MIT, HAYSTACK OBSERVATORY
July 10, 2025

Telephone: 617-715-5517
Fax: 617-715-0590

To: ngEHT Group
From: J. Barrett
Subject: Simulated VDIF Recording Tests, June 2025

1 Objective

The objective of this test was to demonstrate the recording capability of the current Mark6+ hardware and to verify it is able to receive data at up to 64Gbps with an acceptable rate of data loss. The test environment was a typical computer lab at sea level kept at approximately 70°F. No attempt was made to run the recording system in a hypobaric chamber to mimic conditions at high altitude for this test.

2 Equipment and Setup

The equipment used for this test consisted of two Mark6+ recording devices (with expansion chassis), two complete sets of (four) disk modules, and a single pair of 100GbE transceivers with a short length (2ft) of optical fiber. One set of disk modules consisted of 32x 3.5" 20TB hard drives (Seagate ST20000NM007D), while the other set of modules consisted of 32x 2.5" 4TB solid-state hard drives (of the 32 SSDs, 16 were Samsung 870 and 16 were WD Red SA500).

The internal components of both Mark6+ devices used in this test are described in table 1.

Component	Quantity	**Description**
CPU	1	AMD EPYC 7282 16-Core Processor
Motherboard	1	Supermicro H12SSL-CT
RAM	8 × 16GB	DDR4 3200MHz 8x16 (Micron Tech.)
Network Card	1	Intel E810-C for QSFP (rev 02)
HBA	2	ATTO Technology H12F0GT 16-Port 12Gbps

Table 1: Mark6+ Server Components

Each Mark6+ was populated with four disk modules, the first with the Seagate HDD modules, and the second with the SSD modules, the top slot of each Intel e810 100GbE network card were connected together with the optical fiber/transceivers.

Figure 1: The back of the Mark6+ recorders, showing the HBA card with module cables and optical fiber connection (yellow cable to the right).

3 Software and Procedure

The base operating system installed on both Mark6+ units was Ubuntu 22.04 (server) and the cplane/dplane software installation used for this test was git revision `5bfed3ad` from the M6-clone repository[1]. The entire software installation and system configuration was done via ansible using the playbook file (`recorder-playbook.yml`) distributed with the code.

The recording process was handled using using cplane/dplane, while the simulated VDIF data was generated by the program `SimulateDeviceTX`. The program `SimulateDeviceTX` attempts to simulate VDIF data at the requisite rate by populating a VDIF packet with an appropriately configured and time-stamped header but an empty (zero'd) payload. The VDIF payload size was fixed at 8192 bytes (for a total packet size of 8274 bytes). The packet simulator program takes the configurations options described in table 2, and table 6 shows the commands used for each VDIF configuration.

Option	Description
–device-pcie-addr (-d)	device pcie address (default = 81:00.0)
–sock-mem (-s)	pre-allocated memory space (default = 4096 (MB))
–prefix (-x)	EAL file prefix (default = 2)
–threads (-t)	the number of fake VDIF threads to tag the data with (default = 2)
–rate-fraction (-f)	the fraction of the max link speed (between 0 and 1) default = 0.5
–packets-per-sec (-p)	N packets to send per second (overrides -f option)
–n-packets (-n)	send a fixed number of packets (default = 40000000)
–continuous (-c)	run continuously (conflicts with '-n' option)

Table 2: SimulateDeviceTX Options

Testing the Mark6+'s was accomplished by using one Mark6+ as the recording unit and the other as the packet generator. The basic setup procedure for a single test is as follows:

1. Load the recording Mark6+ and expansion chassis with the disk modules.
2. Ensure the cplane/dplane daemons are running
3. Key on the modules and configure the recording group with cplane. See figure 2.
4. Configure the `input_stream` with cplane. See figure 3 for the single VDIF stream configuration, and 4 for the dual VDIF stream configuration.
5. Log into the Mark6+ serving as the packet generator and start SimulateDeviceTX as follows:
   (a) Disable the dplane daemon with `sudo systemctl stop dplane`
   (b) Use `dpdk-devbind.py -s` to verify the e810 NIC is assigned to DPDK
   (c) If not, use `assign_dpdk_dev.py` to assign ownership of the e810 from the OS to DPDK
   (d) Start a `screen` session, and run `sudo SimulateDeviceTX` with the appropriate configuration parameters
   (e) Log out
6. Run the recording on the receiving Mark6+ for the specified duration using:`record=on:X`, where `X` is the duration in seconds.
7. Check the dplane log for recording size and the packet loss fraction.

For all of the recording test a total of four disk modules were used. They were configured and opened as a single recording group via the `da-client` interface using the commands in figure 2.

Once the recording group was opened, the input streams were configured to match the VDIF data coming from SimulateDeviceTX. To do this the expected VDIF thread ID(s) must be specified for the data interface. When recording a single VDIF stream the thread ID was assigned as '0', while for the dual streams, the IDs were '0' and '1'. The listing in figure 3 shows the input stream command used to configure a single VDIF stream, while the listing in figure 4 shows the dual stream command.

Once the recording group and streams were configured, three recording tests were executed, each for 60, 300, and 900 seconds duration. This was done via the `da-client` interface by issuing an timed recording with immediate start via the record command. For example, at 60 second recording would be run with `record=on:60`. Because full communication between cplane and dplane has not yet been implemented, following the completion of a recording, a `record=off` command was issued to update the cplane state.

Since normal OS-level network monitoring tools do not work for devices which have been captured by DPDK, the state of the recording system network interfaces was monitored by the program `MonitorStatus`. An example of the output of this program during a single-stream VDIF recording is shown in figure 5.

[1] `git@github.mit.edu:barrettj/M6-clone.git`

```
>> mod_init=1:8:HDD%0001:sg:new
<<  !mod_init=0:0;
>> mod_init=2:8:HDD%0002:sg:new
<<  !mod_init=0:0;
>> mod_init=3:8:HDD%0003:sg:new
<<  !mod_init=0:0;
>> mod_init=4:8:HDD%0004:sg:new
<<  !mod_init=0:0;
>> group=new:1234
<<  !group=0:0:1234;
>> group=open:1234
<<  !group=0:0:1234;
>> mstat?all
<<  !mstat?0:0:1234:1:HDD%0001/160000/4/8:8:8:159986:160000:open:ready:sg:1234:2:HDD%0002/160000/4/8:8:8:159986:160000:open:ready:sg
     :1234:3:HDD%0003/160000/4/8:8:8:159986:160000:open:ready:sg:1234:4:HDD%0004/160000/4/8:8:8:159986:160000:open:ready:sg;
```

Figure 2: Module setup

```
input_stream=add : RDBE1 : vdif : 8224 : 50 : 42 : dev2 : 172.16.3.16 : 4001 : 1234 : 0
<<  !input_stream=0:0;
>> input_stream=commit
<<  !input_stream=None:0;
```

Figure 3: Single stream `input_stream` configuration

```
input_stream=add : RDBE1 : vdif : 8224 : 50 : 42 : dev2 : 172.16.3.16 : 4001 : 1234 : 0,1
<<  !input_stream=0:0;
>> input_stream=commit
<<  !input_stream=None:0;
```

Figure 4: Dual stream `input_stream` configuration

```
name: mark6-4196
status: recording
device: dev2
   rx packets: 116027960
   rx bytes: 1027395297122
   dropped packets: 8146128
device: dev3
   rx packets: 0
   rx bytes: 0
   dropped packets: 0
vdif_time: dev2
       (id,   epoch, second,       frame                ~delta (s))
       (0,    50,    14401442,     683791,              -0.431088)
stack:
RDBE1_sg_block_stack_0: 3181
dev2_free_rx_block_stack1: 4095
dev2_free_rx_block_stack2: 4096
dev3_free_rx_block_stack1: 4096
dev3_free_rx_block_stack2: 4096
queue:
RDBE1_free_file_queue: 13
RDBE1_mbuf_queue_0: 0
RDBE1_sg_block_queue_0: 0
dev2_rx_block_queue: 0
dev3_rx_block_queue: 0
```

Figure 5: Example ouput from MonitorStatus

4 Results

The results of the recording tests are shown in tables 3 and 4. The first set of tests was performed using 32x Seagate 20TB HDDs, while the second set of tests was done using the 32 SSDs. (16x WD Red SA500 and 16x Samsung 870 4TB SDDs). For each test, the module type, duration, approximate data rate, and data loss fraction were determined. For all tests and configurations, the data loss rate was less than the nominal acceptable rate of 1e-4, except for a single 15 minute test done with the SSD modules (for which the data loss rate was inexplicably 10%). During a follow-up 15 minute recording test, the same SSD modules failed to repeat this behavior. This behavior is suspected to be related to the internal SSD caching mechanism, but more investigation will be needed to find the underlying cause. In general, with the exception of this outlier test, the use of SSD modules resulted in slightly less packet loss than the use of HDD based modules. However, it should also be noted that these test were done starting with completely empty modules, and that the performance of HDD modules may change depending on the write location (outer sectors should perform better than inner sectors). No attempt was made to determine the performance of HDDs modules when near their full capacity.The recording configuration (VDIF stream setup) for each set of tests is described in 5. It should be noted that because the packet generator software is running on a non-real-time OS, and does not have precise timing, the data amount and rate can vary from scan to scan. However, on average it is within a few percent of the specified data rate and serves well enough to approximate a backend.

Configuration	Module type	Duration (s)	Approx. Size (GB)	Approx. Rate (Gbps)	Packet loss
Config A	HDD	58.99	250.249	33.93	0.0
Config A	HDD	300.0	1245.3	33.2	7.3e-7
Config A	HDD	900.0	3725.7	33.1	9.6e-7
Config B	HDD	60.0	506.87	67.58	3.4e-5
Config B	HDD	300.0	2492.04	66.45	2.7e-5
Config B	HDD	900.0	7451.01	66.23	2.75e-5
Config C	HDD	60.0	502.52	67.0	8.08-5
Config C	HDD	300.0	2489.1	66.38	3.06-5
Config C	HDD	900.0	7458.05	66.29	2.32e-5

Table 3: Recording tests on Seagate 20TB HDDs

Configuration	Module type	Duration (s)	Size (GB)	Approx. Rate (Gbps)	Packet loss
Config A	SSD	60.0	250.47	33.39	0.0
Config A	SSD	300.0	1244.3	33.18	9.8e-7
Config A	SSD	900.0	3728.02	33.14	5.5e-7
Config B	SSD	60.0	504.12	67.22	3.5e-6
Config B	SSD	300.0	2486.41	66.3	3.69e-6
Config B	SSD	900.0	7455.82	66.27	5.06e-6
Config C	SSD	60.0	502.57	67.01	9.35e-6
Config C	SSD	300.0	2488.91	66.37	9.83e-6
Config C	SSD	600.0	4970.66	66.28	9.17e-6
Config C	SSD	900.0	7457.45	66.28	**0.119093**
Config C	SSD	900.0	7452.11	66.24	1.0e-5

Table 4: Recording tests on a combination of WD Red SA500 and Samsung 870 SSDs

Configuration Name	Description
Config A	Single VDIF stream 500k packets per sec (~32Gbps)
Config B	Dual VDIF stream 1M packets per sec (~64Gbps)
Config C	Single VDIF stream 1M packets per sec (~64Gbps)

Table 5: Configuration Details

To validate that the data-on-disk was recorded as expected, the program `sg_vcheck` was run to examine the VDIF headers of each scatter gather block across all file fragments on the disk modules. This program uses the block meta-data files (.mblk) to locate each 10MB block, and then checks the VDIF headers of each packet in the block to verify that the frame number count has incremented appropriately. This program takes a very long time to run, so it was only executed on the shortest (60s) scans to verify that the number of frame jumps and packet loss was in accordance with what was reported in the dplane logs.

Configuration Name	Command
Config A	`sudo ./SimulateDeviceTX -d 0000:82:00.0 -t 1 -p 500000 -c`
Config B	`sudo ./SimulateDeviceTX -d 0000:82:00.0 -t 2 -p 1000000 -c`
Config C	`sudo ./SimulateDeviceTX -d 0000:82:00.0 -t 1 -p 1000000 -c`

Table 6: SimulateDeviceTX Commands