

3 * ACQ2106 + 4xACQ424 + AO424 +
DIO432 in low latency control mode to a
single host using ACQPROC.

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

Minimum required release:
acq400-131-20191107115916

Recommended:
RELEASE acq400-166-20200306104841 (or newer)

Revision of document.

Date	Change
17.03.2019	Init
19.03.2019	pgm: add acqproc doc

Low Latency Control Concept

- Low Latency Control LLC :
 - used in PCS over 4 generations D-TACQ HW
- In 4G, the ACQ2106 DAQ appliance (UUT) controls data flow to a HOST PC via AFHBA404, PCIe, DMA.
 - Input : On the Clock, all INPUTS are sent as a single Vector In (VI) to HOST MEMORY
 - Output: On the Clock, all OUTPUT demand values are collected from HOST memory as a single Vector Out (VO).
- A single AFHBA404 can host up to 4 UUTS
- We're using a new configuration file driven software core "ACQPROC". please see
 - <https://github.com/D-TACQ/AFHBA404/blob/master/ACQPROC-README.md>
- Control Script:
 - scripted configuration and control of the UUTS by Ethernet, calling
- Control Program
 - This is the ACQPROC core that contains the REALTIME data handling element.
 - Note that our implementation of ACQPROC is just a stub, we expect users to add their own data sharing, computation, perhaps by subclassing from ACQPROC.SystemInterface
- Post shot data store and analysis
 - Additional scripts to manage the data postshot.

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
```

N.B. Make sure that you use at least:

<https://github.com/D-TACQ/AFHBA404/releases/tag/1.4.1>

Or higher.

Installing acq400_hapi

We assume a USER name “dt100”

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/
```

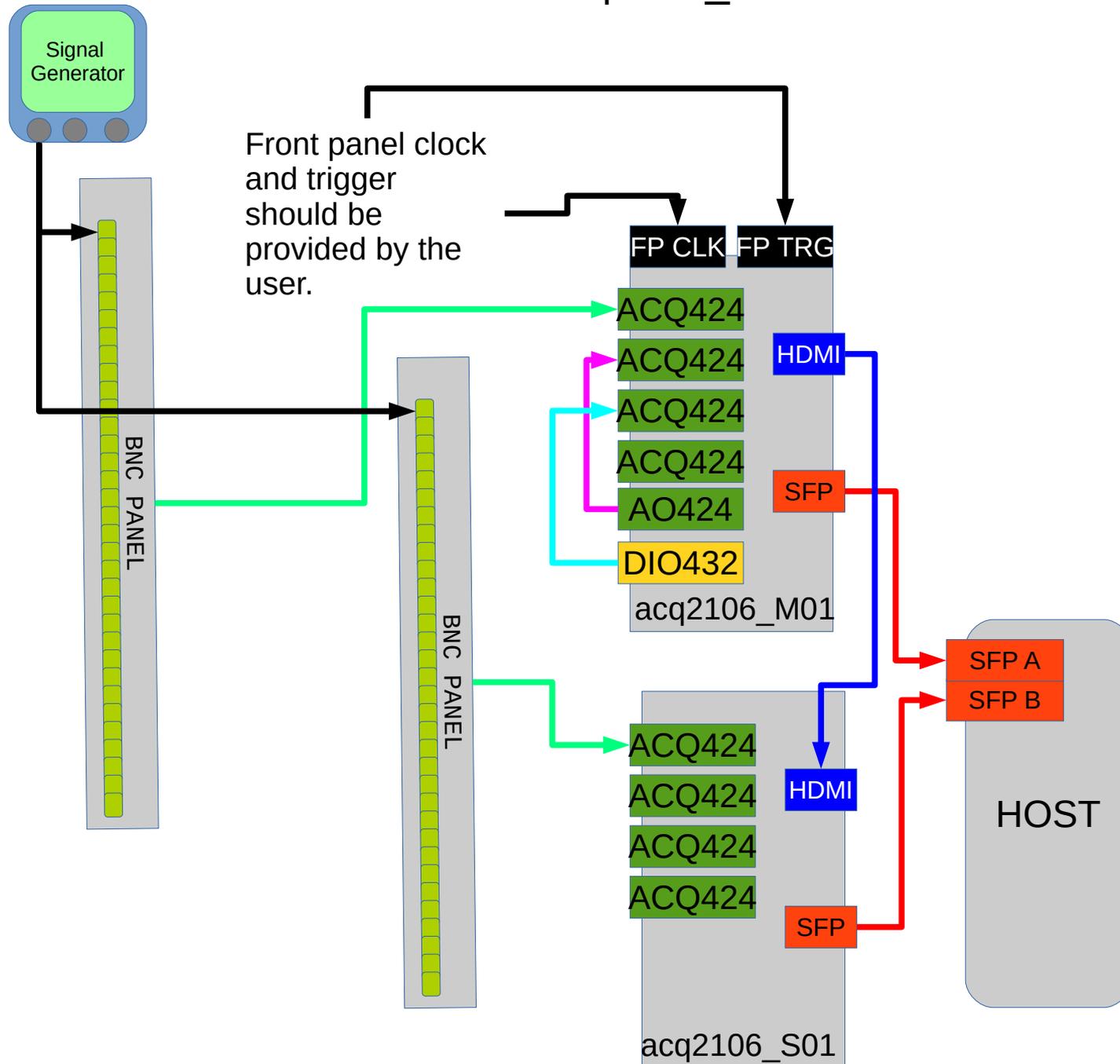
It can be cloned using the following command:

```
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



Set up CS Studio

Configure CS Studio to monitor the UUTs. For only two UUTs it is easier to use two separate workspaces, one for each UUT. Follow the instructions here to install CS Studio and configure the workspaces for the UUTs:

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

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-mkconfig -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_SAVEDDEFAULT=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. If the user wishes to use MDSplus the local MDSplus server must also be specified.

The control script will configure the system **clocks** using the sync_role script. This can also be changed. 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 changed. 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.

Editing the JSON configuration file

The control program is data driven using a JSON configuration file.

- This is defined at the top of `acqproc_multi.sh`
- `ACQPROC_CONFIG=${ACQPROC_CONFIG:-./ACQPROC/configs/swip1.json}`
- For alternative configurations, please review content of `./ACQPROC/configs` for examples, and it's possible to select a new config with
 - `export ACQPROC_CONFIG=./ACQPROC/configs/custom_config`
- Configuration points in `acqproc_multi.sh`
 - UUT1, UUT2, UUTS are created automatically from `ACQPROC_CONFIG`, we assume that the site has working DNS (or at least entry for each UUTx in `//etc/hosts`)
 - Hardcoded `PYTHONPATH=/home/dt100/PROJECTS/acq400_hapi`

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 $\sim 500\text{kHz}$).

Output of the control script

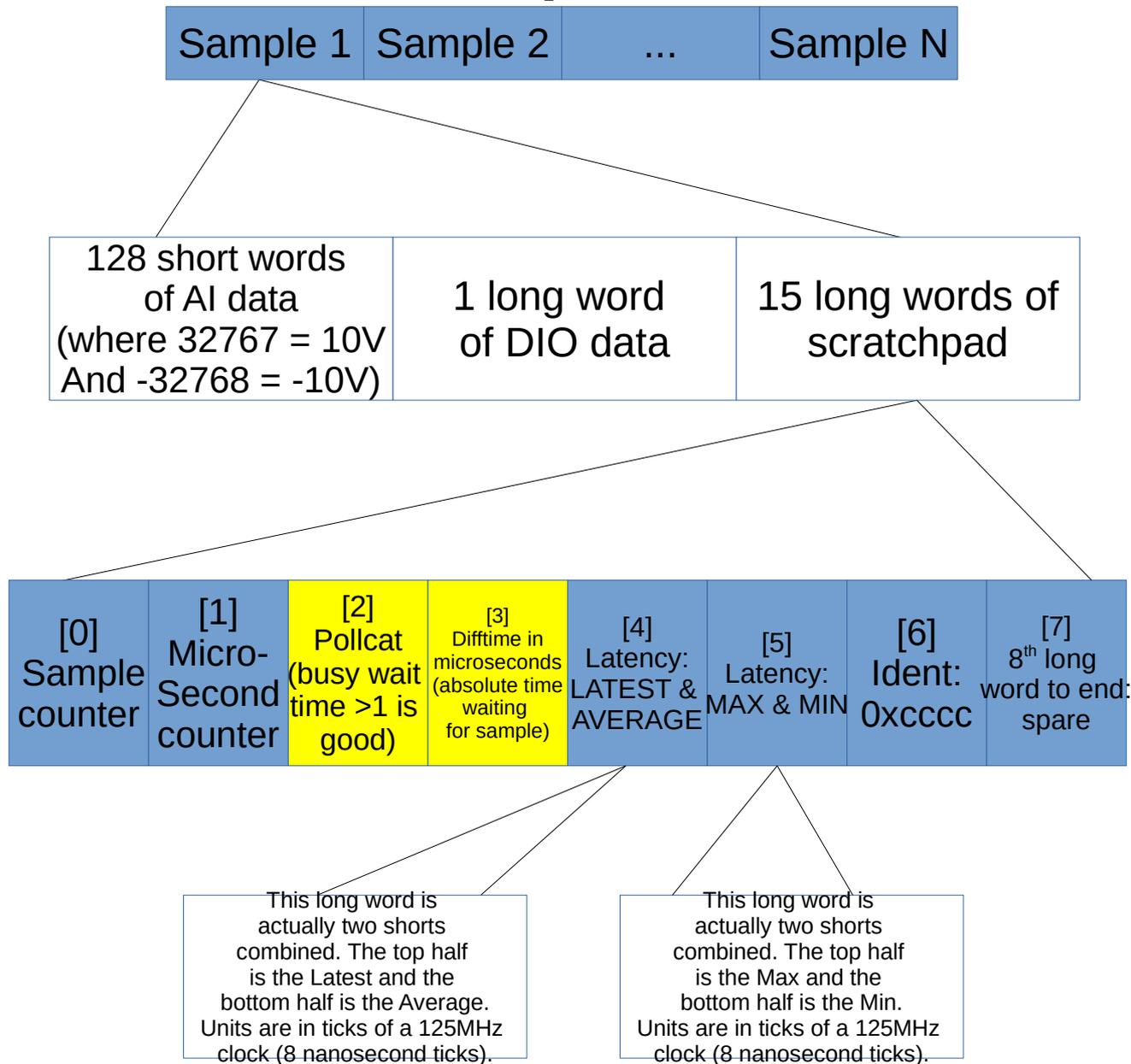
The control script will display a histogram of the T_LATCH values, showing how many samples were missed by the host computer (the T_LATCH is the sample counter, so the difference between any two consecutive samples should be one). Ideally there should be no samples missed. There is also some textual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

The control script will also automatically configure the UUT to calculate some latencies. These latencies are encoded in the scratchpad and will be shown in a histogram, along with some relevant statistics. Please note that the latency registers will not be enabled unless the UUT is running the firmware “RELEASE acq400-125-20190930193608” or above. If your system is not running this release then you can upgrade according to section 29.3 of the the 4GUG available here:

<http://www.d-tacq.com/resources/d-tacq-4G-acq4xx-UserGuide-r28.pdf>

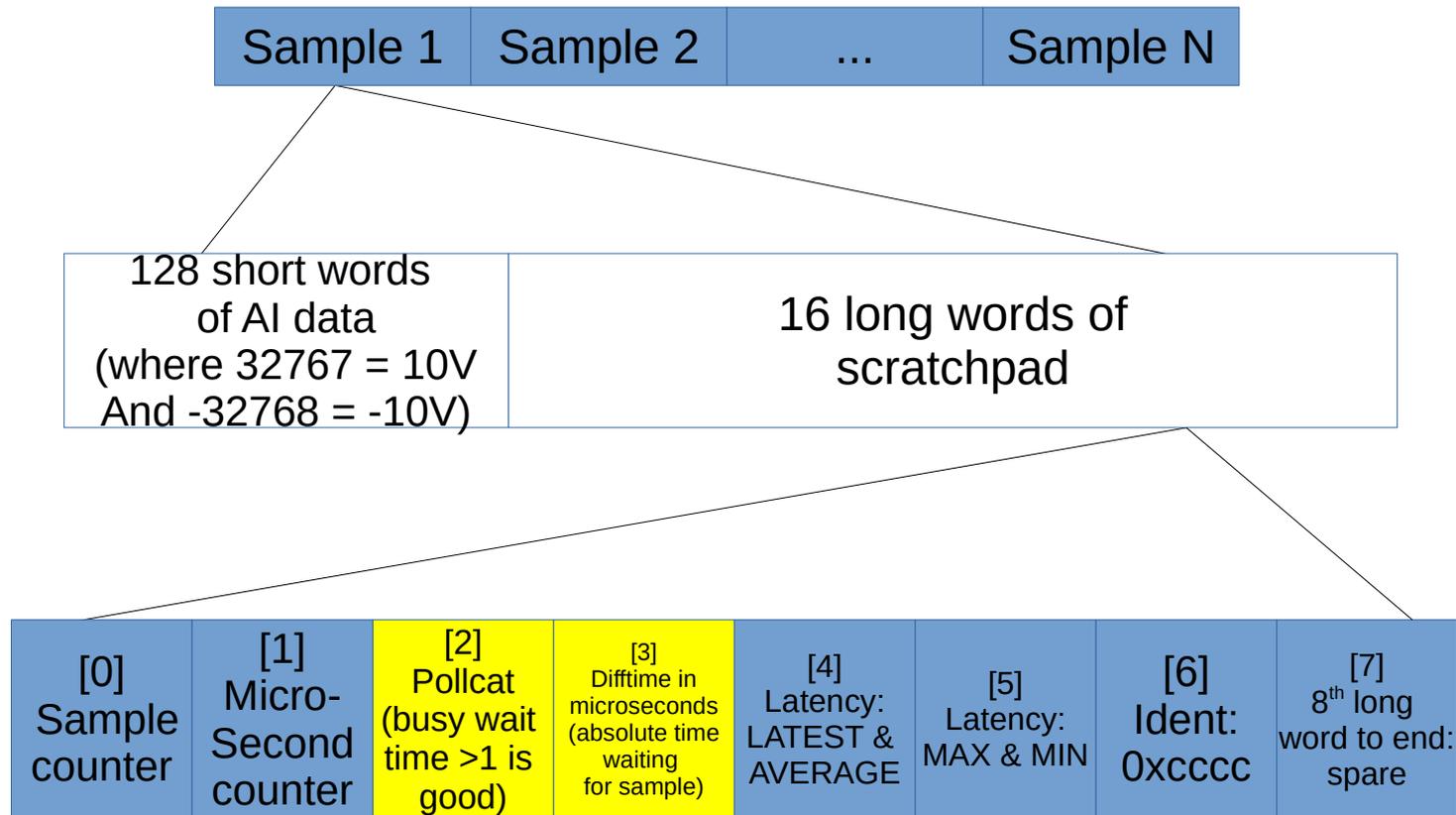
Example histograms are shown in the following pages. Since there are three UUTs in the loop for this test, there will be six histograms shown in total. Once the FAT has been completed these can be turned off inside the script by disabling the analysis.

First UUT sample construction



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

Second UUT sample construction

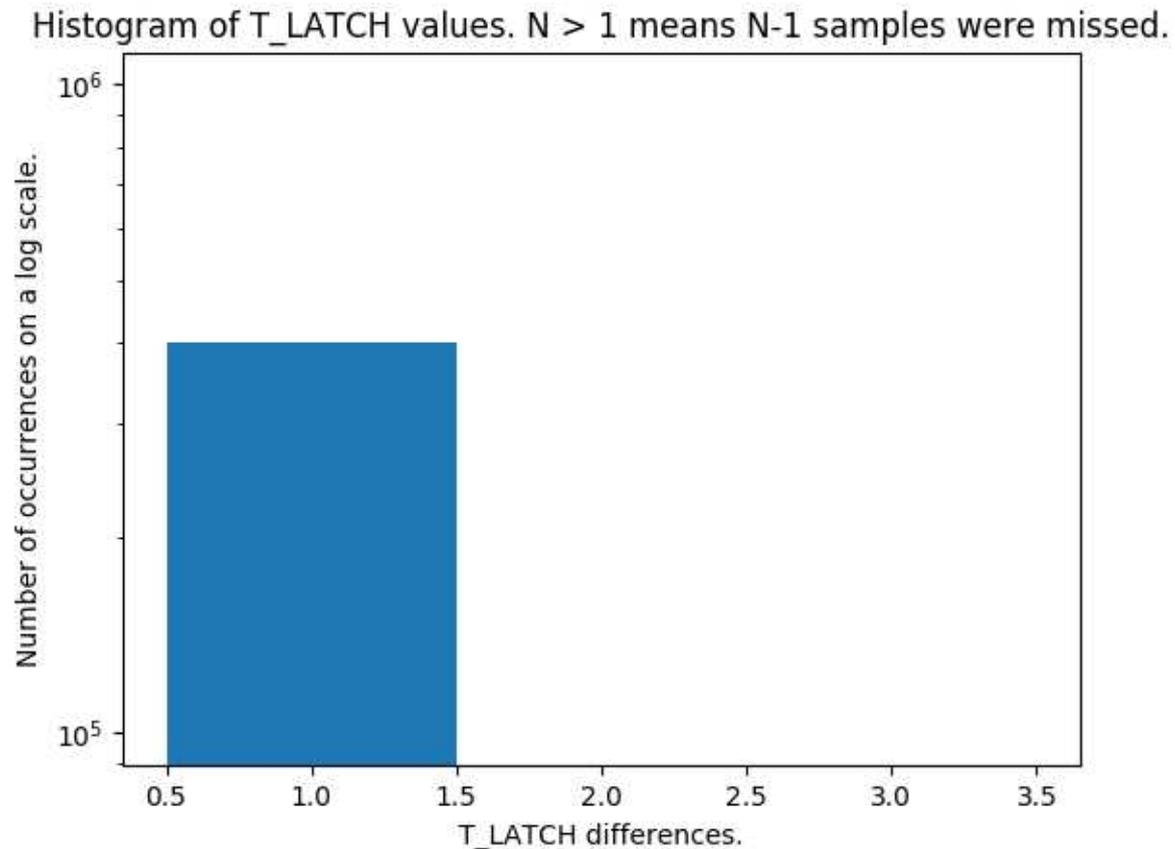


This long word is actually two shorts combined. The top half is the Latest and the bottom half is the Average. Units are in ticks of a 125MHz clock (8 nanosecond ticks).

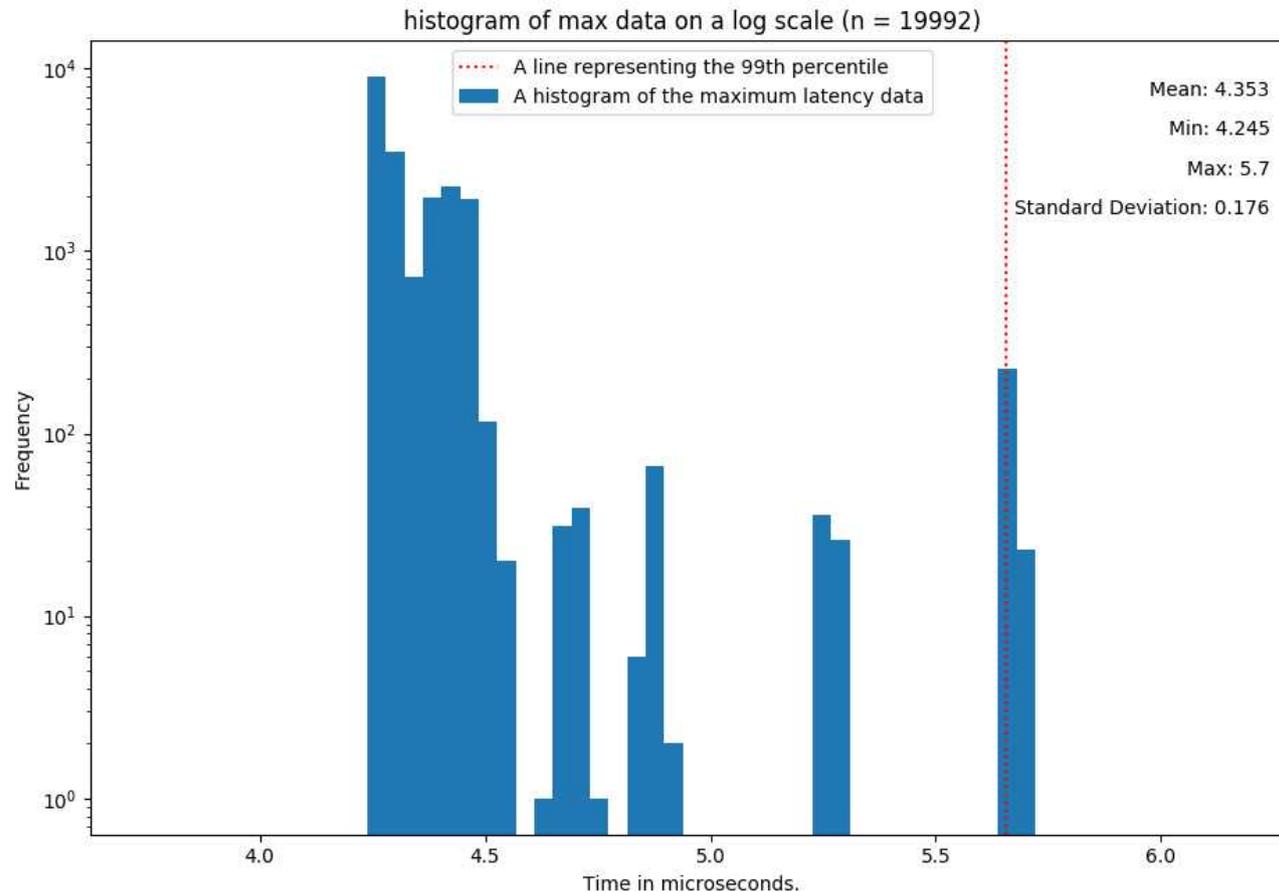
This long word is actually two shorts combined. The top half is the Max and the bottom half is the Min. Units are in ticks of a 125MHz clock (8 nanosecond ticks).

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.



Histogram of the FPGA maximum latency register



Plotting data from the first UUT

Due to the number of channels and number of cards, there are two separate example commands to plot data (one command for each UUT).

The commands use a python script called `host_demux.py` which can be found in the `user_apps/analysis/` directory of `acq400_hapi`. The first command is as such:

```
python ./host_demux.py --src=/home/dt100/PROJECTS/AFHBA404/acq2106_183_VI.dat  
--nchan=160 --pchan=1,33,78 --data_type=16 --plot_mpl=1 --mpl_subrate=1  
acq2106_183
```

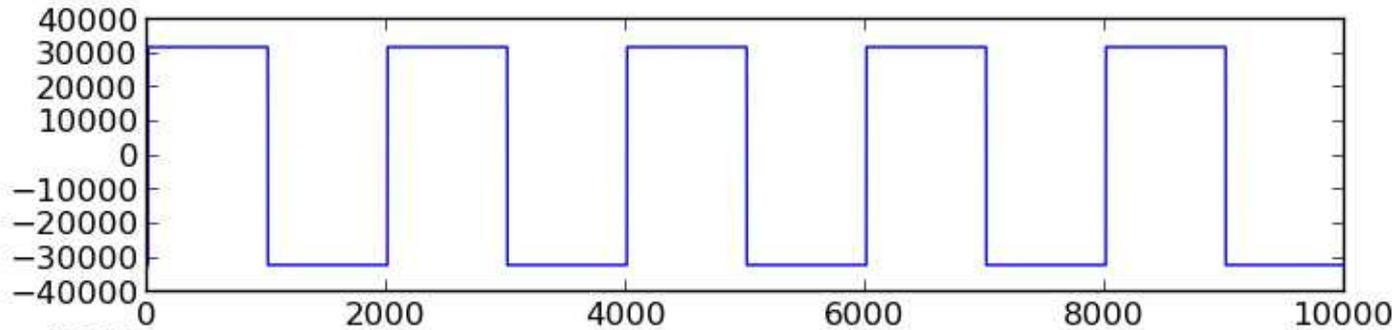
Provided the system has been connected according to the setup diagram on page 4, this will plot:

1. The input channel (CH001),
2. The first AO loopback (CH033),
3. The first DO loopback (any from CH065:CH097, with CH078 chosen for readability),

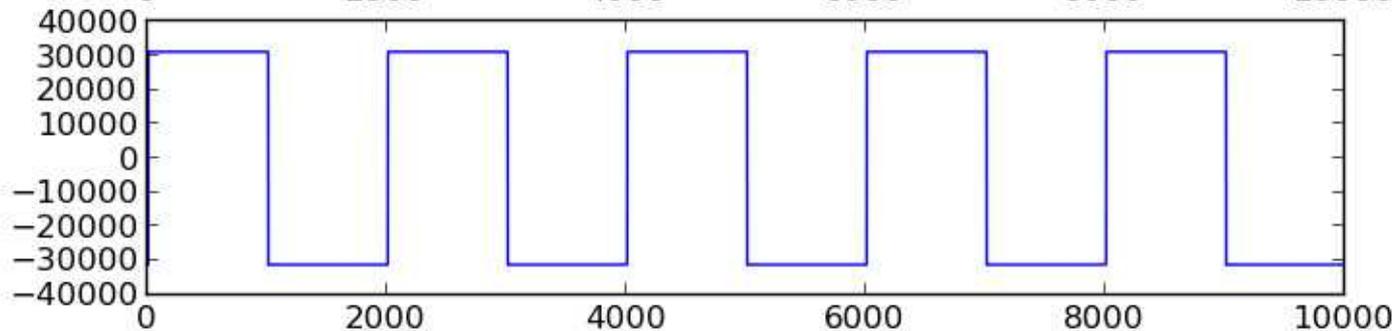
The resultant plot is shown on the next slide.

Data plotted from the first (master) UUT

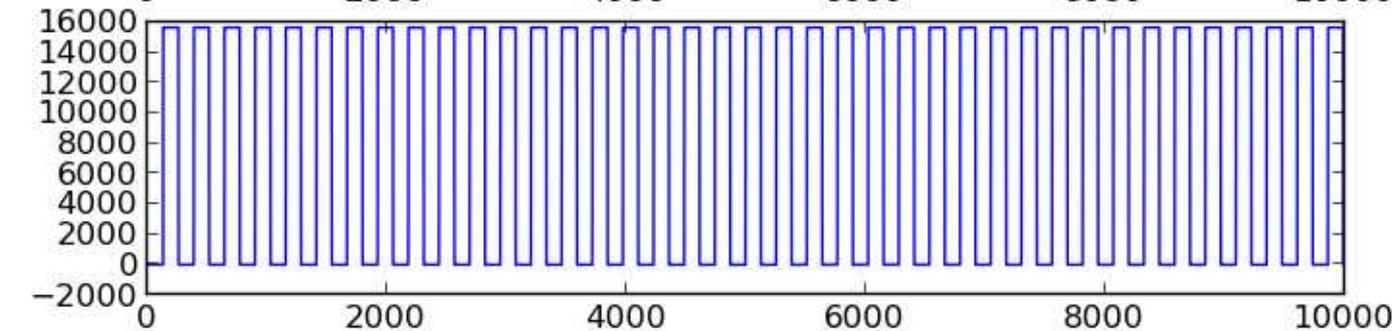
1. Sig gen wave



2. AO1



3. DO1



Note: Capture reduced to 10000 samples for better visibility

Plotting data from the second UUT

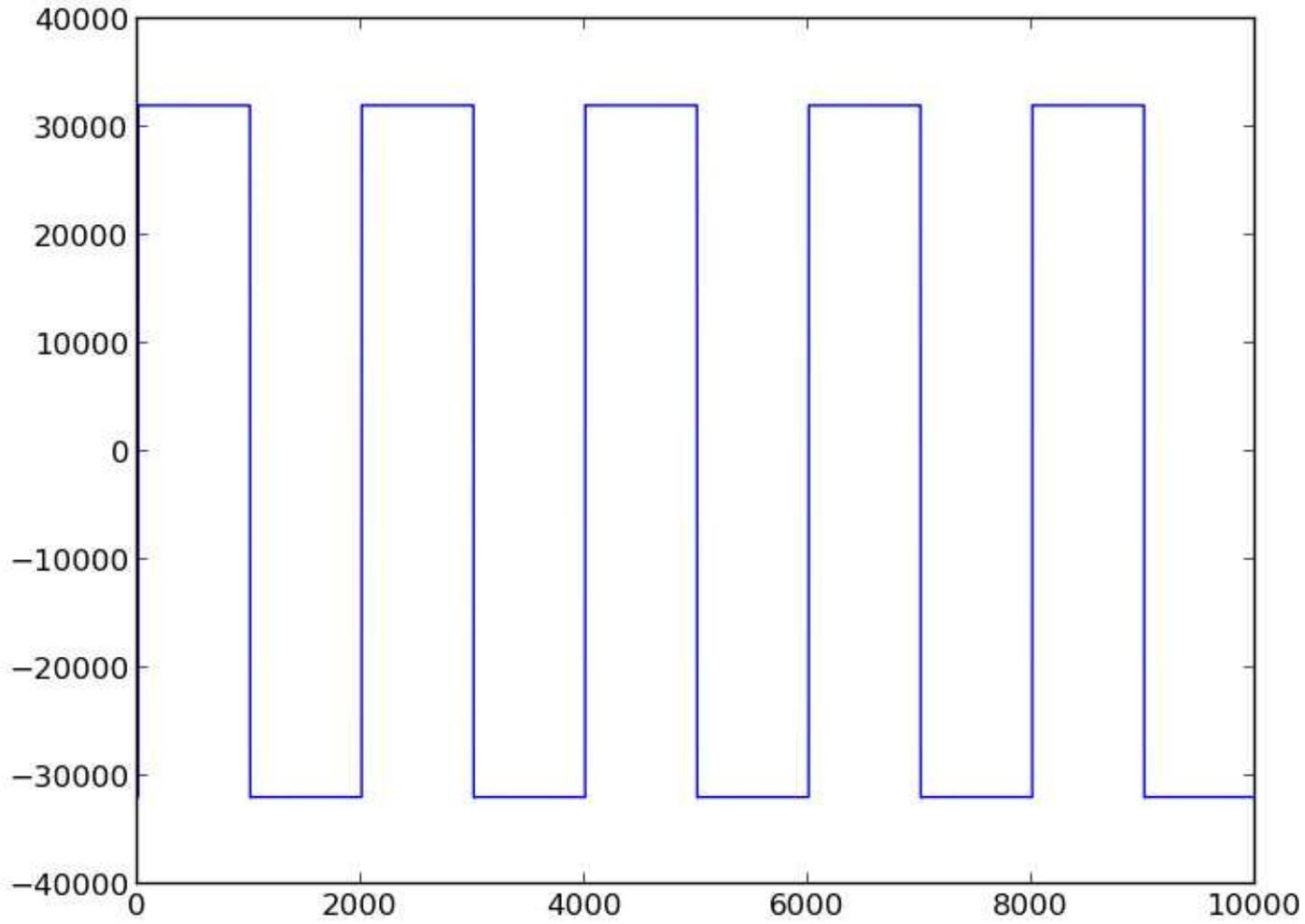
The command to plot data from the second UUT is as follows:

```
python ./host_demux.py  
--src=/home/dt100/PROJECTS/AFHBA404/acq2106_184_VI.dat --  
nchan=160 --pchan=1 --data_type=16 --plot_mpl=1 --  
mpl_subrate=1 acq2106_184
```

This will plot:

1. The first input channel (signal generator)

Data plotted from the second (slave) UUT



1. Sig gen
signal on
CH01

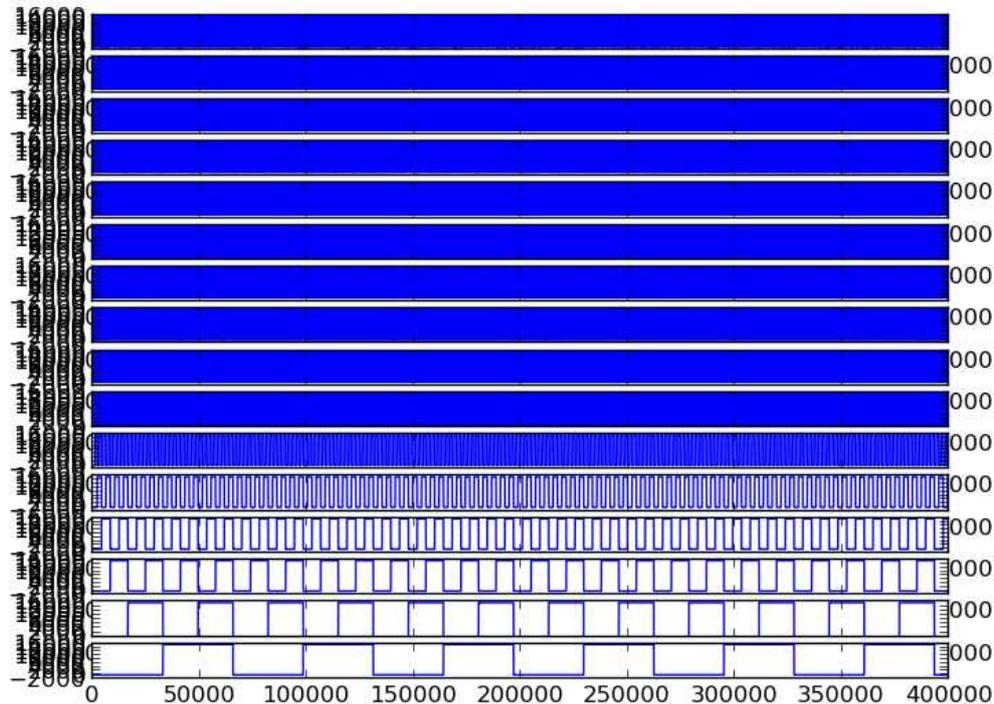
Note: Capture
reduced to
10000 samples
for better
visibility

Demonstration of 16 bits of digital output looped back to an AI module

Here is an example of showing all 16 digital outputs on a DIO432 looped back to 16 channels on an AI module.

The command to do this was:

```
python ./host_demux.py --src=/home/dt100/PROJECTS/AFHBA404/acq2106_183_VI.dat --nchan=160 --pchan=65:80 --data_type=16 --plot_mpl=1 --mpl_subrate=1 acq2106_183
```



Analysing the latency of the data

The control script analyses the FPGA latency registers and plots the histogram of that data, but the latency can also be seen on a scope for an independent verification of the latency. There are already guides written by D-TACQ that explain how to measure the latency of the system. 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.



```

Terminal
File Edit View Search Terminal Help
acq2106_085> get.site 5 OUTPUT_DELAY 4.36
acq2106_085>
acq2106_085>
acq2106_085>
acq2106_085>
  
```

Using hexdump to view the data

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

```
hexdump -e '80/4 "%08x," "\n"' acq2106_183_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 '160/2 "%04x," "\n"' acq2106_183_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.

The screenshot shows the CS Studio software interface. The left panel contains control elements for UUT (acq1001, acq2106_085), SITE (2), Chan (Plot 1), and various acquisition parameters like acq425, acq423 gains, acq420 plot, offsets, acq420, AO420FMC DC, acq435, AO424ELF DC, MONITOR, acq480, DAC20 DC, AI VALUES, bolo8, AWG/DPG, lia, radcelf, AO420FMC AWG, v2f, dds, AO421FMC MUX, anatrg, deltrg, QEN, ODS, DI32, and FIFO Histogram. The right panel displays the 'acq2106_085 COUNTERS' data for CLK, TRIG, and SYN counters across six channels (CLK-d0 to CLK-d6, TRIG-d0 to TRIG-d6, SYN-d0 to SYN-d6). The data is presented in a grid format with frequency values and trigger counts for each channel.

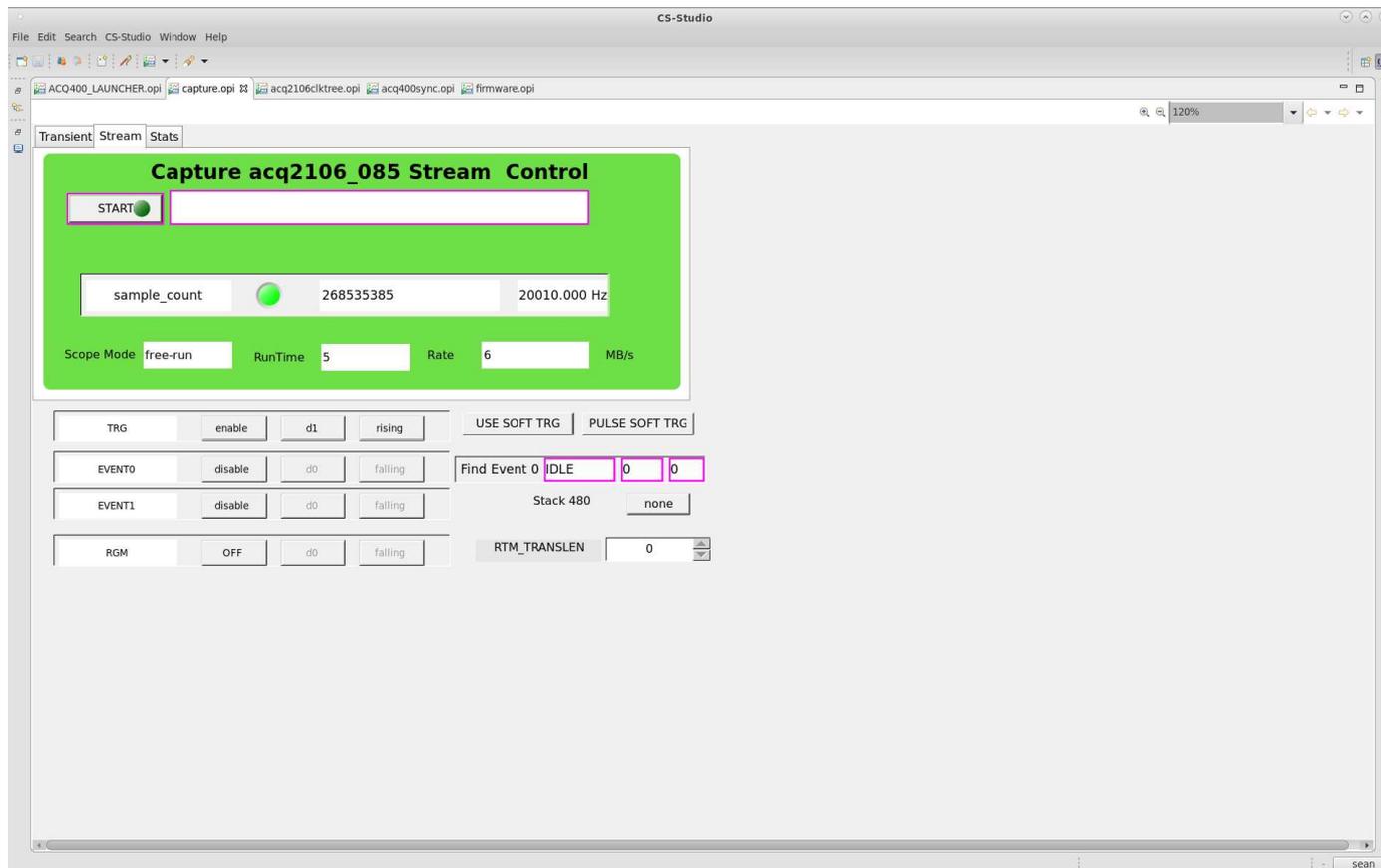
CLK-d0	CLK-d1	CLK-d2	CLK-d3	CLK-d4	CLK-d5	CLK-d6
0.000 Hz	2.001E4 Hz	2.001E4 Hz	0.000 Hz	0.000 Hz	0.000 Hz	2.001E4 Hz
0	227463209	129951677	0	0	0	4939868
EXT	MB	S1	S2	S3	S4	S5

TRIG-d0	TRIG-d1	TRIG-d2	TRIG-d3	TRIG-d4	TRIG-d5	TRIG-d6
0.000 Hz						
1	1	0	0	0	0	1
EXT	MB	S1	S2	S3	S4	S5

SYN-d0	SYN-d1	SYN-d2	SYN-d3	SYN-d4	SYN-d5	SYN-d6
2.001E4 Hz	2.001E4 Hz	0.000 Hz	0.000 Hz	0.000 Hz	0.000 Hz	2.001E4 Hz
4978156	5192178	1	0	0	0	4482120
EXT	MB	S1	S2	S3	S4	S5

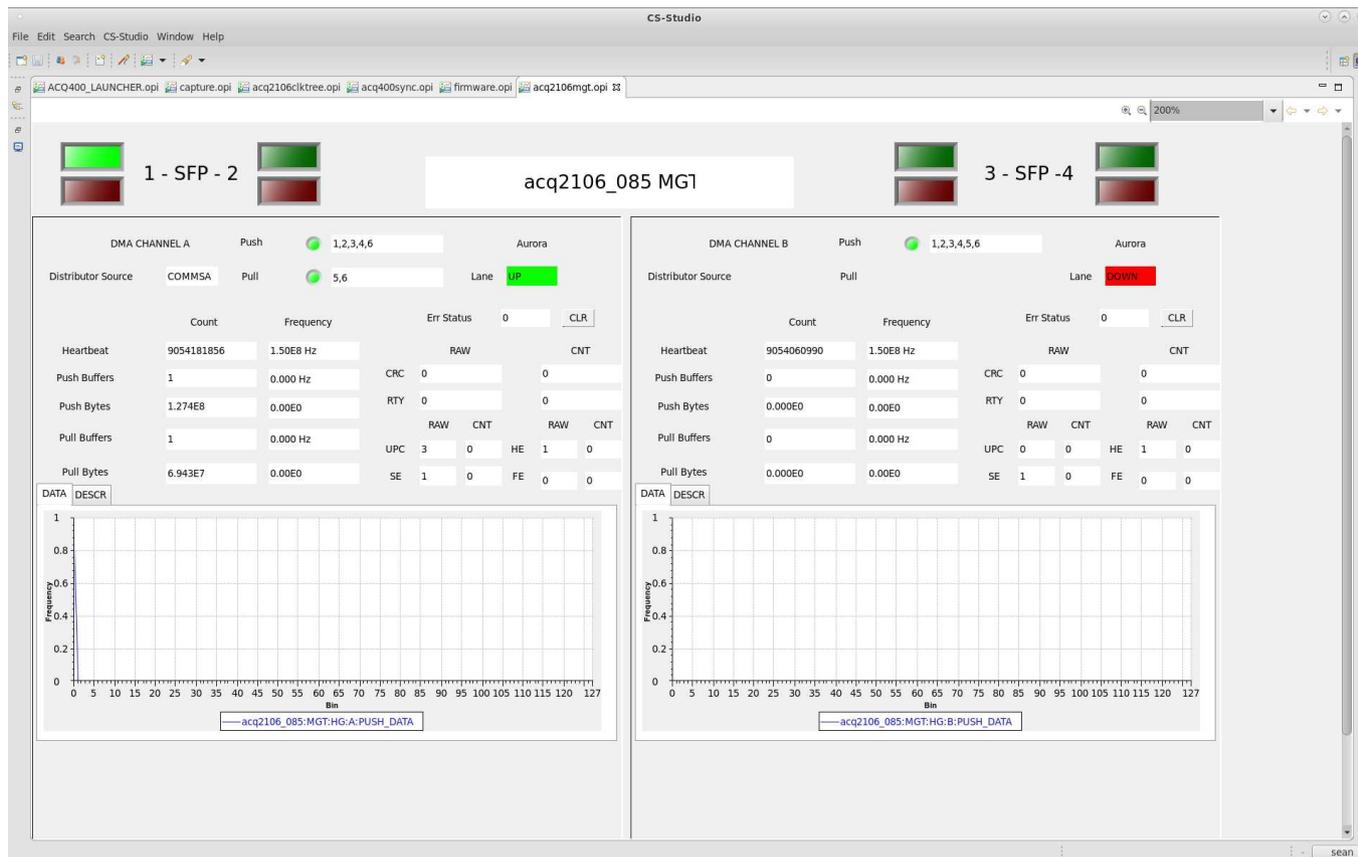
Check UUTs are streaming

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. The sites included in the 'push' box should be AI and DI, and the sites included in the 'pull' box should be AO and DO. Note that DO and DI will likely be the same site (site 6).

Checking the DI

The user can verify the Digital Input by attaching a square wave signal to the DIO432 mezzanine and an AI channel. By default the lower DIO432 will be configured half as DO and half as DI. Connecting channel 17 will toggle the 17th bit of the DIO data. An example of this is shown on the following slide.

17th bit toggling

```
[dt100@seil AFHBA404]$ hexdump -e '80/4 "%08x " "\n"' acq2106_183_VI.dat | awk '{ print $65" "$66 }' | more
```

```
00000000 00000001
```

```
00000000 00000002
```

```
00010001 00000003
```

```
...
```

```
0001000e 00000011
```

```
...
```

```
0001000f 00000012
```

```
00000011 00000013
```

```
...
```

```
00000023 00000026
```

```
00010026 00000029
```

```
00010026 0000002a
```

Hexdump DI vs AI

The following command will show the sampled AI data against the DI data, with the sample counter included. Please note that due to the way the DIO samples it will always lag one sample behind the AI.

```
[dt100@seil AFHBA404]$ hexdump -e '128/2 "%04x," 1/4 "%08x," 1/4 "%10d," 14/4 "%08x," "\n"' acq2106_183_VI.dat | cut -d, -f1-4,129,130 | more
005e,3e46,0038,000f,00000000,      1
0042,3e5d,0031,000a,00000000,      2
002c,3e50,0028,0008,00000000,      3
001f,3e53,0020,0005,00010001,      4
0015,3e57,0018,0003,00010002,      5
ffad,0013,ffd4,fff0,00010003,      6
ffc2,0001,ffd4,fff2,00000004,      7
ffd4,0003,ffd7,fff5,00000005,      8
ffe0,0006,ffdc,fff7,00000006,      9
ffe7,0005,ffe2,fff9,00000007,     10
004c,3e47,0026,000c,00000008,     11
0036,3e5a,0023,000a,00010009,     12
0024,3e54,001e,0006,0001000a,     13
0019,3e55,0017,0003,0001000b,     14
0011,3e3d,0011,0002,0001000c,     15
ffaa,0011,ffcfc,fff0,0001000d,     16
ffc2,fffe,ffd0,fff2,0000000e,     17
ffd2,0001,ffd6,fff4,0000000f,     18
ffdf,0006,ffdb,fff7,00000010,     19
ffe6,0008,ffe2,fff8,00000011,     20
004e,3e46,0022,000b,00000012,     21
0036,3e5a,0023,0008,00010013,     22
0025,3e54,001e,0005,00010014,     23
001a,3e56,0018,0004,00010015,     24
0011,3e58,0010,0000,00010016,     25
ffab,0010,ffcfc,ffef,00010017,     26
ffc1,fffc,ffcfc,fff3,00000018,     27
```

Hexdump DI vs AI with acq196 emulation

The following command is the same as the previous command, except with acq196 emulation enabled (so real CH02 appears demuxed on CH17):

```
[dt100@seil-AFHBA404]$ hexdump -e '128/2 "%04x," 1/4 "%08x," 1/4 "%10d," 14/4 "%08x," "\n"' uut2_data.dat | cut -d, -f1,2,17,129,130 | more
004b,0043,3e4d,00000000,      1
0035,002e,3e46,00000000,      2
0025,001e,3e4a,00000000,      3
0017,0014,3e45,00010000,      4
0013,000c,3e46,00010003,      5
ffbd,ffb9,ffef,00010003,      6
ffcc,ffc9,0009,00000004,      7
ffda,ffd6,fff9,00000004,      8
ffe3,ffdf,fffa,00000006,      9
ffeb,ffe3,fffb,00000007,     10
003c,0035,3e4c,00000008,     11
002c,0025,3e42,00010009,     12
001f,0018,3e43,0001000a,     13
0012,000e,3e47,0001000b,     14
0008,0005,3e4b,0001000c,     15
ffb4,ffb3,ffef,0001000d,     16
ffc6,ffc2,fff0,0000000e,     17
ffd3,ffd0,ffd,0000000e,      18
ffdd,ffd6,fff9,00000010,     19
ffe4,ffdf,fffb,00000010,     20
0038,0030,3e4d,00000012,     21
0023,0021,3e45,00010013,     22
0018,0013,3e41,00010015,     23
0010,000b,3e45,00010015,     24
0008,0003,3e4e,00010017,     25
ffb5,ffa7,fff7,00010017,     26
ffc5,ffc3,fff6,00000018,     27
```