Constructing a dependable side channel data acquisition system for Tektronix 4 Series oscilloscopes

Federico Cerutti, fce201
federico@ceres-c.it

February 16, 2024

Abstract

A successful power analysis attack requires a high-quality and reliable data acquisition system. This report details the construction of a software library to aid in the acquisition of power traces from Tektronix 4 Series Mixed Signal Oscilloscopes (MSOs). The report will also provide information on the different communication protocols used to control test equipment. The reader will be guided through the process of setting up the oscilloscope and acquiring traces.

1 Instrument communication protocols

The majority of measurement instruments come equipped with various communication interfaces, with the most common being USB and Ethernet. Multiple protocols exist to facilitate communication with test equipment, and this section will quickly introduce relevant standards and libraries.

1.1 Transport

1.1.1 USB Test and Measurement Class (USB-TMC)

The USB Implementers Forum has defined a standard for USB communication with test equipment. The standard is called USB-TMC and is supported by most test equipment manufacturers. It defines device descriptors, USB endpoints used for control and data transfer, and encoding of commands and data.

1.1.2 LAN eXtensions for Instrumentation (LXI)

LXI is a standard to leverage Ethernet and TCP/IP technologies to control test equipment. The consortium behind LXI maintains a set of standards that specify communication protocols (VXI-11, HiSLIP), device discovery features, and REST APIs.

1.1.3 Virtual instrument software architecture (VISA)

VISA is an API that hides the details of the transport layer and provides a common interface to communicate with test equipment. It is a standard developed by the Interchangeable Virtual Instrument Foundation (IVI) and is supported by most test equipment manufacturers. It provides a number of operations (`read`, `write`, `flush`...) and an event system to react to changes.

1.2 Control

1.2.1 Standard Commands for Programmable Instruments (SCPI)

SCPI is a transport-independent API that dictates how to communicate with test equipment. It is a standard developed by the IVI and provides a list of commands and queries to control the instrument. It is a text-based protocol, and it is often used on top of VISA to control test equipment. While it should be a standard, often vendors use different commands and extend it with proprietary features.

1.3 VISA drivers

1.3.1 pyVISA and pyVISA-py

pyVISA provides python bindings for VISA libraries: it does not offer any implementation of VISA, but it relies on third party libraries to provide the functionality. Many vendors provide their own VISA libraries, some of which are available on linux as well, but are often commercial and not open source. pyVISA-py, in turn, is an open source pure python implementation of VISA that can be used on any platform and interfaces directly with pyVISA. This library lacks some features of the VISA standard (`viClear`, `viClose`...), but it proved to be sufficient for the purpose of this project.

2 4 Series MSO

The CCI group at the University of Amsterdam has acquired a Tektronix MSO44[1] (note: not MSO44B), a 4 Series MSO with 4 analog channels and a 12-bit ADC at 3.125 GS/s per channel. The ENOB, as stated in the datasheet, is in the range of 8.9 bits (@20MHz) to 7.1 bits (@1.5GHz). These characteristics make it a suitable device for power analysis attacks.

The oscilloscope is equipped with a USB-B 2.0 "device port" (used to control the device from a host computer) and an Ethernet port. There are more USB-A ports, and they can be used for USB HID or storage devices. It supports both USB-TMC and LXI communication protocols with SCPI commands.

2.1 Speculations on the firmware

The software architecture of the oscilloscope seems to be more reminiscent of a monolithic, tightly integrated embedded system rather than a modular RTOS. No SDK is provided for this specific device by the vendor, and the only way to control the oscilloscope is through the graphical UI or VISA.

A quick reverse engineering analysis of the firmware showed that the control of the analog frontend is built into the UI binary, consequently, the LXI server also has to go through the UI to execute commands. This becomes problematic when the LXI server crashes and stops responding, causing the UI to also become unresponsive, thus requiring manual interaction to reboot the oscilloscope. Crashes will happen in multiple circumstances, but they can mostly be summed up in two categories:

- **Buffer overflows**: The oscilloscope has a limited amount of memory, and it is easy to fill it up when acquiring and transferring multiple traces with the `CURVE?` command. Speed does not directly affect this issue, as even at speeds as low as 1 trace/s the oscilloscope will eventually crash after $\sim$ 300 traces. I speculate this is due to some internal elaboration buffer that is not being freed. This problem was solved with CurveStream and FastAcq modes (more in subsection 3.3).
- **Network issues**: The TCP stack running on the oscilloscope will stop responding to requests after a variable number of traces in the 20000 $\sim$ 40000 range, forcing again a manual reboot. This problem was solved by using the USB interface as a backup control interface.

3 pyMSO4

The pyMSO4 library is a python library that provides an interface to control the Tektronix 4 Series MSOs and acquire power traces. It is built on top of pyVISA and pyVISA-py and provides a high-level interface to control the oscilloscope. The library is designed to be easy to use and to provide a high-level interface to control the oscilloscope. It should not be necessary to resort to the programmer manual to configure the basic settings of the oscilloscope.

3.1 Wiring

The oscilloscope should be connected to the host computer via both Ethernet and USB. Ethernet is used to control the instrument as well as acquire data, while USB is used as a backup control interface if the

[1] https://www.tek.com/en/datasheet/4-series-mso

Ethernet connection is lost. This is required because, as stated in subsection 2.1, the visa implementation on the VISA Resource is not reliable and often crashes, requiring a reset of the oscilloscope.

- **Ethernet and switch**: The oscilloscope is connected to a router, to which the host computer is also connected. The router provides a DHCP server to assign an IP address to the oscilloscope.

  
  Utility → I/O… → LAN
  Network Address: Auto
  Apply Changes

  
  Create a standard network connection with DHCP to the router

- **Ethernet direct**: The oscilloscope is directly connected to the host computer. The host computer is configured with a static IP address in the same subnet as the oscilloscope.
  Note: You need either a crossover cable or a modern network card with Auto MDI-X[2].

  
  Utility → I/O… → LAN
  Network Address: Manual
  Instrument IP Address: 128.181.240.130 (example)
  Subnet Mask: 255.255.255.0
  Apply Changes

  
  ```
  sudo nmcli con add con-name "tek-mso44-p2p" ifname <INTERFACE NAME>
  ↪  type ethernet ip4 128.181.240.131/24
  ```

- **USB**: Connect the USB-B "device" port on the back of the oscilloscope to the host computer. The oscilloscope will be recognized as a USB-TMC device.

  
  Utility → I/O… → USB Device Port
  USB Device Port: ON

3.2 Software setup

The library is available on PyPI and does not depend on any proprietary software. On Debian 12, the following commands will install the required software:

```
sudo apt update && sudo apt install python3 python3-pip python3-venv
python3 -m venv venv
source venv/bin/activate
pip3 install pymso4
```

Additionally, the USB device needs to be accessible by the user running the script. This can be achieved on Debian adding the user to the `dialout` group and with udev rules (file available in the repository at Appendix A):

```
sudo -E usermod -a -G dialout $USER
# Now logout
cp 50-newae.rules /etc/udev/rules.d/50-newae.rules
sudo systemctl stop ModemManager && sudo systemctl mask ModemManager
sudo udevadm control --reload-rules && sudo udevadm trigger
# Did you logout?
```

To test the configuration, run the following python script:

[2] https://en.wikipedia.org/wiki/Medium-dependent_interface#Auto_MDI-X

```
source venv/bin/activate
pip3 install psutil # Necessary to discover TCP connected devices
pyvisa-shell
(visa) list
( 0) USB0::1689::1319::C019654::0::INSTR
( 1) TCPIP::192.168.1.140::INSTR
```

There should be at least 2 entries in the output, one for the USB device and one for the Ethernet device.

3.3 Pitfalls

VISA is a complex standard with asynchronous operations, events, and multiple layers of abstraction. The VISA standard is not always implemented correctly by vendors, and the pyVISA-py library is not a complete implementation of the standard. This means there are multiple gray areas and "should's" in the documentation that are not always true. For example, some commands simply do not work as stated in the programming manual[1]. Here are some of the issues encountered during the development of the library:

- **Synchronization issues 1**: It is often the case that swapping the order of two instructions in the code will result in a different behavior of the oscilloscope. This is due to the fact that the oscilloscope is not always able to keep up with the commands sent to it, regardless of what stated in the manual[1, p. 1915], and some commands might not execute in time.
  Solution: Explicit delays might help.
- **Synchronization issues 2**: Some long-running commands can set the bit 0 of the `SESR` register when the execution is complete.
  Solution: A list is available in the manual[1, t. 3-3].
- `*OPC`: The non-query version of the `*OPC?` command does not work as expected[1, p. 1001]. The query version works as expected.
- **Buffer overflows**: As stated in subsection 2.1, extensive usage of the `CURVE?` command will eventually crash the oscilloscope.
  Solution: Use CurveStream mode, which sends data directly to the host with minimal buffering
- **CurveStream**: Using CurveStream mode without FastAcq mode will result in the oscilloscope crashing just like with the `CURVE?` command.
  Solution: Enable FastAcq mode.
- **FastAcq**: The length of the waveform retrieved from the oscilloscope can be freely configured, but its actual value will not be updated on the oscilloscope's end until a normal acquisition is performed. This is (probably) due to the optimizations done in FastAcq mode that skip many of the post-procesing steps that would normally happen in a normal acquisition.
  Solution: Force a trigger after setting the length of the waveform and check if the data length has been updated.

This list is in no way exhaustive, it is just a list of issues encountered during the development of the library that I can remember at the time of writing.

4 Architecture

The library has a main class, `MSO4`, which acts as the main interface to connect to and control the oscilloscope. The configuration is done through properties and methods of this class: reading and writing these properties will send the appropriate commands to the oscilloscope. The `MSO4` class instance also contains instances of other classes:

- `MSO4.sc`: The pyVISA resource used to communicate with the instrument
- `MSO4.acq`: Acquisition settings such as horizontal scale and position, sampling rate, waveform length…
- `MSO4.ch_a`: Per-channel and vertical settings

- `MSO4.trigger`: Trigger settings, different classes implement different trigger types

For more details on the available properties and methods, refer to the documentation at Appendix A.

```
import pyMSO4
mso44 = pyMSO4.MSO4(trig_type=pyMSO4.MSO4EdgeTrigger)
mso44.con(ip="123.234.123.234")
scope.ch_a_enable([True, False, False, False]) # Enable channel 1
scope.ch_a[1].scale = 1
scope.trigger.mode = 'auto'
scope.sc.write("CURVE?") # Interact with the pyVISA resource directly
mso44.sc.read_binary_values(datatype=mso44.acq.get_datatype(),
↪  is_big_endian=mso44.acq.is_big_endian)
```

Listing 1: pyMSO4 minimal example

Acronyms

IVI Interchangeable Virtual Instrument Foundation. 1

LXI LAN eXtensions for Instrumentation. 1, 2

MSO Mixed Signal Oscilloscope. 1, 2

SCPI Standard Commands for Programmable Instruments. 1, 2

USB-TMC USB Test and Measurement Class. 1–3

VISA Virtual instrument software architecture. 1, 2, 4

Glossary

VISA Resource Any instrument that can be controlled using the VISA standard. 3

References

[1] *4, 5, 6 Series MSO Programmer Manual*, https://github.com/ceres-c/pyMSO4/blob/master/docs/4-5-6-MSO_Programmer_077130520_fw2_0_x.pdf, Tektronix, Inc., supports FW version 2.0.x and above.

[2] M. O. Orlando, "oscilloscope," https://freeicons.io/laboratory-solid-51240/oscilloscope-calibration-voltages-laboratory-equipment-icon-2182335.

[3] www.wishforge.games, "laptop," https://freeicons.io/apps-&-programming-2/applications-and-programming-laptop-computer-coding-code-script-icon-41736.

A Software

Source code of pyMSO4 python library can be found at https://github.com/ceres-c/pyMSO4.
Auto generated code documentation is available at https://ceres-c.it/pyMSO4/.

B Hardware

B.1 Bill of materials

The setup was tested with the following components:

- 1 × MSO44 oscilloscope (Firmware version *non-windows V2.0.3.950*)
- 1 × Debian 12 computer with 1 USB-A port and 1 Ethernet port
- 1 × Xiaomi Mi Router 4A with OpenWRT (any switching device will do)
- 2 × Ethernet cables
- 1 × USB-A to USB-B cable

Additionally, to communicate with the CW305 target board, the following components are required:

- 1 × SMA-SMA cable
- 1 × SMA-BNC adapter
- 1 × BNC probe

C If your `uid` is not 0, you don't own it