4 * ACQ2106 + ACQ423 in low latency control mode to a single host.

This factory acceptance test outlines the procedures undertaken to demonstrate that the equipment is functioning as intended.

Minimum required release:
RELEASE acq400-228-20200625141955

Revision of document.

Date	Change
29.06.2020	Init

Load driver for low latency control.

To configure the host computer for low latency control mode there is a script that must be run in order to load the driver for the module. The script can be found in the D-TACQ AFHBA404 github repo here:

https://github.com/D-TACQ/AFHBA404

To load please run the following commands inside the AFHBA404 directory after cloning (or updating) the repository:

```
sudo make

sudo ./scripts/install-hotplug

sudo ./scripts/loadNIRQ
```

Installing acq400_hapi

The user will need to clone acq400_hapi from GitHub and install it on PIP. The repository can be found here:

https://github.com/D-TACQ/acq400_hapi

The repository should be cloned to the following location on the host computer:

/home/$USER/PROJECTS/

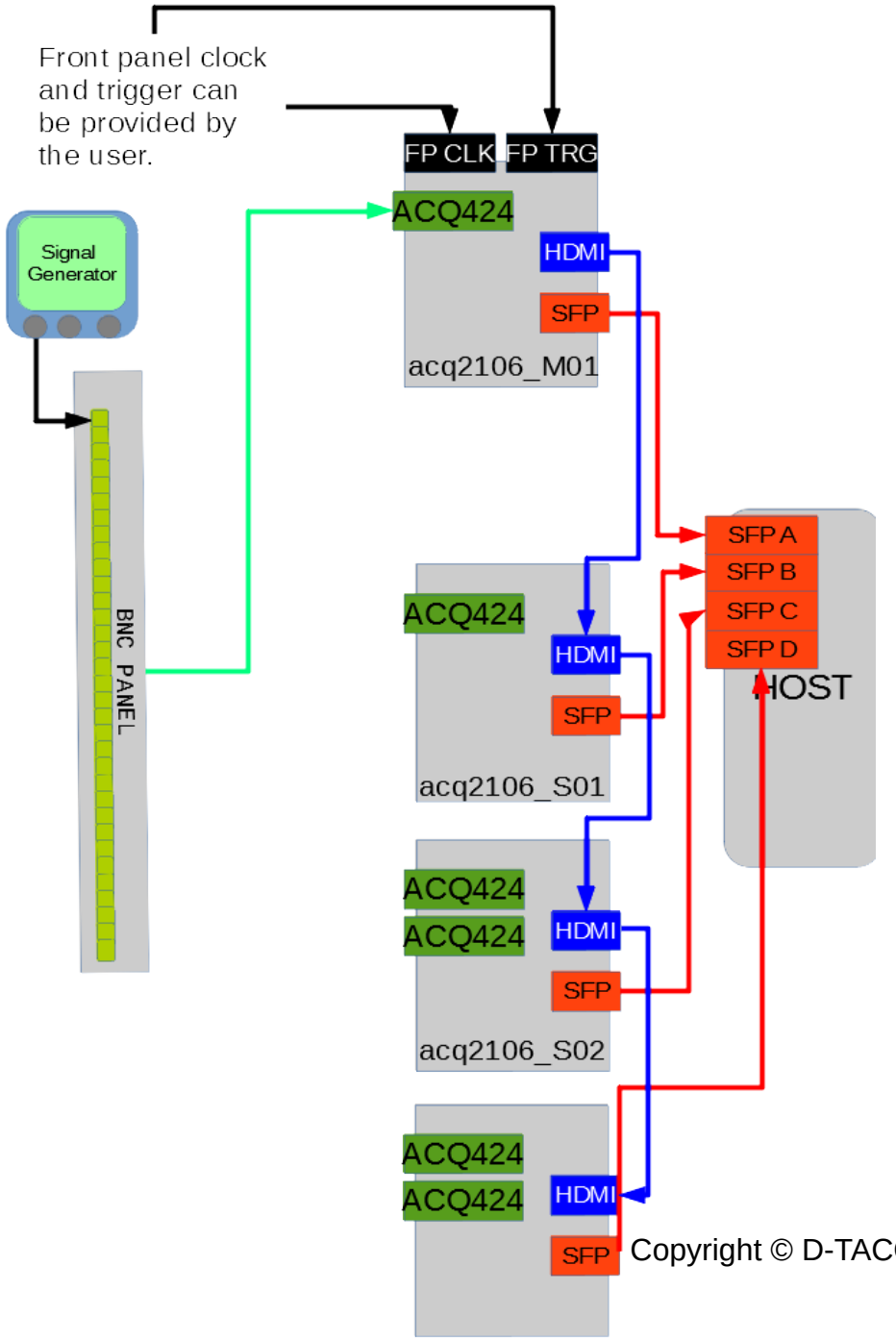
It can be cloned using the following command:

```
git clone https://github.com/D-TACQ/acq400_hapi.git
```

Once this repository has been cloned, acq400_hapi can be installed by running:

```
sudo pip3 install acq400_hapi
```

acq2106_M01 – Master
acq2106_S01 – Slave
acq2106_S02 - Slave-Slave



Set up CS Studio

Configure CS Studio to monitor the UUTs. It is best to use the STREAMVIEW4.opi. Configure a CS Studio workspace as such:

Set macros UUT1 UUT2 UUT3 UUT4 from

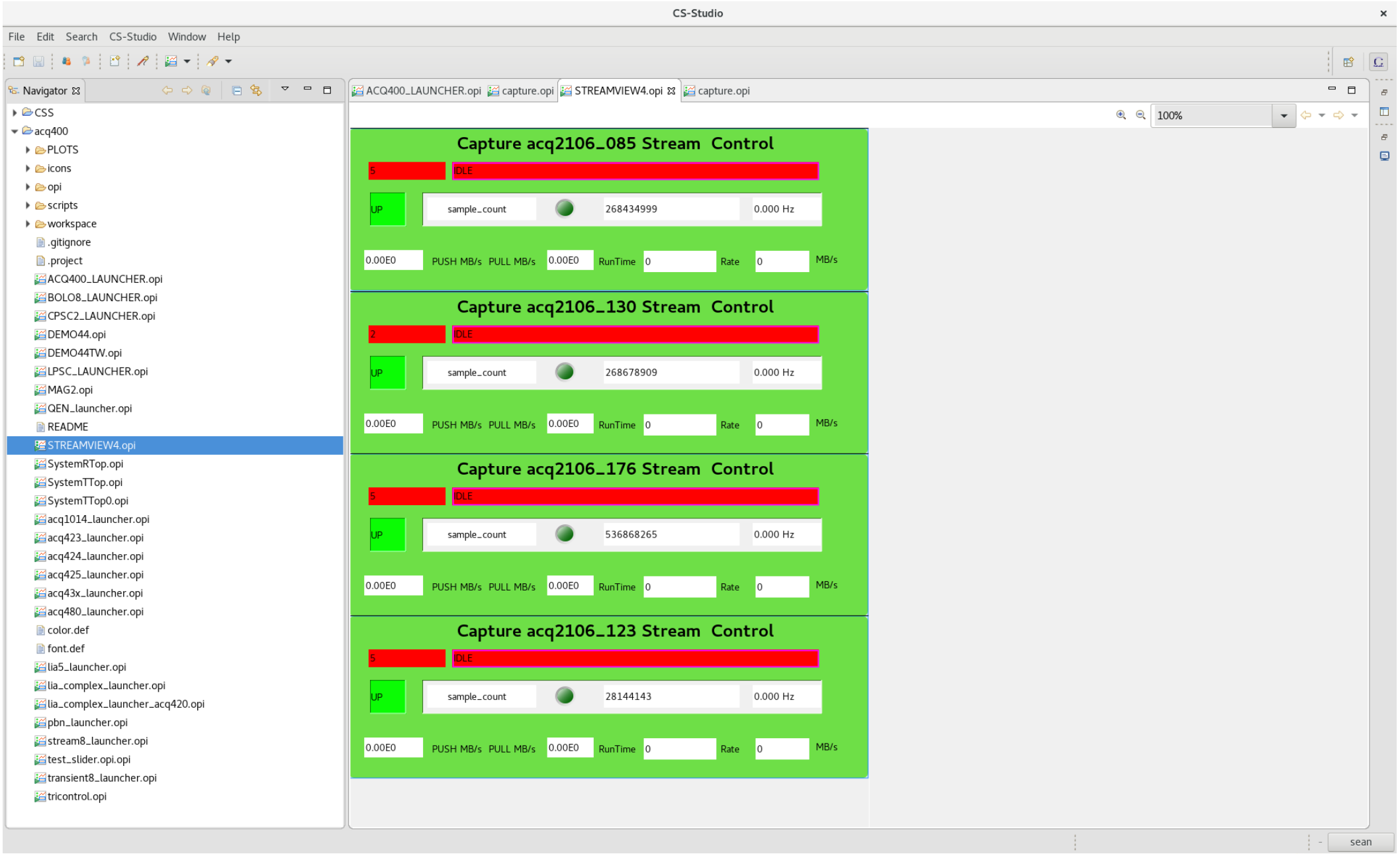
Edit|Preferences|CSS Applications->Display->BOY|OPI Runtime

Run STREAMVIEW4.opi direct from Navigator

new file: ACQ400/STREAMVIEW4.opi

new file: ACQ400/opi/stream_view.opi

An image showing what the CS Studio OPI looks like is included in the following slide.



N.B. The UUTs shown in this image are not the UUTs that the user will be using. Adjust the UUT names accordingly.

Isolating CPUs

For performance reasons it makes sense to isolate a CPU (or more than one) to handle the control program. This means that the linux scheduler will not be allowed to allocate any other processes to the CPUs that have been isolated. To get a task to run on the isolated CPUs the user must explicitly specify which CPUs the program is allowed to run on either using taskset or sched_set_affinity.

To isolate CPUs the user should edit the grub file. An example grub file is provided on the following slide. Once the file has been edited the user should make a new grub config like so:

```
grub-mkconfg -o /boot/grub/grub.cfg
```

Once this has been done a reboot is required for the changes to be implemented. To check the changes were successful the user can use:

```
[dt100@seil ~]$ cat /sys/devices/system/cpu/isolated

0-1
```

New grub file with CPUs isolated.

```
[dt100@seil ~]$ cat /etc/default/grub
GRUB_SAVEDEFAULT=true
GRUB_DEFAULT=0
GRUB_TIMEOUT=5
GRUB_DISTRIBUTOR="$(sed 's, release .*$,,g' /etc/system-release)"
#GRUB_CMDLINE_LINUX_DEFAULT=""
GRUB_CMDLINE_LINUX="console=ttyS1,115200 console=tty0"
GRUB_CMDLINE_LINUX_DEFAULT="isolcpus=0,1"
isolcpus="0,1"
GRUB_TERMINAL="serial"
GRUB_SERIAL_COMMAND="serial --speed=115200 --unit=0 --word=8 --parity=no --stop=1"


GRUB_INIT_TUNE="480 440 1"
```

Control script

A control script has been created to automate the LLC capture. It is contained in the scripts directory of the AFHBA404 GitHub repository referenced in the previous slide.

To run the script “cd” into the AFHBA404 directory and run the following command:

```
./scripts/acqproc_multi.sh
```

There are a few parameters which can be configured inside the script. These include whether or not to use **MDSplus** (entirely optional) and which UUTs are currently being used. The UUTs being used can be changed from the “ACQPROC/config/swip1.json” file and there are example configuration files in ACQPROC/config/. These files can be copied and the UUT names and channel counts adjusted for any future systems the user may have.

The control script will configure the system **clocks** using the sync_role script. This can also be configured to suit the users needs. By default it is set to configure the first system as a “**master**” and all subsequent systems as slaves over HDMI. This setting can be changed to “**fpmaster**” if the user wishes to use a front panel clock and trigger. The default clock speed is **20kHz** although this can also be customized. The slaves always share the same clock as the master system.

Once the system is configured for capture the control program is started. After the control program has been started the system is armed and triggered. The default capture length is **400k samples** and this is also configurable.

Detailed explanation of config file

The configuration file (JSON format) contains all of the information the control program and the control scripts need to configure all of the necessary UUTs. All of the UUTs in the config file will be configured by the control script and the control program will then capture from the specified UUTs. This makes the system entirely data driven and there is now no need to change and then recompile the control program every time the system configuration changes. The system will now intelligently be configured entirely based on the users specification. It will save any user a lot of time and effort switching to the new ACQPROC scheme.

The default configuration file can be changed by setting the following parameter on the command line:

export ACQPROC_CONFIG=./ACQPROC/configs/acq2106_423_4.json

This will change the configuration file used by the control program and control scripts to the one specified by the user. The user can create custom configuration files to drive the system in the way they want. An example configuration file is shown on the following page.

Please note that the default configuration is swip1.json and this **will** need to be changed to another more relevant configuration file using the commands shown above. For more information on ACQPROC it is useful to read and refer to the ACQPROC README on github, which can be accessed from here:

https://github.com/D-TACQ/AFHBA404/blob/master/ACQPROC-README.md

```
{
    "AFHBA": {                              # ONE AFHBA404 HOST CARD
        "DEVNUM": 0,                         # /dev/rtm-t.0
        "UUT": [
            {                               # UUT[0]
                "name": "acq2106_241",      # HOSTNAME: look up ip address in /etc/hosts/
                "type": "pcs",              # pcs style
                "VI": {                     # VI : Input Vector
                    "AI16": 32,               # 32 AI, 16 bit
                    "SP32": 16                # 16 SP32, Status longwords
                }
            },
            {                               # UUT[1]
                "name": "acq2106_243",      # HOSTNAME: look up ip address in /etc/hosts/
                "type": "pcs",              # pcs style
                "VI": {                     # VI : Input Vector
                    "AI16": 32,               # 32 AI, 16 bit
                    "SP32": 16                # 16 SP32, Status longwords
                }
            },
            {                               # UUT[2]
                "name": "acq2106_239",      # HOSTNAME: look up ip address in /etc/hosts/
                "type": "pcs",              # pcs style
                "VI": {                     # VI : Input Vector
                    "AI16": 64,               # 64 AI, 16 bit
                    "SP32": 16                # 16 SP32, Status longwords
                }
            },
            {                               # UUT[3]
                "name": "acq2106_240",      # HOSTNAME: look up ip address in /etc/hosts/
                "type": "pcs",              # pcs style
                "VI": {                     # VI : Input Vector
                    "AI16": 64,               # 64 AI, 16 bit
                    "SP32": 16                # 16 SP32, Status longwords
                }
            }
        ]
    }
}
```

Explanation of the control scripts

There are two control scripts that are used to configure the systems for LLC capture. The first is llc-config-utility.py which configures the aggregator and distributor onboard the FPGA on all of the systems.

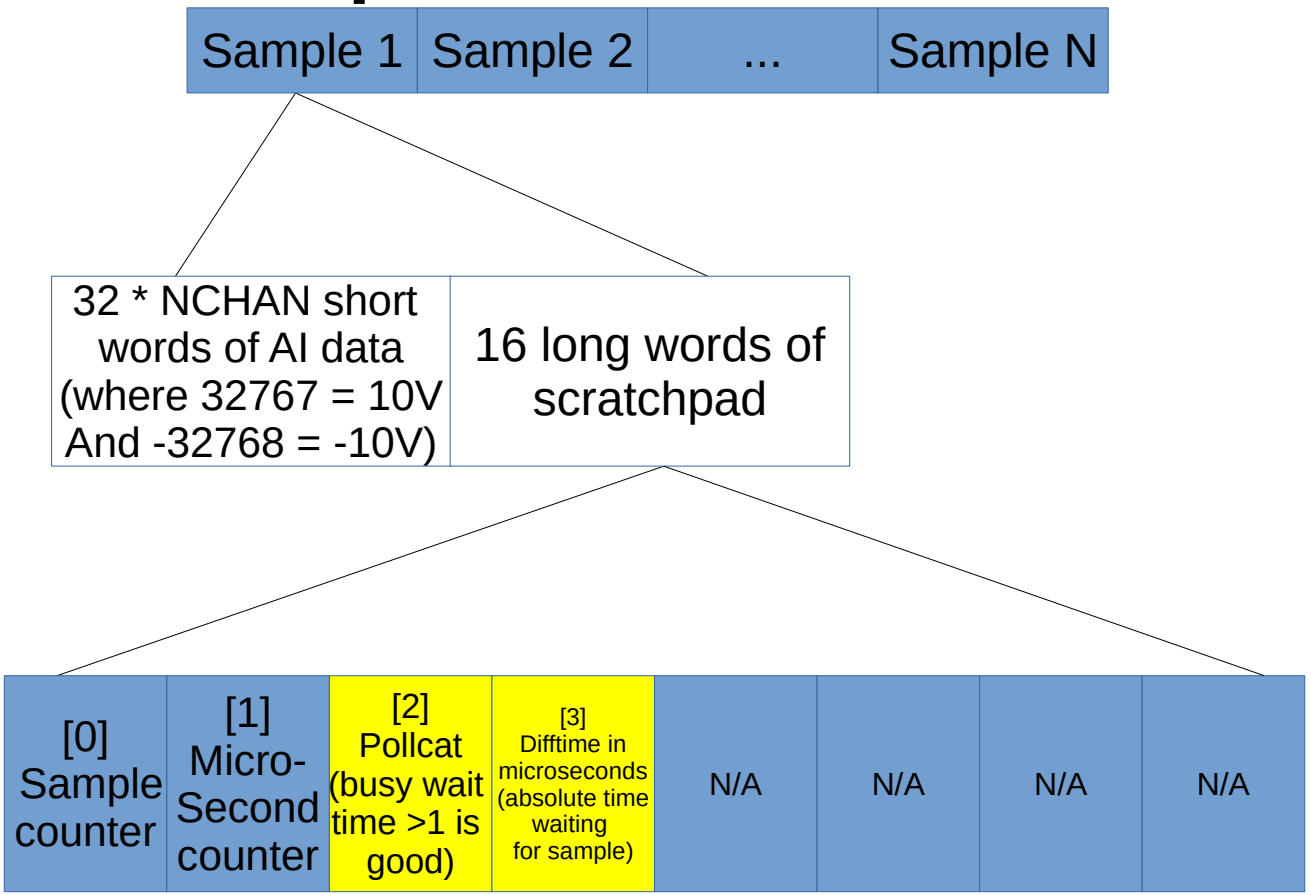
Then the clocks are set by sync_role.py. The clocks can be configured by the user (the clocks should not be set faster than ~500kHz). The first acq2106 is known as the master and is configured as “master” by default. It can also be configured as “fpmaster” to use the front panel clock and trigger instead.

Output of the control script

The control script will display a histogram of the T_LATCH values, showing how many samples were missed by the host computer (the T_LATCH is the sample counter, so the difference between any two consecutive samples should be one). Ideally there should be no samples missed. There is also some textual output of the T_LATCH histogram data.

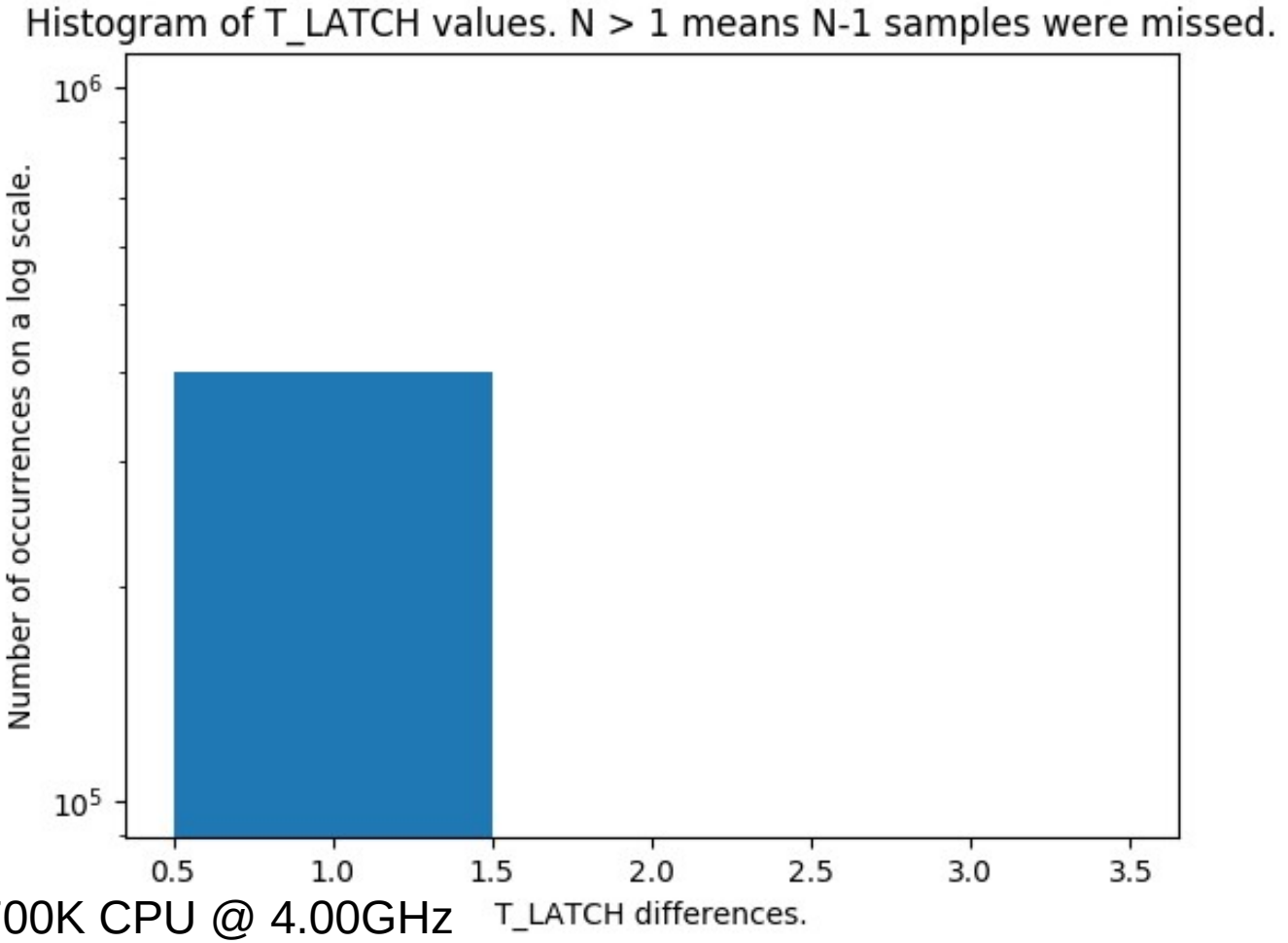
Usually the control script will configure the systems to report latencies in the scratchpad. However, systems that do not contain an AO module WILL NOT report these latencies. D-TACQ prepared a histogram of the latencies by including an AO module for the user to view, but this will not be possible for the user. As such, the user will not receive latency data in the scratchpad for acq423 systems that do not have an AO module.

Sample construction



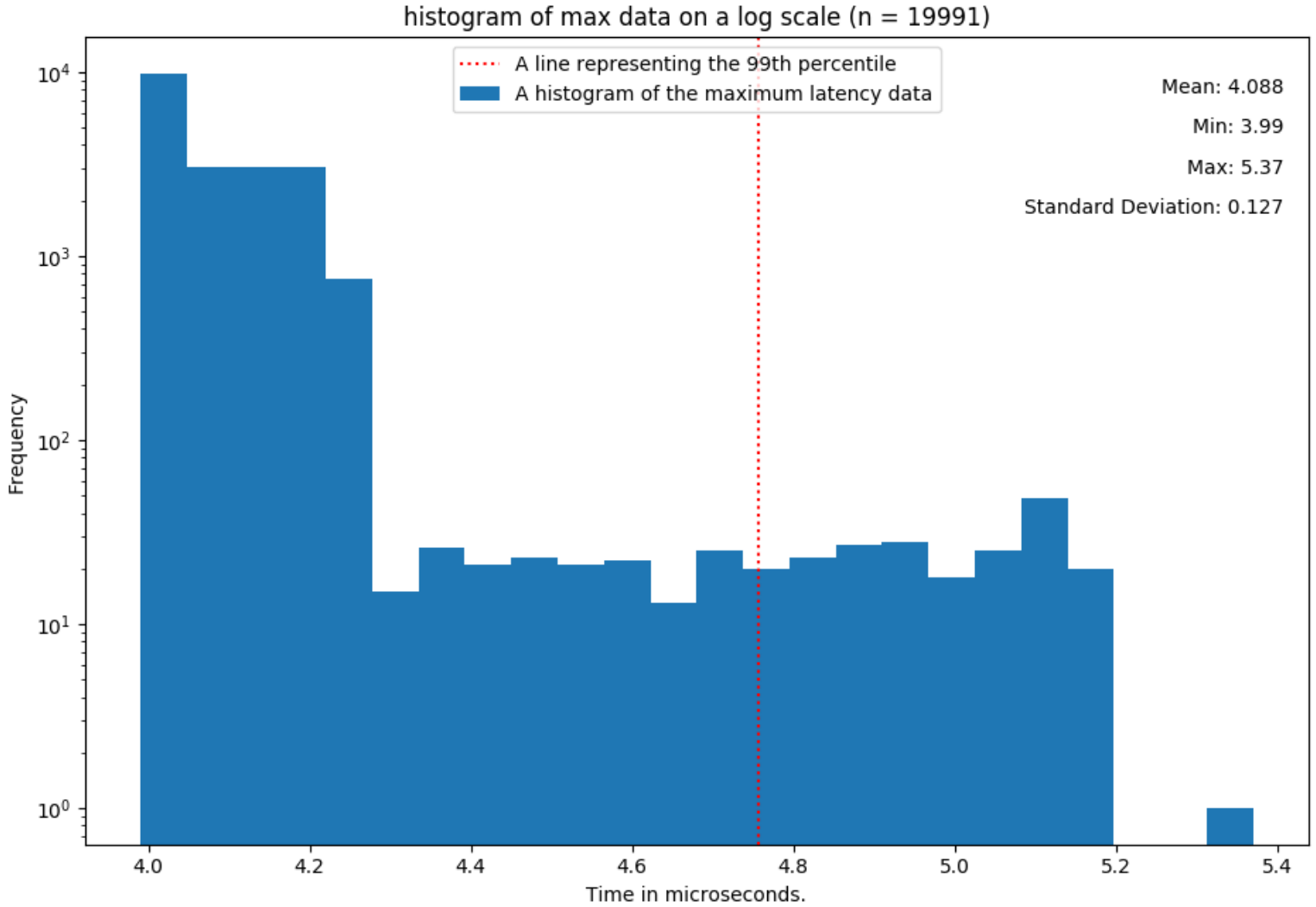
Note: In the scratchpad blue fields are generated on the acq2106 and yellow fields are inserted by the control program

Histogram of the sample counter (T_LATCH) on an ideal run.



N.B. With an i7-6700K CPU @ 4.00GHz there were no recordings of missed samples in the T_LATCH in any of the 4 UUTs sampling at 20kHz.

Histogram of the FPGA maximum latency register.



N.B. Latency registers are not available to users on systems not containing an AO module.

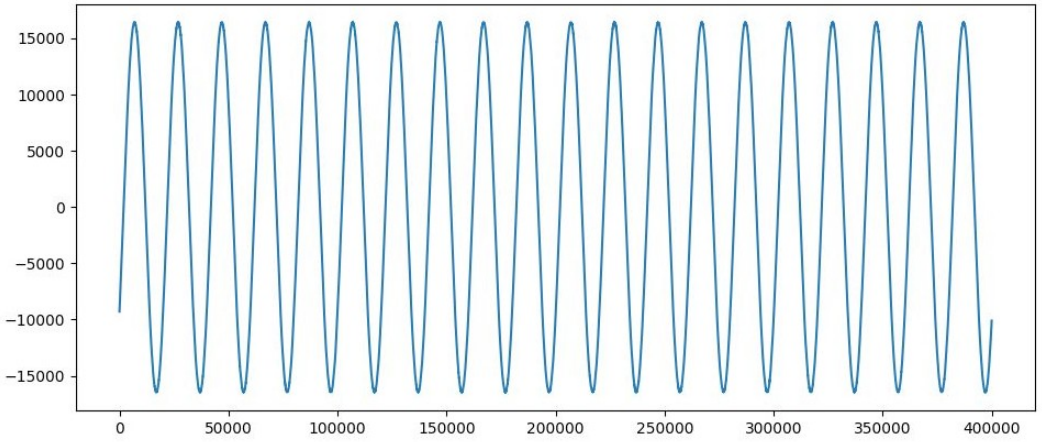
Plotting data from the first UUT

The files saved by previous control programs were not named after the UUT but by their position on the AFHBA404 card. ACQPROC now uses the JSON file provided to output files named after their respective UUTs.

The command to view any specific channel(s) use a python script called host_demux.py which can be found in the user_apps/analysis/ directory of acq400_hapi. An example command is as such:

```
./host_demux.py --src=/home/dt100/PROJECTS/AFHBA404/acq2106_239_VI.dat --nchan=96 \       --
    pchan=1 --data_type=16 --plot_mpl=1 --mpl_subrate=1 acq2106_239
```

This is just an example command. The channel to plot can be changed to any channel the user wishes (or more than one channel by providing --pchan=1,2,3 for example). The output from the above command is included below.



Analysing the latency of the data

While the latency cannot be measured easily on an acq423 system with no AO module, the following documentation is still recommended for the user. The D-TACQ low latency white paper is available here:

http://www.d-tacq.com/resources/LLC_White_Paper.pdf

And the D-TACQ LLC system latency measurement guide, available here:

LLC-system-latency-measurement-guide.pdf

The following page contains a scope trace showing the latency of the system.

Saving channelised data

We can use a first order taylor series expansion to accurately approximate the difference in phase between the systems. If we sample the same sine wave from a signal generator we can compare the signals using python. There are a number of steps to complete.

1. Make a local copy of each channel using host_demux.py. The host_demux.py script was used earlier in the guide to view the data. It can also be used to save the data. The following terminal output text shows how it was used to save data to the AFHBA404 directory.

```
[dt100@seil AFHBA404]$ pwd

/home/dt100/PROJECTS/AFHBA404

[dt100@seil AFHBA404]$ ../acq400_hapi/user_apps/analysis/host_demux.py --nchan=64 --src="./acq2106_241_VI.dat" --
    data_type=16 --plot_mpl=1 --mpl_subrate=1  --pchan="1" --save=acq2106_241 acq2106_241
```

The above pwd shows where the commands are being run from. We call into acq400_hapi user_apps to use host_demux.py. The --save option is specified to be the name of the UUT. This is repeated for each UUT that the user wishes to analyse. Doing this for all of the UUTs results in the following directory structure in side AFHBA404:

```
[dt100@seil AFHBA404]$ ls -ld *acq*/

drwxrwxr-x. 2 dt100 dt100 4096 Jul  2 15:17 acq2106_241/

drwxrwxr-x. 2 dt100 dt100 4096 Jul  2 15:17 acq2106_243/

drwxrwxr-x. 2 dt100 dt100 4096 Jul  2 15:18 acq2106_245/

drwxrwxr-x. 2 dt100 dt100 4096 Jul  2 15:18 acq2106_246/
```

Phase analysis of saved data

Once the data has been saved to disk using the steps outlined in the previous slide the user can then begin to analyse the data. The following is the commands used from inside the AFHBA404 directory to verify the phase. For a greater in depth explanation of the algorithm used please consult the phase_delay.py python script. In the output below some of the text has been removed (the radians and degrees reports) due to space constraints.

```
[dt100@seil AFHBA404]$ ../acq400_hapi/test_apps/phase_delay.py
    --file1=./acq2106_241/acq2106_241_01.dat --file2=./acq2106_243/acq2106_243_01.dat --
    fsig=2 --s_clk=20000 --type=16

Difference in seconds is  3.6e-06

Difference as % of sample clock  7.2

[dt100@seil AFHBA404]$

[dt100@seil AFHBA404]$ ../acq400_hapi/test_apps/phase_delay.py
    --file1=./acq2106_241/acq2106_241_01.dat --file2=./acq2106_245/acq2106_245_01.dat --
    fsig=2 --s_clk=20000 --type=16

Difference in seconds is  5.4e-06

Difference as % of sample clock  10.8

[dt100@seil AFHBA404]$

[dt100@seil AFHBA404]$ ../acq400_hapi/test_apps/phase_delay.py
    --file1=./acq2106_241/acq2106_241_01.dat --file2=./acq2106_246/acq2106_246_01.dat --
    fsig=2 --s_clk=20000 --type=16

Difference in seconds is  2.6e-06

Difference as % of sample clock  5.33637991476
```

Repeating ACQPROC: 4 UUTs each with 4 acq423 cards

To highlight how easy it is to work with different UUT types and stacks ACQPROC will be used again with a different UUT stack configuration and a corresponding different configuration file. The new file is also on GitHub and is called acq2106_423_4x4.json. The line used to run this ACQPROC instance was as follows:

ACQPROC_CONFIG=./ACQPROC/configs/acq2106_423_4x4.json VERBOSE=0 ./scripts/acqproc_multi.sh

The configuration file is loaded and is used to set up all the clocks as before and then ACQPROC uses the information to set up an LLC transfer.

Phase analysis of the 4 UUT stack data.

If the steps outlined in previous slides are followed for these UUTs (using host_demux.py to save the data and then using phase_delay.py to analyse the phase) the following information can be obtained.

UUT (compared to UUT 1)	Phase difference (% of sample clock)	Phase difference (microseconds)
2	7.2	4.6
3	11	5.7
4	5.9	3.0

Repeating ACQPROC: 4 UUTs each with 3 acq423 cards

For the specific case of four acq2106 systems in a stack, each with 3 acq423 cards the following configuration can be used. Again, all that needs to be changed is the configuration file. The command used to run this ACQPROC instance was as follows:

ACQPROC_CONFIG=./ACQPROC/configs/acq2106_423_4x3.json VERBOSE=0 ./scripts/acqproc_multi.sh

If the steps outlined in previous slides are followed for these UUTs (using host_demux.py to save the data and then using phase_delay.py to analyse the phase) the following information can be obtained.

UUT (compared to UUT 1)	Phase difference (% of sample clock)	Phase difference (microseconds)
2	7.3	3.7
3	11	5.7
4	5.8	2.9

Running the same test on 6 UUTs

To show the flexibility and ease of use of the ACQPROC system we will now move to 6 UUTs. To do this a new configuration file is created with 6 UUTs. This file is called acq2106_423_6.json and is available in the configs directory in ACQPROC. Once the configuration file has been changed run acqproc_multi.sh again with

export ACQPROC_CONFIG=./ACQPROC/configs/acq2106_423_6.json

This is all the user needs to change to run with the extra UUTs in the loop. Note that a single AFHBA404 card will only take 4 UUTs, so if the user wishes to expand upon 4 UUTs then they must fit 2 AFHBA404 cards in the one host PC. All other steps are the same.

Phase analysis for 6 UUTs

Repeating the steps outlined for phase analysis earlier in this document for 6 UUTs the following results were obtained:

UUT (compared to UUT 1)	Phase difference (% of sample clock)	Phase difference (microseconds)
2	7.3	3.6
3	10.8	5.4
4	5.7	2.8
5	6.2	3.1
6	6.5	3.2

Using ACQPROC with a PCS system and remote AO

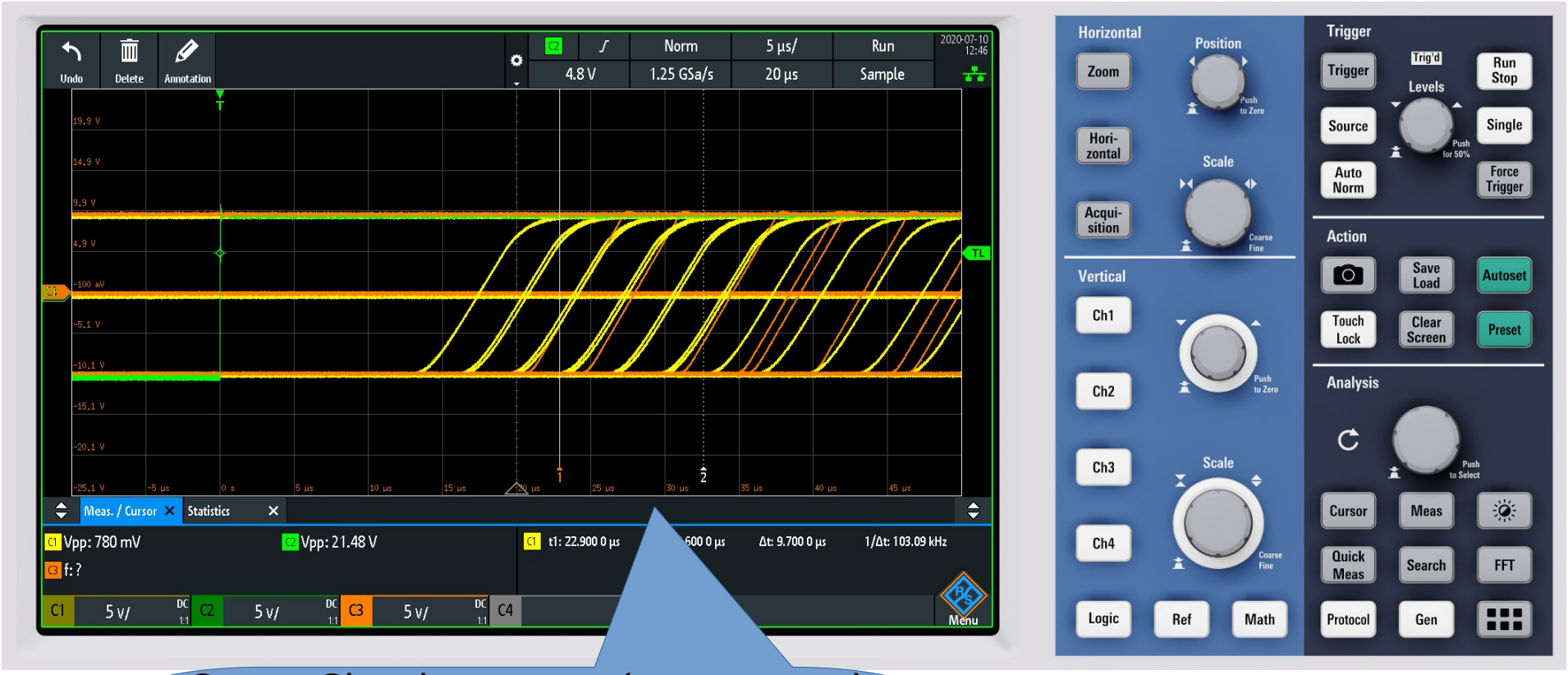
To again demonstrate the flexibility of ACQPROC we can use an acq2106 with 4*acq424 cards, an ao424 and a dio432 paired with another acq2106 with an ao420. To do this use the acq424_remote_ao.json configuration file which can be found in the ACQPROC/configs/ directory. To use the config the user can export the environment variable ACQPROC_CONFIG as shown below:

export ACQPROC_CONFIG=./ACQPROC/configs/acq424_remote_ao.json

Once this has been exported the user can then run ACQPROC on their system and capture data from the systems. Below is a table of phase relationships between the first channel, the ao424 looped back to an acq424 and the ao420 looped back to an acq424. Note that this is a very different number from the previously included phase relationships. Loopbacked AOs have a larger phase difference due to having to be routed through the AFHBA404 and resampled by the AI cards.

AO module (looped back to AI module)	Phase difference (% of sample clock)	Phase difference (microseconds)
ao424	89	44
ao420	98	49

Example of latencies with an acq424 LLC system paired with a remote AO system.



Green: Signal generator (square wave)
Yellow: First AO output
Orange: Second AO output
Round trip latency of approx. 12us.

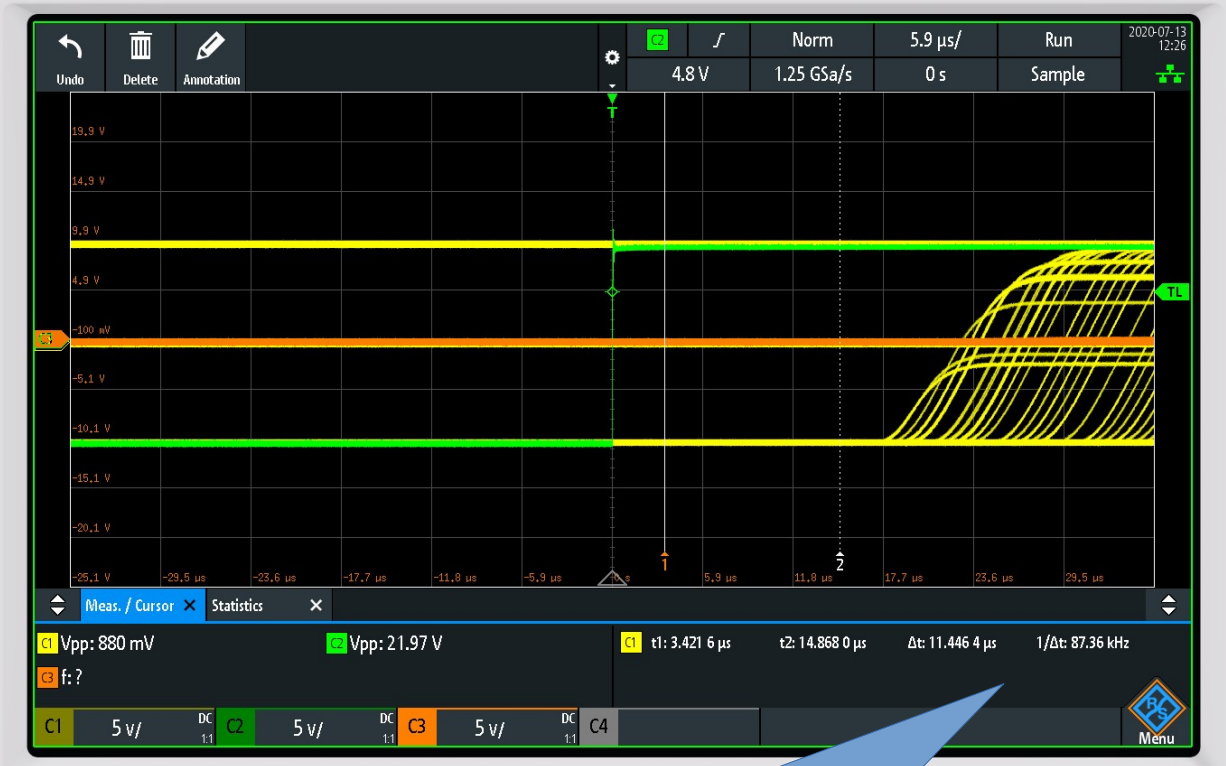
A 7 UUT mixed AI AO system

A 7 UUT mixed acq423, acq424, ao424 and dio432 system was tested to verify that ACQPROC can handle many configurations. This is provided that the correct configuration json file format is used. In order to test the configuration the following configuration file was used:

7_uut_system.json

The phase difference was reduced to 67% of a sample clock or 33 microseconds.

7 UUT system scope trace showing latency



AO424 output in yellow showing a latency of approx. 18 microseconds in an LLC configuration looped back to an acq423

Using hexdump to view the data

It can be useful to hexdump the data to check its composition. The following command may be changed to check different columns of long word data (scratchpad data for instance).

```
hexdump -e '48/4 "%08x," "\n"' acq2106_239_VI.dat | cut -d, -f1,4,8,49-55 |
   head
```

The following command will work for viewing short word data (acq424 channels for instance).

```
hexdump -e '96/2 "%04x," "\n"' acq2106_239_VI.dat | cut -d, -f1-10 | more
```

Debug

If something doesn't work in acqproc_multi.sh then there are a few steps to take to make sure things are configured correctly. The acqproc_multi script assumes that the system configuration is identical to that in the diagram on page 5. It is worth checking that this is the case. HDMI connections and SFP connections are crucial to have wired correctly.

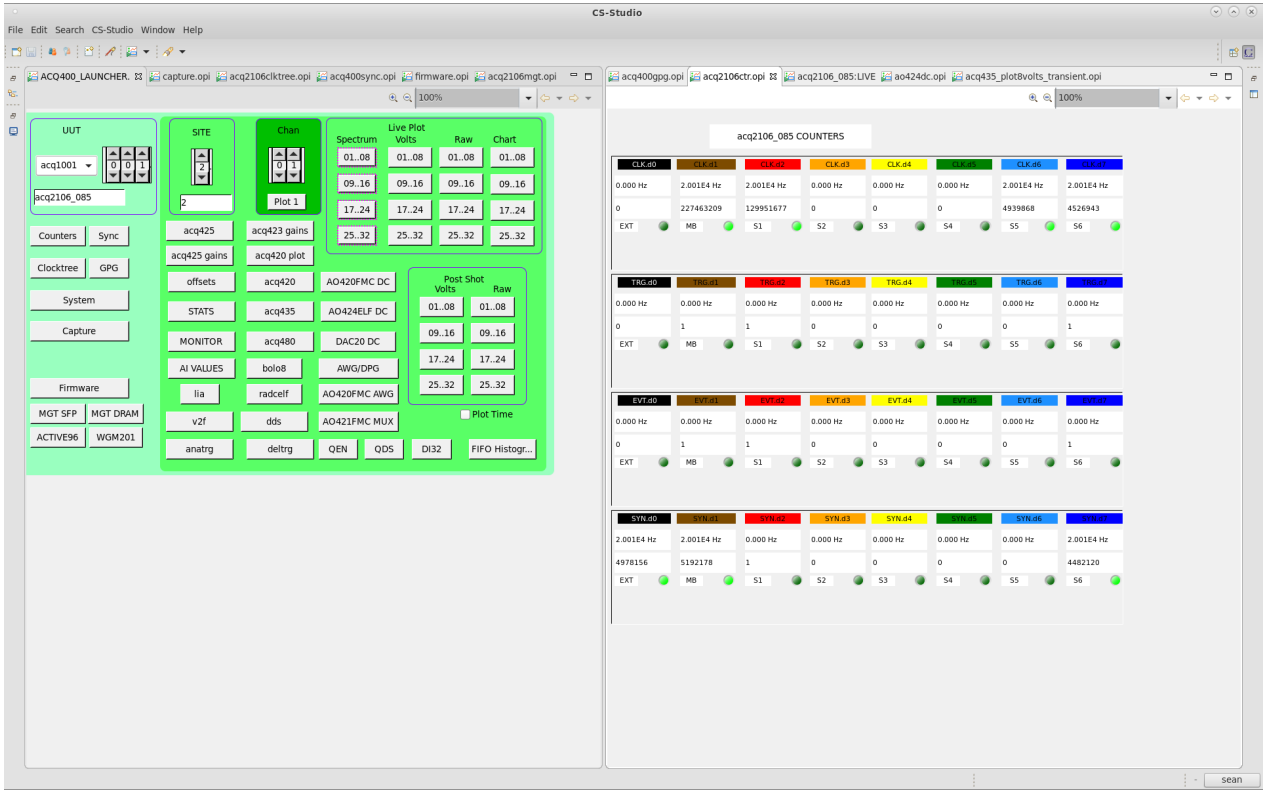
If the system is configured exactly as shown on page 5, then the user should check the clocks and triggers on each UUT individually. The best method of doing this is using CS Studio. Instructions for installing and using CS Studio can be found here:

https://github.com/D-TACQ/ACQ400CSS

Once CS Studio is installed the user should check for each UUT that the clocks and triggers are accounted for.

CS Studio counters page

The CS Studio counters page should look very similar to the image below after running a capture using acqproc_multi.sh. It contains the information for the clocks on each site and information about the triggers. In this case the trigger is set to soft, so we get 1 soft trigger in the d1 trigger counter box, and a corresponding trigger in the d2 trigger box. The user should check this information for each UUT.

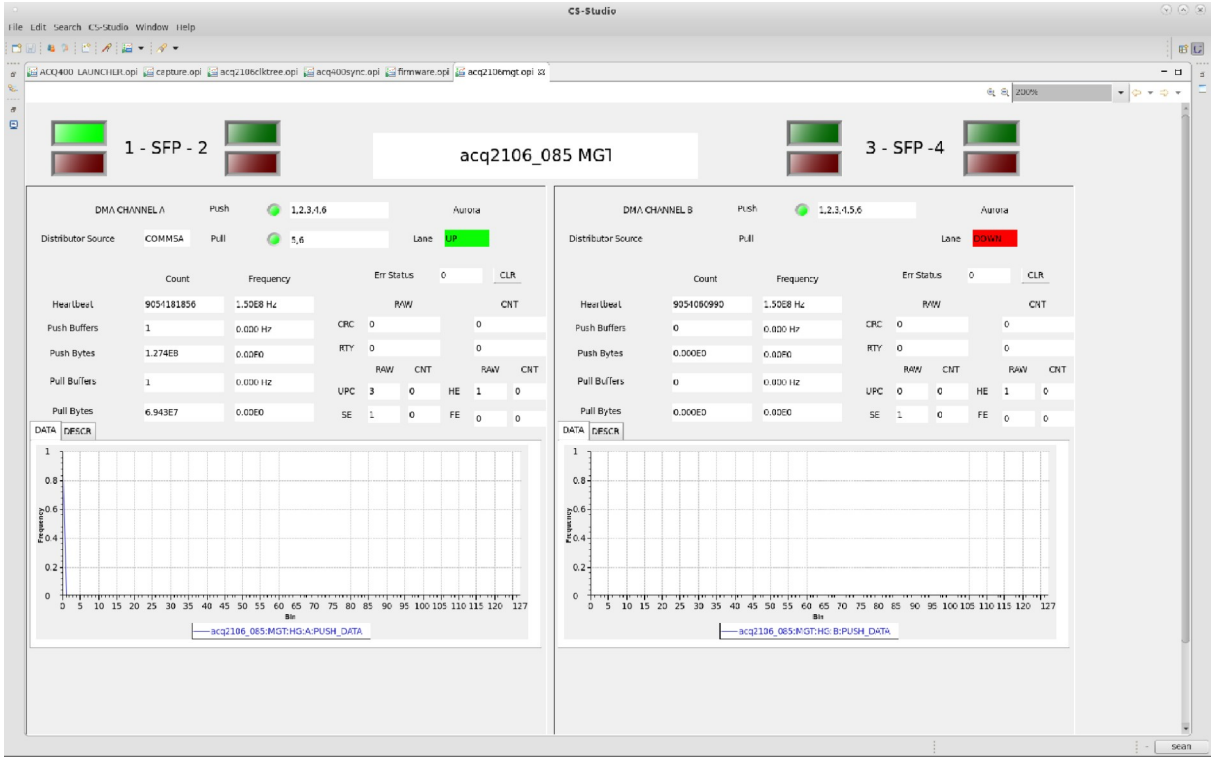


Check UUTs are streaming

To check that the UUTs are streaming data the user should observe the stream tab of the capture OPI as shown below. During a capture the sample count should be ticking up. The clock speed should also be visible in the box next to it. The rate will also be visible in the rate box. Again, this should be verified for all UUTs.

Check CS Studio MGT-SFP page

The MGT-SFP page is useful for checking whether data has been sent to the host and if data has been returned from the host. After running one capture the page should look similar to the image below. There should be 1 push buffer and 1 pull buffer with a non zero value in each (how large depends on how much data has been streamed). Check all the UUTs are the same after rebooting and running a single capture.



Checking which sites are enabled

If there is data going to the host (as shown in the previous page), but the system still isn't working, then check that the correct sites are included in the aggregator. This can also be seen in the image in the previous slide in the 'push' and 'pull' boxes in the MGT-SFP OPI.