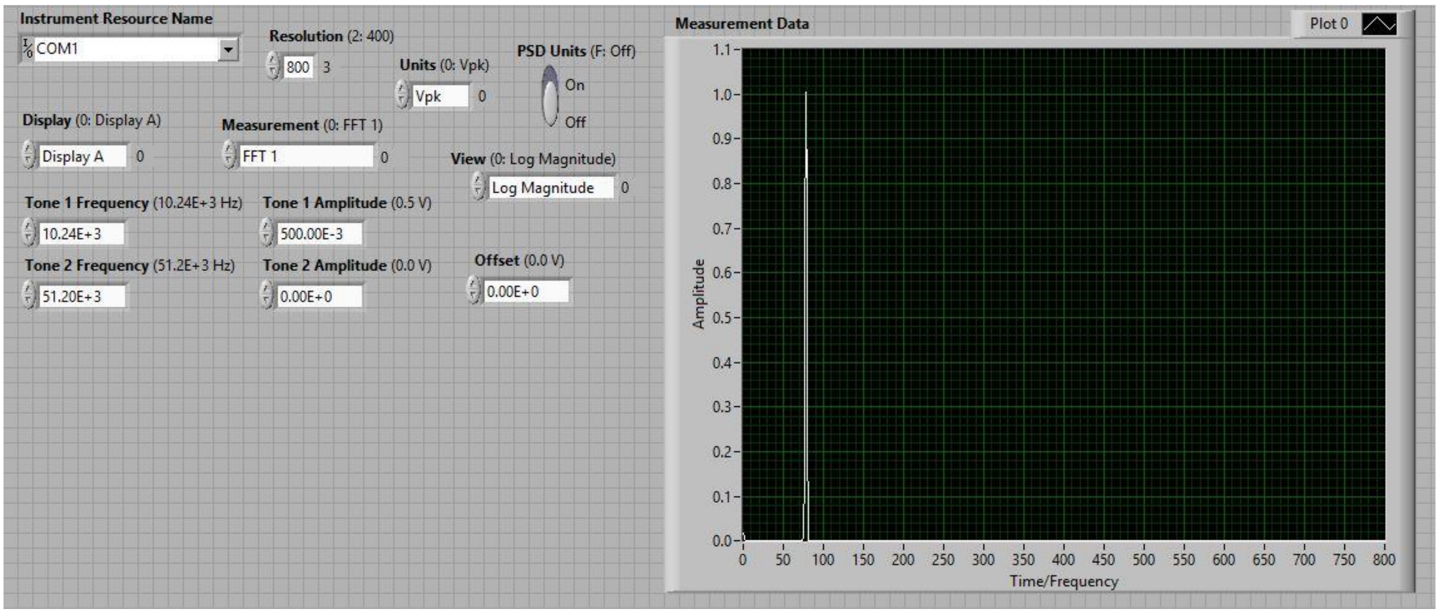

Instrument Resource Name: COM1
Resolution (2: 400): 800 3
Units (0: Vpk): Vpk 0
PSD Units (F: Off): On / Off
Display (0: Display A): Display A 0
Measurement (0: FFT 1): FFT 1 0
View (0: Log Magnitude): Log Magnitude 0
Tone 1 Frequency (10.24E+3 Hz): 10.24E+3
Tone 1 Amplitude (0.5 V): 500.00E-3
Tone 2 Frequency (51.2E+3 Hz): 51.20E+3
Tone 2 Amplitude (0.0 V): 0.00E+0
Offset (0.0 V): 0.00E+0

Figure 4-5-4: Sinusoidal signal FFT with 800 FFT lines

PSD: (800 FFT lines)

Y axis unit: V^2 / Hz (V_{rms}^2/Hz); X axis and Y axis both are in log scale.

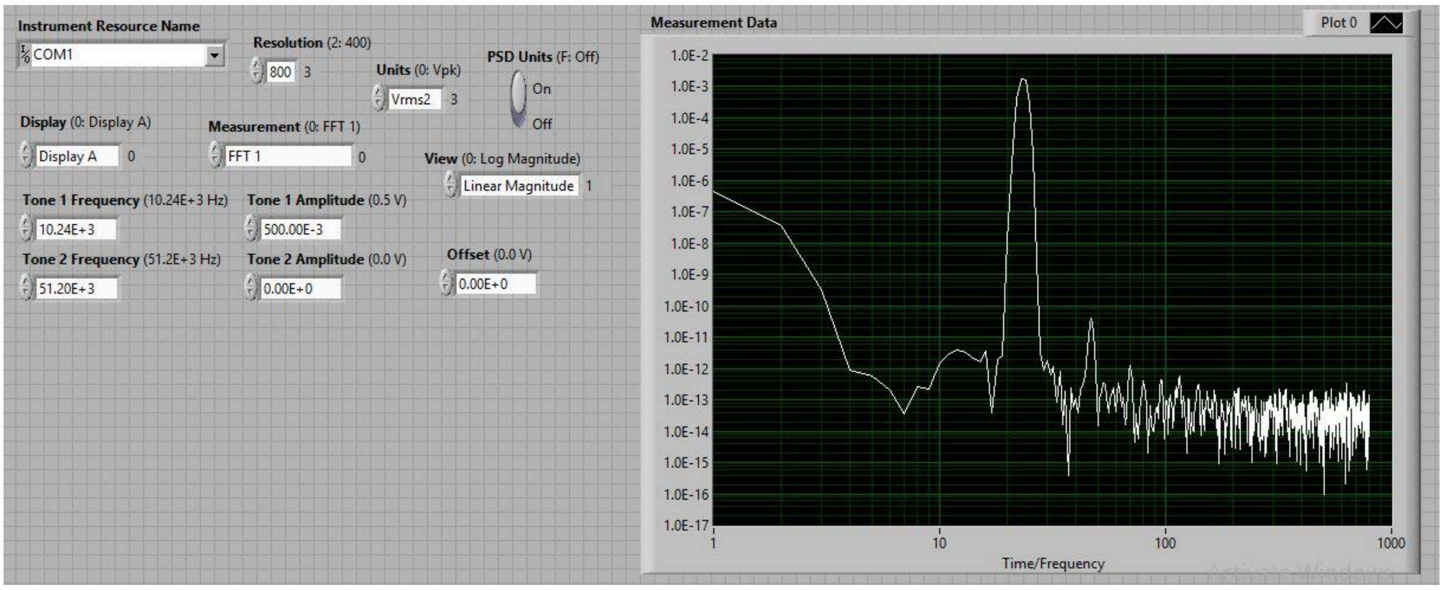

Instrument Resource Name: COM1
Resolution (2: 400): 800 3
Units (0: Vpk): Vrms2 3
PSD Units (F: Off): On / Off
Display (0: Display A): Display A 0
Measurement (0: FFT 1): FFT 1 0
View (0: Log Magnitude): Linear Magnitude 1
Tone 1 Frequency (10.24E+3 Hz): 10.24E+3
Tone 1 Amplitude (0.5 V): 500.00E-3
Tone 2 Frequency (51.2E+3 Hz): 51.20E+3
Tone 2 Amplitude (0.0 V): 0.00E+0
Offset (0.0 V): 0.00E+0

Figure 4-5-5: Sinusoidal signal PSD

B. Square wave Signal

Time domain: (400 FFT lines)

Y axis unit: Volt (pk to pk)

Frequency of the signal= 10.24 kHz

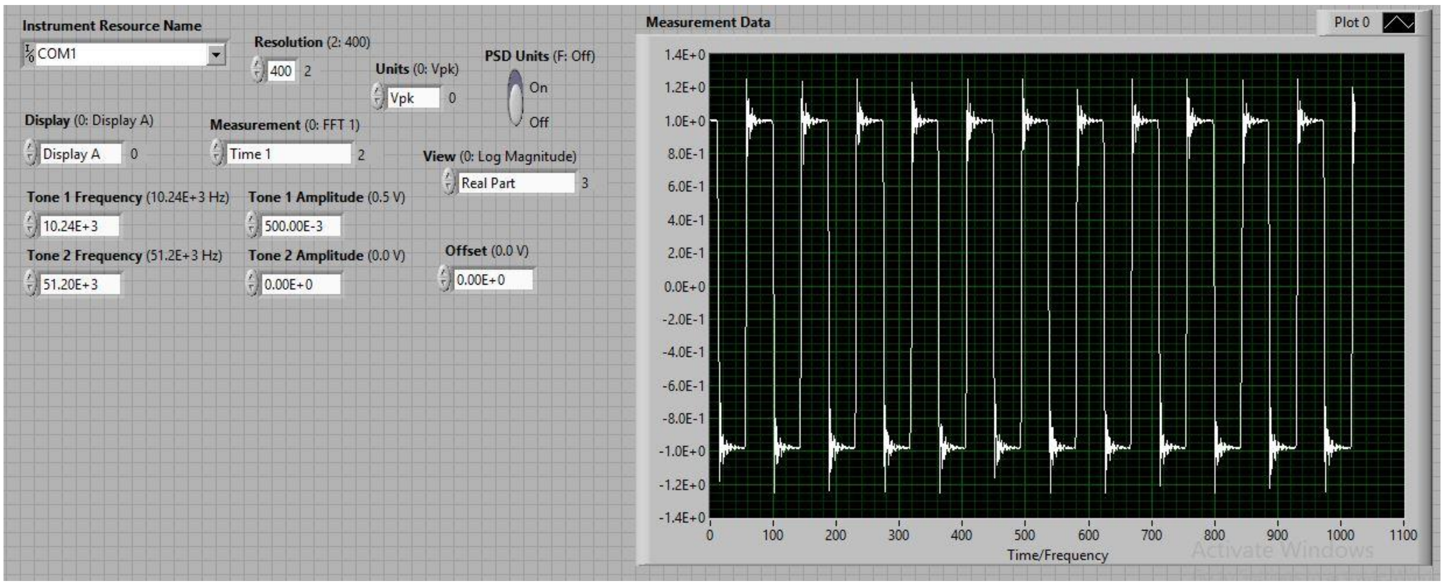
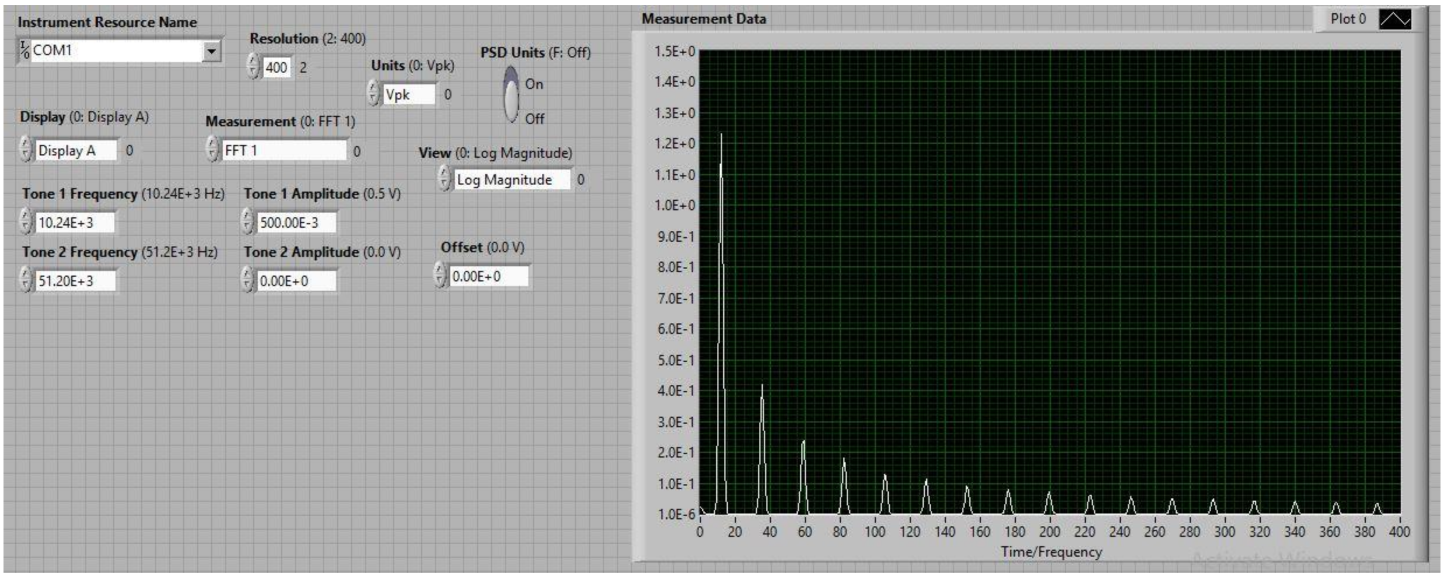

Instrument Resource Name: COM1
Resolution (2: 400): 400 2
Units (0: Vpk): Vpk 0
PSD Units (F: Off): On / Off
Display (0: Display A): Display A 0
Measurement (0: FFT 1): Time 1 2
View (0: Log Magnitude): Real Part 3
Tone 1 Frequency (10.24E+3 Hz): 10.24E+3
Tone 1 Amplitude (0.5 V): 500.00E-3
Tone 2 Frequency (51.2E+3 Hz): 51.20E+3
Tone 2 Amplitude (0.0 V): 0.00E+0
Offset (0.0 V): 0.00E+0

Figure 4-5-6: Square wave signal in time domain

<u>FFT</u>: (400 FFT lines)

Y axis unit: Volt (pk to pk)

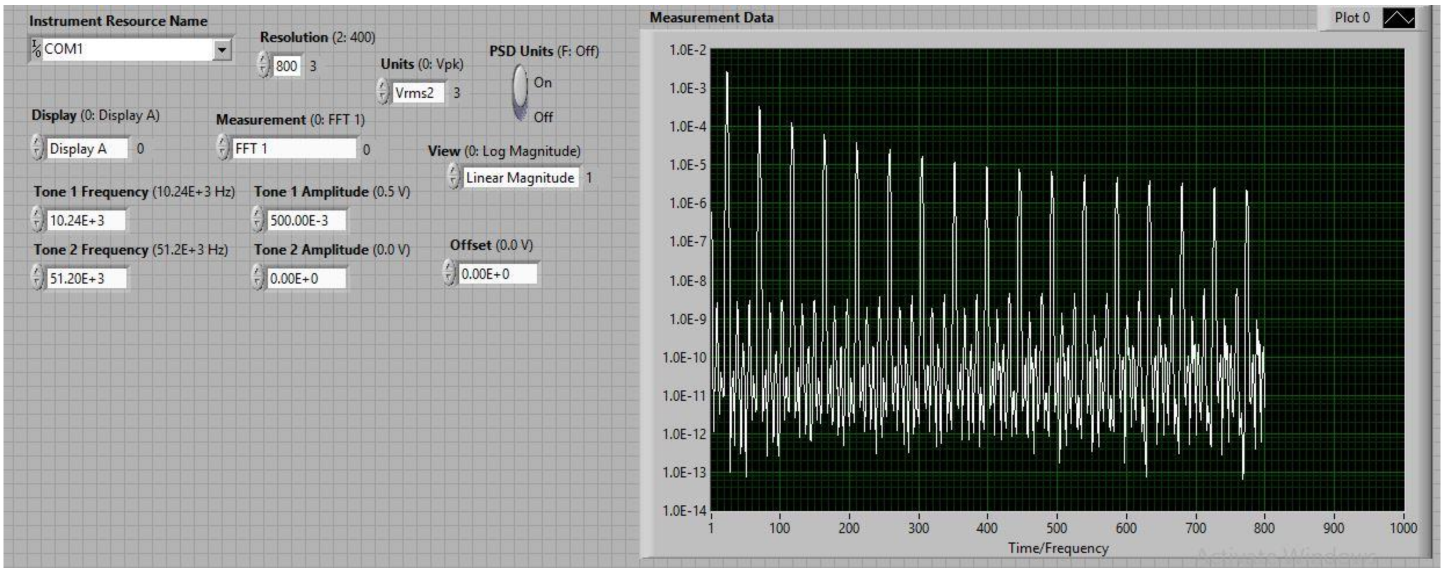


Instrument Resource Name: COM1
Resolution (2: 400): 400 2
Units (0: Vpk): Vpk 0
PSD Units (F: Off): On / Off
Display (0: Display A): Display A 0
Measurement (0: FFT 1): FFT 1 0
View (0: Log Magnitude): Log Magnitude 0
Tone 1 Frequency (10.24E+3 Hz): 10.24E+3
Tone 1 Amplitude (0.5 V): 500.00E-3
Tone 2 Frequency (51.2E+3 Hz): 51.20E+3
Tone 2 Amplitude (0.0 V): 0.00E+0
Offset (0.0 V): 0.00E+0

Figure 4-5-7: Square wave signal FFT

<u>PSD</u>: (800 FFT lines)

Y axis unit: V^2 / Hz (V_{rms}^2/Hz); Only y axis is in log scale.



Instrument Resource Name: COM1
Resolution (2: 400): 800 3
Units (0: Vpk): Vrms2 3
PSD Units (F: Off): On / Off
Display (0: Display A): Display A 0
Measurement (0: FFT 1): FFT 1 0
View (0: Log Magnitude): Linear Magnitude 1
Tone 1 Frequency (10.24E+3 Hz): 10.24E+3
Tone 1 Amplitude (0.5 V): 500.00E-3
Tone 2 Frequency (51.2E+3 Hz): 51.20E+3
Tone 2 Amplitude (0.0 V): 0.00E+0
Offset (0.0 V): 0.00E+0

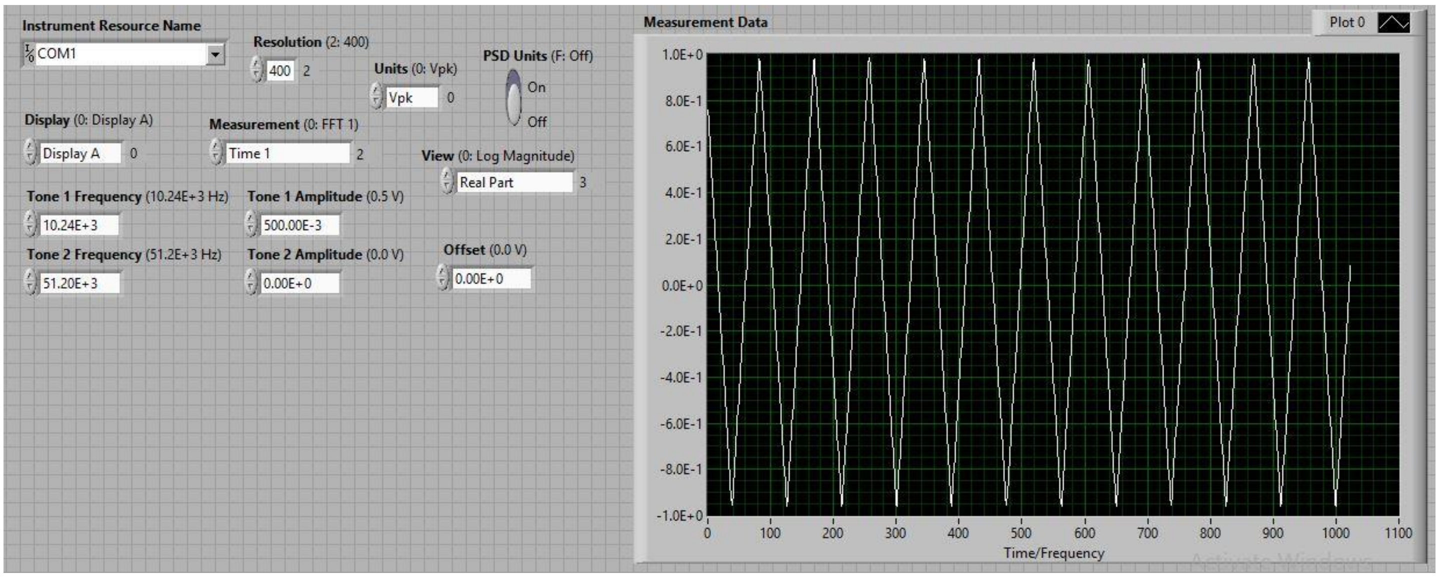
Figure 4-5-8: Square wave signal PSD

C. Triangular wave signal

<u>Time domain</u>: (400 FFT lines)

Y axis unit: Volt (pk to pk)

Frequency of the signal= 10.24 kHz

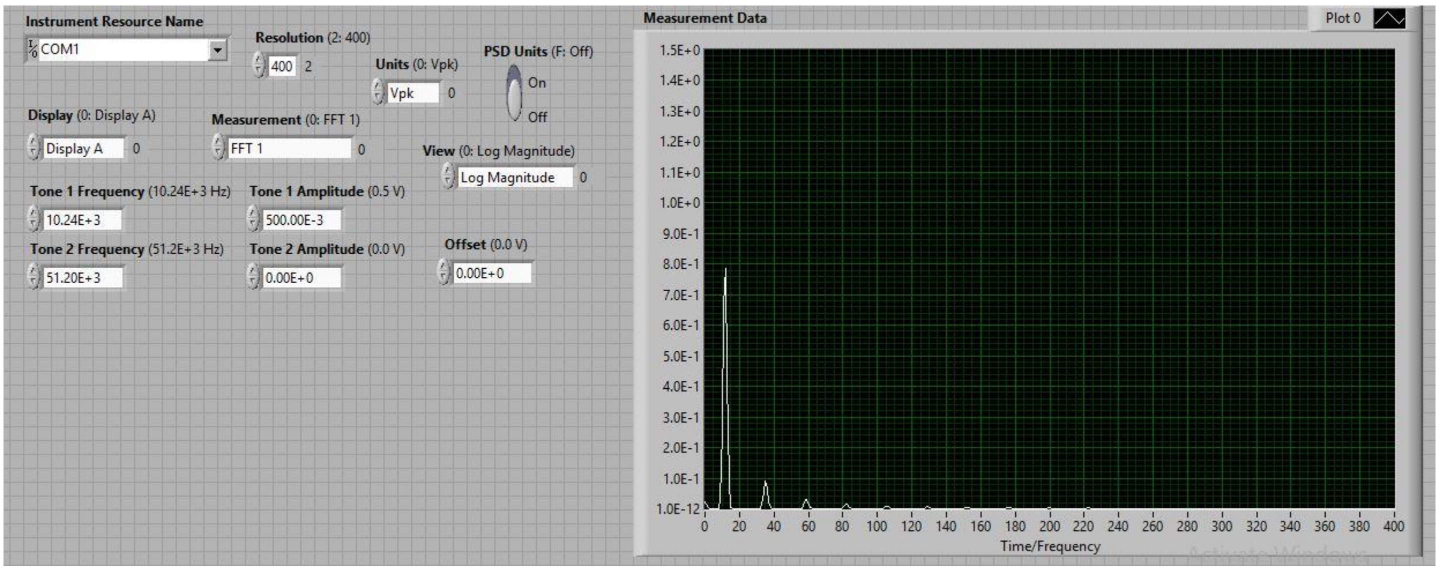


Instrument Resource Name: COM1
Resolution (2: 400): 400 2
Units (0: Vpk): Vpk 0
PSD Units (F: Off): On / Off
Display (0: Display A): Display A 0
Measurement (0: FFT 1): Time 1 2
View (0: Log Magnitude): Real Part 3
Tone 1 Frequency (10.24E+3 Hz): 10.24E+3
Tone 1 Amplitude (0.5 V): 500.00E-3
Tone 2 Frequency (51.2E+3 Hz): 51.20E+3
Tone 2 Amplitude (0.0 V): 0.00E+0
Offset (0.0 V): 0.00E+0

Figure 4-5-9: Triangular wave signal in time domain

<u>FFT</u>: (400 FFT lines)

Y axis unit: Volt (pk to pk)

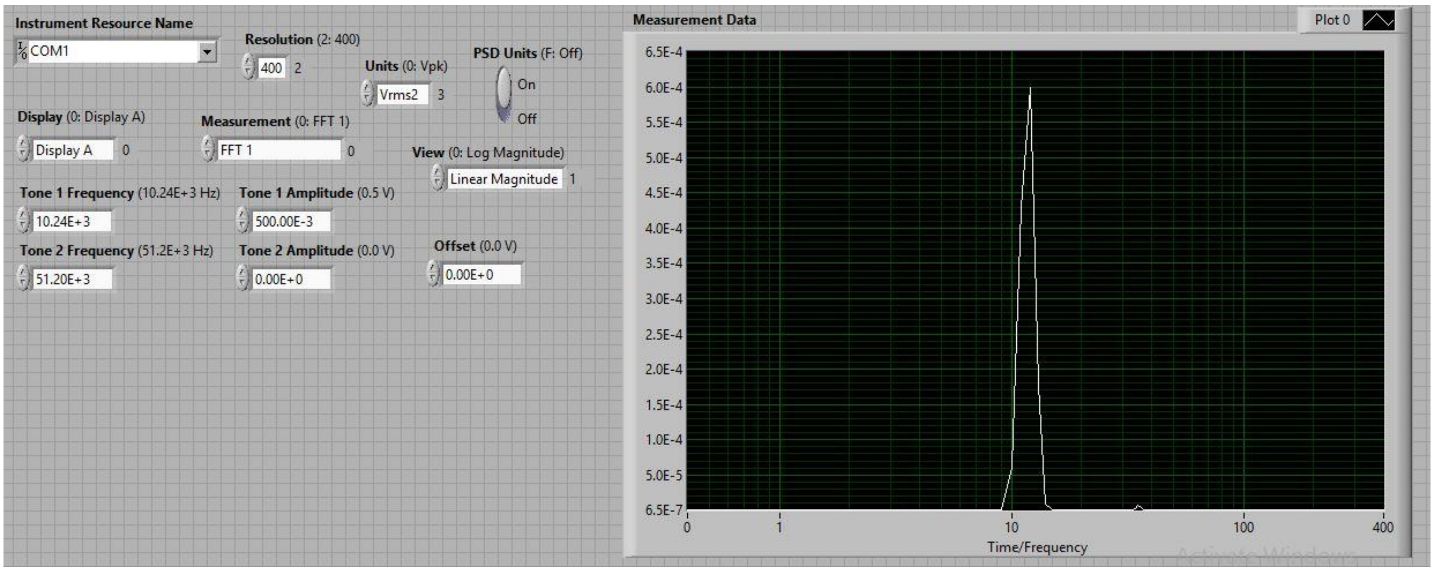


Instrument Resource Name: COM1
Resolution (2: 400): 400 2
Units (0: Vpk): Vpk 0
PSD Units (F: Off): On / Off
Display (0: Display A): Display A 0
Measurement (0: FFT 1): FFT 1 0
View (0: Log Magnitude): Log Magnitude 0
Tone 1 Frequency (10.24E+3 Hz): 10.24E+3
Tone 1 Amplitude (0.5 V): 500.00E-3
Tone 2 Frequency (51.2E+3 Hz): 51.20E+3
Tone 2 Amplitude (0.0 V): 0.00E+0
Offset (0.0 V): 0.00E+0

Figure 4-5-10: Triangular wave signal FFT

<u>PSD</u>: (400 FFT lines)

Y axis unit: V^2 / Hz (V_{rms}^2/Hz); X axis and Y axis both are in log scale.



Instrument Resource Name: COM1

Resolution (2: 400): 400 2

Units (0: Vpk): Vrms2 3

PSD Units (F: Off): On / Off

Display (0: Display A): Display A 0

Measurement (0: FFT 1): FFT 1 0

View (0: Log Magnitude): Linear Magnitude 1

Tone 1 Frequency (10.24E+3 Hz): 10.24E+3

Tone 1 Amplitude (0.5 V): 500.00E-3

Tone 2 Frequency (51.2E+3 Hz): 51.20E+3

Tone 2 Amplitude (0.0 V): 0.00E+0

Offset (0.0 V): 0.00E+0

Figure 4-5-11: Triangular wave signal PSD

4.5.1 Calculation of actual time or frequency from the plotted points

$$\text{Actual time or frequency} = \frac{P}{L} \cdot I \qquad (1)$$

Where,

P= x coordinate of the measured point in LabView waveform graph.

L= Maximum x-axis scale limit in LabView waveform graph.
(For n lines FFT its value is n. n may be equal to 100, 200, 400 or 800)

I= Maximum x-axis scale limit in the instrument (in Hz or kHz).

For above plots:

L= 400 or 800

I= 102.4 kHz

From the figure 4-5-3 (400 FFT lines), Sinusoidal signal FFT peak is at 40.

So, from the equation (1), frequency (f) = $\frac{40}{400}$ x 102.4 kHz = 10.24 kHz (2)

Similarly, from the figure 4-5-4, peak frequency = $\frac{80}{800}$ x 102.4 kHz =10.24 kHz (3)

Chapter 5

Discussion, Challenges and Conclusion

Discussion

Automation of LFN measurement in terms of RTN and flicker noise measurements, for doing all measurements, data acquisition and analysis of data efficiently, was done. Some basic program outputs and plots of automated RTN and flicker noise measurement setups have been shown systemically.

For RTN measurement automation, Visual C++ program has been developed for both I-V and sampling measurements for WGFMU measurement module. Visual C++ programs for I-V measurement have been developed for I_d (drain current) vs V_g (gate voltage) keeping V_d(drain voltage) at 0.05V (50mV) and I_d vs V_d keeping V_g in three different values (0.3V, 0.5V and 0.7V). After connecting DUT to test programs with above specified values, measured outputs are plotted in figure 3-3-1 and 3-3-2. Both the cases gate and drain voltages are increased with a specified voltage step and drain currents are measured with a correct sampling interval to cover the full voltage sweep range of gate or drain voltage. Number of measurement data can be varied by changing the step voltage of V_g or V_d. The results of I-V measurements are used to analyse sampling measurement results properly for RTN measurement. Drain sampling measurements are directly useful to measure RTN.

Sampling measurement programs are developed for both fixed and pulsed gate bias voltages. Sampling interval of measured drain current and number of measurement points can be changed in the program. Creating several (as necessary) fixed bias and pulsed bias sampling measurement program functions (Samplingwithfixedbias() and Samplingwithpulsedbias()) with different drain current sampling intervals (output file names should be different for each program functions) and calling these functions simultaneously in main function of the program, it is possible to get outputs with different sampling interval running the program a single time only. It saves lots of measurement time for different sampling intervals. Figure 3-3-3 and 3-3-4 show the output plot of sampling measurement program with fixed gate bias (V_g=0.3V, V_d=0.05V) and figure 3-3-7 and 3-3-8 show the output plot of sampling measurement program with pulsed gate bias (V_g=0.4V, V_d=0.05V). Here outputs are taken with 10 μs and 1 ms for fixed bias sampling and 5 μs and 50 μs for puled bias sampling measurements. Any number of drain current measurement points can be given in the sampling measurement programs. Sampling interval and number of points of V_g and V_d must be changed in accordance with the I_d sampling interval and the measurement points. Timing error will come in the program if total sampling time of applied V_g or V_d is less than the total sampling time of I_d measurement. Total sampling time of Id measurement must be less than or equal to the total sampling time of applied V_g or V_d. Log files are created to save all log errors and warnings coming from the execution of WGFMU functions. The causes of the errors and warnings can be found in the WGFMU user manual [6] using corresponding error codes saved in the log file. All output files are saved in the specified location in the PC as specified in the output file saving portion of the program code. In figure 3-3-2, I_d vs V_d graphs for different V_g are not exactly parallel (specially graph for V_g = 0.3V is nearly correct). This is due to the defined measured current range in the program which was 1mA in our case and because of that reason plots with higher currents came correctly with good resolution. For sampling measurements in figure 3-3-3 and 3-3-4, noise in the current has come near 5.5 µA current at V_g = 0.3V and V_d = 0.05V which is almost equal to the measured current from I_d vs V_g graph at the same V_g and V_d. So, the outputs data plots of RTN setup automation are justified.

For flicker noise measurement automation, LabView program has been used. Different known signals (sine, square and triangular) are applied to the input of SR780 instrument and time domain, FFT and PSD of the signals are studied (figure 4-5-1 to 4-5-11). The results of this analysis come as output of the SR780 LabView program to the PC and get the plots in LabView front panel waveform graphs. It can also be exported in Microsoft Excel workspace where measured data points can be saved. Time domain signal, FFT and PSD of the signals can be plotted with 100, 200,400 or 800 FFT lines frequency resolution as defined in the instrument. Here plots are shown only for 400 and 800 FFT lines frequency resolution. For 400 FFT lines we can get 1024 time domain data points and for 800 FFT lines we can get 2048 time domain data points. Scale of x axis in LabView waveform graphs are plotted in unit less serial numbers depending on the span and resolution of FFT. The measured time or frequency (in x axis) can be converted into actual frequency using the formula written in section 4.5.1. From the equation (1), figure 4-5-3 and 4-5-4, the calculated peak frequency of the FFT of the applied Sinusoidal signal is 10.24 kHz [equations (2) and (3)] which is exactly same as the frequency of applied signal and same thing also for the PSD of the sinusoidal signal (figure 4-5-5). So, the outputs data plots of Flicker Noise setup automation are justified.

Challenges

In this section the key challenges of this work are described briefly. As this work includes instruments setup, automation programming, measurement and data acquisition, challenges of this work present in all of these sections. The challenges are discussed briefly below.

Programming IDE selection: Proper programming IDE was needed to select for developing C++ automation programs of RTN measurement. Code block, Dev C++, turbo C++ are not compatible with the WGFMU library functions. Microsoft Visual C++ was found compatible and all programs are developed in this IDE.

Functions to open files and save data: 'fopen' function is used to open writable files to save measured output data. But present IDEs show compilation error in this function due to the safety issues and tell to use 'fopen_s' function instead of 'fopen'. But 'fopen_s' is an obsolete function and shows compilation error also. Similar warning was showing for 'sprintf' function also which is used to write anything inside the writable opened file. '#pragma warning (disable 4996)' has been used to turn off the warning related to the 'fopen' and 'sprintf' functions.

Device damage issue: This issue is observed during RTN measurement using Visual C++ programming. In this work, Visual C++ programs are developed based only on the WGFMU measurement module of B1500 instrument. When the EasyEXPERT software which uses SMU measurement module, runs simultaneously with the Visual C++ program (uses WGFMU measurement module) some extra current flows through the corresponding device (DUT) due to the certain switching of RSU and the device is damaged. This issue is solved by running either Visual C++ program or EasyEXPERT software at a time.

Use of LabView program: Flicker Noise measurement instruments can also be automated using C or C++ programming languages. There are three instruments (B2962A, SR570, SR780) in the flicker noise measurement setup which are controlled remotely using lots of commands. To find/remember useful commands from the manuals and input these in the appropriate position of the C or C++ programs to execute the specified work are very cumbersome and time consuming. LabView is a GUI based lab VI programming software where all specified control of the instruments can be done without finding/remembering and writing lots of commands several times.

Interfacing issue of SR570: After installing LabView and the instrument driver, SR570 instrument was not interfacing. This occurs due to the problem in RS232 to USB converter cable or other software/driver issues. RS232 to USB serial converter can be tested using hyper terminal or access port software. Access port is used to test the pins of RS232 to USB serial converters. During loopback test if we write any text in the transmitter window of the access port and send the data, that data will appear in the receiver window of the access port software which shows the correctness of the RS232 to USB serial converter. RS232 to USB serial converter showed its correctness in loopback test. The problem was solved after installing NI VISA software which is necessary software to interface any instrument using LabView through GPIB or RS232 connectors. Details of necessary software and drivers for LabView program environment are discussed in the section 4.1.

Initialisation issue of SR780: SR780 instrument can be interfaced using GPIB or RS232 or LAN. RS232 to USB serial converter is used in this work. We faced some initialisation issues during the LabView program execution. This issue is occurred due to not specifying the baud rate, flow control, parity, data bits and stop bits in the LabView program. This issue is solved after developing the SR780 LabView program compatible for RS232 by putting serial communication cluster in the program and specifying baud rate, flow control, parity, data bits and stop bits same for the instrument, LabView program and in the port settings of the computer which are discussed in the section 4.4.1.

SR780 resetting issue: Every time when LabView program of SR780 instrument is initialised and executed, the instrument resets. Resetting of the instrument means the instrument starts with the default settings. As SR780 is using in a flicker noise measurement setup, every time instrument resetting during program execution may give wrong measurement result. But it was observed after several measurements and also found in the SR780 manual, the SR780 remote control settings don’t change after resetting the instrument, only pre-set the instrument.

Conclusion

In view of the limitation in speed and applicability of using USB A to USB B connector and RS232 to USB serial converter an attempt has been made here to make the automation more cost effective. Automation of RTN and Flicker Noise measurements can also be done using only GPIB interface of all measurement instruments (except SR570 which supports only RS232 interface) with better speed and functionality. In the same way like this work, any measurement which delivers huge volume of measurement data output and consume huge time to measure, can possible to interface all measurement instruments to a computer and automate the control of those instruments using various programming software. Another measurement module of B1500 instrument called SMU has better current measurement resolution (in pA range) than WGFMU (in µA range). So, I-V measurement can be done using SMU with more accuracy. SMU can be automated using Visual Basic software. All RTN and Flicker Noise characterization setups can be automated only using LabView programming for making instruments control much easier and users friendly. In this report pre reliability testing automation is done using Visual C++ (for RTN measurement) and LabView (for flicker noise measurement) programming but during post reliability testing analysis, after automation and all measurements, results may be analysed in Matlab.

References

[1] Chengqing Wei, Yong-Zhong Xiong, and Xing Zhou, “Test structure for characterization of low-frequency noise in CMOS technologies”, IEEE Transactions on Instrumentation and Measurement, vol. 59, no. 7, p. 1860, July 2010.

[2] Maurício Banaszeski da Silva, “A Physics-Based Statistical Random Telegraph Noise Model”, IEEE Transactions on Electron Devices, pp. 14-21, September 2016.

[3] K. Takeuchi, T. Nagumo, K. Takeda, S. Asayama, S. Yokogawa, K. Imai and Y. Hayashi, “Direct Observation of RTN-induced SRAM Failure by Accelerated Testing and Its Application to Product Reliability Assessment” Symposium on VLSI Technology, p. 189, 17 June 2010.

[4] Qianying Tang, and Chris H. Kim,“Characterizing the Impact of RTN on Logic and SRAM Operation Using a Dual Ring Oscillator Array Circuit” IEEE Journal of Solid-State circuits, vol. 52, no. 6, p. 1654, June 2017.

[5] Agilent B1500A Semiconductor Device Analyzer User’s Guide, 6th ed., Agilent Technologies, Santa Clara, CA, November 2007.

[6] Agilent B1530A Waveform Generator/Fast Measurement Unit (WGFMU) Technical Overview, 5th ed., Agilent Technologies, Santa Clara, CA, April 14 2008.

[7] Agilent B1530A Waveform Generator/Fast Measurement Unit User’s Guide, 5th ed., Agilent Technologies, Santa Clara, CA, August 2012.

[8] Kolton T. Drake, “Biometric application of ion-conducting-based memristive devices in spike-timing-dependent-plasticity” pp. 47-119, August 2015

[9] Keysight Technologies B2961A/B2962A low noise power source User’s Guide, 3rd ed., Agilent Technologies, Santa Clara, CA, May 3rd 2016.

[10] Keysight Technologies B2961A/B2962A low noise power source Programming Guide, 4th ed., Agilent Technologies, Santa Clara, CA, May 4th 2016.

[11] Stanford Research System, SR570 current pre amplifier User’s Guide and Data sheet, Revision 1.7., SRS, Sunnyvale, CA, August 2015.

[12] Stanford Research System, SR780 spectrum analyser User’s Guide and Data sheet, SRS, Sunnyvale, CA, August 2015.