International Journal of Modern Engineering and Management | IJMEM
ISSN No: 3048-8230
Volume 2 Issue 5 May - 2025

Software Defined Oscilloscope

Svansh[1], Anant Raj[2] ,Dr. Aditya Agarwal[3]
[1,2,3]Department of Electronics and Communication Engineering
[1,2,3]SRM Institute of Science and Technology NCR Campus, Ghaziabad, India

Abstract: **The Software-Defined Oscilloscope (SDO) is revolutionizing waveform analysis by introducing a level of flexibility and efficiency that traditional oscilloscopes cannot match. Unlike conventional models, SDOs rely heavily on software for signal acquisition, analysis, and display, making them highly customizable and adaptable to a wide range of applications. Software-Defined Oscilloscopes (SDOs) leverage digital signal processing (DSP) techniques to achieve flexibility and software-based waveform analysis, allowing integration of features such as FFT and spectrum analysis [1], [2]. This reliance on software allows for frequent updates, enabling the addition of new features and functionalities without requiring hardware upgrades. Traditional oscilloscopes are limited by hardware constraints, while SDOs are continuously upgradable through software patches and new algorithms [3], [13]. As a result, SDOs offer significant advantages, including high-speed sampling rates, wide bandwidth support, and advanced analytical tools like FFT and spectrum analysis. These capabilities allow users to capture and study complex signals and transient phenomena with exceptional precision. Furthermore, these devices are often more cost-effective than traditional high-end oscilloscopes, offering a compelling alternative for industries seeking efficient and scalable solutions. The applications of SDOs span multiple sectors, including electronics design, telecommunications, automotive, and aerospace. Technological advancements are propelling the capabilities of SDOs even further. Improvements in FPGA technology allow for faster real-time processing, while cloud computing facilitates seamless remote operations and data management. The integration of artificial intelligence automates complex analyses and improves the identification of anomalies, enhancing operational efficiency**

Keywords- *Software-Defined Oscilloscope, signal acquisition, spectrum analysis, high-speed sampling rates, bandwidth support, identification of anomalies and enhancing operational efficiency*.

1. INTRODUCTION

A software-defined oscilloscope (SDO) is an advanced measurement tool that utilizes software and digital signal processing (DSP) to analyze and visualize electrical waveforms. Unlike traditional oscilloscopes, which rely heavily on dedicated hardware components for signal acquisition and processing, SDOs use a software-driven approach, where the core functionalities are implemented through algorithms running on a computer, tablet, or embedded system.This transition from hardware-centric to software-centric design provides greater flexibility, cost-effectiveness, and scalability, making SDOs an attractive alternative to conventional oscilloscopes.The key advantage of software-defined oscilloscopes lies in their flexibility. Traditional oscilloscopes come with predefined hardware features that limit their adaptability to new measurement needs. In embedded system development, SDOs are essential for debugging microcontrollers, FPGA-based circuits, and digital interfaces such as SPI, I2C, and UART[1]. The automotive and IoT sectors also benefit from SDOs, as they are used to analyze sensor signals, CAN bus data, and power electronics, ensuring the reliability of modern automotive and smart devices. Furthermore, in education and research, SDOs provide a cost-effective and flexible solution for academic institutions, allowing students and researchers to experiment with waveform analysis without the need for expensive hardware[2].Their ability to provide flexible, high-performance, and cost-effective signal analysis makes them an indispensable tool for engineers, researchers, and educators. By redefining how waveforms are acquired and analyzed, SDOs represent the future of oscilloscope technology, bridging the gap between traditional hardware-based instruments and modern, software-driven solutions.

2. SYSTEM DESCRIPTION

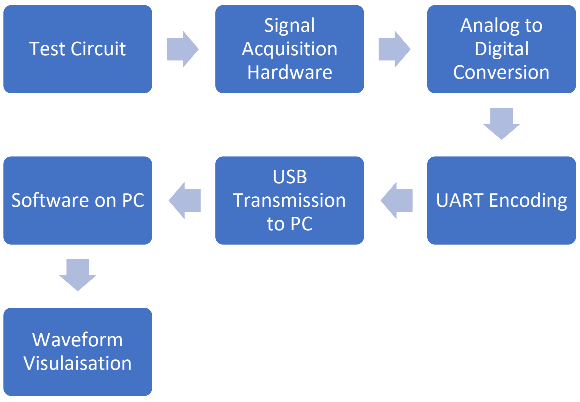
In this section, the theory of the system and implementation of each of the oscilloscope's sub-systems is discussed, as well as the principles of operation and implementation of all. Each of the following subsections details the function and operation of a subsystem of the oscilloscope, beginning with the input stage ending to the final output stage. The overall system is divided into 2 major parts –

1. **Hardware Interface**

    a. Triggering circuit
    b. Level Shifter and Scaler
    c. Low Pass Filter Circuit
    d. Firmware

    The triggering circuit employs a Schmitt trigger to avoid false triggering due to signal noise, stabilizing waveform capture [4], [18].Level shifting is necessary to ensure safe ADC sampling, especially when input signals include negative voltages, as discussed in [10], [16]. Proper design of the low-pass filter stage helps in removing high-frequency noise components before digitization [6], [20].

2. **Software Interface**
    For software we are using python to generate the grid and the waveform using in built modules like - matplotlib, pyqt, numpy.



A. Triggering Circuit

A **triggering circuit** in a **software-defined oscilloscope (SDO)** is essential for capturing signals at precise moments, ensuring accurate waveform analysis using **Schmitt Trigger circuit**, a type of comparator with hysteresis that provides stable digital output by eliminating noise from an analog input signal. The circuit operates by first comparing the input signal with an **upper threshold voltage**; if the signal exceeds this threshold, the output is set **HIGH**. However, if the input remains below the upper threshold, it is further compared to a **lower threshold voltage**. When the input falls below this lower threshold, the output switches to **LOW[4]**.

If the signal remains between the upper and lower thresholds, the circuit maintains its **previous state**, preventing rapid oscillations due to minor fluctuations or noise[3]. This hysteresis effect is reinforced by a **feedback mechanism**, which ensures that once the output state changes, it remains stable until the input crosses the opposite threshold. This design makes Schmitt Trigger circuits highly effective in **noise immunity**, preventing unintended switching caused by small variations in the input signal. The circuit is widely used in **switch debouncing, waveform shaping**, and **level detection** applications in digital electronics, oscilloscopes, and communication systems, where stable and predictable signal processing is essential.

Hysterisis Thresholds

Upper threshold voltage (V_{UT}) and lower threshold voltage (V_{LT}) are determined by:

$$V_{UT} = \frac{V_{REF}R_1}{R_1 + R_2} + \frac{V_{Sat}R_2}{R_1 + R_2}$$

$$V_{LT} = \frac{V_{REF}R_1}{R_1 + R_2} - \frac{V_{Sat}R_2}{R_1 + R_2}$$

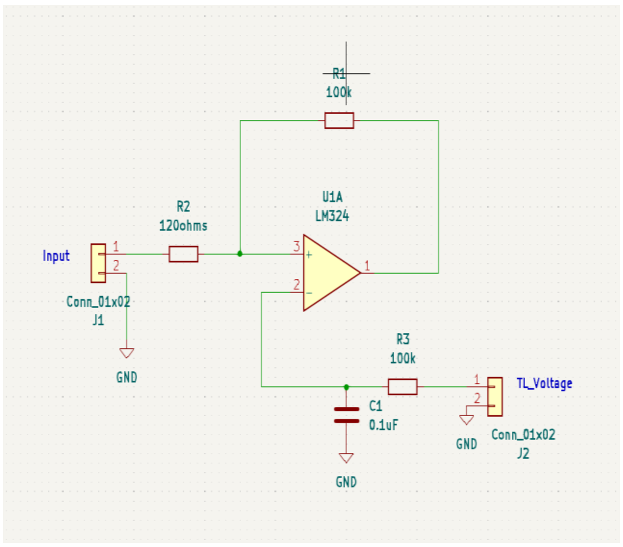


Fig. 1 Triggering Circuit

B. Level Shifter and Scaler Circuit

The **Level Shifter and Scaler Circuit** in the Software-Defined Oscilloscope (SDO) is crucial for adapting incoming waveforms to levels compatible with the ADC input of the microcontroller. Typical signals coming from a circuit under test may span both **positive and negative voltages** (e.g., ±3.3V sine waves), which are **unsafe for direct ADC sampling**, as most ADCs operate only within a **0V to 3.3V** or **0V to 5V** range.

To address this, the Level Shifter and Scaler Circuit performs two key functions:

- **Voltage Scaling:**
  It attenuates the amplitude of the input signal using a resistive voltage divider. This ensures that even the highest peaks of the signal remain within the acceptable voltage range of the ADC, preventing damage and clipping.
- **Level Shifting (DC Biasing):**
  After scaling, the signal is shifted upwards by adding a **positive DC offset**. This is achieved using biasing resistors or op-amp-based circuits.
  This shift ensures that originally negative parts of the waveform (such as a -3.3V trough) are moved into the **positive domain** (near 0V), making them safe for ADC sampling.

Example:

- A sine wave ranging from -3.3V to +3.3V is first attenuated to a smaller swing (e.g., -1.65V to +1.65V).
- Then, a +1.65V DC offset is added, resulting in a final signal swinging between **0V and 3.3V**, perfectly suited for ADC input.

The circuit ensures **linear preservation** of the input signal's shape while staying within safe ADC input levels. It is essential for the accurate and safe acquisition of a wide variety of analog signals without introducing distortion or clipping.

This stage ensures that signals of different amplitudes and polarities can be **correctly sampled** and **analyzed** in the software without hardware damage or loss of fidelity. Techniques for safe signal level translation are detailed in [10], [16]

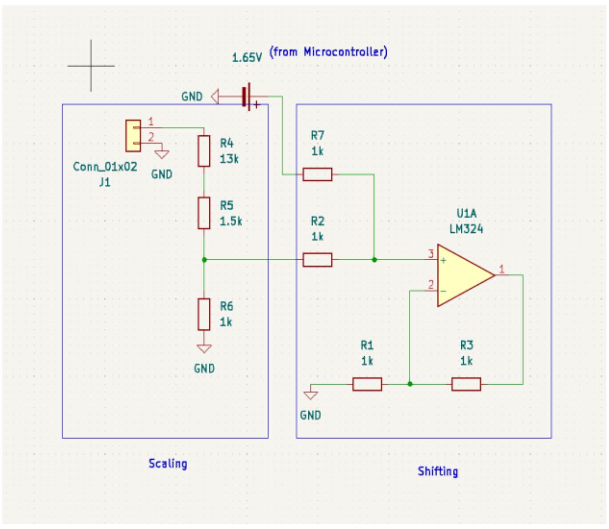


Fig. 2 Level Shifter and Scaler Circuit

C. Low Pass Filter Circuit

The **low-pass filter (LPF) circuit is used as a filter circuit**, which allows low-frequency signals to pass while attenuating high-frequency components. The process begins when the **input signal enters** the circuit and first **passes through a resistor**. This resistor plays a crucial role in setting the circuit's cutoff frequency by working in conjunction with the capacitor[6]. After the resistor, the signal reaches a **capacitor**, which acts as a frequency-dependent component. The capacitor **filters out high-frequency components** by providing a low impedance path for them to ground, thereby attenuating unwanted noise or interference. The remaining signal, which consists mostly of **low-frequency components**, is then passed as the **filtered output signal**. Additionally, an **optional feedback loop** can be included to fine-tune the filtering characteristics, further enhancing signal stability and performance. Low-pass filters are widely used in **audio processing, signal conditioning, and communication systems**, where maintaining low-frequency integrity while removing high-frequency noise is essential for optimal performance as shown in figure 3

Cutoff Frequency Equation

Defined by values of Resistor (R1) and (R2) and Capacitor (C1) and (C2):

$$f_c = \frac{1}{2\Pi\sqrt{R_1 R_2 C_1 C_2}}$$

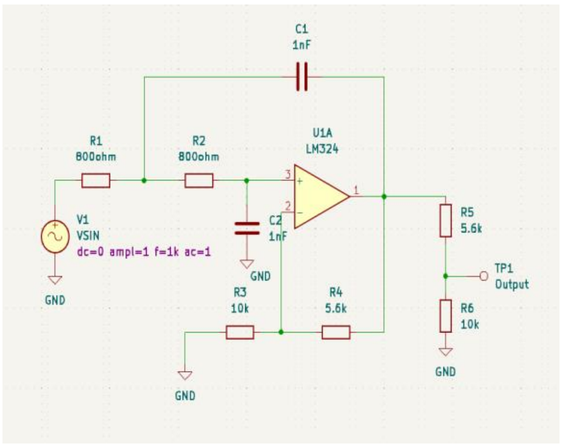


Fig. 3 Sallen-Key 2nd Order LPF Circuit

Firmware

The firmware developed for the Software-Defined Oscilloscope (SDO) plays a critical role in signal acquisition, digitization, and communication between the hardware front-end and the software interface running on a computer. The firmware ensures that the data capture process is reliable, synchronized, and efficient, enabling accurate signal reconstruction at the software side. ADCs are configured to sample at rates satisfying the Nyquist criterion, ensuring that sampled data can reconstruct the input signal without aliasing [5], [7]. The firmware detects trigger events based on user-defined conditions such as threshold voltage and edge polarity [8], [18].

1. Signal Acquisition and Sampling

The firmware initializes the microcontroller's Analog-to-Digital Converter (ADC) to continuously sample the incoming analog signal from the output of the level shifter and low-pass filter circuit. The ADC is configured with an appropriate sampling rate based on the expected input frequency range, ensuring that the Nyquist criterion is satisfied to avoid aliasing. Typically, the ADC resolution is set to 10 or 12 bits to balance between precision and transmission speed. A timer peripheral is employed to trigger ADC conversions at fixed intervals, providing consistent timebase sampling. The sampled data is temporarily stored in a buffer within the microcontroller's memory. This buffered approach allows for a controlled and organized transfer of data to the communication module once triggering conditions are met.

2. Trigger Detection Mechanism

One of the core functionalities implemented in firmware is **trigger detection**. To avoid random or unstable waveforms, the microcontroller continuously monitors incoming ADC samples and evaluates them against user-defined trigger conditions:

- **Trigger Level:** A voltage threshold set by the user.
- **Trigger Slope:** Rising edge (low-to-high crossing) or falling edge (high-to-low crossing).

The firmware compares successive ADC values to detect if the signal crosses the trigger level according to the selected slope. When operating in **Normal** or **Single** trigger mode, the firmware remains in a "waiting" state until the trigger event occurs. Once the trigger is detected, the firmware flags the event and captures a defined number of pre-trigger and post-trigger samples, ensuring a complete snapshot of the waveform is available for display. This behavior replicates the working of real-time oscilloscopes, as documented in [1][2], ensuring a steady and analyzable waveform presentation.

3. Data Formatting and UART Transmission

After the triggering event and sample acquisition, the firmware formats the ADC data for transmission. The data is packaged into a simple UART data stream, often consisting of raw voltage values or encoded packets with headers to signify transmission beginning and end.
Key aspects of this step include:

- **Baud Rate Optimization:** To balance real-time performance and data integrity, a high baud rate (e.g., 115200 or higher) is configured.
- **Packetization:** Each packet may include sample number, channel information (if multi-channel is implemented), and the ADC result.
- **Synchronization:** Special start markers (such as 0xAA, 0x55) are used to ensure the receiving Python application correctly identifies packet boundaries.

The use of UART is ideal for simple, low-latency communication, as explored in embedded system protocols [14].
A timer-based sampling strategy ensures that ADC conversions occur at fixed intervals, critical for consistent timebase creation [12].

4. Trigger Mode Management

The firmware supports three major trigger modes:

- **Auto Mode:** Samples and sends data continuously, regardless of trigger conditions.
- **Normal Mode:** Samples only after trigger condition is satisfied; otherwise waits.

- **Single Mode:** Captures a single triggered waveform and halts until manually rearmed.

The management of these modes is implemented through state machines, ensuring organized transitions between "waiting", "triggered", and "idle" states. This behaviour is essential for proper oscilloscope operation, mirroring standard practices [2][4].

After trigger detection, captured waveform data is transmitted via UART using efficient packetization methods to avoid loss of information [14].

5. System Reliability and Error Handling

The firmware incorporates basic error detection mechanisms:

- **Timeout counters**: To prevent indefinite waiting if the trigger condition is rare.
- **Buffer overrun detection**: Ensures that sampling does not overwrite critical data before transmission.
- **UART transmission error detection**: Includes parity or checksum validation if needed.

Such robustness improves the reliability of the Software-Defined Oscilloscope, especially during prolonged operation.

6. Power Management

Since the firmware operates in a real-time embedded environment, it also ensures efficient power management by using interrupt-driven sampling instead of continuous polling. This helps minimize CPU usage, allowing low-power operation when idle between triggers.

7. Future Expandability

The current firmware design allows future extensions, including:

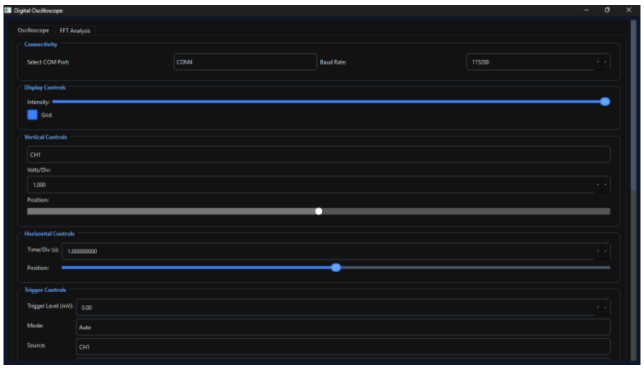
- Multi-channel signal acquisition.
- Adjustable sampling rates through software commands.
- Basic waveform analysis (peak detection, frequency measurement) onboard.
- Wireless transmission using modules like Bluetooth or Wi-Fi.

By separating signal acquisition, trigger evaluation, and communication tasks cleanly, the firmware maintains a modular structure that can evolve with software updates, aligning with modern software-defined instrumentation practices [2][3].

2. Software Interface

In the software implementation of our system, we use **Python** to generate the **grid and waveform** using built-in modules such as **Matplotlib, PyQt, and NumPy**. These libraries provide powerful tools for visualization, user interface development, and numerical computations. **Matplotlib** is used to plot and display the waveform, allowing real-time visualization of signals. **PyQt** is employed to create a graphical user interface (GUI), enabling interactive control of parameters such as scaling, zooming, and signal analysis. **NumPy** plays a crucial role in handling numerical operations, such as waveform generation, signal processing, and data manipulation. By combining these modules, we can efficiently simulate, analyze, and visualize waveforms, making Python an ideal choice for implementing software-defined oscilloscopes and other signal processing applications. The software is built on Python, utilizing Matplotlib for real-time waveform plotting, PyQt for GUI creation, and NumPy for signal handling [21].

Following is a screenshot of the software we are developing in its beta stage in Fig 5.–



Connectivity
Select COM Port: COM4
Baud Rate: 115200

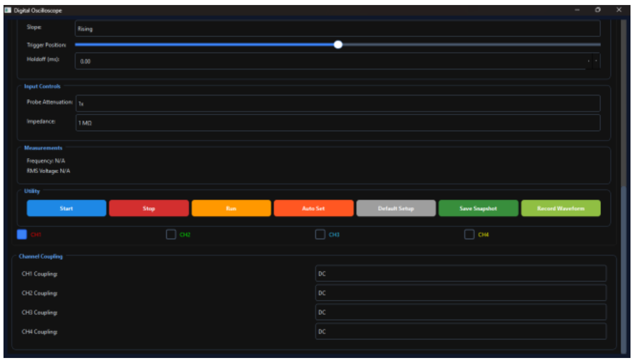
Display Controls
Intensity:
Grid

Vertical Controls
CH1
Volts/Div: 1.000
Position:

Horizontal Controls
Time/Div (s): 1.000000000
Position:

Trigger Controls
Trigger Level (mV): 0.00
Mode: Auto
Source: CH1

Fig. 5.1



Slope: Rising
Trigger Position:
Holdoff (ms): 0.00

Input Controls
Probe Attenuation: 1x
Impedance: 1 MΩ

Measurements
Frequency: N/A
RMS Voltage: N/A

Utility

CH1 CH2 CH3 CH4

Channel Coupling
CH1 Coupling: DC
CH2 Coupling: DC
CH3 Coupling: DC
CH4 Coupling: DC

Fig. 5.2

And this is how we are projecting the waveforms in the software –

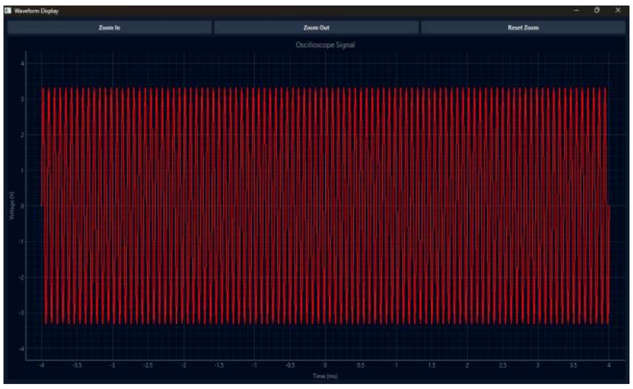
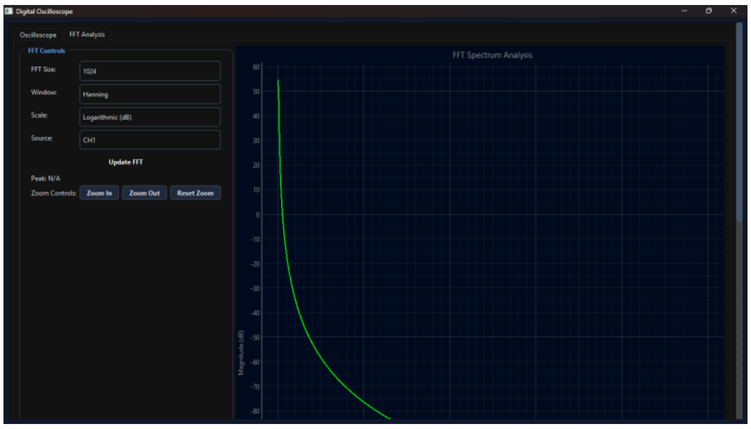


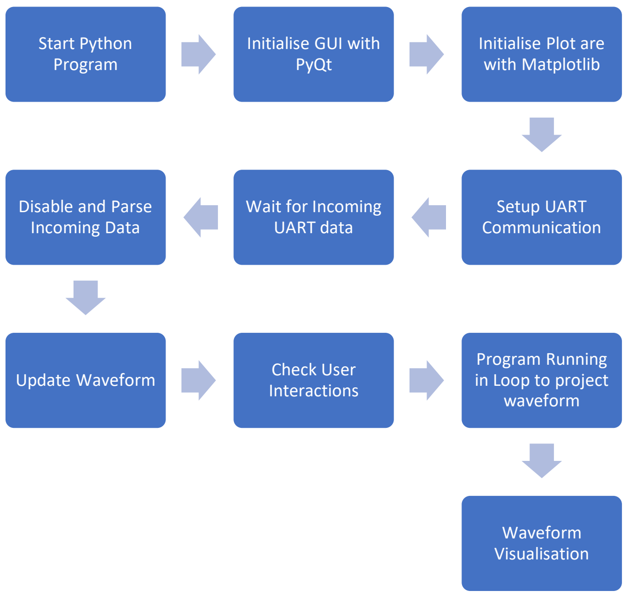
Fig. 5.3



FFT Controls
FFT Size: 1024
Window: Hanning
Scale: Logarithmic (dB)
Source: CH1
Peak: N/A
Zoom Controls:

Fig. 5.4

The above image shows a 3.3V AC Sinusoidal waveform being projected. Serial data reception from the microcontroller is handled in non-blocking mode to ensure responsive GUI operation [14], [15].


Start Python Program
Initialise GUI with PyQt
Initialise Plot are with Matplotlib
Setup UART Communication
Wait for Incoming UART data
Disable and Parse Incoming Data
Update Waveform
Check User Interactions
Program Running in Loop to project waveform
Waveform Visualisation

CONCLUSION

The implementation of the software-defined oscilloscope successfully demonstrated its ability to capture, process, and display waveforms in real-time using software-based techniques. By leveraging Python libraries such as Matplotlib, PyQt, and NumPy, the oscilloscope provided a flexible and interactive user interface for signal analysis. Further improvements could integrate cloud-based remote access and AI-based anomaly detection, enhancing the functionality of SDOs [2], [15], [17]. The triggering mechanism played a crucial role in stabilizing the waveform display, ensuring precise and reliable signal visualization. Compared to traditional hardware-based oscilloscopes, the SDO offers greater flexibility, cost-effectiveness, and adaptability, allowing for software-driven modifications and enhancements. While the system effectively processed and displayed signals, some limitations, such as sampling rate constraints and latency in high-frequency applications, were observed. These challenges can be addressed in future improvements through higher-performance ADCs and optimized signal processing algorithms. Increasing the sampling bandwidth by using faster ADCs and improving transmission speed could reduce latency in high-frequency applications [7], [12]. Overall, the software-defined oscilloscope proved to be an efficient and reliable tool for waveform analysis, making it a valuable solution for applications in education, research, and embedded system development. With further advancements, SDOs have the potential to revolutionize signal measurement and analysis by integrating AI-based processing and cloud-based remote access for enhanced functionality.

REFERENCES

[1] Texas Instruments, "Embedded system debugging with software-defined oscilloscopes," 2019.

[2] Tektronix, "Software-defined oscilloscopes: Future of waveform analysis," Tektronix White Paper, 2022.

[3] A. Agrawal and R. Malhotra, Digital Signal Processing and Software-Defined Instrumentation: A Modern Approach, 1st ed. 2021.

[4] A. V. Oppenheim and R. W. Schafer, Discrete-Time Signal Processing, 3rd ed. Pearson, 2010.

[5] T. H. Lee, The Design of CMOS Radio-Frequency Integrated Circuits, 2nd ed. Cambridge University Press, 2019.

[6] P. Horowitz and W. Hill, The Art of Electronics, 3rd ed. Cambridge University Press, 2015.

[7] National Instruments, "Understanding Oscilloscope Fundamentals," Application Note 1412, 2014.

[8] Keysight Technologies, "Oscilloscope Measurement Guide," 2017.

[9] R. J. Baker, CMOS: Circuit Design, Layout, and Simulation, 3rd ed. Wiley-IEEE Press, 2010.

[10] Analog Devices, "Precision Op Amps for Oscilloscope Front-Ends," Technical Article, 2019.

[11] A. S. Sedra and K. C. Smith, Microelectronic Circuits, 7th ed. Oxford University Press, 2014.

[12] Microchip Technology, "Getting Started with ADC on PIC Microcontrollers," Application Note AN900, 2020.

[13] J. G. Proakis and D. G. Manolakis, Digital Signal Processing: Principles, Algorithms, and Applications, 4th ed. Pearson, 2007.

[14] Cypress Semiconductor, "UART Design Guide for PSoC Microcontrollers," Application Note, 2016.

[15] A. Z. Sadik, "Software-Based Instrumentation for Embedded Systems," International Journal of Computer Applications, vol. 120, no. 15, 2015.

[16] Maxim Integrated, "Understanding Signal Level Shifting and Scaling Techniques," 2018.

[17] R. Prasad, Wireless Communications. McGraw-Hill, 2009.

[18] G. Smith, "Advanced Triggering in Digital Oscilloscopes," Electronic Design, 2017.

[19] H. W. Johnson and M. Graham, High-Speed Digital Design: A Handbook of Black Magic. Prentice Hall, 1993.

[20] S. Franco, Design with Operational Amplifiers and Analog Integrated Circuits, 4th ed. McGraw-Hill Education, 2014.

[21] MathWorks, "Signal Processing Toolbox User’s Guide," R2021b Documentation, 2021.

Would you like any adjustments or further formatting?