IMPLEMENTATION OF A SOURCE METER CONTROL SYSTEM WITH PYTHON USING SCPI AND VISA

Alexandr SEREACOV[1*], **Adrian BIRNAZ**[1]

[1]*Department of Microelectronics and Biomedical Engineering, Faculty of Computers Informatics and Microelectronics, Technical University of Moldova, Chisinau, Republic of Moldova*

*Corresponding author: Alexandr Sereacov, alexandr.sereacov@mib.utm.md

Tutor/coordinator: Oleg LUPAN, doctor habilitatus, *Department of Microelectronics and Biomedical Engineering, Faculty of Computers Informatics and Microelectronics, Technical University of Moldova, Chisinau, Republic of Moldova*

Abstract. *This paper describes existing communication with electronic devices open standards and experience of their integration in measurement system software. The presented measurement system software is designed to use SCPI over VXI bus to communicate with measurement device. Also, there is described a modular software architecture to make possible to interchange communication logic without touching other modules.*

Keywords: *SCPI, VISA, clean architecture, Python, modular software design*

Introduction

Nowadays laboratories are equipped with a very large variety of measurement devices which require unique software to work with. Building of a complex measurement system based on multiple different devices produces by different vendors requires usage of expensive software products. It this paper we describe our approach to implement measurement system using open standards and Python language to implement a user interface and data acquisition software for a third-party measurement device.

Communication layer

SCPI (Standard Commands for Programmable Instruments) is a standard language that defines a common syntax for communicating with test and measurement instruments, such as oscilloscopes, signal generators, and multimeters. SCPI provides a standardized set of commands and parameters that can be used to control and query instrument settings, as well as to read measurement data and status information. These program commands and parameters are sent from a controller to an instrument using IEEE 488.1, VXIbus, RS-232C, etc., interfaces [1].

One of the key benefits of using SCPI is that it provides a consistent programming interface across different types of instruments from different manufacturers. This means that a programmer can write a single program that can control a wide range of instruments without having to learn different command sets for each one.

SCPI commands and parameters are designed to be flexible, allowing instruments to accept a range of formats for programming. For example, a command to set the frequency of a signal generator might be accepted in multiple formats, such as "FREQ 1kHz", "FREQ:FIXED 1000", or "FREQ:CW 1000MHz" [1-3].

When an instrument responds to a SCPI command, it can send back both data and status information. Data information can be formatted in a device- and measurement-independent way, making it easier for the programmer to work with the data regardless of the specific instrument being used.

Overall, SCPI provides a powerful and flexible standard for controlling and querying test and measurement instruments, reducing the programming effort required to work with a variety of instruments and improving the consistency and reliability of instrument control.

SCPI command set for generic source meter:

- :ABORt
- :FETCh?
- :MEASure:<function>?
- :READ?
- :CALCulate
- :DIGital
- :DISPlay
- :FORMat
- :ROUTe
- :SENSe1
- :SOURce
- :STATus
- :SYSTem
- :TRACe
- :TRIGger

Horizontal and vertical consistency are two important concepts in SCPI programming that help to ensure a consistent and standardized programming interface across different instruments and functions [1].

Horizontal consistency refers to the consistency of SCPI commands and parameters across different instruments of the same type. For example, all oscilloscopes from different manufacturers should accept the same SCPI commands and parameters for setting the time base, triggering, and acquiring waveform data. This makes it easier for programmers to write generic code that can be used with different instruments, and also reduces the learning curve when switching between instruments.

Vertical consistency, on the other hand, refers to the consistency of SCPI commands and parameters across different functions within the same instrument. For example, the same command and parameter format should be used to set the trigger level on a scope, whether the trigger is for a single-shot acquisition or a repetitive waveform capture. This ensures that programmers can reuse code and commands across different functions within the same instrument, and also reduces the likelihood of errors due to differences in command formats [1].

A key to consistent programming is the reduction of multiple ways to control similar instrument functions. The philosophy of SCPI is for the same instrument functions to be controlled by the same SCPI commands. To simplify learning, SCPI uses industry-standard names and terms that are manufacturer and customer supported.

SCPI provides several different levels of instrument control. Simple Measure commands provide users easy and quick control of SCPI instrumentation, while more detailed commands provide traditional instrument control [1].

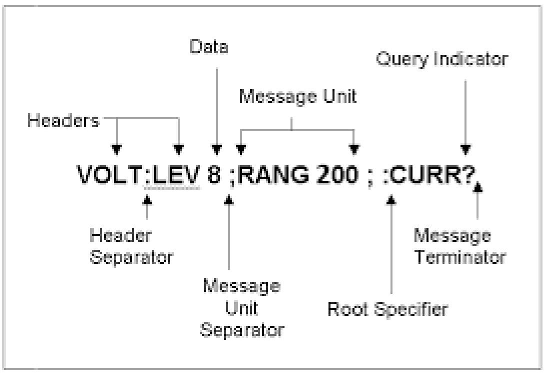

Figure 1. Example of SCPI commands syntax

One of the key design goals of SCPI is to allow for future expansion with new commands and parameters, without breaking compatibility with existing instruments and programs. This is achieved through a standardized command structure and syntax, which allows new commands to be added without conflicting with existing ones.

However, while SCPI commands and parameters can be expanded in the future to support new instruments, test programs designed for newer instruments may not be fully compatible with older instruments. This is because the new commands may not be recognized by the older instruments, or the parameters may be interpreted differently.

Additionally, SCPI provides a standardized mechanism for querying an instrument's capabilities, allowing a program to check whether a specific command or parameter is supported before sending the command.

However, SCPI does not provide complete instrument interchangeability, as it only defines the programming commands and responses, and not the instrument functionality, accuracy, resolution, connectors, etc. These factors are critical for ensuring that a replacement instrument can be seamlessly integrated into an ATE system without requiring hardware or software modifications.

In practice, replacing an SCPI instrument with another SCPI instrument typically requires less effort than replacing a non-SCPI instrument, since the programming commands and responses are standardized. However, even with SCPI instruments, there may be some differences in the specific commands and parameters used by different manufacturers or models, which can require some adjustments to the ATE system software.

To achieve true instrument interchangeability, additional standards are needed to define the physical and functional characteristics of instruments, such as the mechanical and electrical interfaces, accuracy specifications, and performance characteristics. Standards such as VXIbus, PXI, and LXI provide additional levels of standardization beyond SCPI, defining physical and electrical interfaces that can help to ensure that instruments from different manufacturers can be easily integrated into an ATE system.

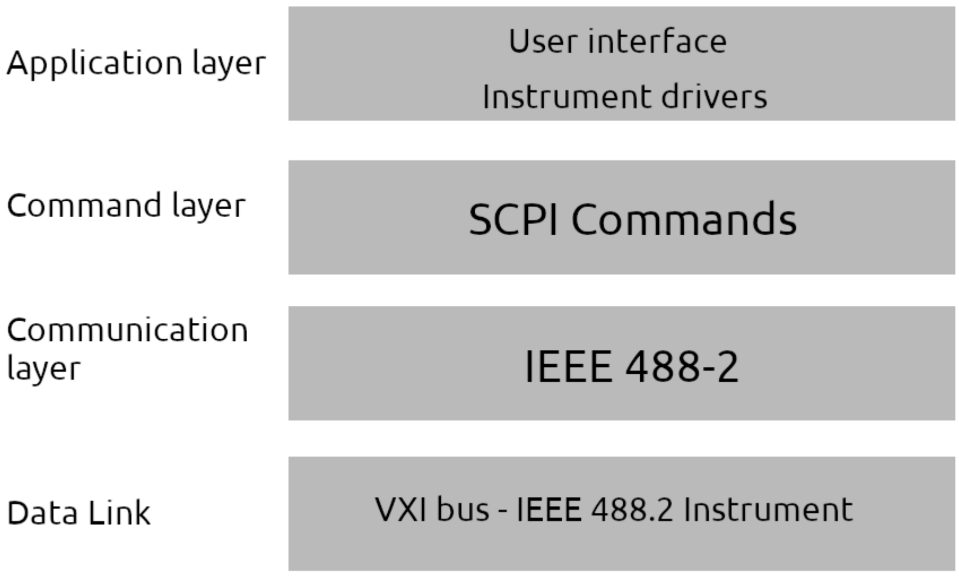

Figure 2. Protocol stack used for system implementation

Implementing clean architecture-based software

When designing software systems that involve user interfaces, it is often beneficial to combine a user interface implementation pattern with the clean architecture pattern. This can help to ensure that the system is easy to use and maintain, while also providing a high degree of modularity and flexibility.

The user interface implementation pattern typically involves separating the user interface code from the business logic and data access code. This is achieved by creating a set of classes and objects that are responsible for presenting information to the user and responding to user input, while delegating the actual processing of that input to separate layers of the system.

The clean architecture pattern, on the other hand, emphasizes the separation of concerns and the creation of independent modules that can be easily swapped in and out as needed. It achieves this by organizing the system into layers, each with a specific responsibility and purpose.

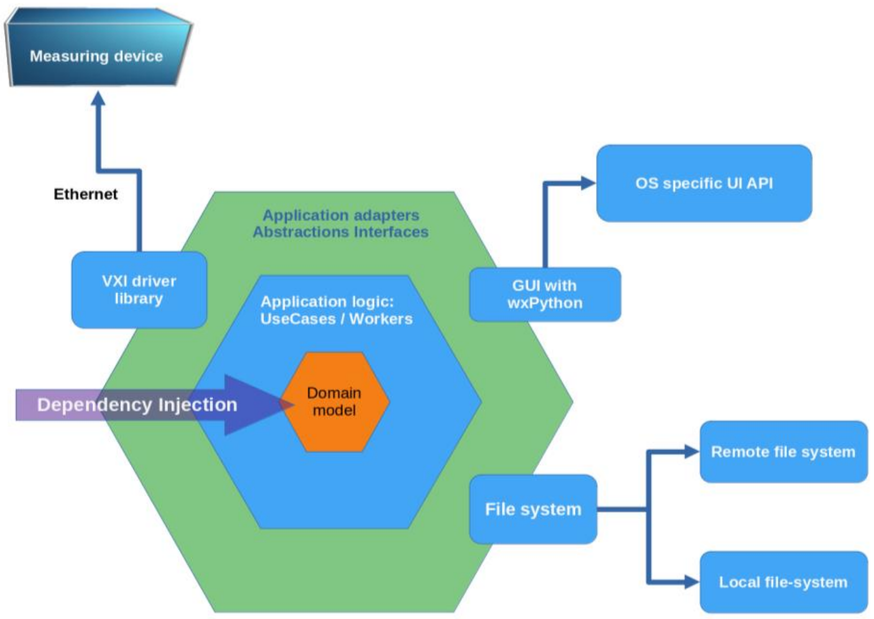

Figure 3. Clean architecture layers diagram

Combining these two patterns involves creating a user interface layer that interacts with the business logic layer in a clean and modular way. This can be achieved by creating interfaces or protocols that define the communication between the two layers, and implementing those interfaces in separate classes or modules.

For example, the user interface layer might define a set of protocols or interfaces that describe how user input should be handled, and the business logic layer might implement those protocols in separate classes or modules. This allows the two layers to communicate with each other in a way that is flexible and modular, and makes it easy to modify or replace individual components of the system without affecting the rest of the codebase.

Overall, combining a user interface implementation pattern with the clean architecture pattern can help to create a software system that is easy to use, maintain, and modify over time. By separating concerns and organizing the system into independent modules, it is possible to achieve a high degree of flexibility and modularity, while still providing a seamless and intuitive user experience.

User interface structure and implementation

This program provides the possibility to perform real time measurements with data saving and visualization features. Two work modes are supported:

- Volt-Ampere characteristics
- Real time transient analysis

Real time transient analysis has several adjustable parameters:

- Source voltage
- Maximum current limit
- Experiment temperature
- Agent to test sensitivity on. (Ex: gas, humidity level, light wavelength)

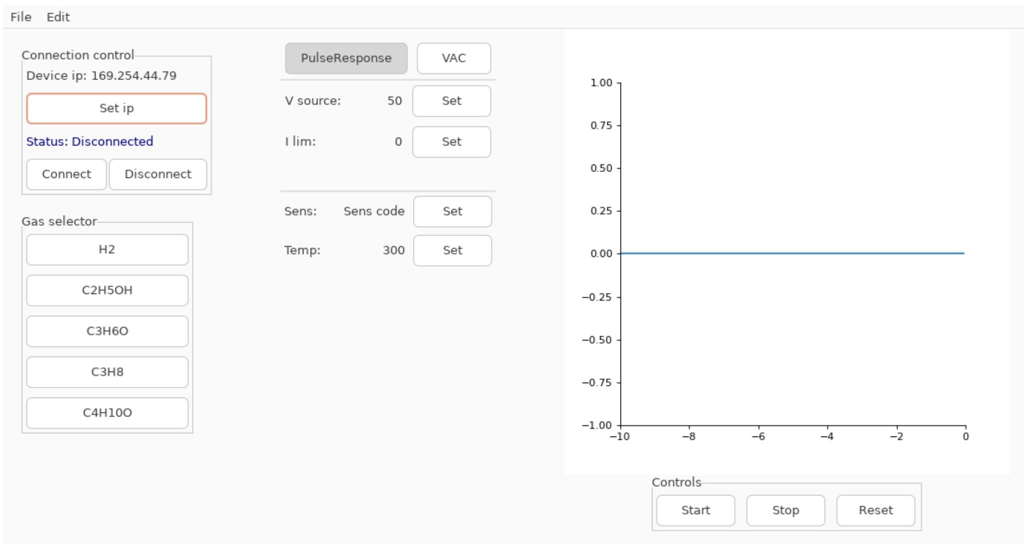
Connection control
Device ip: 169.254.44.79
Status: Disconnected
V source: 50
I lim: 0
Sens: Sens code
Temp: 300

Figure 4. Graphic user interface for real time transient analysis.

Volt-Ampere characteristics has 3 main parameters:

- Start voltage level
- End voltage level
- Number of measurement points (how many measurements of electrical current level will be performed during voltage change from it's start level to it's end level).

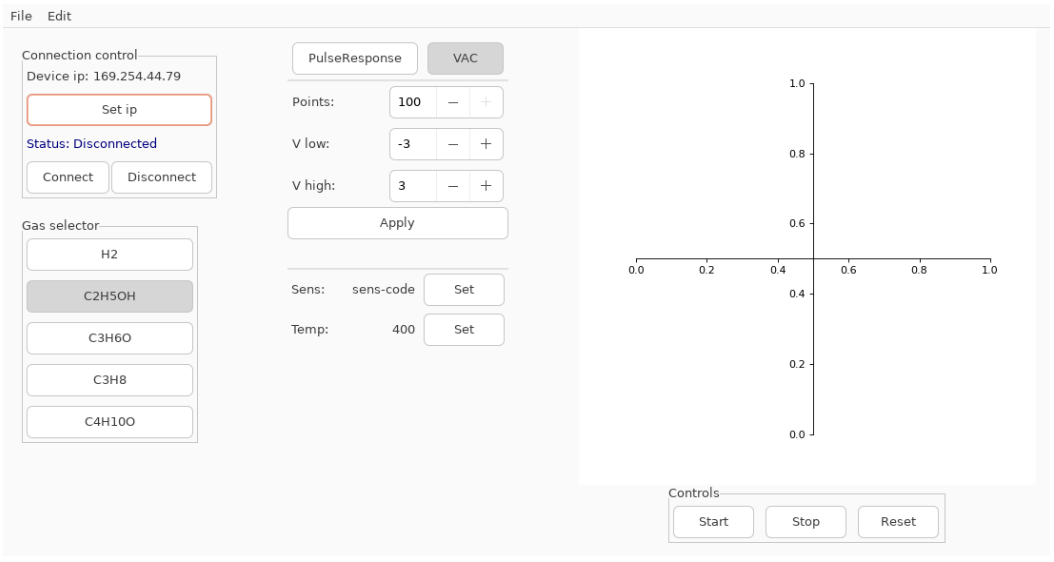
Connection control
Device ip: 169.254.44.79
Status: Disconnected
Points: 100
V low: -3
V high: 3
Sens: sens-code
Temp: 400

Figure 5. Graphic user interface for volt-ampere analysis

The source voltage can be set within the limits of millivolts up to 250 volts. Should be mentioned, that this limit is based on the measurement devices capabilities and on the software ones.

Graphic user interface structure is presented on Fig. 6. We knowingly left the GUI structure without implementation details, to keep it is structure cleaner. User interface handling is implemented in reactive manner. Then some event occurs, visual element responsible for this kind of event is changing it's properties.

User interface logic works on it's separate thread to keep it independent from the communication and data capture logic.

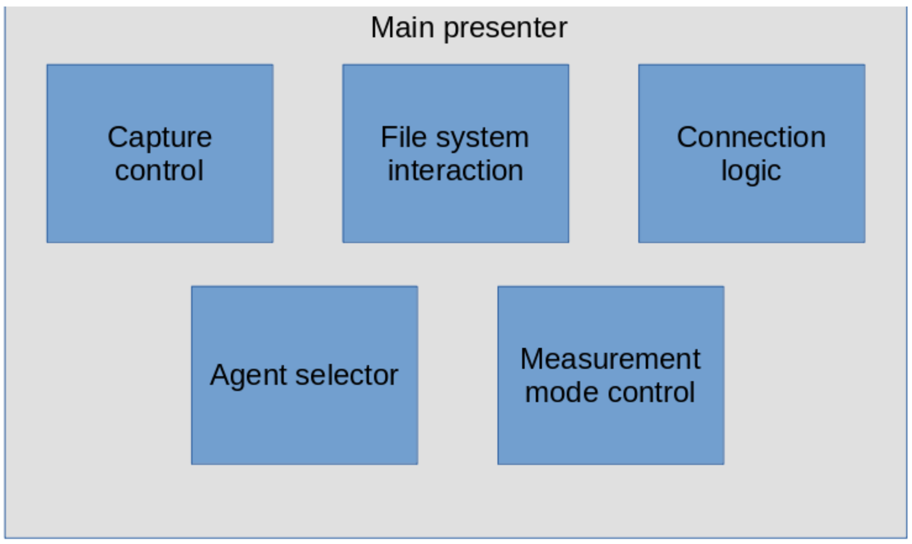

Figure 6. Graphic user interface main modules

Conclusions

Presented measuring system architecture has as a main goal to be standardized and opened for modifications and evolution. This lead us to the system with a communication layer based on open standards like: SCPI command standard, VXI network protocol, Ethernet protocol. And also, software part was based on the free licensed solutions: Python programming language, wxPython cross-platform framework for user interface implementation, matplotlib for graph building and data visualization in real time.

The proposed architecture has flexibility to be modified in accordance with actual individual for every laboratory requirement. The multi-modular approach enables us to integrate new devices with minimal efforts of new driver and communication protocol implementation.

This system is flexible enough to be extended to fully automatic measuring system.

References

1. SCPI Consortium, Standard Commands for Programmable Instruments (SCPI). USA, 1999
2. *IEEE*, International Standard EEE 488.1, *2004*
3. AARØEN, O, KIÆR, H, RICCARDI, E. PYVISA: Visualization and Analysis of path sampling trajectories. *J Comput Chem*. 2021; 42: 435– 446. https://doi.org/10.1002/jcc.26467
4. JOHN M. PIEPER.: SCPI and VISA a valuable combination. (2002)