INDUSTRY INTERNSHIP PROJECT REPORT

ON

AUTOMATED RF AND MICROWAVE SYSTEM FOR QUANTUM SENSING SYSTEM

In partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

In

Computer Science & Technology

Department of Computer Science and Engineering

Amity University, Uttar Pradesh

Submitted by

Vindhya Sree (A2305221413)

Under the guidance of

Dr. S. K. Dubey

Microwave Metrology, CSIR – National Physical Laboratory, New Delhi

DECLARATION

I, Vindhya Sree, a student of Bachelors of Technology in Computer Science and Engineering, hereby declare that the project report entitled 'Automated Microwave system for Rydberg Atom based Quantum sensing systems' submitted to my industry guide Dr. S. K. Dubey at Council of Scientific and Industrial Research – National Physical Laboratory, New Delhi as well as to my faculty guide Dr. Garima Aggarwal, Department of Computer Science and Engineering, ASET, Amity University, Noida, is a project report of the work carried out by me under the guidance of Dr. S. K. Dubey. This work is submitted in partial fulfilment for the award of the degree of B. Tech CSE. The results presented in this project report have not been submitted to any other University or Institute for the recognition of any degree or diploma.

Vindhya Sree

A2305221413

7CSE4Y

COMPLETION CERTIFICATE

ACKNOWLEDGEMENT

It gives me profound sense of pride and pleasure to present the project report undertaken during the six weeks of Industrial training. I owe special debt to my industry guide, Dr. S. K. Dubey, Microwave Metrology, Council of Scientific and Industrial Research – National Physical Laboratory, New Delhi for his constant support and guidance throughout the course of my work. My deepest thanks to my faculty guide Dr. Garima Aggarwal, Associate professor, ASET(CSE), Amity University Uttar Pradesh for guiding and assisting with the paperwork with her attention and assisted me in finalizing and completing the project within a stipulated time.

Vindhya Sree

A2305221413

7CSE4Y

INDEX

LIST OF FIGURES

ABSTRACT

Electrical measurements encountered in the microwave region of the electromagnetic spectrum are analyzed and controlled through microwave measurement techniques. Automation of these instruments is, therefore, a need of the hour in measurement labs for quicker execution with less manual intervention. Specifically, microwave measurements require advanced electronics, embedded systems and microcontrollers. Proper synchronization of these devices is important for developing measurement automation systems in research labs where precision is key.

This project report is concerned with an internship carried out at the Microwave Metrology Laboratory of the Council of Scientific and Industrial Research – National Physical Laboratory, New Delhi. The internship was conducted in the period between May 2024 and June 2024. The purpose of the activity was to acquire the skills and abilities to ensure the use and mastery of the LabView software in order to create a graphical interface for controlling instruments. The instrumentation used was an SMM100A signal generator, an RTM3004 oscilloscope and an FSV Signal Analyzer by Rohde & Schwarz Instruments.

This project report provides a detailed overview on creating a graphical user interface (GUI) using LabVIEW programming for a precise and automated microwave measurement system. It includes chapters on hardware, software and interconnections to explain the hardware setup and software programming which are consolidated to form an automated measurement system for a signal generator, oscilloscope and a spectrum analyzer. Some of the hardware specifications, their key features and important measurements of the instruments are mentioned in the hardware chapter. The software chapter explores the features of LabVIEW programming in detail. The chapter of automation of microwave systems discusses the projects created in LabVIEW to measure important units with precision.

INTRODUCTION

In the field of quantum sensing systems, achieving precise and accurate electrical measurements is essential, particularly in the microwave region of the electromagnetic spectrum. Scientists and engineers depend on synchronized instruments to conduct experiments and develop test stations that meet high accuracy standards. The complexity and demand for precision in these measurements make the automation of instruments crucial for enhancing efficiency and minimizing manual intervention.

Microwave measurements involve advanced electronics, embedded systems, and microcontrollers. Proper synchronization of these devices is vital for developing automated measurement systems in research laboratories, where precision is critical. This project report focuses on creating a graphical user interface (GUI) using LabVIEW programming to achieve a precise and automated microwave measurement system.

The report provides a comprehensive overview, starting with the hardware setup, followed by software programming, and the interconnections required to form an integrated automated measurement system. It covers the specifications and key features of the hardware components, such as signal generators, oscilloscopes, and spectrum analyzers. The software chapter details the capabilities of LabVIEW programming, while the automation chapter discusses the implementation of various projects in LabVIEW to measure critical parameters with high precision.

By automating the measurement process, this project aims to enhance the efficiency, reliability, and throughput of microwave measurements, making it a valuable resource for scientists and engineers working in research of quantum sensing systems.

LITERATURE REVIEW

In light of the developing technology, metrology needs to be at par with the growing demand for precision and accuracy in the measurements. The automation of RF system measurements is essential to reduce manual intervention and increase productivity. RF signals operate within a frequency range of approximately 20 kHz to 300 GHz. Microwave signals encompass a frequency range of around 300 MHz to 30 GHz. The RF system features a consolidated instrumentation system that includes a signal generator, an oscilloscope and a spectrum analyzer to monitor, detect, and analyze the generated RF signals.

Electromagnetically Induced Transparency (EIT) refers to a phenomenon resulting from a laser inducing coherence between atomic states, leading to quantum interference that nullifies absorption and refraction at a resonant frequency, rendering the medium transparent to specific light frequencies. EIT has numerous applications in RF systems for quantum sensing:

1. In the domain of RF Field Sensing, EIT amplifies the detection of weak RF signals using Rydberg atoms, providing high sensitivity and a wide frequency range essential for communication and radar systems.
2. In medical imaging and fundamental physics research, EIT-based magnetometers offer high precision in measuring magnetic fields.
3. EIT enables the development of secure communication channels resistant to eavesdropping and supports the establishment of quantum networks for information transmission and processing.
4. EIT enhances the resolution of spectroscopic measurements, assisting in the accurate identification and characterization of materials.
5. EIT is utilized in developing atomic clocks for precise timekeeping and synchronization in communication networks and GPS systems, providing stable frequency references for metrology and telecommunications. [1]

Microwave electric field sensing is crucial for various applications in remote sensing, radar astronomy, and communications. Rydberg atoms have been employed in highly sensitive microwave electric field sensing due to their exaggerated response to MW E-fields and abundant optical energy levels. [2] The large size, dipole moment, and higher lifetimes of Rydberg atoms are used to gauge the amplitude of microwave E-fields.

Microwave quantum sensors aim to offer more efficient, robust, and secure strategies for remote sensing and communication [3]. For the said applications, automation of instrumentation is utmost necessary when it comes to experimentation and testing procedures of the RF system for Rydberg atom-based quantum sensing systems.

1. HARDWARE:

This section deals with a detailed study of hardware specifications and measurement units of the instruments which are essential to develop the software for automation of measurement.

1.1 Rohde & Schwarz SMM100A signal generator:

Rohde & Schwarz SMM100A signal generator is a midrange vector signal generator that generates high quality radio frequency (RF) signals across a broad frequency range. It is well-suited for testing and development in wireless communication including 5G NR (New Radio) for both FR1 which are sub-6 GHz and FR2 which are mmWave bands. The latest instrument drivers for the SMM100A vector signal generator have been installed in the computer for seamless communication and LabVIEW VI development.

1.1.1 Key Features of SMM100A Vector Signal Generator

The vector signal generator covers a frequency range from 100 kHz to 44 GHz, including sub-6 GHz and mmWave frequencies. The SMM100A supports up to 1 GHz RF modulation bandwidth suitable for broadband signal generation. It is known to offer single-sideband (SSB) phase noise and error vector magnitude (EVM) performance. The SSB phase noise denotes a measure of the purity of the signal in terms of phase stability. Lower the phase noise, much cleaner the signal. Error vector magnitude (EVM) performance indicates the quality of the modulation. Lower EVM values represent higher modulation accuracy.

1.1.2 Important Terminologies of SMM100A Vector Signal Generator

Some of the important terms and measurements of the signal generator include the following with the respective terminologies:

1. RF frequency: It indicates the frequency of the radio signal generated by the instrument. The SMM100A covers a range from 100 kHz to 44 GHz.
2. Limit: Limit denotes a predefined threshold value for a particular measurement parameter. An alert is triggered if the threshold value exceeds this limit.
3. Offset: A value is either added to or subtracted from a measurement in order to compensate for systematic errors or adjust the signal to a desired frequency range.
4. Digital Attenuation: The reduction of signal strength using digital control components which allows precise control over the signal amplitude without introducing significant noise or distortion in the generated signals.

The above signal generator measurements are automated using LabVIEW software which is discussed in Chapter 4.

1.2 Rohde & Schwarz RTM3004 Oscilloscope:

The Rohde & Schwarz RTM3004 is a high-resolution digital oscilloscope designed for precise signal analysis and debugging in various electronic applications. The instrument drivers for the RTM3004 oscilloscope have been installed in the computer for seamless communication and LabVIEW VI development.

1.2.1 Key Features of RTM3004 Oscilloscope:

Some of the key features of RTM3004 oscilloscope include the following:

1. Sample Rate: The RTM3004 is capable of up to 5GSa/s (giga samples per second) for high temporal resolution for accurate signal representation.
2. Bandwidth: It covers a wide range of signal frequencies ranging from 100 MHz to 1 GHz.

3. 10-bit ADC: The oscilloscope offers four times more vertical resolution as compared to the standard 8-bit ADCs facilitating a more detailed signal capture.
4. Memory depth: It offers to sample in interleaved mode which allows for long acquisition times without compromising on accuracy.
5. Display: It features a 10.1-inch high resolution capacitive touchscreen with gesture support.

1.2.2 Important Terminologies of RTM3004 Oscilloscope

Some of the important measurement terminologies of RTM3004 Oscilloscope include the following:

1. Channels: They are the number of independent signal paths available for measurement. The RTM3004 oscilloscope offer four channels for simultaneous observation and analysis of multiple signals.
2. Trigger Level: It indicates the specific voltage level at which the oscilloscope triggers to start capturing data that helps in stabilizing repetitive waveforms for analysis.
3. Trigger Slope: It determines whether the oscilloscope triggers on the rising or falling edge of the signal.
4. Vertical Range: It is the range of voltages that can be displayed on the vertical axis. It determines the amplitude range of the signal.
5. Vertical Coupling: It is a metric by which the input signal is connected to the vertical system of the oscilloscope. Common settings include DC coupling that passes all frequencies and AC coupling that blocks DC components.
6. Vertical Offset: It is the adjustment metric that shifts the entire waveform up or down on the display for a better visualization of the signal.
7. Horizontal Range: It is the range of time that can be displayed on the horizontal axis. It determines the time span of the observed signal.

8. Horizontal Coupling: It is a metric by which the time base is connected to the horizontal system of the oscilloscope. It can be set to different modes to filter out unwanted frequencies.
9. Horizontal Offset: It is the adjustment metric that shifts the entire waveform left or right on the display for a better alignment of the signal with respect to the time axis.
10. Probe Attenuation: It is the factor by which the probe reduces the signal amplitude before it reaches the oscilloscope.

The above oscilloscope measurements are automated in the LabVIEW software which are discussed in Chapter 4.

1.3 Rohde & Schwarz FSV Signal Analyzer

Rohde & Schwarz FSV Signal Analyzer is a signal analyzer designed for a wide range of applications in signal and spectrum analysis. The latest instrument drivers for the FSV Signal Analyzer have been installed in the computer for seamless communication and LabVIEW VI development.

1.3.1 Key Features of FSV Signal Analyzer

1. The FSV signal analyzer' s frequency range covers from 10 Hz to 40 GHz covering a broad spectrum for various applications.
2. The signal analyzer supports signal analysis bandwidth up to 160 MHz for a detailed analysis of wideband signals.
3. It offers a phase noise -110 dBc at 10 kHz frequency offset and 0.4 dB level measurement uncertainty up to 7 GHz.

1.3.2 Important Terminologies of FSV Signal Analyzer

1. Start Frequency: It is the lowest frequency at which the analyzer begins its measurement.

2. Stop Frequency: It is the highest frequency at which the analyzer ends its measurement.
3. Input Impedance: It is the impedance presented by the analyzer' s input, typically set to 50 ohms, to ensure proper matching with the signal source.
4. Reference Level: It is the maximum signal level that can be displayed without distortion, and it is used as a reference point for measurements.
5. Reference Level Offset: It is an adjustment that shifts the reference level to accommodate signals with different amplitudes.
6. Centre Frequency: It is the midpoint frequency of the analyzer' s measurement range around which the analysis is centred.
7. Resolution Bandwidth (RBW): It refers to the bandwidth of the filter used to resolve signals in the frequency domain.
8. Video Bandwidth (VBW): It refers to the bandwidth of the filter applied to the detected signal which also includes smoothing the trace and reducing noise.
9. Power Spectral Density (PSD): It denotes the measure of signal power per unit frequency. This metric is useful to analyze noise and signal strength.
10. Peak Search Frequency: The frequency at which the peak signal level is detected within the measurement range.
11. Attenuation: It indicates the reduction of signal amplitude which is often adjusted to prevent overload and ensure accuracy in measurements.
12. Time Sweep: It is the duration over which the analyzer sweeps through the frequency range, affecting the resolution and speed of the measurement.

The above signal analyzer measurements are automated using LabVIEW software which is discussed in Chapter 4.

2. SOFTWARE:

2.1 Introduction of LabVIEW:

LabVIEW is a graphical programming environment created by National Instruments that provides powerful features for instrument control and simulation, DAQ (Data Acquisition) applications, developing test benches and stations for automated measurement in the many fields of research and development in industry and academia. It was first released for the Apple Macintosh in 1986. It is widely used in various industries and applications, such as science, research, manufacturing, testing, measurement, automation, control, medical devices, aerospace, defense, energy, utilities, automotive, and telecommunications.

2.2 Advantages of LabVIEW

LabVIEW provides its users with numerous benefits:

1. LabVIEW is recognized for its user-friendly and visual nature of programming due to its graphical interface, making it accessible to engineers and researchers without programming expertise.

2. It offers support for multiple operating systems, including Windows, MacOS, and Linux, making it suitable for various types of systems and their associated applications.

3. LabVIEW provides a wide-ranging library of built-in functions and tools for data acquisition, instrument control, simulation, automation, analysis, and visualization applications, owing to its graphical programming environment and cross-platform functionality.

4. The software boasts a large user community and comprehensive online support resources, including discussion forums, tutorials, and example codes and programs.

5. LabVIEW is capable of communicating with other systems and tools, such as PLCs, microcontrollers, and various software, using interfaces like Ethernet, GPIB, and USB, all of which will be further discussed in upcoming chapters.

6. The scalability of LabVIEW enables applications ranging from simple data acquisition to complex control systems, making it suitable for automated test and measurement projects of varying requirements.

7. LabVIEW can be seamlessly integrated with other software such as MATLAB, Python, and Microsoft Excel, offering compatibility for data analysis and visualization purposes.

2.3 Pitfalls of LabVIEW

1. Expense: LabVIEW can be costly, which can be a hurdle for some developers and organizations.
2. Steep learning curve: Learning LabVIEW can be tough for individuals in academia and industry who have little to no programming experience.
3. Limited portability: Code sharing and reusability across different platforms can be more difficult compared to traditional programming languages.
4. Limited support for distributed systems: While LabVIEW does support distributed systems, its capabilities are not as robust as languages like C++ and Python.

2.4 Data flow programming

Data flow programming is a fundamental aspect of the LabVIEW development environment that sets it apart from other programming languages. We are already aware that data flow refers to how data moves between different program elements such as functions, controls, and indicators. Data moves through

programs by following connections made by wires that link the terminals of different components. Data moves from the output terminals of one component to the input terminals of another component, making it easy for users to interpret and visualize the data and logic flow of the program.

The data flow in LabVIEW differs from the traditional programming model, in which the program flow is determined by the order of code statements. In LabVIEW, data flow is determined by the connections between elements, and data is processed as it moves through the program. This allows for the creation of highly concurrent applications that can take advantage of multiple processors and cores.

2.3 Performance of LabVIEW:

The performance of LabVIEW is influenced by a variety of factors, including the intricacy of the program, the hardware, the configuration settings, and the specific operations and functions executed within the application. It is widely recognized for excelling in simple data acquisition and instrument control applications as it has the capability to gather and process substantial amounts of data in real-time while maintaining low-latency control over instruments and other hardware. It can also be affected by the algorithm’s complexity and the volume of data processed in intricate applications such as image processing or real-time control applications.

2.4 Compiled language

LabVIEW, being a compiled language, involves the conversion of a created program into machine code. This machine code is then directly processed and executed by the computer's CPU. This differs from interpreted languages, where the translation into machine code occurs at runtime. Compiled languages offer

advantages over interpreted languages, such as faster execution rates and the capability to create stand-alone executables.

2.5 Virtual Instruments:

LabVIEW programs or subroutines are called virtual instruments. Each virtual instrument consists of three components:

1. Front panel: The user interfaces created in LabVIEW is the front panel for the virtual instrument (VI). It includes controls and indicators which can be used as input and output terminals of the VI. Controls make up the input devices like knobs, push buttons, dials, and many others while indicators are LEDs and other displays that require indication of units or levels in accordance with the instrument involved. Any components placed on the front panel are automatically placed on the block diagram.
2. Block diagram: Block diagram contains the code. It resembles a flowchart showing the dataflow from one element to the next. Structures like while, for loops and case structures graphically encompass the operational code within it.
3. Connector Pane: The connector pane specifies the VI's inputs and outputs by linking the controls and indicators of the front panel to the icon's terminals.

2.6 Interaction of LabVIEW with hardware:

The software developed with LabVIEW interacts with the required hardware with an appropriate processor with the following:

1. A Graphical user interface (GUI)
2. Interface with instruments through GPIB, Ethernet, USB, PCI, RS-422
3. Measuring and controlling signals with NI hardware (analog or digital)

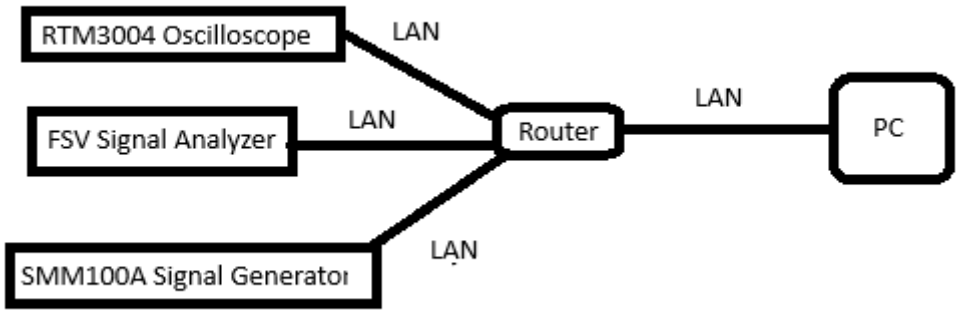

Figure 1 : Instrument Connection Diagram

2.7 Environments supported by LabVIEW:

LabVIEW supports a wide range of platforms such as the following:

1. All modern Windows versions
2. X86 and x64 processors
3. ARM processors (Advanced RISC machines)
4. PXI and PXI Express which are PCI extensions for instrumentation
5. CompactRIO also known as reconfigurable input/output devices
6. Real-time controllers
7. FPGA devices

2.8 Example Code of LabVIEW

The figure below depicts an example of a LabVIEW VI that operates a query about the system date to the oscilloscope instrument connected to the PC via LAN or GPIB interface. It consists of a VISA resource selection control with which the source instrument can be selected. After successful connection of the instrument to the computer, the month, day and year is displayed in the respective indicator components along with the error in and error out dialog box components.

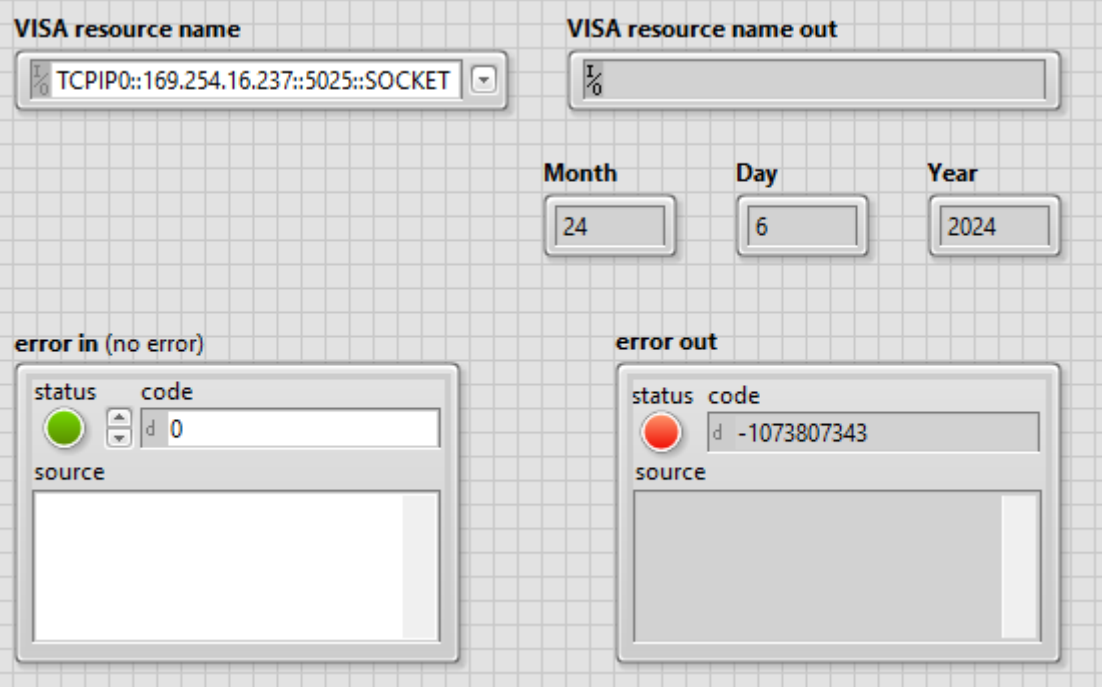

Figure 2: Front panel of rsscope Query System Date.vi

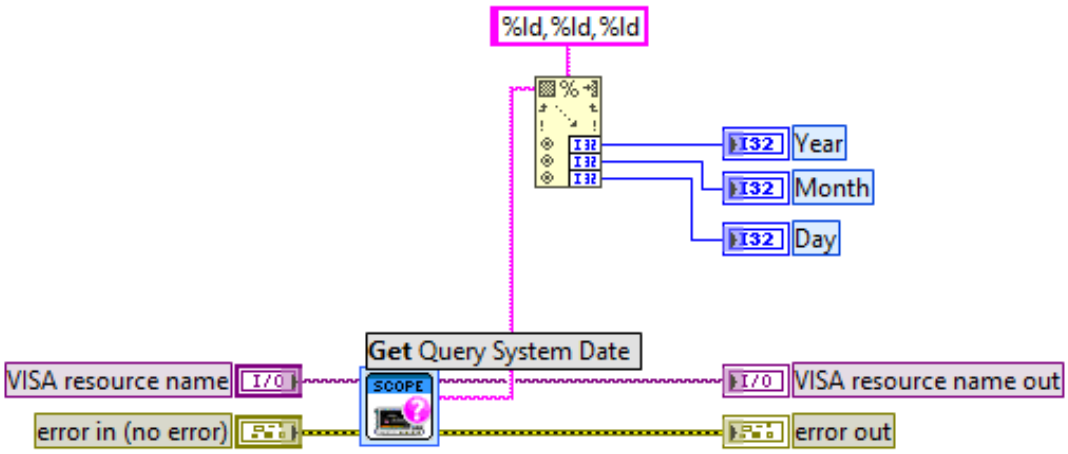

Figure 3: Block Diagram of rsscope Query System Date.vi

3. INTERCONNECTIONS

3.1 Ethernet connection

Ethernet, also known as Local Area Network (LAN), is a widely used interface cable for connecting devices within a limited area such a laboratory. It is crucial for modern instrumentation and automation due to its high-speed, reliable and scalable nature. It enables efficient communication and data transfer between multiple devices essential for automated measurement and control systems. With the use of Ethernet, researchers and engineers can achieve seamless integration of various instruments increasing the efficiency and accuracy of the testing procedures and experiments.

3.1.1 Key Features

1. High Data Transfer rates: Ethernet supports high-speed data transmission with current standards offering speeds up to 100 Gbps (giga bytes per second).
2. Reliability and stability: It provides a stable and secure connection with consistent performance and low latency which is an utmost requirement for the software developed.
3. Scalability: It is easily scalable to accommodate additional devices and increased data traffic without significant infrastructure changes.
4. Flexibility: It is compatible with a wide range of devices and operating systems proving to be a versatile choice for various instrumentation applications.
5. Cost-Effective: Ethernet interface is relatively inexpensive to install and maintain as compared to the other interfacing options for networking.

Ethernet or LAN cables are used for connecting the instruments to the computer via a router for measurement automation with the LabVIEW software already installed in the computer which is detailed in Chapter 4.

3.2 VISA

VISA (Virtual Instrument Software Architecture) is a standardized interface for configuring, programming, and troubleshooting instrumentation systems that utilize various communication protocols like GPIB, PXI, Serial, Ethernet, and USB. It streamlines the process of connecting and communicating with diverse instruments from different manufacturers by abstracting hardware details. This allows users to concentrate on higher-level programming and control. The standardization ensures compatibility and interoperability in complex measurement and automation systems, making VISA indispensable in modern laboratory and industrial settings.

NI-VISA, or National Instrument VISA, offers robust support for a wide array of instruments and interfaces. It enables seamless communication and control of instruments such as oscilloscope, vector signal generators and spectrum analyzers.

NI-VISA is installed in the controller computer from the National Instruments website. The installation process involves running the NI Package Manager that manages the installation and upgrading of NI software. However, it is recommended to install the programming environment of NI LabVIEW before installing NI-VISA for proper integration.

3.2.1 NI-VISA utilities

NI-VISA includes several utilities and application services that facilitate instrument control and data acquisition:

1. NI DAQmx: It is a driver software designed for NI data acquisition (DAQ) hardware that streamlines the configuration and programming of DAQ devices offering high-performance data acquisition and signal conditioning.

2. NI MAX (Measurement and Automation Explorer): It is a configuration tool for managing NI hardware and software that provides a graphical interface for setting up devices, creating tasks and viewing data making it an essential tool for system setup.
3. NI VISA Interactive Control: NI VISA Interactive Control is a utility that facilitates communication and control of instruments using the VISA API. Users can interactively test their VISA commands and view the responses from their instruments.
4. NI IO Trace: NI IO Trace is a debugging tool that captures and displays the communication between software applications and instruments. It logs VISA, and other I/O calls, allowing users to analyze and troubleshoot their instrument control code.
5. NI Launcher: This utility provides quick access to the NI software and tools by organizing installed NI applications and utilities in a single interface.
6. NI Device Monitor: This utility automatically detects and displays information about connected NI hardware. It provides real-time status updates and notifications from their devices and ensures that their systems are properly configured and operational.
7. NI Remote Server: This enables remote access and control of NI hardware and software. It allows users to manage and interact with their measurement and automation systems from a remote location.
8. NI Hardware Configuration Utility: It is a tool for configuring and managing NI hardware devices that provides a graphical interface to set up device parameters, perform firmware updates, and manage device settings.

3.2.2 Controlling Instruments with NI-VISA

With the use of NI-VISA, users can control oscilloscopes, signal generators, and spectrum analyzers through a standardized API for communication. This allows

for the creation of programs in LabVIEW environment to send commands and receive data from these instruments. For instance, an oscilloscope can be set to capture waveforms, a signal generator can be instructed to produce specific signals, and a spectrum analyzer can be configured to measure signal spectra. NI-VISA manages the underlying communication protocols that ensure reliable and efficient data transfer between the computer and the instruments.

3.3 Standard Commands for Programmable Instruments (SCPI) commands

Standard Commands for Programmable Instruments (SCPI) commands is a standardized universal language used to control and communicate with test and measurement instruments. It is developed to provide a consistent and universal way to program instruments. These commands are widely used in laboratories and industries for automating measurements and tests.

1. SCPI commands follow a uniform syntax allowing users to control different instruments from various manufacturers from various manufacturers using a common set of instructions.
2. These commands are organized in a hierarchical manner enabling systematic navigation through different functions and settings of an instrument.
3. SCPI commands simplify the process of writing and interpreting instrument control scripts and are designed to be easily understood.
4. They support a wide range of functions from basic settings like frequency and power level to complex level operations such as data acquisition and error handling.
5. SCPI-supported instruments can be integrated into automated test systems facilitating seamless communication and control across different devices.

Some SCPI commands used in the instrument automation:

1. SYSTem:PRESet : This command resets the instrument to its default settings. It is used to ensure a known starting state before beginning a new measurement.
2. SOUR:FREQ 1 GHz : This command sets the frequency of the signal to 1 GHz. It is used to configure the output frequency of a signal generator.
3. READ<ch>[:POWer]? : This command queries the power measurement from the specified channel (<ch>). It is used to retrieve the power level measured by the instrument.
4. FREQ:CENT 400 MHz : This command sets the center frequency of the instrument to 400 MHz. It is typically used in spectrum analyzers to define the central point of the frequency span.
5. FREQ 500kHz : This command sets the frequency to 500 kHz. It is used to configure the frequency of the instrument, similar to the SOUR:FREQ command but without specifying the source.
6. FREQ:MODE SWE : This command sets the frequency mode to sweep. It is used to configure the instrument to sweep across a range of frequencies, which is useful for analyzing frequency-dependent behavior.
7. :SYST:ERR:ALL? : This command queries all system errors. It retrieves a list of all errors currently recorded by the instrument, helping diagnose issues and ensure proper operation.

4. AUTOMATION OF MICROWAVE SYSTEM FOR QUANTUM SENSING SYSTEM

4.1 Initial Python Programming for RTM3004 oscilloscope

The microwave system consists of a signal generator and oscilloscope wherein the generation of RF signal takes place and the signal analysis and control is done by the oscilloscope. Initially, the test setup was conducted to observe the clear and accurate functioning of the instruments and to test their responses to functions developed using Python with a PySimpleGUI framework which comprised of Standard Commands for Programmable Instruments or SCPI commands. The initial software testing functions will be elaborated in the software chapter ahead. The signal generator and oscilloscope responded well to basic Python and SCPI functions, hence, verifying their functionality. The python programming of the probes, oscilloscope and the interfaces took a lot of time which resulted in the shift to LabVIEW programming.

4.1.1 SCPI Programming with Python and RsInstrument

RsInstrument is a Rohde & Schwarz developed library which can be used while Python programming for instrument communications or measurement. Initial Python code to establish connection and enable instrument communication with the computer using SCPI commands is as shown below:

```
# For RTM3004 Oscilloscope
# Preconditions:
# - Installed RsInstrument Python module from pypi.org
# - Installed VISA e.g. R&S Visa 5.12.x or newer

from RsInstrument import *
from time import time
RsInstrument.assert_minimum_version('1.53.0')

try:
   rtm = RsInstrument('TCPIP::169.254.16.237::INSTR', True, False)
   rtm.visa_timeout = 3000
```

```
    rtm.opc_timeout = 15000
    rtm.instrument_status_checking = True
except Exception as ex:
    print(f'Error initializing the instrument session:\n{ex.args[0]}')
    exit()

print(f'RTM3004 IDN: {rtm.idn_string}')
print(f'RTM3004 Options: {","}.join(rtm.instrument_options)}')

rtm.clear_status()
rtm.reset()
#rtm.write_str("SYST:DISP:UPD ON")  # Enable display update (switch OFF after debugging)

# Basic Settings:
rtm.write_str("ACQ:POIN:AUTO ON")  # Enable automatic acquisition points # not working in RTO2044
rtm.write_str("TIM:RANG 0.01")  # 10 ms Acquisition time
rtm.write_str("ACQ:POIN 20002")  # 20002 X points  #not working in RTO2044
rtm.write_str("CHAN1:RANG 2")  # Horizontal range 2V
rtm.write_str("CHAN1:POS 0")  # Offset 0
rtm.write_str("CHAN1:COUP AC")  # Coupling AC 1MOhm
rtm.write_str("CHAN1:STAT ON")  # Switch Channel 1 ON

# Trigger Settings:
rtm.write_str("TRIG1:MODE AUTO")  # Trigger Auto mode
rtm.write_str("TRIG1:SOUR CHAN1")  # Trigger source CH1
rtm.write_str("TRIG1:TYPE EDGE;:TRIG1:EDGE:SLOP POS")  # Trigger type Edge Positive
rtm.write_str("TRIG1:LEV1 0.04")  # Trigger level 40mV
rtm.query_opc()  # Wait until all instrument settings are finished

# Arming the RTM for single acquisition
rtm.visa_timeout = 2000  # Set acquisition timeout higher than the acquisition time
rtm.write_str("SING")

# Wait for acquisition to finish
rtm.query_opc()

# Fetch the waveform in ASCII and binary format
t = time()
```

```
trace_ascii = rtm.query_bin_or_ascii_float_list('FORM ASC;:CHAN1:DATA?')
# Query ASCII array of floats
print(f'Instrument returned {len(trace_ascii)} points in the ASCII trace, query duration {time() - t:.3f} secs')

t = time()
rtm.bin_float_numbers_format = BinFloatFormat.Single_4bytes #not working in RTO2044
trace_binary = rtm.query_bin_or_ascii_float_list('FORM REAL,32;:CHAN1:DATA?')  # Query binary array of floats
print(f'Instrument returned {len(trace_binary)} points in the binary trace, query duration {time() - t:.3f} secs')

# Capture a screenshot and transfer the file to the PC
rtm.write_str('HCOP:DEV:LANG PNG')  # Set the screenshot format
rtm.write_str(r"MMEM:NAME 'c:\temp\Dev_Screenshot.png'")  # Set the screenshot path
rtm.write_str("HCOP:IMM")  # Capture the screenshot
rtm.query_opc()  # Wait for the screenshot to be saved
rtm.read_file_from_instrument_to_pc(r'c:\temp\Dev_Screenshot.png', r'c:\Temp\PC_Screenshot.png')  # Transfer the file
print(r"Screenshot file saved to PC 'c:\Temp\PC_Screenshot.png'")
# Close the session
rtm.close()
```

4.1.2 PySimpleGUI code

The Python code developed initially to initiate a graphical user interface (GUI) using PySimpleGUI framework and pyvisa module for the connected RTM3004 oscilloscope is shown as follows:

```
import PySimpleGUI as sg
from RsInstrument import RsInstrument
import os
import pyvisa
import math
import csv

SIZE_X = 200
SIZE_Y = 100
NUMBER_MARKER_FREQ = SIZE_X // 8
AMPLITUDE = SIZE_Y // 2
```

```
FREQUENCY = 0.1
PHASE_SHIFT_SPEED = 0.1
rm = pyvisa.ResourceManager()

os_address = ""  # Correct address for the oscilloscope
# sig_gen_address = ""  # Correct address for the signal generator
# sig_gen = rm.open_resource(sig_gen_address)

try:
   oscilloscope = RsInstrument(os_address)
except Exception as e:
   print(f"error init oscilloscope: {e}")

# GUI layout
oscilloscope_tab_layout = [
   [
      sg.Button("Capture Screenshot"),
      sg.Button("Transfer Waveform"),
      sg.Button("Show oscilloscope waveform"),
   ],
   [sg.Text("Select Channel:"), sg.Combo(["CH1", "CH2", "CH3"], key="-
CHANNEL-")],
   [sg.Text("Set Horizontal scale"), sg.Stretch(), sg.Text("Set Vertical scale")],
   [
      sg.Slider(range=(0, 10), default_value=5, orientation="h", key="-
HORIZONTAL-"),
      sg.Slider(range=(0, 10), default_value=5, orientation="h", key="-
VERTICAL-"),
   ],
   [sg.Button("Auto"), sg.Button("Normal"), sg.Button("Single")],
   [sg.Multiline(size=(30, 3), key="-OSC_DETAILS-", disabled=True)],
   [
      sg.Slider(
         (0, SIZE_Y),
         orientation="h",
         enable_events=True,
         key="_SLIDER_1",
         expand_x=True,     )
   ],
   [
      sg.Slider(
         (0, SIZE_Y),
         orientation="h",
```

```
            enable_events=True,
            key="_SLIDER_2",
            expand_x=True, )
    ],
]

# Layout for the Signal Generator Controls Tab
signal_generator_tab_layout = [
    [sg.Text("Frequency (Hz):"), sg.InputText(key="-FREQ-")],
    [sg.Text("Power Level (dBm):"), sg.InputText(key="-POWER-")],
    [sg.Button("Start Signal"), sg.Button("Stop Signal")],
    [
        sg.Text("Select Waveform Type:"),
        sg.Combo(["Sine", "Square", "Triangle"], key="-WAVEFORM_TYPE-"),
    ],
]

# Main layout with tabs
layout = [
    [
        sg.TabGroup(
            [
                [sg.Tab("Oscilloscope Controls", oscilloscope_tab_layout)],
                # [graph_frame],
                [sg.Tab("Signal Generator Controls", signal_generator_tab_layout)],
            ]
        )
    ],
    [sg.Button("Exit")],
]
# Window
window = sg.Window("Instrument Automation", layout, resizable=True,
element_justification="center")

def create_waveform_data():
    waveform_data = [[t, 5 * math.sin(0.1 * t) + 5] for t in range(100)]
    return waveform_data

def show_waveform_data_as_table(waveform_data):
    layout = [
        [
            sg.Table(
                values=waveform_data,
```

```
            headings=["Time", "Voltage"],
            display_row_numbers=True,
            auto_size_columns=True,
            num_rows=min(25, len(waveform_data)),  )
        ]
    ]
    window = sg.Window("Waveform Data", layout)
    event, values = window.read()
    window.close()
def plot_real_time_oscilloscope():
    SIZE_X = 200
    SIZE_Y = 100
    AMPLITUDE = SIZE_Y // 2
    FREQUENCY = 0.1
    PHASE_SHIFT_SPEED = 0.1

    # GUI layout
    layout = [
        [
            sg.Graph(
                canvas_size=(800, 400),
                graph_bottom_left=(-SIZE_X, -SIZE_Y),
                graph_top_right=(SIZE_X, SIZE_Y),
                background_color="black",
                key="-GRAPH-",        )
        ],
        [
            sg.Text("Time Axis:"),
            sg.Slider(range=(0,  100),  default_value=0,  orientation="h",  key="-
TIME-"),
            sg.Text("Offset:"),
            sg.Slider(
                range=(-SIZE_Y, SIZE_Y),
                default_value=0,
                orientation="h",
                key="-OFFSET-",   ),
            sg.Text("Channel:"),
            sg.Combo(
                ["CH1",  "CH2",  "CH3",  "All"],  default_value="CH1",  key="-
CHANNEL-"    ),
            sg.Text("Concurrency:"),
            sg.Radio("Active", "CONCURRENCY", default=True, key="-ACTIVE-
"),
```

```
            sg.Radio("Inactive", "CONCURRENCY", key="-INACTIVE-"),
        ],
        [sg.Button("Exit")],
    ]
    # Window
    window = sg.Window("Real-Time Oscilloscope", layout, finalize=True)
    # Function to draw the sine wave
    def draw_sine_wave(graph, amplitude, frequency, phase_shift):
        graph.erase()
        draw_axis(graph)  # Draw axes
        prev_x = prev_y = None
        for x in range(-SIZE_X, SIZE_X):
            y=amplitude*math.sin(frequency*(x*2*math.pi/SIZE_X) + phase_shift )
            if prev_x is not None:
                graph.draw_line((prev_x, prev_y), (x, y), color="yellow")
            prev_x, prev_y = x, y

    # Function to draw the axes
    def draw_axis(graph):
        graph.draw_line((-SIZE_X, 0), (SIZE_X, 0), color="white")  # X-axis
        graph.draw_line((0, -SIZE_Y), (0, SIZE_Y), color="white")  # Y-axis

    phase_shift = 0

    # Event loop
    while True:
        event, values = window.read(timeout=100)  # Update every 100
milliseconds
        if event == sg.WINDOW_CLOSED or event == "Exit":
            break
        amplitude = AMPLITUDE
        frequency = FREQUENCY
        channel = values["-CHANNEL-"]
        if channel == "All":
            for ch in ["CH1", "CH2", "CH3"]:
                draw_sine_wave(window["-GRAPH-"], amplitude, frequency,
phase_shift)
                amplitude -= 10  # Offset amplitude for multiple channels
                frequency += 0.05  # Offset frequency for multiple channels
        else:
            draw_sine_wave(window["-GRAPH-"], amplitude, frequency,
phase_shift)
        phase_shift += PHASE_SHIFT_SPEED
```

```
    # Window close
    window.close()


def PopUp_SS(pc_path):
    answer = sg.popup_yes_no("Screenshot captured. Would you like to view it?")
    if answer == "Yes":
        if os.path.exists(pc_path):
            screenshot_layout = [
                [sg.Image(filename=pc_path)],
                [sg.Button("Close")],
            ]
            screenshot_window = sg.Window("Screenshot", screenshot_layout)
            while True:
                screenshot_event, _ = screenshot_window.read()
                if screenshot_event == sg.WIN_CLOSED or screenshot_event ==
"Close":
                    screenshot_window.close()
                    break
        else:
            sg.popup_error("Screenshot file not found.")

# Event loop
while True:
    event, values = window.read()
    pc_path=
"C:\\Users\\Vindhya Sree\\OneDrive\\Desktop\\CSIR\\GUI_Images\\GUI2.png"
    waveform_filename = (
    "C:\\Users\\Vindhya Sree\\OneDrive\\Desktop\\CSIR\\CSVs\\WFM02.CSV")
    if event == sg.WIN_CLOSED or event == "Exit":
        break
    elif event == "Capture Screenshot":
        PopUp_SS(pc_path)
    elif event == "Transfer Waveform":
        waveform_data = create_waveform_data()
        show_waveform_data_as_table(waveform_data)
        sg.popup("Waveform data transferred to PC.")
        oscilloscope.write_str_with_opc("FORM REAL,32")
        oscilloscope.write_str_with_opc("CHAN1:DATA?")
        waveform_data = oscilloscope.read_bin_float_data()
    elif event == "Show oscilloscope waveform":
        plot_real_time_oscilloscope()
```

```
    elif event == "Set Frequency and Power":
        frequency = values["frequency"]
        power = values["power"]
        # SCPI command to instrument that sets freq and power level.
        sig_gen.write(f"SOUR:FREQ {frequency}MHz")
        sig_gen.write(f"SOUR:POW {power} dBm")
        sg.popup(f"Frequency set to {frequency} MHz and Power to {power} dBm",  size=(400, 150), )
    elif event == "Set Trigger":
        oscilloscope.write_str_with_opc("TRIGger:MODE LEVel")
        oscilloscope.write_str_with_opc("TRIGger:LEVel:SOURce CHANnel1")
        oscilloscope.write_str_with_opc("TRIGger:LEVel CHANnel1,0.5")
        sg.popup("Trigger set on oscilloscope.", size=(400, 150))
    elif event == "Set FFT":
        oscilloscope.write_str_with_opc("CALCulate:FFT:WINDow HANNing")
        oscilloscope.write_str_with_opc("CALCulate:FFT:SOURce CHANnel1")
        oscilloscope.write_str_with_opc("DISPlay:FFT:WINDow:STATe ON")
        sg.popup("FFT set on oscilloscope.", size=(400, 150))
# Window close
window.close()
```

The code above lets the user select channels, set horizonal and vertical scales, select modes like auto, normal and single, capture screenshot, transfer waveform data to the computer and show the oscilloscope waveform and control the waveform by activating and inactivating concurrency in channels in the oscilloscope controls tab. The output of the above code is shown in Figure 4:

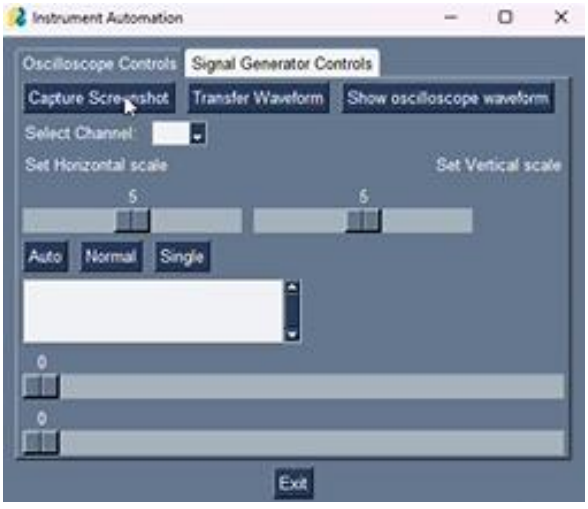

Figure 4 : PySimpleGUI for oscilloscope controls

4.2 Signal generator in LabVIEW

The signal generator connected with the computer with the instrument drivers and NI VISA already installed to establish communication and facilitate measurement automation is used to generate a RF signal which will be further analyzed by the oscilloscope connected with Ethernet cables via a router. The VI is developed to send and set measurement level from the panel to the signal generator as shown in Figure 5. The measurements of RF Frequency (in GHz), limit, offset and attenuation is entered by the user in the front panel. The power limit is set by converting the entered floating point value into ASCII string. The result is then concatenated into the SCPI command ":SOURn:POW", where n is the number of controlled output. The error out box displays the errors sent by the signal generator. The complete programming of the VI is shown in the block diagram in Figure 6.

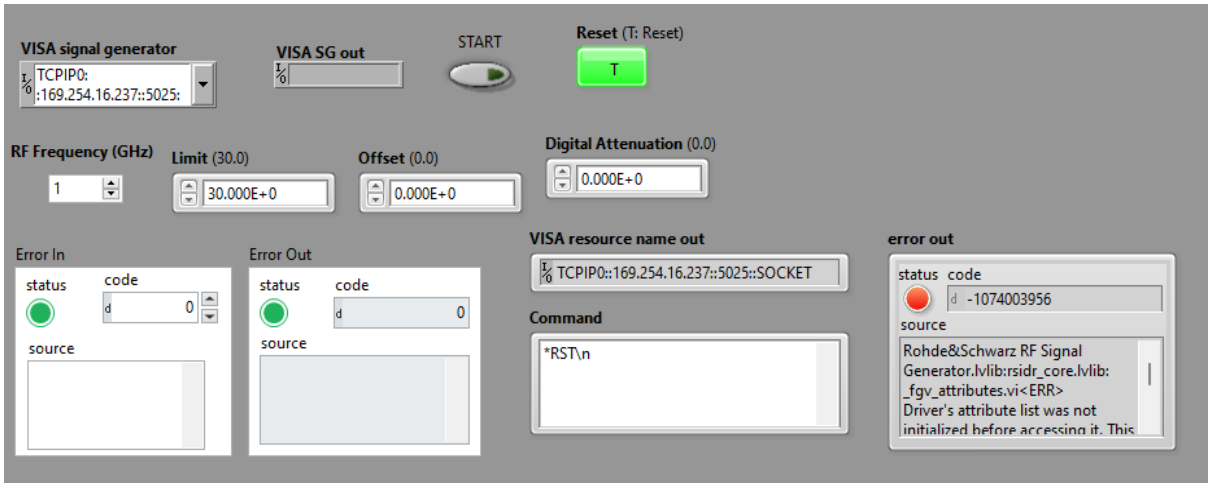

Figure 5: Front panel of signalGenerator.vi

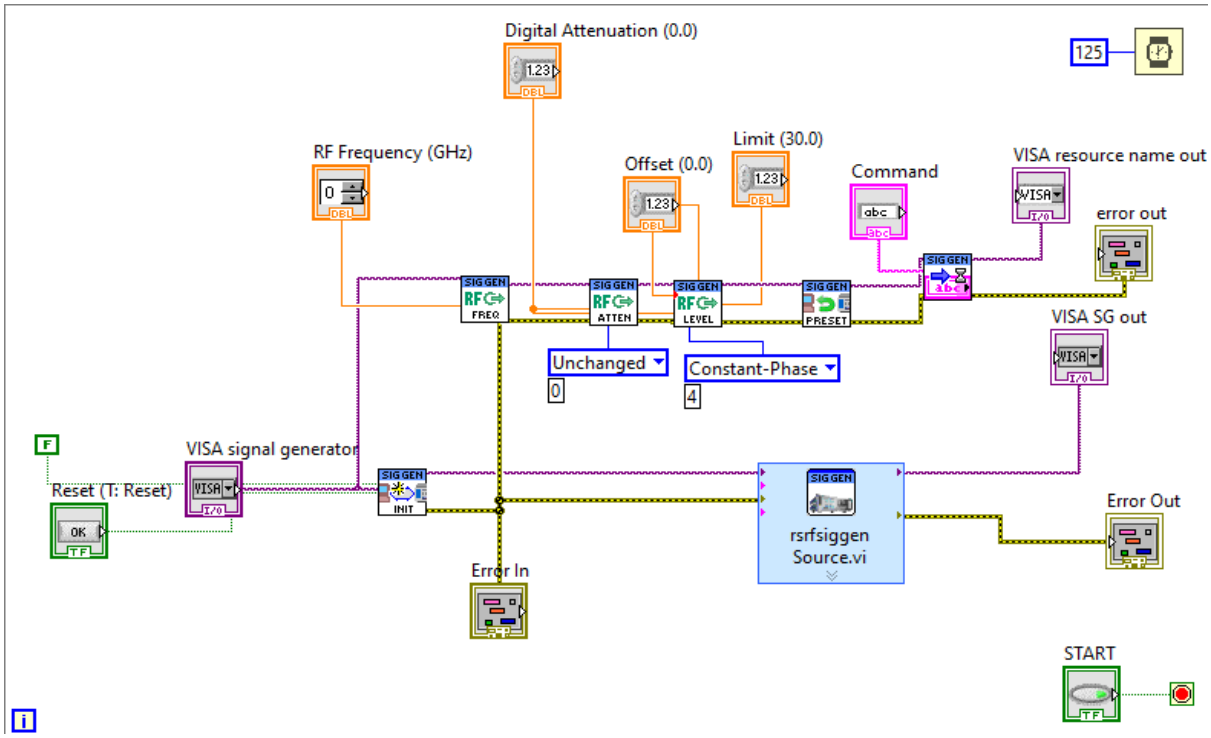

Figure 6: Block Diagram of signalGenerator.vi

4.3 Oscilloscope in LabVIEW

The oscilloscope connected with the computer is used to display, control and analyze the RF signal generated by the signal generator. The VI is developed to send and set measurement levels from the front panel which is the user interface to the oscilloscope as shown in Figure 7. The measurements of trigger level, trigger slope, vertical ranges, offsets and coupling for four channels are entered by the user in the front panel. There exists a function to retrieve waveform data points of the oscilloscope trace to the computer. The error out box displays the errors sent by the signal generator. Since the instrument was not functioning properly and not reading commands appropriately, the oscilloscope VI was not used. The complete programming of the oscilloscope VI is shown in the block diagram in Figure 8.

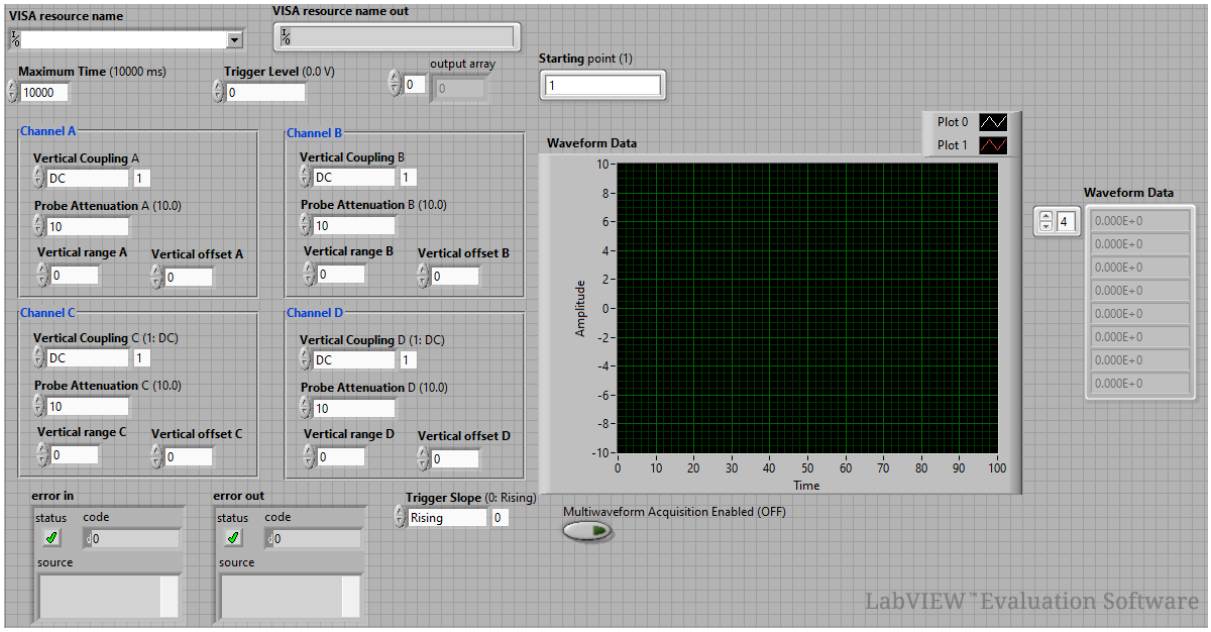

Figure 7: Front panel of oscilloscope.vi

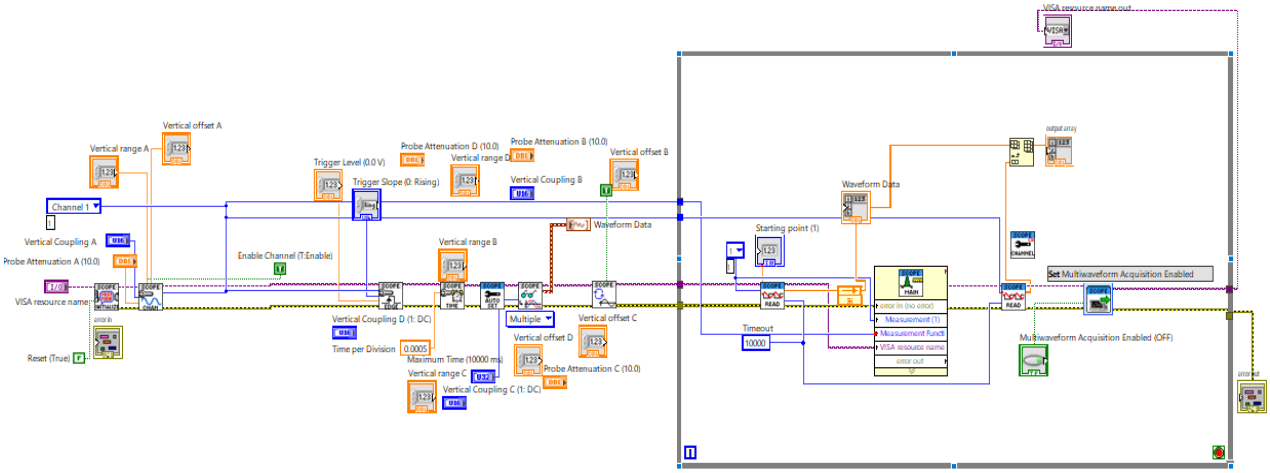

Figure 8: Block Diagram of oscilloscope.vi

4.3 Spectrum Analyzer in LabVIEW

The spectrum analyzer VI is used to read power spectral density and peak search frequency, set reference level and its offset, start and stop frequency, attenuation, time sweep, centre frequency, resolution and video bandwidth and to open waveform data in an XML file as input to the waveform trace in the spectrum analyzer as shown in Figure 9. These measurements are entered by the user in the

front panel that is the user interface. The complete programming of the spectrum analyzer VI is shown in the block diagram in Figure 10.

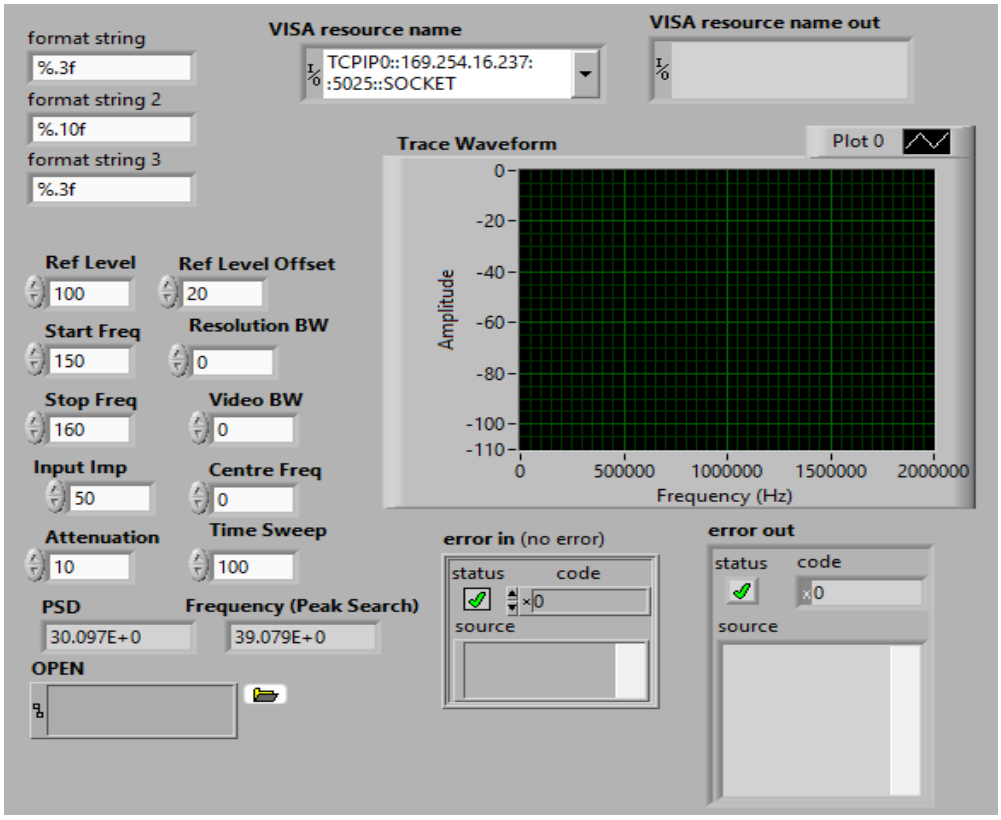

Figure 9: Front panel of SpectrumAnalyzer.vi

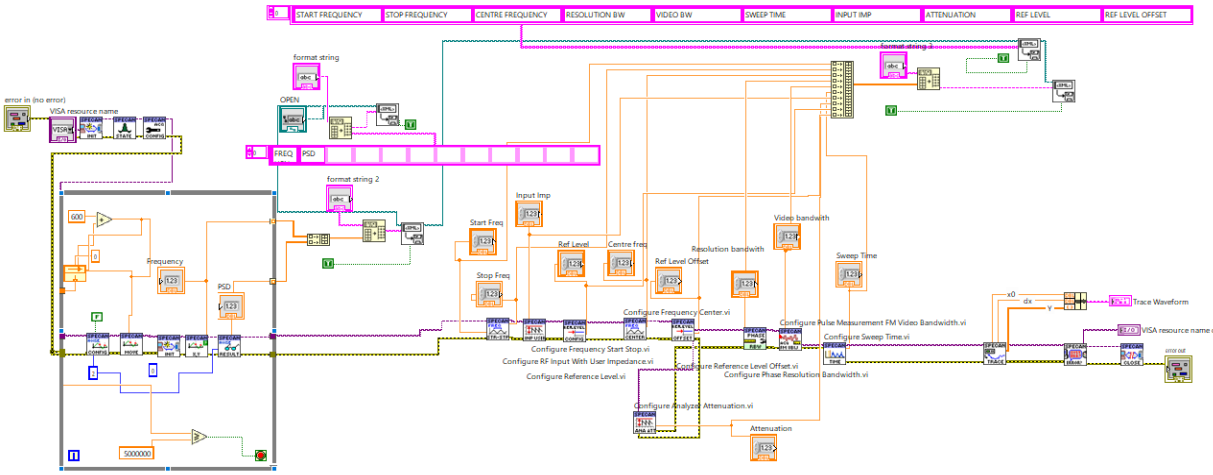

Figure 10: Block Diagram of SpectrumAnalyzer.vi

CONCLUSION:

This project report has outlined the development and implementation of an automated RF and microwave measurement system developed for a Rydberg atom-based quantum sensing system. By integrating advanced instruments, embedded systems, and interconnections, the project successfully utilized LabVIEW programming to create a graphical user interface (GUI) that automates the process of querying and receiving microwave measurements.

The synchronized operation of instruments such as signal generators, oscilloscopes, and spectrum analyzers is essential for achieving accurate and reliable measurements in research laboratories where precision is key. Automation of these instruments proves to minimize manual intervention and significantly improves the speed and consistency of the testing and measurement processes in terms of time efficiency, reliability, and throughput. The detailed chapters on hardware, software, and interconnections provides a thorough understanding of the setup and programming required to form an automated measurement system.

In conclusion, the automation of RF and microwave systems for Rydberg atom-based quantum sensing marks a significant advancement in the domain of quantum metrology. The implementation of this project highlights the potential for further research and development in this area, paving the way for new applications and improvements in quantum sensing technologies. This work contributes to the ongoing efforts to enhance the accuracy, efficiency, and reliability of quantum measurements, supporting broader scientific and technological progress.

BIBLIOGRAPHY:

- https://www.ni.com/en/shop/labview.html
- https://www.viewpointusa.com/labview/what-is-labview-used-for/
- https://www.s5solutions.com/labview-graphical-programming.html
- https://labviewwiki.org/wiki/LabVIEW
- https://www.rohde-schwarz.com/us/products/test-and-measurement/vector-signal-generators/rs-smm100a-vector-signal-generator_63493-834088.html
- https://www.rohde-schwarz.com/us/products/test-and-measurement/vector-signal-generators_333559.html
- https://www.everythingrf.com/products/signal-generators/rohde-schwarz/564-92-r-s-smm100a
- https://www.rohde-schwarz.com/us/products/test-and-measurement/oscilloscopes/rs-rtm3000-oscilloscope_63493-427459.html
- https://www.tequipment.net/Rohde-&-Schwarz/RTM3004/Digital-Oscilloscopes/
- https://www.geeksforgeeks.org/what-is-ethernet/
- https://www.zgsm-wireharness.com/blog/ethernet-cable-overview/
- https://www.theemcshop.com/emc-test-equipment/emc-receivers-emi-analyzers/rohde-schwarz-fsv40-10-hz-40-ghz-spectrum-analyzer/
- https://www.rohde-schwarz.com/us/manual/r-s-fsva-and-r-s-fsv-user-manual-manuals_78701-29310.html
- https://www.ni.com/pdf/manuals/ni-visa-windows-2023-q3.html
- https://www.ni.com/en/support/downloads/drivers/download.ni-visa.html

REFERENCES:

[1] M. Fleischhauer, A. Imamoglu, and J. P. Marangos, “Electromagnetically induced transparency: Optics in coherent media,” Rev. Mod. Phys., vol. 77, no. 2, pp. 633–673, 2005, doi: 10.1103/revmodphys.77.633.

[2] J. Yuan, W. Yang, M. Jing, H. Zhang, Y. Jiao, W. Li, L. Zhang, L. Xiao and S. Jia, "Quantum sensing of microwave electric fields based on Rydberg atoms," Rept. Prog. Phys., no.10, 106001 (2023), doi:10.1088/1361-6633/acf22f

[3] J. Bourassa and C. M. Wilson, "Progress Towards Quantum-Enhanced Microwave Remote Sensing and Communication Applications," 2024 United States National Committee of URSI National Radio Science Meeting (USNC-URSI NRSM), Boulder, CO, USA, 2024, pp. 204-204, doi: 10.23919/USNC-URSINRSM60317.2024.10464708.

[4] S. E. HARRIS, “Electromagnetically Induced Transparency,” Opt. Photonics News, vol. 2, no. 12, p. 29, 2009, doi: 10.1364/opn.2.12.000029.

[5] S. E. H. K. J. Boller, A. Imamoglu, “Observation of electromagnetically induced transparency,” Phys. Rev. Lett., vol. 66, no. 20, pp. 2593–96, 1991.