Hardware UDP

High Performance Simultaneous Data Acquisition

Summary and reference document for details relating to D-TACQ support for Hardware UDP on ACQ2106 Carrier via SFP Port D on MGT482-SFP.

Revision History

Revision	Date	Author(s)	Description
1.0	26/05/2022	SR	Created
2.0	06/07/2022	SR	New diagrams. More prose. Added HUDP to Host section
3.0	15/07/2022	SR	Section better specifying packet structure and max limits
4.0	18/01/2023	SR	Some formatting fix-ups
5.0	27/04/2023	SR	More usage examples for end users

Contents

1 Low Latency Demo Setup

1.1 HW

- AI Box : acq2106_189
- AO Box : acq2106_274

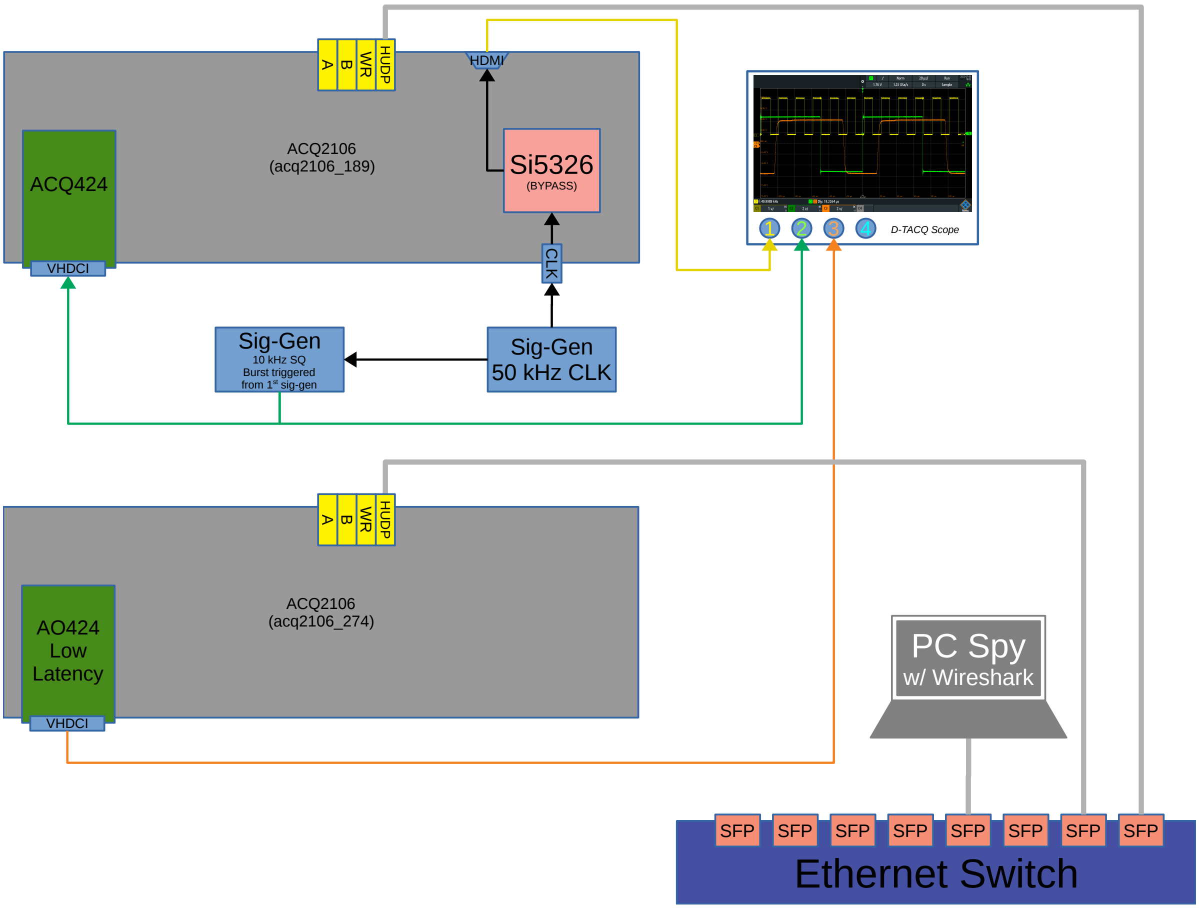

Figure 1: HUDP Demo Hardware Configuration

We set the AI box clock to bypass so that there is no phase delay between the clock from the signal generator and the clock which the ADC sees. The AO is in Low Latency mode (update as soon as you have data) so the clock on the AO box is less important.

Bring the clock back out of HDMI from the 2106 before displaying on scope. This tells us for certain that the Si5326 is successfully in bypass mode and has the added benefit of showing the user where the sample clock lies in relation the their excite signal.

The signal generator which is providing the clock signal, is also driving the External Trigger input of a secondary function generator. This secondary function generator outputs a square wave burst on the rising edge of the clock (not on every edge, burst period is longer than clock period).

We can then control the phase of the burst signal relative to the rising edge of the clock to account for the aperture delay and slew rate of ACQ424 input.

See figures in Section 9 for a detailed look at the oscilloscope traces and latency measurements.

1.2 HUDP Core Configuration

D-TACQ have provided a handy Python interface to help when configuring the HUDP Core. This is further discussed in Section 5.1.

For our example above (Figure 1), the configuration is as follows.

We execute a `sync_role` command to place the AI box's clock synthesiser in bypass.

```
./user_apps/acq400/sync_role.py --fin=50k --fclk=50k --si5326_bypass 1 --toprole=fpmaster,strg acq2106_189
```

Then we set up a TX-RX pair between box acq2106_189 and acq2106_274. Here we use the broadcast address .255 so that the PC which is also in the network can easily spy on the packets as they fly past.

```
./user_apps/acq2106/hudp_setup.py --tx_ip 10.12.198.128 --rx_ip 10.12.198.255 --run0='1 1,16,0' --play0='1 16' \
--broadcast=1 --disco=16 acq2106_189 acq2106_274
```

1.3 CSS OPI

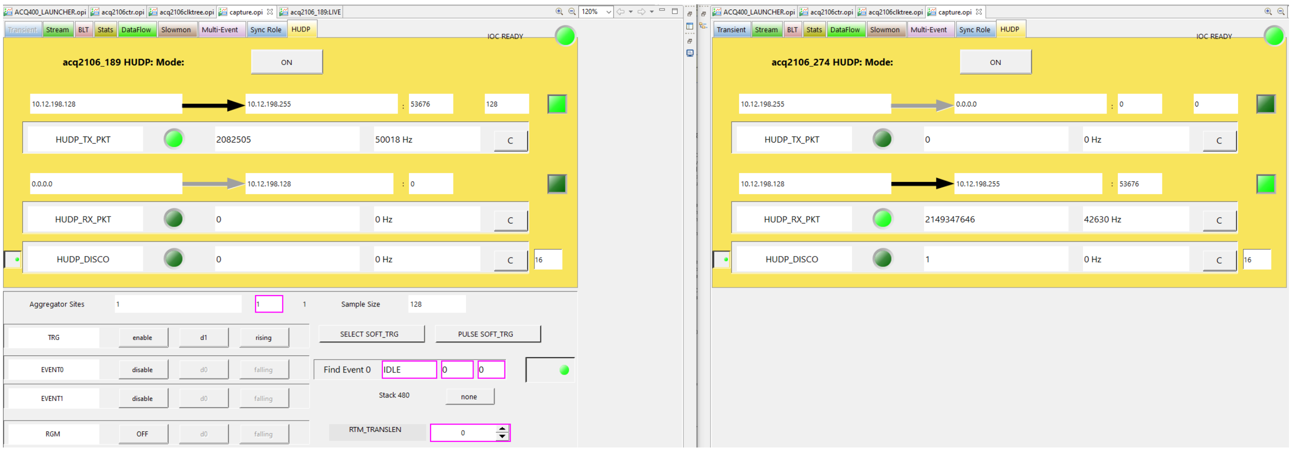

Figure 2: HUDP OPIs : A TX-only box on the left and an RX-only box on the right

1.4 Discontinuity Counter

The discontinuity (disco) counter uses the embedded sample counter (SPAD0) contained within every sample/-packet to verify that we have had no skips or lost packets throughout the course of a run.
Check that the latest received sample count is equal to the previously decoded sample count plus 1. Note, this will fire once at the start of every shot (when our previous sample count is assumed to be zero and the first received sample count will also be zero.) It will also fire spuriously on the rollover of the 32-bit counter (once a day at 50 kHz). We could enhance the logic to account for these exceptions.

2 Clock and Excite Signal Phase Relationship

Here we demonstrate the effect of sampling the analogue rising edge at different points in the low-to-high transition. Even though the signal generator produces a very sharp rising edge, a combination of the bandwidth and slew rate limitations of the front end of the ADC mezzanine, ”flatten out” the rising edge somewhat.

- With the sample clock almost coincident with the analogue rising edge (far left) the ADC only manages to capture the very beginning of the rising edge, resulting in a very low amplitude update on the DAC.
- Pulling the excite signal forward in time (middle section) means that the sample clock is in effect, delayed, and thus samples further up the slope of the rising edge. This results in a higher amplitude update from the DAC.
- With the rising edge now having almost completed its low-to-high transition before the sample clock arrives, the ADC samples much further up the slope and the output from the DACs is almost full scale.

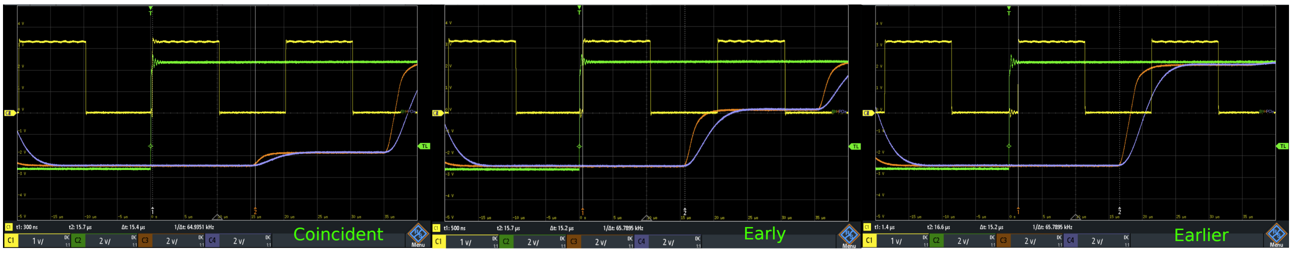

Figure 3: The result of the sample clock sampling the rising edge at various points during the signal slew

ADC sample clock   Sig-gen   LLC Filter DAC Output   Std. Filter DAC Output

3 HUDP - Internal workings

Here we have a more detailed diagram which helps visualise data flows, both within an individual box employing the HUDP core, and in a larger system utilising HUDP as the backbone for data transfer.

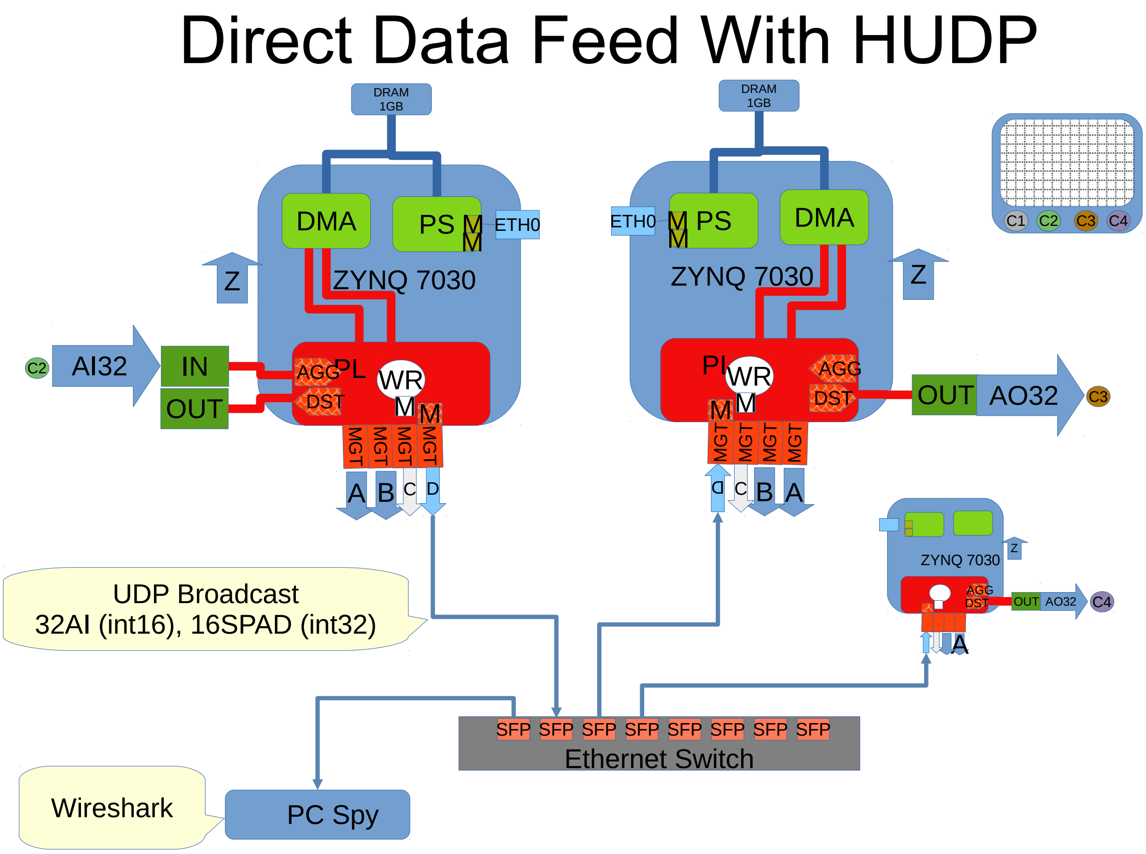

Figure 4: More detailed example of HUDP dataflows

This helps to visualise the entire datapath. Beginning at C2 AI32 we see the data flow into the site, through the Aggregator, down to the HUDP core and associated Mulit-Gigabit Transceiver (MGT) port.
We push packets out on SFP, through the Ethernet switch and back to the RX box. These packets hit the HUDP core on the AO box, data is stripped and passed down to the Distributor which is responsible for sharing out the data to selected sites. From here we carry on to the AO32 site and eventually back out into the analogue domain to be sampled by the oscilloscope (C3).

4 Setup

4.1 Python - ACQ400 HAPI Installation

Python code and full install instructions can be retrieved from : https://github.com/D-TACQ/acq400_hapi

To install :

```
mkdir PROJECTS; cd PROJECTS
git clone https://github.com/D-TACQ/acq400_hapi.git
cd acq400_hapi
```

on Linux , run `source ./setpath`

on Windows, run `SETPYTHONPATH.BAT`

5 Streaming Sampled data to host via UDP

5.1 Setup

We begin by running the `hudp_setup` script from the host. Usage and arguments below. There is also extensive explanation provided in the comments at the beginning of the script.

```
[dt100@naboo acq400_hapi]$ python3 ./user_apps/acq2106/hudp_setup.py -h
usage: hudp_setup.py [-h] [--netmask NETMASK] [--tx_ip TX_IP] [--rx_ip RX_IP]
                     [--gw GW] [--port PORT] [--run0 RUN0] [--play0 PLAY0]
                     [--broadcast BROADCAST] [--disco DISCO] [--spp SPP]
                     [--hudp_decim HUDP_DECIM]
                     txuut rxuut

hdup_setup

positional arguments:
  txuut                 transmit uut
  rxuut                 transmit uut

optional arguments:
  -h, --help            show this help message and exit
  --netmask NETMASK     netmask (default: 255.255.255.0)
  --tx_ip TX_IP         rx ip address (default: 10.12.198.128)
  --rx_ip RX_IP         tx ip address (default: 10.12.198.129)
  --gw GW               gateway (default: 10.12.198.1)
  --port PORT           port (default: 53676)
  --run0 RUN0           set tx sites+spad (default: 1 1,16,0)
  --play0 PLAY0         set rx sites+spad (default: 1 16)
  --broadcast BROADCAST
                        broadcast the data (default: 0)
  --disco DISCO         enable discontinuity check at index x (default: None)
  --spp SPP             samples per packet (default: 1)
  --hudp_decim HUDP_DECIM
                        hudp decimation, 1..16 (default: 1)
```

This configures all of the necessary HUDP variables on the box. Many of these can be monitored using the HUDP tab in the capture.opi.

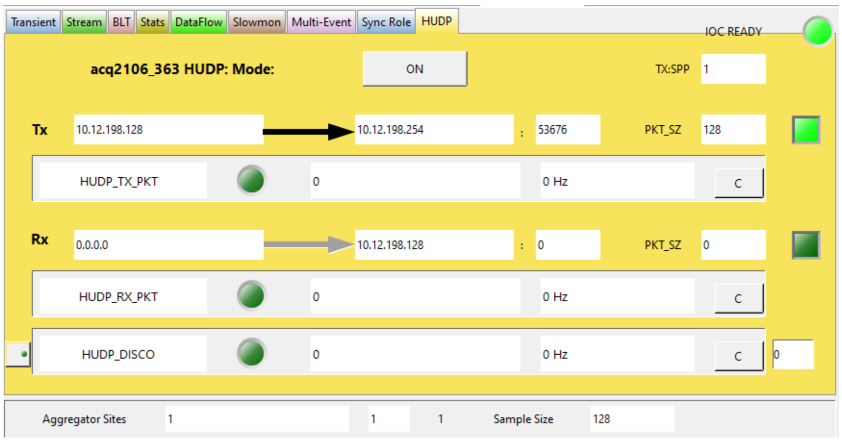

Figure 5: HUDP OPI after configuration by udp_setup.py

Here we give the acq2106 an address of 10.12.198.128 and we plan to receive packets on host (naboo:10.12.198.254). Our port on both ends is the default 53676.

```
[dt100@naboo acq400_hapi]$ python3 ./user_apps/acq2106/hudp_setup.py --tx_ip 10.12.198.128 --rx_ip 10.12.198.254 \
--run0='1 1,16,0' acq2106_363 none
txuut acq2106_363
enable disco at 16
```

The `run0='1 1,16,0'` argument tells us that we have requested data from Site 1, and that we have enabled a Scratchpad (SPAD) of length 16 LWords.
The Scratchpad is a series of writable elements that we append to the end of each sample of analog data. The first of these is always a sample counter. This is very useful to verify that no packets have been lost on the wire.

5.2 Streaming

Packets will be received on the host by the netcat program and redirected to a file for later analysis.
We start the `nc` process on the host, then begin a capture on the acq2106. In this way we always catch the beginning of the shot.

Open two terminals on your host. From the first we will run the `nc` command and from the other we will control the start/stop of the acq2106 stream process.

```
$ nc -ulv 10.12.198.254 53676 | pv > shot_data
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Listening on 10.12.198.254:53676
Ncat: Connection from 10.12.198.128.
 113MiB 0:00:15 [23.2MiB/s] [           <=>
]
```

```
$ python3 ./user_apps/acq400/acq400_continuous.py \
--run=1 acq2106_363
$
$ # Wait for the nc > file to fill with data
$
$ python3 ./user_apps/acq400/acq400_continuous.py \
--stop=1 acq2106_363
```

We can use hexdump to take a quick look at the data. Here we have received data comprised of 32 LWord samples, where the first 16 LWords are analog data and the remaining 16 are SPAD values. The SPAD values begin on the 17th entity (counting from 1) and we can see this clearly by looking for the sample counter. I have forced some fake headings into the hexdump output below to illustrate this.
The use of `cut` here means we don't have to look at all 32 values, thus saving ourselves some space in our terminal.

```
[dt100@naboo UDP_TEST]$ hexdump -e '32/4 "%08x " "\n"' shot_data | cut -d ' ' -f 1-4,17-20 | head
ANALOG   ANALOG   ANALOG   ANALOG   COUNTER  XX       XX       SIGNATURE
ea631a50 f571ff61 fffefff9 fff10000 00000001 00000000 00561657 00000000
ea651a21 f571ff5f fffffff8 fff10000 00000002 00000000 0056171f 00000000
ea6219f0 f572ff5f fffffff9 fff10000 00000003 00000000 005617e7 00000000
ea6319c2 f572ff5f fffffff9 fff10000 00000004 00000000 005618af 00000000
ea671993 f572ff5f fffffff8 fff10000 00000005 00000000 00561977 00000000
ea691964 f572ff61 fffffff9 fff10001 00000006 00000000 00561a3f 00000000
ea691935 f572ff60 fffffff9 fff10000 00000007 00000000 00561b07 00000000
ea641904 f572ff63 fffffff8 fff00000 00000008 00000000 00561bcf 00000000
ea6718d5 f572ff63 fffffff9 fff10000 00000009 00000000 00561c97 00000000
ea6518a6 f572ff64 fffffff8 fff10000 0000000a 00000000 00561d5f 00000000
```

We will plot this data graphically in the following section.

5.3 Plotting Sampled Data on Host

```
[dt100@naboo acq400_hapi]$ python3 ./user_apps/analysis/host_demux.py --save=DATA --nchan=64 --nblks=-1 --pchan=1,33 \
      --src="/home/dt100/UDP_TEST/shot_data" acq2106_363 --plen=20000
INFO: plotext available as plot backup when no graphics
data_type 16 np <class 'numpy.int16'>
args.pc_list [0, 32]
data saved to directory: /home/dt100/UDP_TEST/DATA
Plotting with MatPlotLib. Subrate = 1
plot_mpl num 1
```

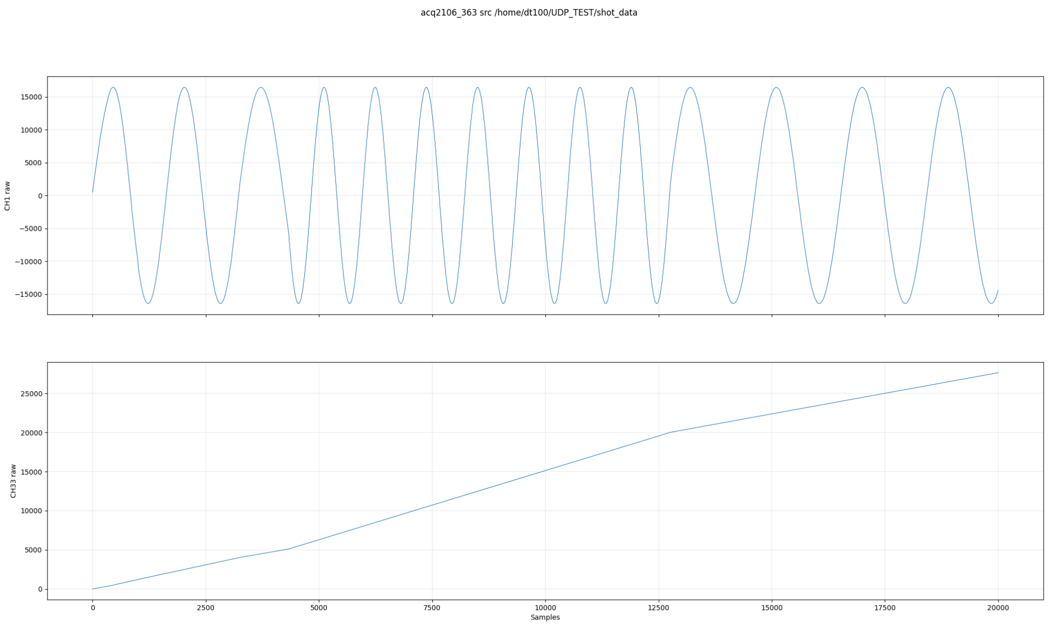

Figure 6: An example of packet loss when we overwhelm the host with too many packets per second

Note the unstable sine wave and non-contiguous sample counter indicating that packets have been lost. In this case we know through previous testing that this particular host can only cope with 80,000 samples per second, and here we are sending 200,000 per second.

For low latency applications we want to minimise our packet size so that we do not incur a delay by waiting for a packet to fill with user data. For streaming applications, this is obviously terribly inefficient.

We have two ways of dealing with this problem.

- Firstly, we can pack multiple samples into a packet; hence reducing the packet rate.
- Secondly, we can apply decimation in the HUDP module by discarding every X samples.

5.4 Multiple Samples per Packet

This time we run our setup script but include an argument for samples per packet. `--spp 10`

```
[dt100@naboo acq400_hapi]$ python3 ./user_apps/acq2106/hudp_setup.py --tx_ip 10.12.198.128 --rx_ip 10.12.198.254 \
--run0='1 1,16,0' acq2106_363 none --spp 10
txuut acq2106_363
TX configured. ssb:128 spp:10 tx_pkt_size 1280
```

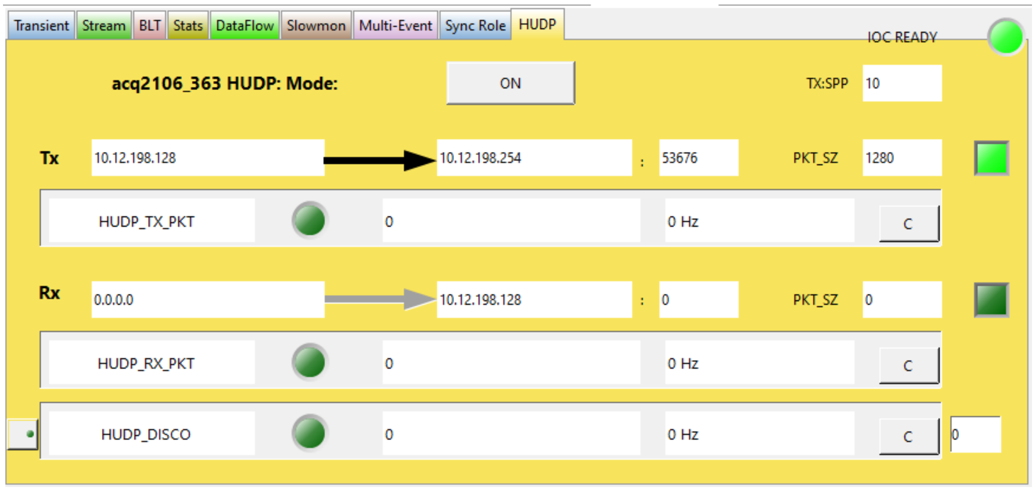

Figure 7: HUDP OPI with Multiple Samples per Packet showing new PKT_SZ

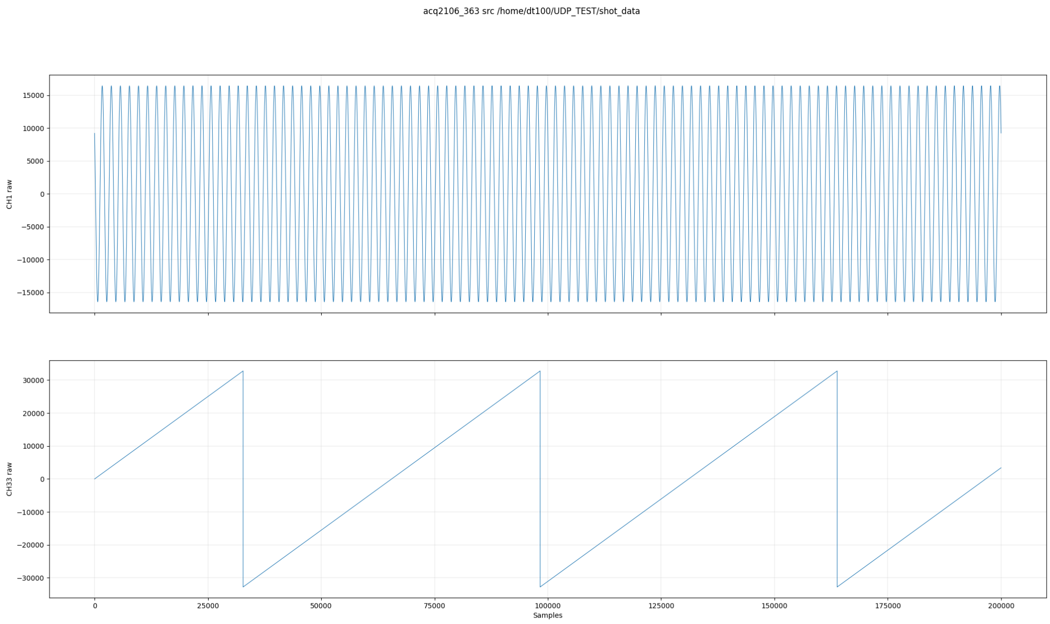

Figure 8: One whole seconds worth of data with zero packet loss

CH33 contains the bottom 16 bits of an embedded sample counter so it wraps at 32767, but what we are focusing on here is the lack of any other unexpected jumps across the entire shot. There are more robust ways of validating this transmission but this test is presented here as an easily understood, basic example.

5.5 Decimation

We re-run the setup script again to return to 1 sample per packet. But now we will apply sample decimation in the HUDP module to lower the packet rate **AND** the data rate, instead. `--hudp_decim 10`

```
[dt100@naboo acq400_hapi]$ python3 ./user_apps/acq2106/hudp_setup.py --tx_ip 10.12.198.128 --rx_ip 10.12.198.254 \
--run0='1 1,16,0' acq2106_363 none --hudp_decim 10
txuut acq2106_363
TX configured. ssb:128 spp:1 tx_pkt_size 128
```

We requested a decimation of 10. This means we discard 9 samples between outputs, a "decimation" of factor 10.

We can see here that the `MiB/s` rate has dropped by a factor of 10 relative to full rate example in Section 5.2.

```
[dt100@naboo UDP_TEST]$ nc -ulv 10.12.198.254 53676 | pv > shot_data
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Listening on 10.12.198.254:53676
Ncat: Connection from 10.12.198.128.
50.5MiB 0:00:31 [2.44MiB/s] [                              <=>
```

The effective sample rate has now decreased so our sampled waveform has a higher apparent frequency than the plots earlier in this document.

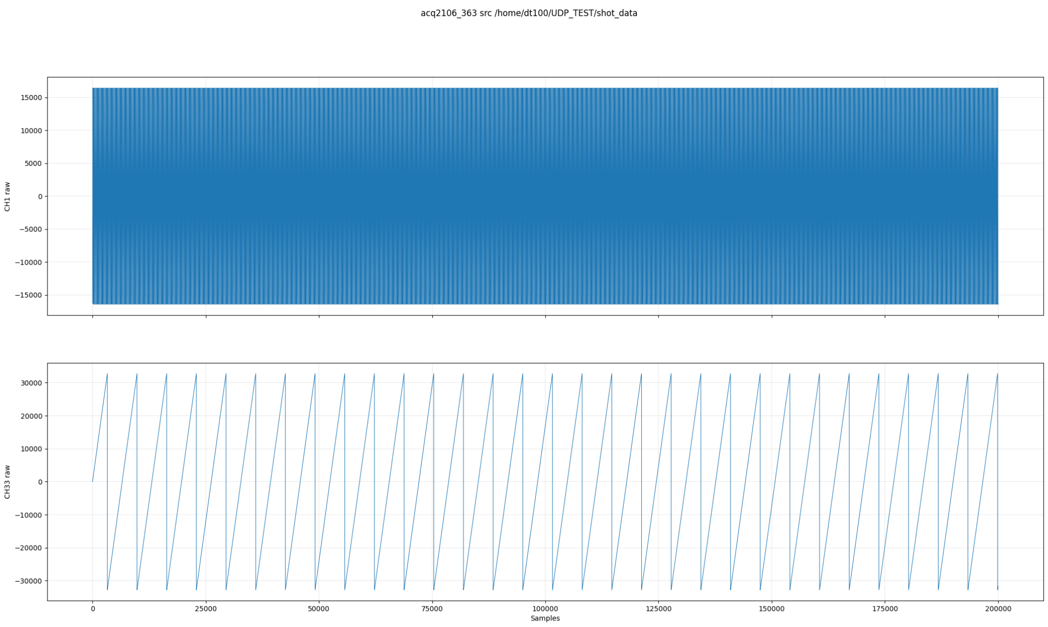

Figure 9: Decimated data showing effective decreased sample rate

Note also that the sample counter now jumps in increments of 10 samples.

```
[dt100@naboo UDP_TEST]$ hexdump -e '32/4 "%08x " "\n"' shot_data | cut -d ' ' -f 1-4,17-20 | head
ANALOG   ANALOG   ANALOG   ANALOG   COUNTER  XX       XX       SIGNATURE
e5c0c068 f490ffd7 fffbfff3 ffef0006 00000001 00000000 204ac4af 00000000
e5b3c0b4 f490ffd7 fffbfff3 fff00007 0000000b 00000000 204acc7f 00000000
e5b1c113 f491ffd6 fffcfff2 ffee0006 00000015 00000000 204ad44f 00000000
e5b6c184 f491ffd6 fffafff2 fff00006 0000001f 00000000 204adc1f 00000000
e5aac202 f490ffd5 fffafff3 ffef0006 00000029 00000000 204ae3ef 00000000
e5a5c290 f491ffd5 fffbfff4 fff00006 00000033 00000000 204aebbf 00000000
e5a9c32f f491ffd4 fffbfff3 fff00006 0000003d 00000000 204af38f 00000000
e5a2c3dd f491ffd4 fffbfff3 fff00005 00000047 00000000 204afb5f 00000000
e592c498 f491ffd2 fffbfff3 fff00005 00000051 00000000 204b032f 00000000
e58ec562 f491ffd1 fffcfff4 fff00005 0000005b 00000000 204b0aff 00000000
```

5.6 Streaming Multiple Sites of Data

```
   1      D-TACQ Solutions       ACQ423ELF   ACQ423ELF-32-200-16-FFC N=32 M=09       E42310229
   2      D-TACQ Solutions       ACQ423ELF   ACQ423ELF-32-200-16-FFC N=32 M=09       E42310230
   6      D-TACQ Solutions       DIO482ELF                 DIO482ELF N=32 M=6B       E48220160
```

Let us now include all of the "input" capable sites in the Aggregator set. We will also reduce the length of the SPAD to give a nice round number. Note, `run0='1,2,6 1,15,0'`
We'll also decimate by 10 to keep the packet rate to the host down.

```
[dt100@naboo acq400_hapi]$ python3 ./user_apps/acq2106/hudp_setup.py --tx_ip 10.12.198.128 --rx_ip 10.12.198.254 \
--run0='1,2,6 1,15,0' acq2106_363 none --hudp_decim 10
txuut acq2106_363
TX configured. ssb:192 spp:1 tx_pkt_size 192
```

The output from the `hudp_setup` script helpfully tells us our samples size in bytes (ssb). So we can see that our sample size in LWords is now 48 (192÷4). The new hexdump is presented below. Again, context headings have been added to aid understanding.

```
[dt100@naboo UDP_TEST]$ hexdump -e '48/4 "%08x " "\n"' shot_data | cut -d ' ' -f 1,2,17,18,33-37 | head
1        2        17       18       33       34       35       36       37
ACQ423_1 ACQ423_1 ACQ423_2 ACQ423_2 DIO482   COUNTER  XX       XX       SIGNATURE
ffff0000 ffffffff 00060005 00000000 00000000 00000001 00000000 906accb5 00000000
ffffffff 00000000 00060006 00000000 00000000 0000000b 00000000 906ad485 00000000
0000ffff ffffffff 00050005 00000000 00000000 00000015 00000000 906adc55 00000000
ffff0000 ffff0000 00060006 00000000 00000000 0000001f 00000000 906ae425 00000000
0000ffff ffffffff 00060005 ffff0000 00000000 00000029 00000000 906aebf5 00000000
ffffffff 00000000 00050005 0000ffff 00000000 00000033 00000000 906af3c5 00000000
ffffffff 00000000 00060005 00000000 80000000 0000003d 00000000 906afb95 00000000
ffffffff 00000000 00060005 00000000 80000000 00000047 00000000 906b0365 00000000
ffffffff 0000ffff 00060005 00000000 80000000 00000051 00000000 906b0b35 00000000
ffffffff 00000000 00060006 00000000 80000000 0000005b 00000000 906b1305 00000000
```

6 Streaming UDP Data to acq2106 XO Modules

In this Section we demonstrate sending data from the host to the acq2106 over UDP and playing it out on the AO424 module. For simplicity we will use previously sampled data to replay on the AO424. Process below :

- Sample analog data on an ADC module
- Stream this data over UDP to the host
- Store the data in a file
- Stream this data file back to the box over UDP to be played out by the DAC module

We will limit the sample/update rate to 20 kHz so as not to stress the host through high frequency packet generation. A C program has been created to take a source file and send fixed length packets to a specific IP address and port. This is available here : https://github.com/petermilne/HUDP

```
# Set the sample rate to 20 kHz
python3 ./user_apps/acq400/sync_role.py --fclk=20k --toprole=master,strg acq2106_363

# Configure the acq2106 HUDP TX to send us 32 channels of analog data with 16 LWords of SPAD
python3 ./user_apps/acq2106/hudp_setup.py --tx_ip 10.12.198.128 --rx_ip 10.12.198.254 --run0='1 1,16,0' acq2106_363 none

# Capture some data to file
nc -ulv 10.12.198.254 53676 | pv > shot_data

# Configure the acq2106 HUDP RX to accept packets from IP addr and port
# Also, set up the distributor to send 32 channels of data to Site 3 and discard the SPAD LWords
python3 ./user_apps/acq2106/hudp_setup.py --tx_ip 10.12.198.254 --rx_ip 10.12.198.128 --play0='3 16' none acq2106_363

# Stream the data file back to the acq2106
# Here 50 usecs equates to 20 kHz, matching our ADC sample rate
[dt100@naboo HUDP]$ ls -lh shot_data
-rw-r--r-- 1 dt100 d-tacq 104M Nov 23 11:00 shot_data
[dt100@naboo HUDP]$
[dt100@naboo HUDP]$ ./udp_client
usage:  udpclient <IP address> <port> <sz> <file> <usecs>
[dt100@naboo HUDP]$ ./udp_client 10.12.198.128 53676 128 shot_data 50
```

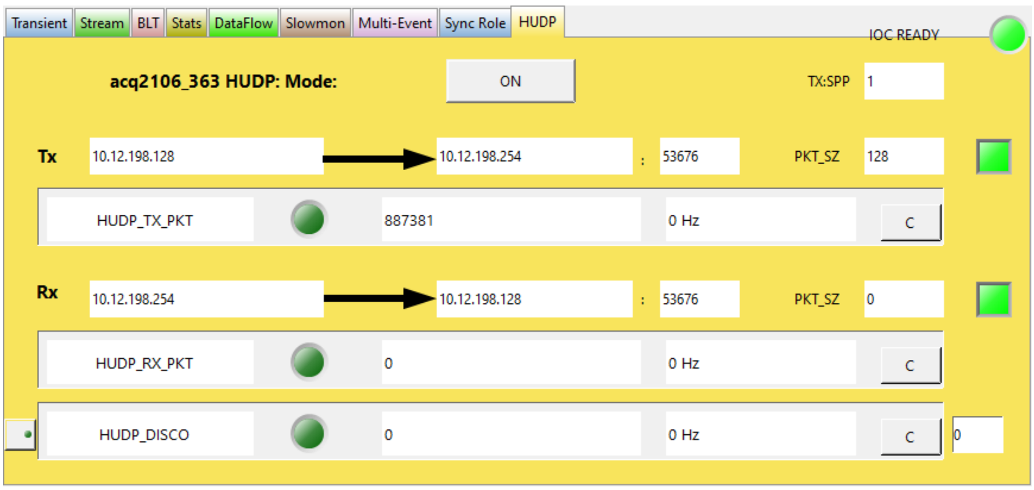

Figure 10: HUDP OPI showing TX packet count and RX variables set

After we run the last command in the listing above, the RX packet counter should being counting and the DAC should now be updating.
We can see the result of this in Figure 11. The yellow trace shows us what the signal generator is producing, and hence what the ADC had captured. The blue trace is live output from the DAC showing the signal faithfully recreated.

Figure 11: Oscilloscope trace showing data from signal generator in yellow and similar data from DAC in blue

7 Max Specifications

Frame Section	Size (Bytes)	Comment	Size (Bytes)
Inter-Frame Gap (IFG) (96 ns)	12	(125 MB/s × 96 ns = 12B)	12
MAC Preamble	8	including Start of Frame Delimiter (SFD)	8
MAC Destination Address	6		6
MAC Source Address	6		6
EtherType (or Length)	2		2
IPv4 Header UDP Header User Data (CALC_PKT_SIZE)	20 8 1472	Payload MTU 1500B	20 8 1472
Frame Check Sequence (FCS)	4		4
Total Frame Size	1518	Total w/ overheads (includes IFG and Preamble)	1538

Table 1: Breakdown of an Ethernet Frame from HUDP Core

Gigabit Ethernet raw line rate = 1.25 Gb/s,
accounting for 8B/10B encoding leaves us with 1 Gb/s,
this equates to 125 MB/s.

So we can transfer 125 million bytes per second, and our maximum frame size is 1538 bytes. This gives us a maximum packet rate (with a 1538B frame size) of :

$$\frac{125 \times 10^6}{1538} \approx 81,274 \text{ frames per second}$$

This means that our maximum rate for transferring (useful) user data is as follows :

$$\begin{aligned} 81,274 \times 1472\text{B} &= 119.635 \text{ MB/s} \\ &\approx 114 \text{ MiB/s} \end{aligned}$$

From the above we can see that our total overhead is equal to :

$$1538 - 1472 = 66\text{B}$$

To calculate our max packet rate for any sample size, and hence our maximum sample rate in a low latency system, we would use the following equation :

$$\frac{125 \times 10^6}{(\text{Sample Size} + 66)} = \text{Max. packets per second} \tag{1}$$

7.1 Jumbo Frames

Ethernet Jumbo Frames commonly carry payloads of up to 9000 bytes. Support is still under development.

$$\frac{125 \times 10^6}{9000} \approx 13,888 \text{ frames per second}$$

We still have the same overall overhead, so our max user data length is :

$$9000 - 66 = 8934\text{B}$$

This means that our maximum rate for transferring (useful) user data is as follows :

$$\begin{aligned} 13,888 \times 8934\text{B} &= 124.075 \text{ MB/s} \\ &\approx 118 \text{ MiB/s} \end{aligned}$$

7.2 10G

This is a future development for ACQ2206 carriers.

10 Gigabit Ethernet raw line rate = 10.3125 Gb/s,
accounting for 64B/66B encoding leaves us with 10 Gb/s,
this equates to 1.25 GB/s.

So we can transfer 125 million bytes per second, and our maximum frame size is 1538 bytes. This gives us a maximum packet rate (with an 8934B frame size) of :

$$\frac{1.25 \times 10^9}{9000} \approx 138,888 \text{ frames per second}$$

This means that our maximum (theoretical) rate for transferring (useful) user data is as follows :

$$\begin{aligned} 138,888 \times 8934\text{B} &\approx 1.24 \text{ GB/s} \\ &\approx 1.15 \text{ GiB/s} \end{aligned}$$

Knowing that our test host can only cope with ≈ 80,000 packets per second :

$$\begin{aligned} 80,000 \times 8934\text{B} &= 714.72 \text{ MB/s} \\ &\approx 681.61 \text{ MiB/s} \end{aligned}$$

$$\begin{aligned} 90,000 \times 8934\text{B} &= 804.06 \text{ MB/s} \\ &\approx 766.81 \text{ MiB/s} \end{aligned}$$

7.3 Worked Examples

7.3.1 Small Sample - Low Latency

Let's look at an example with small sample sizes. 1 ACQ424 card with a Scratchpad of 8 LWords.

$$1 \text{ ACQ424} = 32\text{CH} \times 2\text{B} = 64\text{B}, 8 \text{ SPAD LWords} = 8 \times 4\text{B} = 32\text{B}$$

$$\text{Total Sample Size} = \ = 64 + 32 = 96\text{B}$$

Our 96B sample size means that we can manage :

$$\frac{125 \times 10^6}{(96 + 66)} = 771,604 \text{ packets per second, (74 MB/s)}$$

7.3.2 Multiple Samples per Packet - High Throughput

6 ACQ424 cards with a Scratchpad of 16 LWords and 3 samples per packet.

$$6 \text{ ACQ424} = 6 \times 32\text{CH} \times 2\text{B} = 384\text{B}, 16 \text{ SPAD LWords} = 16 \times 4\text{B} = 64\text{B}$$

$$\text{Total Sample Size} = \ = 3 \times (384 + 64) = 1344\text{B}$$

Our 96B sample size means that we can manage :

$$\frac{125 \times 10^6}{(1344 + 66)} = 88,652 \text{ packets per second, (119 MB/s)}$$

8 Monitoring Packets with Wireshark

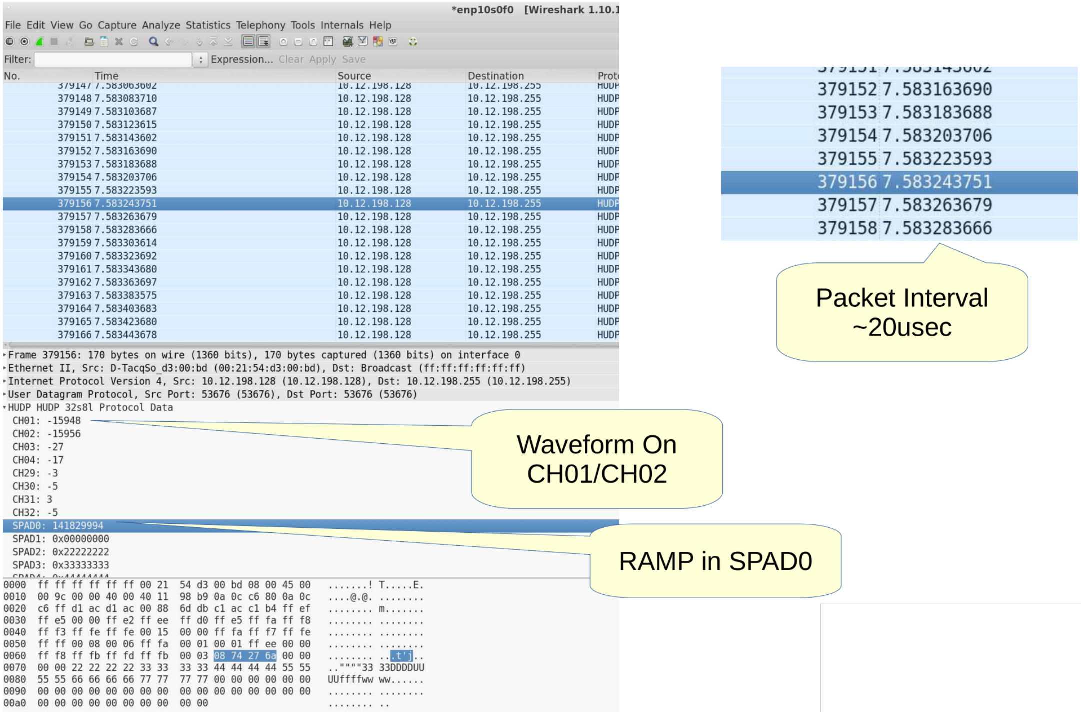

Figure 12: Monitoring packets with Wireshark plugin

https://github.com/D-TACQ/CUSTOM_WRPG/blob/master/CARE/dot.wireshark.plugins.hudp32s_8l.lua

9 Latency Measurements

9.1 Point-to-Point

Best possible result. AI box straight to AO box. Very stable. 12.6 µs. We should repeat this test using a 2 MHz ACQ425 (smallest possible aperture delay) and an AO424-LLC in 16CH mode to minimise both input and output delay as well as the smallest possible packet. This could then be compared to the Aurora latency record of 5.1 µs from the D-TACQ LLC White Paper.

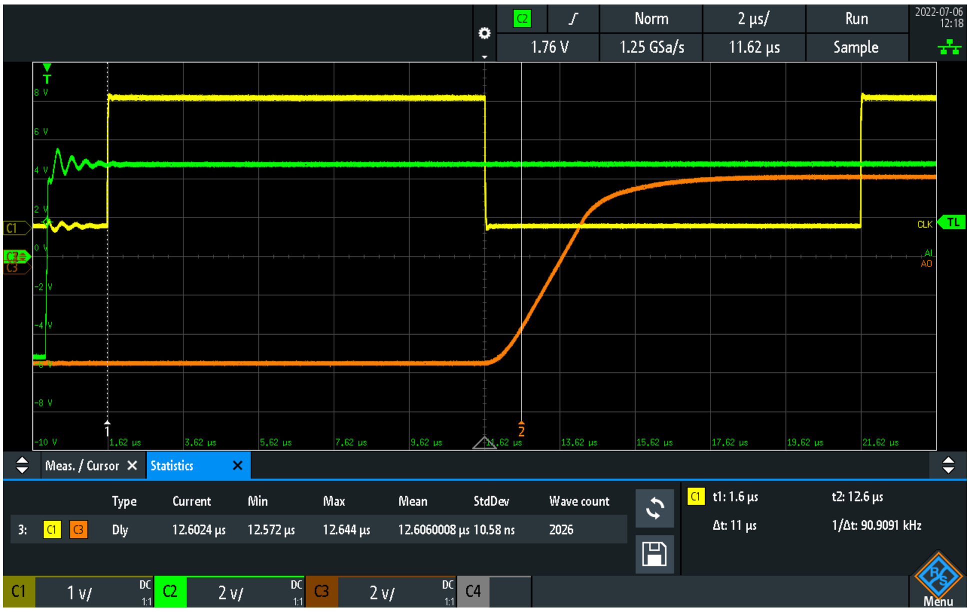

Figure 13: HUDP Point-to-Point Latency

9.2 Through Switch

Note the extra jitter incurred by the trip through the switch in addition to the further ≈ 5 ns of latency. 17.4 µs.

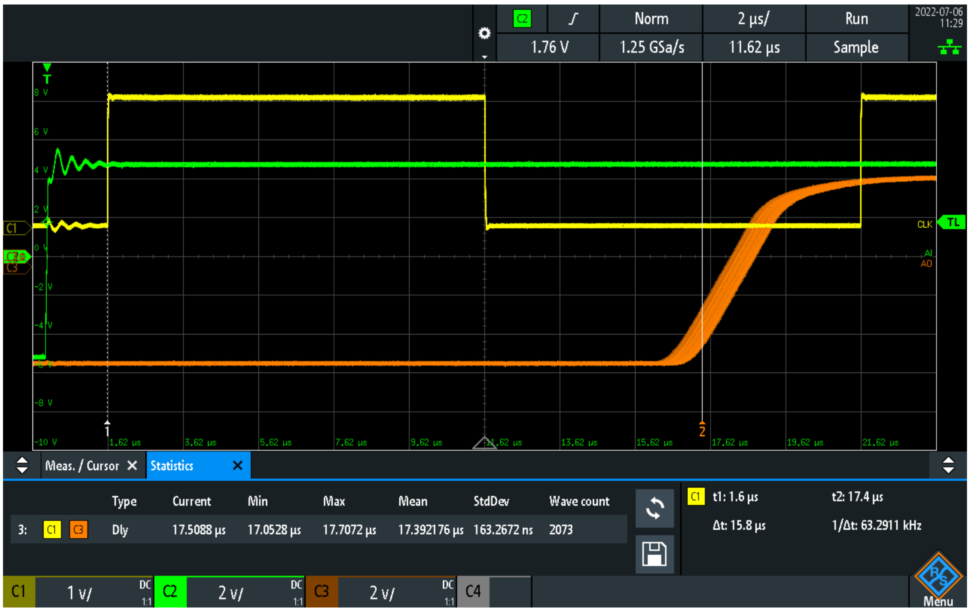

Figure 14: HUDP Latency through Switch

A Resource Usage

UDP Standard = (FIFO Depth 4096)
UDP Jumbo = (FIFO Depth 16384)

Site Type	Available	Used (No UDP)	Used (UDP Std.)	Std. Diff	Std. Util
Slice LUTs	78,600	36,794	40,019	3,225	4.10 %
Slice Regs	157,200	49,353	54,211	4,858	3.09 %
BRAM Tile	265	149.5	160	10.5	3.96 %

Table 2: Table Showing Resource Utilisation for UDP with Standard MTU

Site Type	Available	Used (No UDP)	Used (UDP Jumbo)	Jumbo Diff	Jumbo Util
Slice LUTs	78,600	36,794	40,050	3,256	4.14 %
Slice Regs	157,200	49,353	54,255	4,902	3.12 %
BRAM Tile	265	149.5	167	17.5	6.60 %

Table 3: Table Showing Resource Utilisation for UDP with Jumbo Frames

Site Type	Available	Std. Diff	Jumbo Diff	Diff	Diff %
Slice LUTs	78,600	3,225	3,256	31	0.04 %
Slice Regs	157,200	4,858	4,902	44	0.03 %
BRAM Tile	265	10.5	17.5	7	2.64 %

Table 4: Table Showing Extra Resource Required to implement Jumbo Frames

B HUDP 1G Logic Schematic View

Overleaf

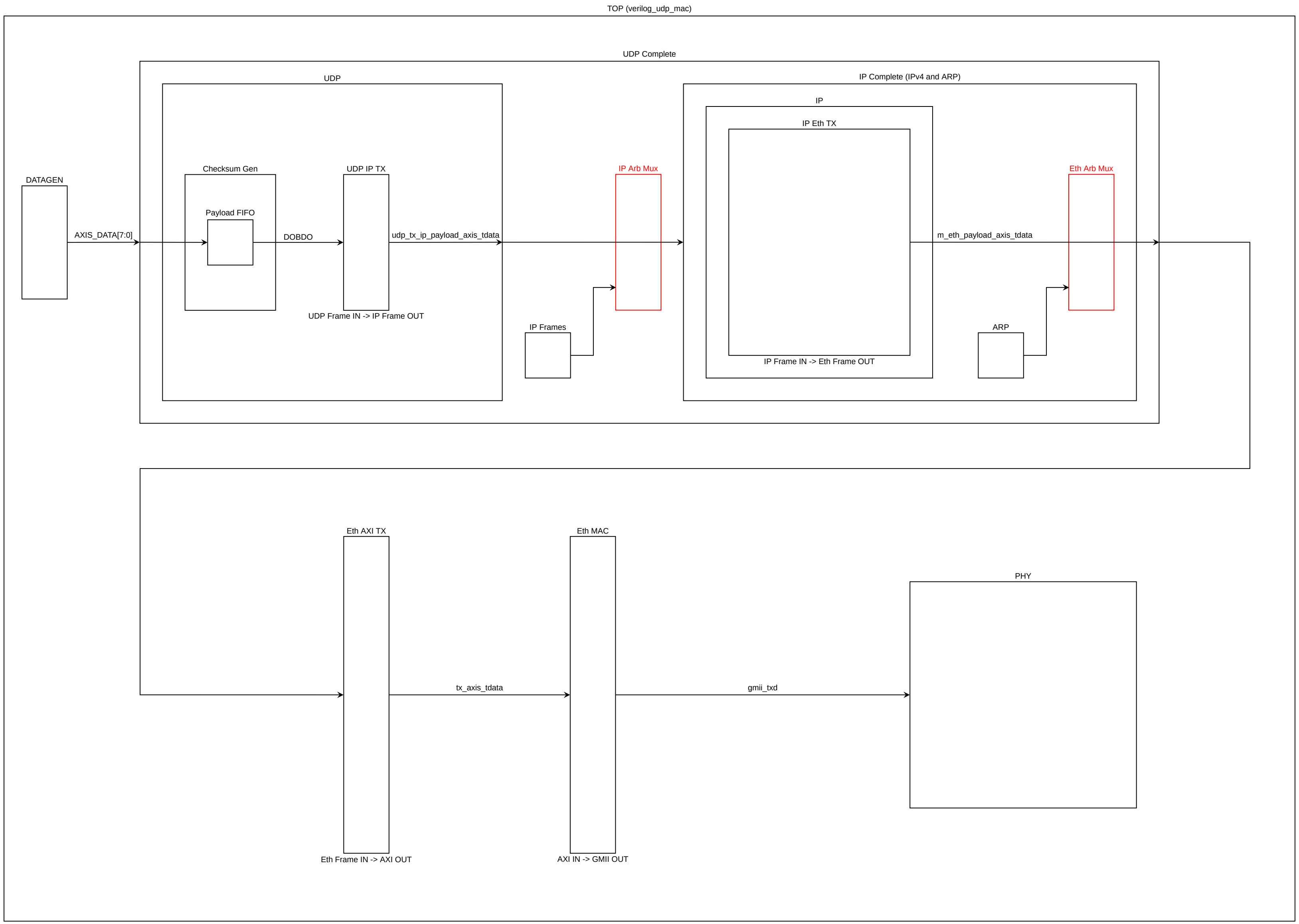

Figure 15: 1G HUDP Logic Schematic View