PRL-TN-2025-119
Technical Note

Python-Based GUI for Real-Time Digital and Analog Signal Monitoring Using NI USB-6363 DAQ

by

Kush Shah, Aaditya Sarda

भौतिक अनुसंधान प्रयोगशाला, अहमदाबाद
Physical Research Laboratory, Ahmedabad

Published by
The Dean's office, PRL.

Python-Based GUI for Real-Time Digital and Analog Signal Monitoring Using NI USB-6363 DAQ

Kush Shah*[1], Aaditya Sarda**[2]

Received 25 April 2025; revised 21 Jul 2025; accepted 11 Aug 2025; published 26 August 2025.

Abstract
This technical note presents a Python-based graphical user interface (GUI) designed for the real-time acquisition and intuitive visualization of both analog and digital signals using a National Instruments (NI) USB-6363 data acquisition (DAQ) device. By leveraging open-source libraries, including PyQt5, pyqtgraph, and nidaqmx, the system enables seamless multi-channel signal processing with precise control, high sampling rate, efficient data handling, and responsiveness. Designed with the open-source modular architecture and advance multithreading technique, the GUI ensures low latency performance, making it highly scalable and adaptable to diverse application needs. The system has been rigorously evaluated against industry-standard equipment, demonstrating robust performance and accuracy compare to commercial oscilloscope.

Keywords
NI USB-6363, Data Acquisition, Python, GUI, Real-Time Visualization

[1]*Devang Patel Institute of Advance Technology and Research, CHARUSAT, Anand*
[2]*Space and Atmospheric Sciences Division, Physical Research Laboratory, Ahmedabad*
***Project Trainee**: kush300603@gmail.com
****Corresponding author**: aaditya@prl.res.in

Contents

1. Introduction

Signal acquisition and analysis are fundamental requirements across domains including electronics, telecommunications, embedded systems, and scientific research. While high-end commercial oscilloscopes provide exceptional performance and advanced features, they often fall short in flexibility and customizability due to proprietary interfaces, limited on-device data processing, and long-term data logging constraints. However, hardware devices like National Instruments (NI) Data Acquisition (DAQ) systems offer high-speed, multichannel signal processing with exceptional capabilities for capturing high-speed digital and precision analog signals. The potential of these sophisticated DAQ devices is often limited by proprietary software like LabVIEW, which can be costly, complex to learn, and offer limited customization options. Recent advancements in Python-based instrumentation frameworks have further validated its suitability for high-performance, real-time applications. Studies have demonstrated successful implementation of Python for data acquisition, signal visualization, and instrument control in both academic and industrial settings (Hughes, 2010; Koerner et al., 2019; Martins, 2023; Akam and Walton, 2019; Jäger and Gümmer, 2023). In this technical note, we present an open-source Python-based graphical user interface (GUI) that integrates seamlessly with the

NI USB-6363 DAQ, enabling real-time signal monitoring with exceptional accuracy and ease of use. By utilizing open-source libraries such as `PyQt5`, `PyQtGraph` , and `NIDAQmx`, the system provides a flexible platform for visualizing and analyzing a wide range of waveforms, including sine, ramp, pulse, and square signals. It also enables users to measure various key signal parameters such as voltage amplitude, frequency, and time period. This solution breaks through traditional proprietary limitations, empowering students, researchers, and industry professionals with a robust and accessible tool that interfaces advanced hardware with customized research applications. By critically evaluating the limitations of the existing tools and technologies, we define the specific problems and present a scalable, user-friendly solution that balances performance and accessibility.

2. Technical Objective

The objective of this work is to develop a modular, Python-based graphical user interface (GUI) for real-time, multi-channel analog and digital signal acquisition, visualization, and parameter analysis using the National Instruments USB-6363 DAQ device. The tool is designed to serve as an open-source, cost-effective alternative to commercial oscilloscopes and proprietary data acquisition software, with direct applications in satellite payload checkout systems and laboratory testing environments.

3. System Architecture

The system integrates data acquisition device with the Python-based software stack to enable real-time signal processing, visualization, and analysis. There are two separate GUIs for analog and digital channel read to optimize the use and make it modular, enabling 16-digital input lines and 4-analog input lines for their simultaneous operation, respectively. Visualization and analysis are performed on a desktop PC equipped with Python and the necessary open-source libraries.

Figure1 below illustrates the overall architecture of the real-time analog and digital signal monitoring system developed using the NI USB-6363 DAQ, Python, and a graphical user interface (GUI).

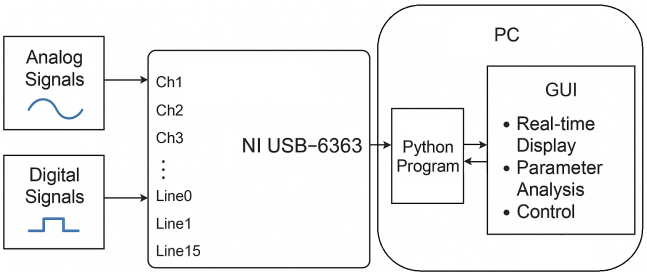


Figure 1. A general block diagram of the system

Figure 2 shows the software architecture block diagram. The system uses the NI-DAQmx Python API to interface directly with the NI USB-6363 hardware. Dedicated acquisition threads for analog and digital inputs run in parallel to capture real-time data. This data is forwarded to the Signal Processing module, which computes key parameters such as rising-edge detection, RMS, and peak-to-peak voltage. The processed results are routed to two outputs: the GUI layer (developed using PyQt5 and PyQtGraph) for real-time visualization, and an optional logging module that allows users to save data in CSV or JSON formats for offline analysis or archival. Users also have control over certain signal processing parameters through the interface.

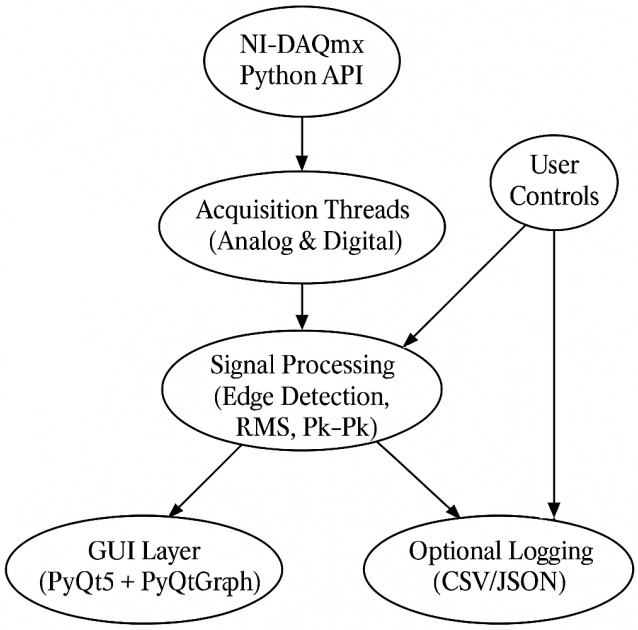


Figure 2. A software architectural block diagram

Figure3 below presents a visual representation of the sequence of actions and operations within the system, illustrating the overall program flow.

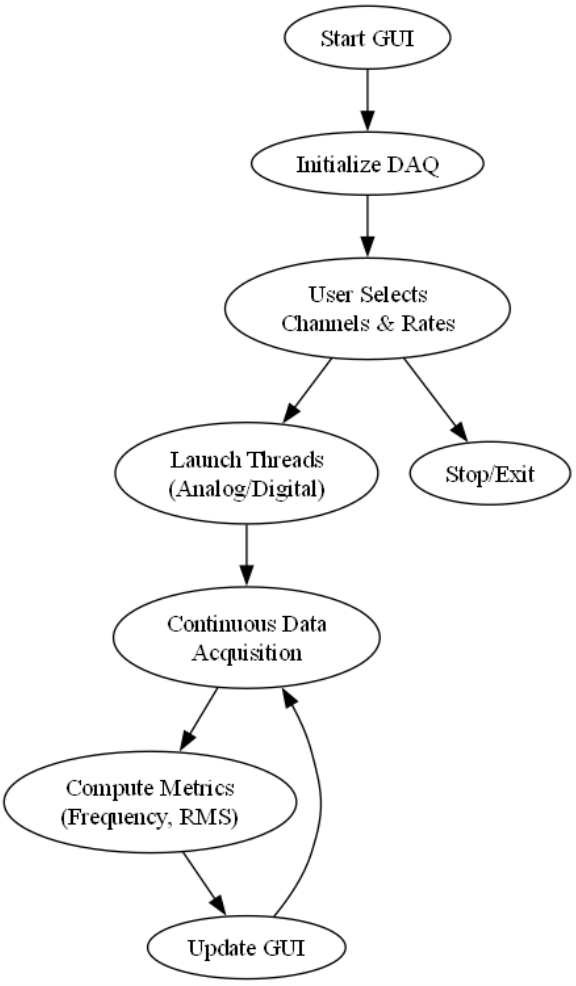


Figure 3. Program Flow Diagram Illustrating the Sequence of Operations

3.1 Hardware Layer

The data acquisition hardware used in this project is the NI USB-6363 DAQ, a research-grade device offering 32 single-ended analog input (AI) channels and 48 digital input\ output (DIO) lines, as shown in Figure4. For analog acquisition, the device supports 16-bit resolution sampling at up to 500 kS\s for multi-channel operations, while digital acquisition supports up to 4 MS\s for parallel inputs. To maintain signal visibility in the GUI's plotting pane, the system utilizes a subset of the available channels:Port0\line0:15 as shown in Figure5, designated for 16-channel digital read operations, typically connected to external signal generators or logic outputs. For analog channels, AI0 to AI3 are configured for continuous acquisition at 500 kS\s, enabling the capture of low-to-medium frequency waveforms ranging from −10 V to +10 V The hardware is connected to the PC via USB, and screw terminal connectors are used to secure input signal wires to the DAQ device.

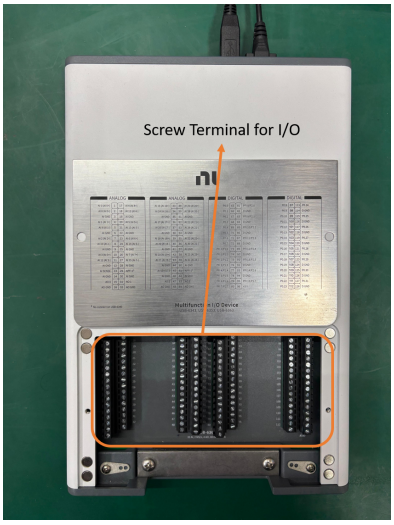


Figure 4. Image of NI USB-6363 DAQ

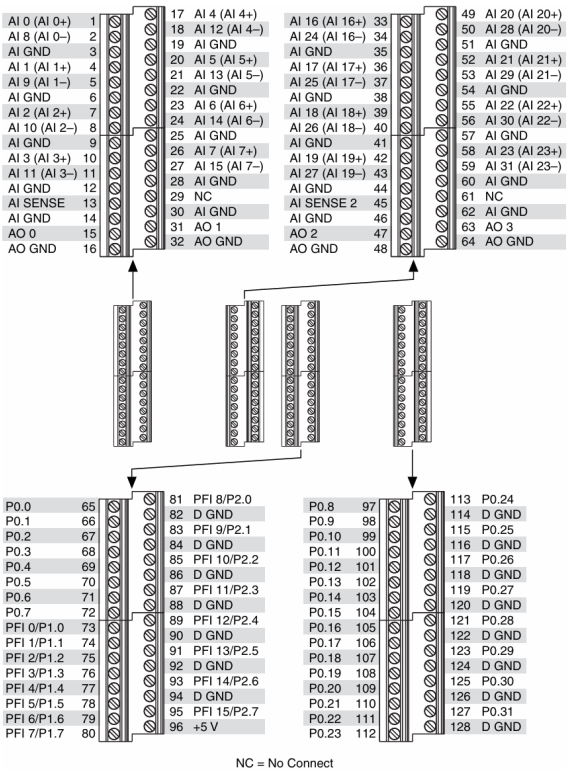
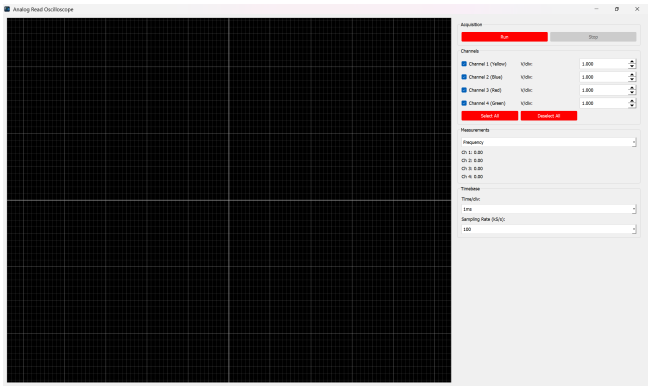


Figure 5. Figure NI USB-6363 Screw Terminal Pin Layout. (Courtesy- `https://www.ni.com/docs/en-US/bundle/pcie-pxie-usb-63xx-features/resource/370784k.pdf`)

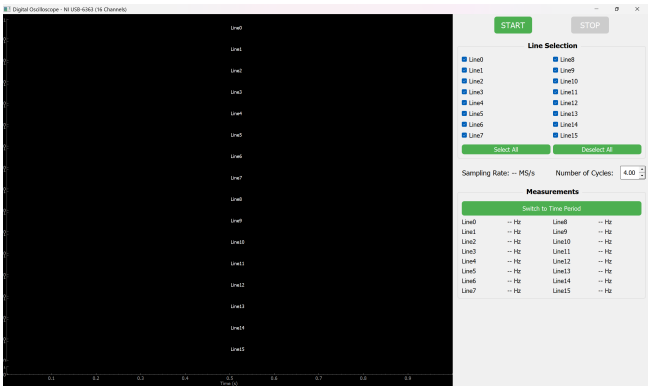
3.2 Software Layer

The software stack is implemented as a modular Python application, utilizing a combination of high-performance libraries to manage device communication, data processing, and graphical visualization. The PyQt5 library provides the graphical user interface (GUI) layer, handling window layouts, in-

teractive widgets, and event-driven control mechanisms. Users can configure channels, trigger acquisition, and visualize real-time outputs directly through the GUI. At the core of hardware communication is the NIDAQmx Python API, which interfaces directly with the NI DAQ device. This layer is responsible for creating acquisition tasks, setting up channels and sampling rates, handling timing and triggering configurations, and managing memory buffers. The PyQtGraph library is used for real-time data visualization, enabling high-speed plotting of waveforms. It supports digital signal visualization using step plots and analog signal plotting with smooth continuous curves. Real-time interaction features such as zooming, panning, and axis scaling are integrated to allow detailed signal inspection. The digital read GUI configures DAQ's digital input lines to capture logic states and compute signal frequencies through edge detection algorithms. The analog read GUI, which samples analog voltages continuously at high speed and streams the data to the GUI for plotting. Each GUI runs on its own processing thread to maintain responsiveness of the UI and handle real-time data updates efficiently. This separation of responsibilities ensures scalability, maintainability, and high reliability during continuous acquisition tasks. Figure6 shows the outlook of the analog and digital GUIs.



(a) Analog GUI



(b) Digital GUI

Figure 6. Overview of the a) Analog GUI and b) Digital GUI window showing control panels, waveform display area, and real-time measurement indicators.

3.3 Data Processing and Control Flow

The core of the system's signal processing and acquisition control resides in the threading-based architecture implemented using PyQt5's 'QThread' class. Two 'AcquisitionThread' classes are designed separately for analog and digital modes. The GUI interacts with the NI USB-6363 device through the NIDAQmx Python API.

For analog acquisition, the 'AnalogMultiChannelReader' reads data from up to four voltage channels configured via `add_ai_voltage_chan`. A continuous acquisition task with a configurable sample rate (up to 500 kS/s per channel) ensures sustained signal capture, buffered by a double-length internal buffer. Each acquired block is corrected with an offset to compensate for signal offset and analysed for frequency measurement. Additionally, metrics such as RMS voltage, peak-to-peak amplitude, min/max voltage, and period are computed per channel, forming a parameter dataset for display.

For digital acquisition, the system uses 'DigitalMultiChannelReader' to read signals from 16 parallel digital lines. Frequency estimation is based on the detection of rising edges in the sampled bit stream. To ensure robustness, the algorithm computes the median period of detected edges, filters outliers, and calculates the average frequency. This digital thread dynamically adjusts the buffer size depending on frequency to optimize acquisition performance and minimize memory overhead.

Data is pushed from the acquisition threads to the GUI layer using custom PyQt5 signals `data_ready`, ensuring thread-safe updates to real-time plots and measurement displays.

3.4 User Interface Layer

The user interface, developed using PyQt5 and PyQtGraph, provides an interactive, multi-pane oscilloscope-style environment tailored separately for analog and digital monitoring modes. In the analog GUI, the GUI window embeds:

- A PlotWidget for real-time waveform display with custom time per division settings as shown in Figure7.

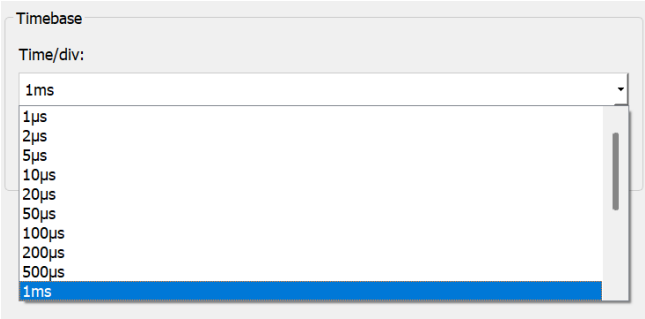


Figure 7. Drop-down menu option to change time per division setting of the graph palate in the Analog GUI

- Checkboxes and V/div controls for enabling/disabling and scaling individual channels as shown in Figure8.

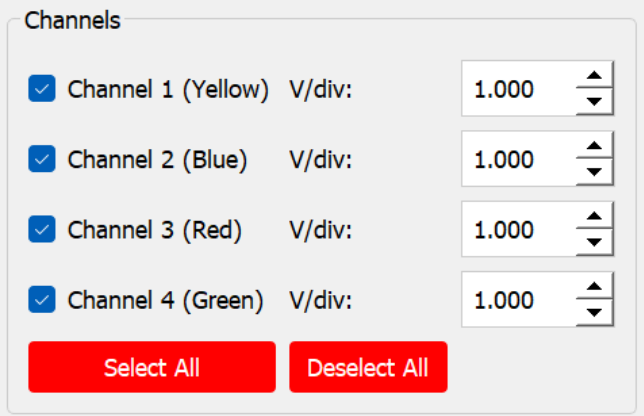


Figure 8. Checkbox option for enabling/disabling and V/div option for scaling individual channels

- A measurement panel that displays frequency, RMS, peak-to-peak, and other parameters as shown in Figure9.

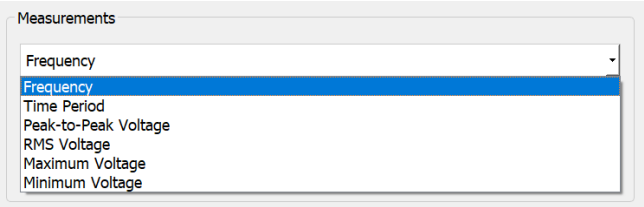


Figure 9. Selection option for parameters to be measured in the Analog GUI

- Controls of the sampling rate to dynamically change the acquisition logic, as shown in Figure10.

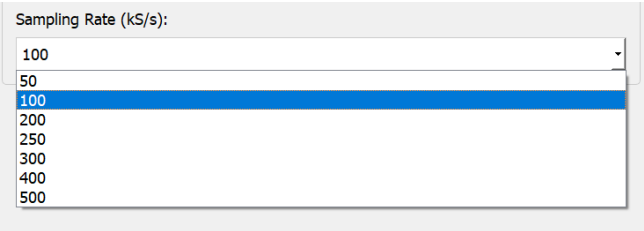


Figure 10. Selection option to change sampling rate setting of the graph palate in the Analog GUI

In the digital GUI, the GUI supports:

- 16 vertically stacked digital waveform plots with color-coded traces as shown in Figure11.
- A global time-axis plot to correlate all selected lines as shown in Figure11.

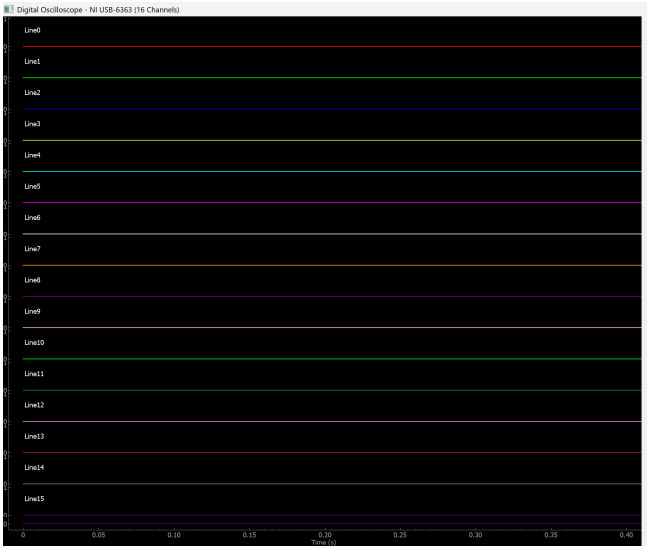
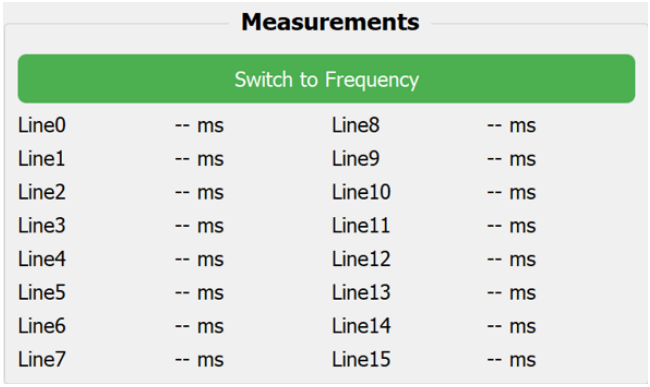
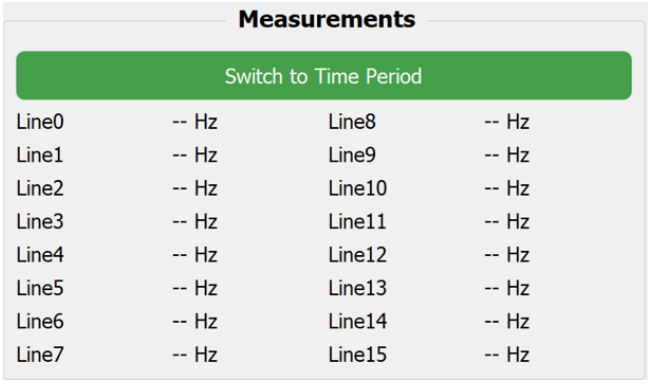


Figure 11. Digital GUI all channels colour palate

- Measurement controls including toggle buttons to switch between frequency and time period view as shown in Figure12.



(a) Switch to Time period button



(b) Switch to Frequency button

Figure 12. Toggle button to switch between a) Frequency display tab and b) Time period tab in Digital GUI.

- Real-time update of frequency or time period per channel in a formatted grid layout., as shown in Figure13.
- Detection of the sampling rate and the options to adjust the number of signal cycles displayed.

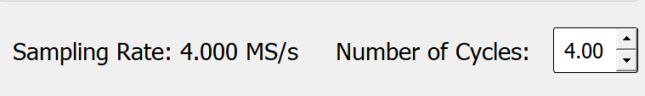


Figure 13. Spin box option to change number of signal cycles for proper visibility

- Channel visibility through checkboxes and double click deselection, as shown in Figure14.

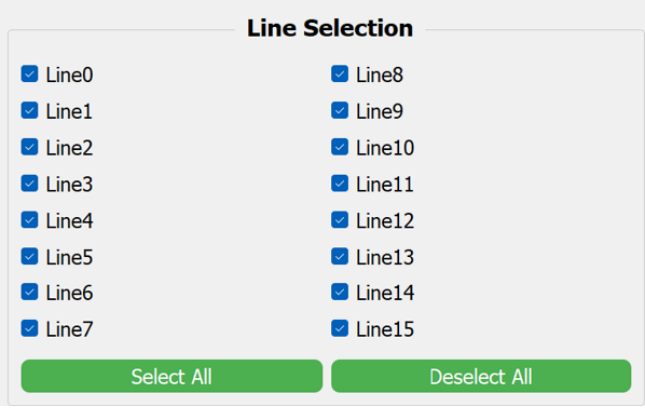


Figure 14. Checkbox option for enabling/disabling for selecting individual channels

Both GUIs include robust feedback mechanisms with user alerts for error conditions and invalid configurations. The interface ensures usability while preserving access to advanced configuration settings, ideal for laboratory research environments.

4. Implementation Methodology

The implementation of the real-time signal monitoring system using the NI USB-6363 DAQ was centered around translating the planned architectural components into a robust, responsive, and modular application. This section elaborates on the techniques used to ensure thread safety, responsiveness, and precision signal handling during real-time acquisition.

4.1 Acquisition Thread Management and Execution

The data acquisition process was implemented using PyQt5's QThread, allowing analog and digital signal acquisition to run independently of the GUI event loop. Each acquisition thread initializes and starts its corresponding DAQ task in continuous sampling mode and continuously reads data from the onboard hardware buffers using NIDAQmx stream readers. For analog input, real-time readings are streamed into NumPy arrays, and signal metrics such as RMS voltage, min/max levels, and peak-to-peak amplitudes are computed. For digital signals, rising edge detection is applied to each sampled logic trace to estimate signal frequency, using median filtering for stability. The acquisition thread passes this processed data through PyQt signals, which are consumed asynchronously by the GUI.

4.2 Real-Time Data Flow and Synchronization

To handle real-time performance, the system employs non-blocking signal-slot mechanisms. GUI updates are decoupled from acquisition threads through the use of PyQt signals. All live waveform data, along with computed metrics, is pushed from the background threads into the main GUI thread using structured packet. Update rates are tuned using QTimer events within the GUI to match the buffer size and plot frame rate, ensuring optimal refresh without overloading the hardware. This design enables simultaneous monitoring of multiple channels with consistent frame pacing, even during high sampling-rate operations.

4.3 Buffered Acquisition and Memory Optimization

Circular buffers are used internally to manage data flow from DAQ to GUI. For analog acquisition, this involves dynamically allocating and rotating NumPy arrays to preserve a window of recent signal history. In the case of digital acquisition, edge indices are extracted and used to compute frequency estimates in a sliding window fashion, reducing memory overhead while maintaining signal integrity. Dynamic buffer resizing is also incorporated for extremely low or high frequencies, adapting the number of samples read per cycle based on the detected period of the incoming signal. This adaptive design enhances the accuracy of frequency measurements without increasing latency.

4.4 User Interaction Control

The GUI allows users to start or stop acquisition, select channels, and modify acquisition parameters. Each of these interactions is routed through Qt's signal-slot mechanism, event triggering methods in the acquisition threads to safely restart or reconfigure tasks without crashing or halting the application. Care was taken to ensure that GUI actions do not interrupt ongoing buffer reads or waveform rendering. For example, updating the sample rate or number of displayed cycles queues the changes until the current acquisition window completes, ensuring atomic and glitch-free transitions.

4.5 Error Handling and Operational Stability

All acquisition threads are wrapped in robust error handling logic. Device disconnection, buffer overruns, and invalid configuration states (e.g., unsupported sample rates or missing channels) are caught and relayed to the user through GUI messages. Upon error detection, the corresponding thread is safely terminated, and its DAQ task is closed before allowing re-initialization. Visual indicators and message boxes in

the GUI guide the user toward corrective actions. A status bar tracks the connection state, task activity, and sampling rate, providing ongoing feedback during operation.

5. Performance Evaluation

The system's validation involved the use of a commercial function generator. For digital readings, it successfully measured frequencies up to 2 MHz across 16 channels, accurately capturing pulse trains, such as a 2 MHz signal being displayed as "2.000 MHz." Median filtering was applied to improve noise resistance. Table1 presents the tested specifications of the system for digital signal input.

Table 1. Tested specifications of the system for digital signal input

Parameters	Specifications
Threshold Voltage for Discretizations	1.44 V
Frequency	Up to 2 MHz
Number of Channels	16
Sampling Rate	4 M Samples/s

In terms of analog readings, the system handled a 30 kHz sine wave, a 1.8 kHz pulse with a 1% duty cycle, a 50 kHz square wave with a rise time of 2.5 µs, and ramp signals at 100 kHz ranging from −10 V to +10 V. The GUI plots were consistent with reference measurements, maintaining a resolution of less than 2 mV, a DC offset of less than 1 mV, and an error margin of ±0.5 %Vin ± 3 mV. Table2 shows the detailed tested specification of the system for analog signal input.

Table 2. Tested specifications of the system for analog signal input

Parameters	Specification
Minimum Voltage	−10 V
Maximum Voltage	+10 V
Minimum Frequency (Sine Wave)	1 Hz
Maximum Frequency (Sine Wave)	30 kHz
Minimum Frequency (Pulse, 1% Duty Cycle)	1 Hz
Maximum Frequency (Pulse, 1% Duty Cycle)	1.8 kHz
Minimum Frequency (Square Wave)	5 Hz
Maximum Frequency (Square Wave)	50 kHz
Rise Time (Square Wave)	2.5 µs
Voltage Resolution	<2 mV
DC Offset	<1 mV
Tested Waveform Types	Sine, Ramp, Pulse, Square

Figure15 shows the digital GUI interface with multiple-channel digital input, demonstrating a 2 MHz square wave. Figure16 displays the analog GUI interface with multi-channel input of a 30 kHz sine wave at an amplitude of ±10 mV. Table3 shows the performance metrics of the system for both the digital and analog acquisition modes.

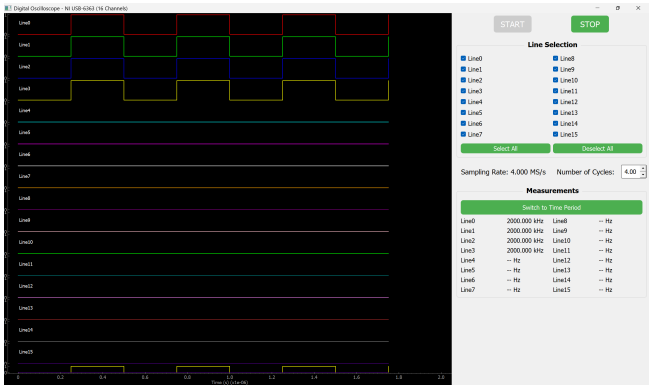


Figure 15. Digital read GUI showing 16-channel frequency measurement at 2MHz

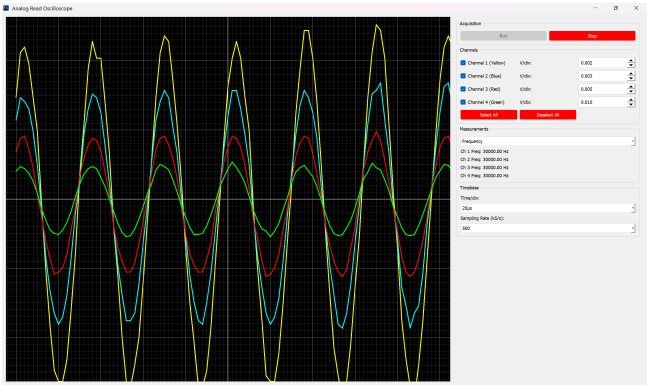


Figure 16. Analog read GUI displaying a 30 kHz sine wave within-10 V

Table 3. Performance metrics for both the digital and analog acquisition modes

Mode	Achieved Frequency	Use Case
Digital Read	2 MHz	Digital bus, High-speed signals
Analog Read	30 kHz	Sensor data, waveform analysis

Table4 below provides a consolidated overview of the software functionalities and measurable signal parameters, including the applicable signal types and the tested measurement ranges achieved by the system.

Table 4. Overview of Software Functionalities and Measurable Signal Parameters

Category	Feature / Parameter	Signal Type	Details	Measurement Range / Resolution
Signal Acquisition	Channel Selection	Analog & Digital	Selectable from GUI; 4 analog and 16 digital channels	Analog: ai0–ai3, Digital: port0/line0–15
	Sampling Rate Configuration	Analog & Digital	Set via GUI and validated in software	Analog: up to 500 kS/s, Digital: up to 4 MS/s
	Buffer Size Control	Analog & Digital	Dynamically adjusted based on frequency and number of cycles	10,000 to 1,000,000 samples
Signal Processing	Frequency Measurement	Digital	Edge-detection method per channel	Up to 2 MHz, ±2–3% error above 2 MHz
	Frequency Measurement (FFT)	Analog	FFT-based estimation per acquisition buffer	1 Hz to 30 kHz
	Time Period Calculation	Analog & Digital	Inverse of frequency	ms (analog) or µs (digital) resolution
	Peak to Peak Voltage	Analog	Max-Min per channel	Within ±10 V, Resolution: <2 mV
	RMS Voltage	Analog	RMS over each buffer window	Resolution: <2 mV, Accuracy: ±0.5% of Vin ± 3 mV
	Max/Min Voltage	Analog	Per window max and min detection	Full range: -10 V to +10 V
	DC Offset Correction	Analog	Software bias adjustment (+8.5 mV applied)	Final DC error <1 mV
Visualization (GUI)	Real-Time Waveform Display	Analog & Digital	PyQtGraph-based multi-channel plots	Time base adjustable from ms to seconds
	Grid Overlay & Time Base	Analog & Digital	Centered oscilloscope-style view with time zoom	Adjustable: 1 ms/div to 1 s/div (GUI setting)
	Channel Toggle and Scaling	Analog & Digital	Enable/disable display; adjust volts/div per channel	GUI-controlled
	Measurement Display Table	Analog & Digital	Frequency, RMS, Pk-Pk, min/max, time period	Updated in real-time, channel-wise
User Control	Start / Stop Acquisition	Analog & Digital	GUI button-triggered	Immediate effect, via PyQt signal-slot
	Parameter Configuration	Analog & Digital	Channels, sample rate, cycles, V/div, etc. via GUI widgets	Dynamically applied
	Signal Type Mode	Analog or Digital	User switches between analog and digital acquisition mode	Separate scripts / GUI windows
Data Logging	Optional CSV/JSON Logging	Analog & Digital	On-demand, user-triggered logging	Raw samples or processed values
	Configurable Logging Parameters	Analog & Digital	Selectable: full waveform, computed stats, timestamp	File formats: .csv, .json

6. Discussion

6.1 Comparative Evaluation with Existing Tools

Our Python-based GUI solution for real-time signal monitoring fills a unique gap between high-end professional tools and low-cost educational platforms. It delivers robust performance with open-source flexibility, a clean GUI, and seamless integration with research-grade DAQ hardware like the NI USB-6363. This makes it especially suitable for academic institutions, research labs, and prototyping environments looking for cost-effective and customizable solutions without compromising core functionalities. These findings we offer here also supported by comparative studies, which highlight that Python-based DAQ frameworks can offer real-time responsiveness and configurability on par with commercial software when designed around specific use cases (Vipond et al., 2023; Saha et al., 2022; Martins, 2023).

a) Comparison Between Our Tool & Commercial Oscilloscopes (e.g., Tektronix, Keysight, R&S): While commercial scopes outperform in speed and features, our tool matches essential functionality for educational and research tasks at a fraction of the cost. Below Table5 shows the comparison of our tool with the commercial oscilloscopes.

Table 5. Comparison with Commercial Oscilloscopes

Feature	Commercial Oscilloscopes	Our Tool
Performance	High bandwidth (MHz to GHz), deep memory, advanced triggering	Mid-range performance with NI USB-6363 (500 kS/s analog, 4 MS/s digital)
Cost	Very expensive (Rs. 2–20 Lakhs+)	Low-cost (uses existing PC + DAQ)
Customizability	None (proprietary firmware/UI)	Fully customized (open-source Python code)
Data Access	Limited long-term storage, vendor-specific formats	Full access to raw data for logging/ export
Target Users	Professional labs, industries	Students, educators, research labs

b) Comparison Between Our Python-Based GUI & NI LabVIEW: Our solution offers similar device control and performance but with transparency, flexibility, and zero software licensing cost, making it suitable for institutions with budget limitations. Below Table6 shows the comparison of our python-based GUI with NI LabVIEW.

Table 6. Comparison with NI LabVIEW

Feature	LabVIEW	Our Python-Based GUI
Software Cost	High licensing fees	Free (open-source stack)
Learning Curve	Steep (graphical programming model)	Easier for Python users
Custom Workflow	Possible but complex	Easily extensible via Python
Multithreading & Buffer Control	Available but hidden under GUI layers	Explicitly implemented using QThread and custom buffers

c) Comparison Between Our Tool & Open-Source Alternatives (e.g., Sigrok, OpenHantek): Most open-source tools suffer from poor GUI or limited DAQ support. Our tool bridges this gap by providing a refined GUI and industrial DAQ performance. Below Table7 shows the comparison of our tool with open-source tools.

Table 7. Comparison with Open-Source Tools

Feature	Sigrok / OpenHantek	Our Tool
Device Compatibility	Limited to supported hardware	Specifically optimized for NI USB-6363
GUI Usability	Basic, often non-intuitive	Professional-grade GUI using PyQt5 + pyqtgraph
Real-Time Performance	Inconsistent at high data rates	Stable at 500 kS/s analog, 4 MS/s digital via optimized multithreaded handling
Feature Set	Basic triggering, limited plotting	Frequency, RMS, time period, Vpp measurements, toggles, multi-channel scaling, and buffer tuning

d) Comparison Between Our Tool & Arduino-Based Signal Monitoring Systems: Arduino is useful for educational prototypes but lacks the precision, throughput, and GUI maturity of our NI-based solution. Below Table8 shows the comparison of our tool with Arduino-based systems.

Table 8. Comparison with Arduino Systems

Feature	Arduino Tools (Serial Plotters, IDE-based GUIs)	Our Tool
Sampling Rate	<10 kS/s (limited by USB Serial and 10-bit ADC)	Up to 500 kS/s (analog), 4 MS/s (digital)
Channels	Typically 6 analog, 14 digital (non-simultaneous)	4 analog (16-bit), 16 digital (parallel) simultaneous channels
Accuracy	10-bit ADCs, basic I/O	16-bit ADC, high signal fidelity
GUI Interface	Minimal, often CLI or Serial Monitor	Full-featured oscilloscope-like GUI

e) Summary Comparison Table: Table9 below shows the overall comparison of our tool with all previously discussed systems.

Table 9. Overall Comparison Summary

Tool	Sampling Speed	GUI Quality	Multi-channel Support	Open-Source	Cost	Customizability
Our Python Tool	Analog: 500 kS/s, Digital: 4 MS/s	4 out of 5	4 analog, 16 digital	Yes	Low	Very High
LabVIEW + NI DAQ	Same as above	3 out of 5	Same as above	No	High	Moderate
Commercial Oscilloscope	MHz to GHz	5 out of 5	Varies	No	Very High	No
Sigrok/OpenHantek	Low to mid	2 out of 5	Limited	Yes	Low	Moderate
Arduino Systems	<10 kS/s	1 out of 5	Few channels	Yes	Very Low	Moderate

6.2 Scientific Context and Broader Impact

The developed Python-based GUI for real-time signal monitoring has significant implications for both immediate and extended scientific applications. One of the most impactful contexts is its direct applicability in the development of Checkout Systems for scientific payloads at PRL.

Checkout systems serve as ground-based satellite simulators that are essential for the verification, test, and calibration of scientific payloads before launch. The presented GUI tool, with its capabilities for multi-channel analog and digital acquisition, real-time waveform analysis, frequency and voltage metrics, and custom signal visualization, forms the core data interface required during such simulations.

Key Applications in Payload Checkout Systems:

- Health Monitoring: Enables continuous observation of voltage levels, payload temperature, etc. during testing.
- Calibration Logging: Captures and archives scientific data during thermal-vacuum, vibration, and EMI/EMC testing phases, essential for verifying payload performance under environmental extremes.
- Post-Launch Validation: Provides reference datasets of payload behavior under controlled lab conditions, which can be compared with in-orbit telemetry to assess any performance degradation, drift, or anomaly.
- Interface Testing: Simulates on-board satellite systems for electrical interface validation of the payload, helping detect I/O mismatches or signal anomalies early in development.

Beyond the immediate scope of scientific payload testing, the developed tool presents a broader impact across scientific and educational domains:

- Open-Source Accessibility: Unlike commercial tools, this system promotes low-cost deployment, allowing widespread adoption across academic institutes, student satellite programs, and resource-limited research labs.
- Scalability: The modular, multithreaded architecture makes the tool adaptable for future use cases including multi-sensor systems, laboratory automation, and embedded system development.
- Cross-Platform Research Integration: It can interface with Python-based data analysis pipelines or machine learning models for automated anomaly detection, long-duration tests, and parameter optimization, useful in various disciplines.

6.3 Error Margins and Limitations

To evaluate the reliability of the system, we conducted extensive performance tests under diverse signal conditions. These tests helped establish the practical operating limits of the developed Python-based DAQ system, primarily due to the hardware constraints of the NI USB-6363 device such as ADC sampling rate, FIFO buffer size, etc. While software-level design choices—such as multithreaded streaming and buffer management—help optimize performance, the fundamental limitations are hardware-defined. The observed error margins and functional constraints are summarized in Table 10.

Table 10. Observed Limitations and Error Margins

Mode	Parameter	Observed Limitation/ Error	Remarks/ Root Cause
Digital Read	Max reliable frequency	2 MHz	Beyond this, rising edge detection becomes unreliable due to buffer overrun and violation of Nyquist criteria caused by onboard DAQ sampling clock limitations.
	Frequency accuracy above 2 MHz	±2–3% error	Missed edges due to limited DAQ USB throughput and system buffer saturation.
	Channel count	16 channels (parallel)	Stable at 4 MS/s aggregate, but performance degrades if CPU resources are constrained.
Analog Read	Max tested sine wave frequency	30 kHz	Clean sine reproduction up to 30 kHz; higher frequencies show slight distortion.
	Max usable square wave frequency	50 kHz	Above this, edge sharpness drops due to limited ADC sampling resolution at higher frequencies.
	Voltage accuracy	±0.5% of Vin ± 3 mV	Minor noise and quantization error; improved via signal averaging.
	Voltage resolution	< 2 mV	Limited by 16-bit DAQ resolution at ±10 V range combined with LSB errors.
	DC offset	< 1 mV	Compensated internally by applying +8.5 mV bias in software.
	Sampling rate cap	500 kS/s per channel (aggregate 2 MS/s)	Achieved using buffer-based streaming and dynamic rate control in multithreaded architecture.

6.4 Platform Compatibility and System Configuration

Table 11 summarizes the minimum and recommended system specifications necessary to achieve optimal performance and avoid data loss, or GUI lag during operation.

Table 11. Platform Compatibility and System Configuration

Component	Minimum Specifications	Recommended Specifications
CPU	Intel i5 (8^{th} Gen) or AMD Ryzen 5	Intel i7 / Ryzen 7 or higher
RAM	8 GB	16 GB or higher
Storage	256 GB HDD/SSD	512 GB HDD/SSD
OS	Windows 10 / 11	Windows 10 / 11
USB	USB 2.0/3.0 port	USB 3.0
Python	Version ≥ 3.7	Python 3.10+
Libraries	nidaqmx, PyQt5, pyqtgraph, numpy, scipy	nidaqmx, PyQt5, pyqtgraph, numpy, scipy
Display	1366×768 min	Full HD for waveform clarity

7. Conclusion

This study presents a Python-based graphical user interface designed for real-time monitoring of digital and analog signals using the NI USB-6363 data acquisition device. The system enables students, researchers, and engineers to conduct high frequency digital and precise analog waveform measurements. Its open-source software and modular architecture offer a solution to the high costs and complexity associated with proprietary tools. The current implementation, therefore, not only meets the immediate needs of signal monitoring and testing but also aligns with ongoing research trends favoring open, reproducible, and extensible Python-based instrumentation platforms (Hughes, 2010; Koerner et al., 2019; Martins, 2023; Akam and Walton, 2019; Jäger and Gümmer, 2023; Vipond et al., 2023; Saha et al., 2022). Future improvements could broaden its range of applications.

8. Future Scope

The current implementation establishes a solid foundation for real-time signal acquisition and monitoring using a Python-based GUI with NI USB-6363 DAQ hardware. However, there are several avenues for extending and improving the system both functionally and in terms of application:

a) Hardware Abstraction for Cross-DAQ Compatibility

- The present system is tightly integrated with the NI-DAQmx API. Future work could focus on abstracting the data acquisition layer to support additional hardware backends (e.g., from open-source USB ADCs).
- This would require defining a standard interface layer and device drivers adaptable to non-NI DAQs.

b) Integration with Checkout Systems for Automated Payload Testing

- The system can be extended to become an integral part of satellite payload checkout systems by interfacing with command-response simulators, telemetry channels, and environmental test chambers (thermal, vacuum, EMI/EMC).
- This includes capturing and storing calibration logs across multiple test phases.

9. Acknowledgment

The authors express their sincere gratitude to Prof. Kashyap Patel of the Devang Patel Institute of Advanced Technology and Research, CHARUSAT, for his invaluable guidance and mentorship throughout the project. We also extend sincere gratitude to the Physical Research Laboratory, Ahmedabad, for providing the resources and support necessary during the internship. This work is supported by Department of Space, Govt. of India.

The Editor, D. Pallamraju acknowledges the reviewers Pranav Adhyaru, Pinky Brahmbhatt, Himanshu Mazumdar and an anonymous reviewer for their help in evaluating this article.

References

https://www.riverbankcomputing.com/software/pyqt/intro.

http://www.pyqtgraph.org.

https://nidaqmx-python.readthedocs.io/.

https://www.ni.com/docs/en-US/bundle/usb-6363-specs/page/specs.html.

https://doc.qt.io/qt-6/qthread.html.

T. Akam and M. E. Walton. pyphotometry: Open source python based hardware and software for fiber photometry data acquisition. *Scientific reports*, 9(1):3521, 2019.

J. M. Hughes. *Real World Instrumentation with Python: Automated Data Acquisition and Control Systems*. " O'Reilly Media, Inc.", 2010.

D. Jäger and V. Gümmer. Pythondaq–a python based measurement data acquisition and processing software. In *Journal of Physics: Conference Series*, volume 2511, page 012016. IOP Publishing, 2023.

L. J. Koerner, T. A. Caswell, D. B. Allan, and S. I. Campbell. A python instrument control and data acquisition suite for reproducible research. *IEEE Transactions on Instrumentation and Measurement*, 69(4):1698–1707, 2019.

S. A. M. Martins. Pydaq: Data acquisition and experimental analysis with python. *Journal of Open Source Software*, 8 (92):5662, 2023.

S. Saha, P. K. Mallisetty, S. Sen, and S. Giri. Design and development of a high-speed data acquisition system for acquiring acoustic emission signals. *Materials Today: Proceedings*, 66:3830–3837, 2022.

N. Vipond, A. Kumar, J. James, F. Paige, R. Sarlo, and Z. Xie. Real-time processing and visualization for smart infrastructure data. *Automation in Construction*, 154:104998, 2023.

Appendix A

GitHub Link: https://github.com/kush-nirav-shah/ni-usb-daq-signal-processing

PRL research encompasses the earth the sun immersed in the fields and radiations reaching from and to infinity, all that man's curiosity and intellect can reveal

पीआरएल के अनुसंधान क्षेत्र में समविष्ट हैं पृथ्वी एवं सूर्य जो निमीलित हैं चुंबकीय क्षेत्र एवं विकिरण में अनंत से अनंत तक जिन्हे प्रकट कर सकती है मानव की जिज्ञासा एवं विचारशक्ति