High Speed Continuous Voltage Data Capture from Picoscope

By: Keerthana Sudarshan
Supervisor: Dr. Prabhu Thiagaraj
19th January 2024

1 Introduction

The Picoscope is a PC oscilloscope from Picotech, that can be connected directly to a computer via USB. The input waveform is observed and analysed either through the Picoscope GUI application, or via API commands in the terminal.

Some of the features of the Picoscope are:

1. 4 input channels
2. Vertical resolution of 8 bits
3. Horizontal resolution: minimum sampling interval of 800 picoseconds (1250 MHz)
4. Capture memory size of 4 GS
5. Sampling rate of 5 GS/s
6. Up to 50 MHz Arbitrary Waveform Generator (AWG)
7. 1 kHz square-wave output

The Picotech Oscilloscope can record data in two modes- Block mode and Streaming mode. A variant of block mode known as rapid block mode may also be used to reduce the time lost in between recorded waveforms.

2 Installation

First, the drivers of the picoscope must be installed. The specific commands for each individual model can be found on the PicoTech website linked in the references. For the Picoscope 6404E model, the command is `sudo apt-get install libps6000a`.

Next, to install the software development tools, Python and PIP are required. If not already installed. install pip using `sudo apt-get install python3-pip python.dev`

To use API commands for data retrieval, the PicoSDK toolkit is required. This is available on the Picotech github page (linked in the References). For this installation, we will be using the Python wrapper provided. Download the 'picosdk-python-wrappers' contents as a .zip file. Extract into a folder, and in the command line, use `pip install .` to install the software. Then, the API commands may be used by importing `picosdk.ps6000a` and `picosdk.functions` in the python file.

Note- The API commands were written in C. So, whenever giving numerical input to the commands, it expects certain specific C datatypes. These can be specified in python using the `ctypes` library.

Finally, the code uses the matplotlib package for plotting waveforms and spectra. This can be installed using `sudo apt-get install python3-matplotlib`

Also helpful to test parts of the code a time is IPython software, which can be installed using `sudo pip3 install ipython`.

3 Block Mode

In block mode, the picoscope records an entire waveform, with the sampling rate and number of samples specified, and stores the information in its internal memory. This data can later be retrieved asynchronously, from the buffer. Thus, the data file size is limited by the internal memory available- for any given resolution, there is a maximum number of samples that can be recorded and stored at once. This mode is more suitable for recording short durations, and has greater flexibility of sample rate and resolution.

To record data in block mode, we go through the following steps:

1. Open the device with ps6000aOpenUnit(*handle, *serial, resolution)
2. Turn on the required channel with ps6000aSetChannelOn(). Turn other channels off with ps6000aSetChannelOff()
3. Set trigger using ps6000aSetSimpleTrigger(). If you want to set it by a particular voltage value, first get the ADC limits, then use the max ADC value and the required voltage value to get the required ADC as input to set the trigger.
4. Set the timebase (corresponding to some sample interval), or obtain the fastest sampling rate from ps6000aGetMinimimTimebasStateless()
5. Set the number of samples to be recorded pre- and post-trigger.

6. Create the buffers with length equal to the total number of samples. Without downsampling, only one buffer is required. With downsampling, two buffers are required. Use ps6000aSetDataBuffer()

7. Run using ps6000aRunBlock(). Wait for data collection to finish by running ps6000aIsReady() until it returns 1 instead of 0.

8. Extract data using ps.ps6000aGetValues()

The data is extracted in the format of raw ADC counts, which can be plotted as is to view the waveform, or can be converted to millivolts using the appropriate picosdk function and then plotted.

We can also view the spectrum of the waveform by performing an FFT using the numpy fft library.

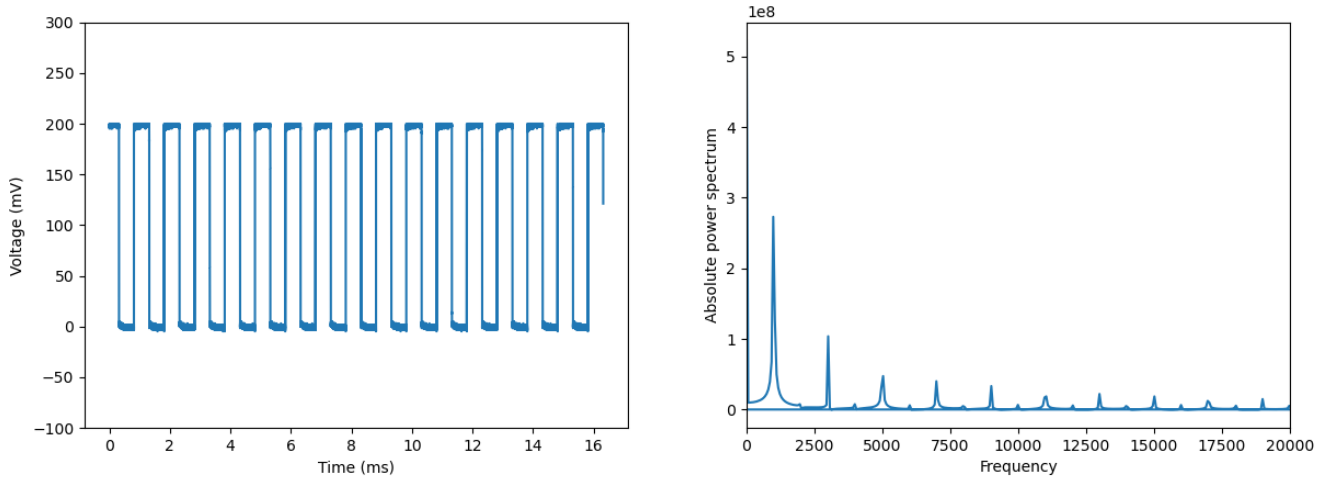

Figure 1: Waveform and corresponding spectrum of a 1 kHz sqaure-wave input, captured using the block mode, with sampling interval of 3.2 nanoseconds. The waveform is constructed from 5 million samples.

4 Streaming Mode

Streaming mode facilitates recording of data over much longer periods, at the cost of flexibility in sampling rate, etc. In this mode, data is transferred in parts from the device to the computer, at high speeds. The speed of transfer is limited by the USB connection, and the volume of data that can be recorded is limited by the resources of the computer. Data can be captured continuously, without any time gap between buffer sets, as would occur in block mode or rapid block mode.

In streaming mode, the data is collected in sets. The size of an individual set is limited

cannot exceed the maximum buffer size of the device, but the number of sets that can be collected is not limited by the picoscope. Each set of data is recorded separately, and these can later be combined to form a singe, continuous waveform.

To record data in streaming mode, we go through the following steps:

1. Open the device with ps6000aOpenUnit(*handle, *serial, resolution)
2. Turn on the required channel with ps6000aSetChannelOn(). Turn other channels off with ps6000aSetChannelOff()
3. Set trigger using ps6000aSetSimpleTrigger(). If you want to set it by a particular voltage value, first get the ADC limits, then use the max ADC value and the required voltage value to get the required ADC as input to set the trigger.
4. Decide the number of samples per set and the number of sets required.
5. Create the buffer for the channel. The dimensions of the buffer should be (number of samples per set) X (number of sets).
6. Using ps6000aSetDataBuffer(), set the receiving buffer as the first row of buffer created above.
7. Set the desired sample interval, specifying the value as well as the order of magnitude with the PICO_TIME_UNITS. Note that this may not be the final sample interval used- the device will choose an interval close to the desired one that is can accommodate (see the 'Timebases' section of the Programmer's Guide).
8. Create two structures using the picostructs package- one to contain the streaming data information, and one to contain the trigger information.
9. Check the status of the buffer using ps6000aGetStreamingLatestValues(). If it gives PICO_OK, the buffer is not full and can keep recording. If the check returns PICO_WAITING_FOR_DATA_BUFFERS, the next buffer (next row of the buffer created in 4.) needs to be assigned. Repeat this until the entire buffer array has been filled.

Once again, the data can be converted from ADC counts to millivolts , and plotted against time.

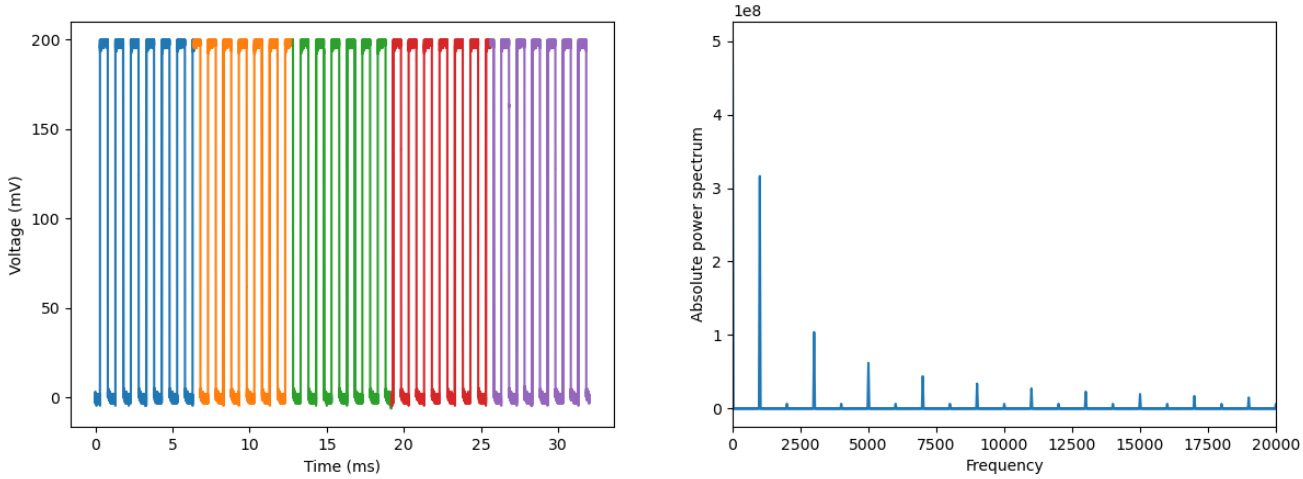

Figure 2: Waveform and corresponding spectrum of a 1 kHz sqaure-wave input, captured using the streaming mode, with 5 buffers and a sampling interval of 6.4 nanoseconds. Each buffer contains 1 million samples.

We can see that the peaks in the frequency spectrum become sharper as more oscillations are recorded, as we would expect.

4.1 Streaming Mode Limitations

Using streaming mode imposes a constraint on the sampling frequency, beyond the constraints of the device itself. This in turn restricts the range of frequency values for which the FFT spectrum can be computed.

1. For 1 channel streaming: the minimum sample interval is 800 picoseconds. This gives an FFT spectrum upto 600 MHz.
2. For 2 channel streaming: the minimum sample interval is 1.6 nanoseconds. This gives an FFT spectrum upto 300 MHz.
3. For 4 channel streaming: the minimum sample interval is 3.2 nanoseconds. This gives an FFT spectrum upto 150 MHz.

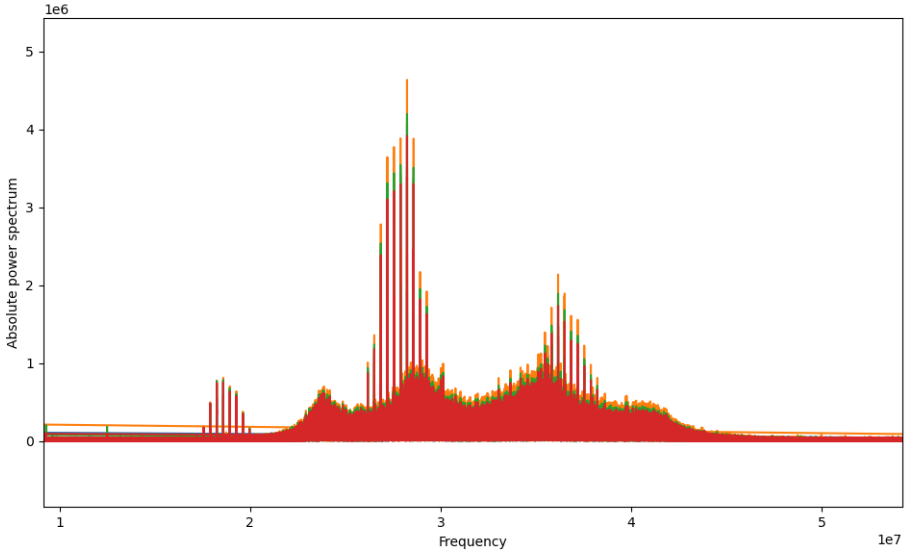

Figure 3: FFT of noise input with 30 MHz bandpass filter. 4 channels, with sampling interval of 6.4 ns.

Additionally, the Picoscope 6404E model only supports 8-bit resolution, which divides the voltage range into 256 levels.

4.2 Saving to file

Using `numpy`, the data extracted from the picoscope can be saved as a text file (.txt format) or as a binary file (.bin format). In order to save as text, use:

```
np.savetxt('file_name.txt', array_name)
```

This can later be read using `np.loadtxt('file_name.txt')`

In order to save as binary file, use:

```
array_name.astype('float').tofile('file_name.bin')
```

This can later be read using `np.fromfile('file_name.bin', dtype=float)`

5 Correlation Coefficients

To obtain the correlation coefficient, we begin with the dataset (the values of voltage at different times) in 4 separate 1-dimensional arrays. Then:

1. Divide each 1D array into bins of 1024 successive values.

2. For each bin, calculate the FFT of the series. Discard the second half of the values (it will be a mirror image of the first half) to be left with 512 values in each bin.

3. For the two channels for which the correlation coefficient is to be obtained, do a point-wise multiplication of the spectrum values for each bin.

4. Average the spectrum across the bins, by doing a point-wise addition of the spectrum of all bins, and then dividing by the number of bins. This yields a 512 point array we shall call the correlation spectrum (CS).

5. To get the correlation coefficient, first calculate the modulus squared of each value. Then, divide the spectrum by square-root of the product of the correlation spectrums of the two individual channels. For example, to find the correlation coefficient of channels A and B, divide CS(A and B) by $\sqrt{\text{CS(A and A)} * \text{CS(B and B)}}$ This gives the correlation coefficient for each frequency bin.

6. To get the phase correlation, get the angle of the complex values in the correlation spectrum by taking $\tan^{-1}(\text{Im/Re})$.

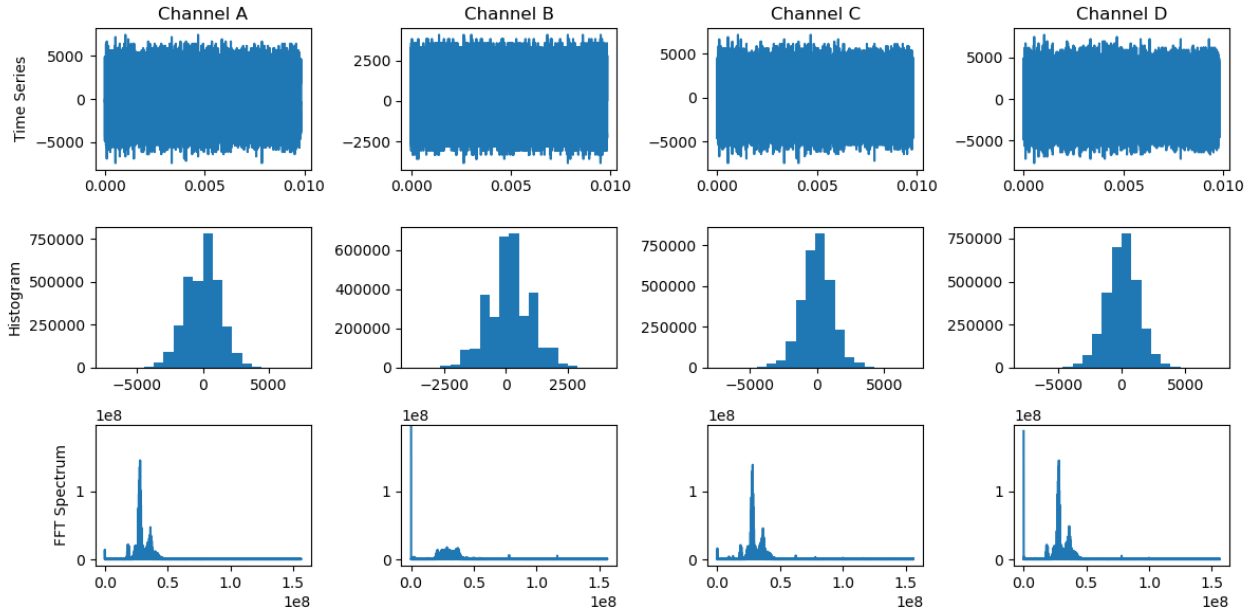

Figure 4: Waveform, histogram of ADC counts and FFT spectrum of all 4 channels in streaming mode. 3 channels (A, C, D) have input from the same noise source with 30 MHz bandpass filter, while B has input from a different noise source, also with bandpass filter at 30 MHz

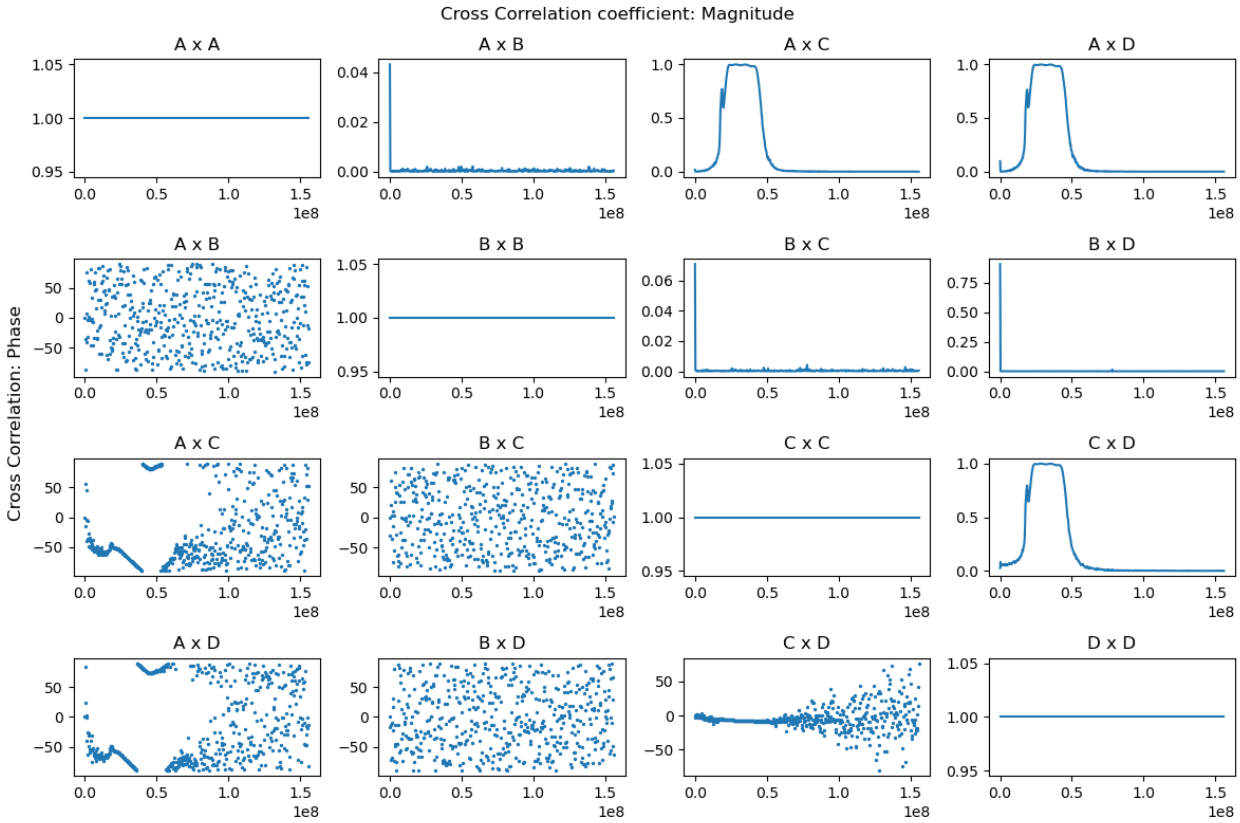

Figure 5: Correlation graphs of all four channels. The upper triangle shows the correlation coefficient plots, and the lower triangle shows the phase coefficient plots. As before, 3 channels (A, C, D) have input from the same noise source with 30 MHz bandpass filter, while B has input from a different noise source, also with bandpass filter at 30 MHz

6 References

[1] PicoScope 6000E Series (ps6000a API) Programmer's guide

[2] Picotech PicoSDK Python Wrappers and examples

[3] PicoTech: Software and drivers installation for Linux