INFORMATION ON „WIDEBAND“ VLBI OPERATION

Summaries for operating the Mark6 and FILA10G,
DiFX correlation, and extracting Phase Cal

Version 1.0
10 Feb 2015
Jan Wagner

1 CONTENTS

2 Location of Software and Tools 5

3 Details on VDIF 7

4 Details on Mark6 8

4.1 Mark6 Network Setup 9

4.2 Recording with Mark6 Software 10

4.2.1 Configuration Commands before Recording 10

4.2.2 Manual Recording and “record=” Command Syntax 11

4.2.3 Manual Recording via Script 11

4.2.4 Automated Recording using a “Field System” 12

4.2.5 Automated Recording based on VEX/XML without “Field System” 12

4.3 “Playback” with Mark6 Software 12

4.4 Recording setup with jive5ab software 13

4.5 Issues Recording VOA (also OCTA) data with Mark6 14

5 Useful tools 16

5.1 Inspect VDIF Data Files 16

5.2 Verify Backend Time Synchronization in Real-Time (*vdiftimeUDP*) 17

5.3 Verify VLBI Backend Data and Spectra 19

5.4 Verify R2DBE Data using Internal ADC Samples 20

6 Details on FILA10G 22

6.1 Setting up FILA10G-SA 22

6.1.1 Cabling 22

6.1.2 Firmware Upload after Power-On 22

6.1.3 USB Programming Cable Issues on Mark 6 24

6.1.4 FILA10G Serial Console 25

6.1.5 Configuration over Serial Console 25

6.2 Issues with the ADS sampler 26

6.3 Data Permutation on FILA10G (Sample Encoding Conversion, IF Interleaving) 27

6.3.1 Example `chan_perm`: Direct 128->128 bit copy with no permutation (default) 27

6.3.2 Example `chan_perm`: Change 2-bit sample encoding from VLBA/K5/Mark5B into VDIF 28

6.4 Configuration File for KVN (K/Q/W/D) 28

6.5 Future developments required for FILA10G integration 29

7 Details on R2DBE 31

7.1 Boot Process 31
7.2 Configuration after Booting 31
8 Phase Calibration Tone 32
8.1 Details 32
8.2 PCal Extraction in DiFX 34
8.3 Multi-Tone PCal Extraction from Files (m5pcal) 35
8.4 Single-Tone PCal Extraction from Files (m5tone) 35
9 Details on DiFX Correlation 36
9.1 DiFX Zoom Band Setting 36
9.2 DiFX FORMAT=VDIF or FORMAT=INTERLACEDVDIF 37
9.3 Correlating Multi-Threaded VDIF Data 37
9.4 Fixing VOA Issues 38
10 Details on H-Maser Rate Monitoring 39
11 APECS-Based alternative to Field System 40

LIST OF TABLES

Table 1 – Possible 10G network configuration on Mark6 10G, and backend 10G IPs as used at APEX........ 9
Table 2 - The Mark6 network configuration (/etc/network/interfaces) 9
Table 3 - Mark6 configuration commands. 11
Table 4 - Script to record with Mark6 software "immediately" for 30 seconds. 11
Table 5 – Example XML file produced by vex2xml.py and ready for recording with m6cc.py. 12
Table 6 – Example of automatic recording by a VLBI schedule using vex2xml.py and m6cc.py. 12
Table 7 – Decoding VDIF headers with vdifheader2.pl. 16
Table 8 - Checking the integrity of a VDIF file with vdifcontinuitycheck.py. 16
Table 9 – Example commands (script) to capture VLBI backend data and produce spectra. 19
Table 10 – Example DBBC configuration file (.conf). 24
Table 11 – Example FILA10G bit permutation settings to convert to VDIF 2-bit sample encoding. 28
Table 12 - VEX file entries for PCal extraction. 34
Table 13 - DiFX v2d file entry for PCal extraction. 34
Table 14 - Multi-tone (or single-tone) PCal extraction with m5pcal. 35

ACRONYMS

AF	Baseband signal that is the output of DDC/PFB processing
IF	Full bandwidth signal that is the input to DDC/PFB processing
DDC	Direct Digital Downconversion used for tunable AFs in a digital backend
PFB	Polyphase Filterbank used to get fixed AFs in a digital backend
PSN	Packet Sequence Number, sometimes added to start of UDP/IP before VLBI data
R2DBE	Roach2 Digital Backend (https://github.com/sma-wideband/r2dbe)
DBBC	Digital Base Band Converter (http://www.hat-lab.com/)
FILA10G	First/Last 10GbE, a VSI-to-10GbE converter board (http://www.hat-lab.com/)
VOA	VLBI Optical Adapter, a VSI-to-10GbE converter developed for NAOJ

(A number of publications have details on the above systems, search http://adsabs.harvard.edu)

2 LOCATION OF SOFTWARE AND TOOLS

The MIT Haystack Mark 6 software (***dplane, cplane***) is at:

Software is at	http://www.haystack.mit.edu/tech/vlbi/mark6/software.html
Memos are at	http://www.haystack.mit.edu/tech/vlbi/mark6/memo.html
Manuals are at	http://www.haystack.mit.edu/tech/vlbi/mark6/documentation.html

Much newer versions are usually not online but are available by email.
Contacts are Chet Ruszczyk (chester@haystack.mit.edu), Geoff Crew (gbc@haystack.mit.edu)

One can also use JIVE recording software (***jive5ab***) since the Mark 6 is a normal 10 GbE Linux computer:

Software is at http://www.jive.nl/~verkout/evlbi/

Contact is Harro Verkouter (verkouter@jive.nl)

The JIVE recording software is based on Metsähovi FlexBuff (***vlbi-streamer***) by Tomi Salminen.

Software is at	https://code.google.com/p/vlbi-streamer/
Documents at	http://www.jive.nl/nexpres/doku.php?id=nexpres:nexpres_wp8
Instructions at	https://code.google.com/p/vlbi-streamer/w/list

and https://deki.mpifr-bonn.mpg.de/@api/deki/files/5308/=vlbi_streamer_fieldnotes.txt
and https://deki.mpifr-bonn.mpg.de/02VLBI_Group/Group_Projects/PV_Upgrade

Note, *vlbi-streamer* seems to be buggy and not maintained anymore?
Contact is unknown

Additional software can be found in 'git' source code repositories at

https://bitbucket.org/jwagner313/vdifstream
VDIF UDP tools ***vdifstream, vdifsnapshotUDP, vdiftimeUDP, vdifheader2.pl***
VDIF tool ***vdifcontinuitycheck.py,*** is similar to vdiftimeUDP but is not real-time, and uses files
```
$ git clone https://git@bitbucket.org/jwagner313/vdifstream.git
```

https://bitbucket.org/jwagner313/kvnvdiftools
Generic VDIF tools, Mark 6, OCTA: ***modifyVDIF***, ***m5pcal***, ***m6sg_mount***, ***fuseMk6***, ***kvnVDIF2VDIF***, ***kvnMark5B2VDIF, recv_octa***, …
```
$ git clone https://git@bitbucket.org/jwagner313/kvnvdiftools.git
```

https://bitbucket.org/jwagner313/apex-tools
DiFX and Mark 6 zero baseline testing **scripts** and ***VEX/v2d***, and other things
```
$ git clone https://git@bitbucket.org/jwagner313/apex-tools.git
```

https://bitbucket.org/jwagner313/fila10gtools
Tools for FILA10G: ***consoleFILA10G.py*** and FILA10G setup scripts
```
$ git clone https://git@bitbucket.org/jwagner313/fila10gtools.git
```

Contacts are Jan Wagner (jwagner@kasi.re.kr, jan.wagner@iki.fi) for most of the tools, and Duk-Gyoo Roh (dgroh@kasi.re.kr) for OCTA tools (also on *polaris:~oper/bin/*)

Most backends produce data in standard VDIF format.

For the VDIF specification see http://vlbi.org/vdif/

DBBC/FILA10G firmware and documentation are available at http://www.hat-lab.com/
The most recent versions are available from Gino Tuccari (g.tuccari@ira.inaf.it).

R2DBE firmware and documentation are available from the EHT project wiki and
the SMA correlator repository https://github.com/sma-wideband/r2dbe

The ADS sampler and VOA VSI-to-10GbE converter documentation is available at ??
No firmware files are available.

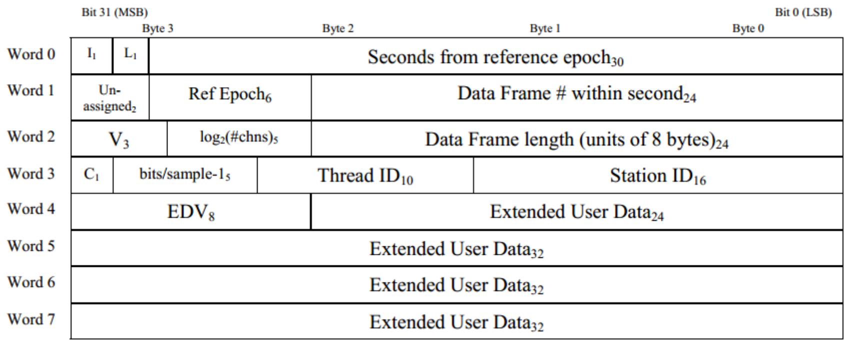
How to access ‘git’ repositories?
For tutorials please use Google, one tutorial is http://www.vogella.com/tutorials/Git/article.html.
Perhaps this summary also helps:

```
# The first time to get a new copy of a project:
$ cd /usr/local/src/
$ git clone <URL or location of git repository>

# Later to update your existing copy to most the recent version:
$ cd /usr/local/src/nameofproject/
$ git pull

# To submit/share changes you made to a file under /usr/local/src/nameofproject/:
$ cd /usr/local/src/nameofproject/
$ git commit -m “some message” <file> # copies modified file into local history
$ git push # uploads local history to server, if you have an account&permissions
```

3 DETAILS ON VDIF

VDIF is a standard file format for VLBI. Most backends can produce VDIF data. Software correlators can correlate station recordings that are in VDIF format. The VDIF specification is at http://vlbi.org/vdif/ and the current version is at http://vlbi.org/vdif/docs/VDIF_specification_Release_1.1.1.pdf.

The VDIF header is shown in Figure 1. The raw data VLBI comes immediately after the header.



Bit 31 (MSB) · Byte 3 · Byte 2 · Byte 1 · Byte 0 · Bit 0 (LSB)

Word 0	I_1	L_1	Seconds from reference epoch$_{30}$	
Word 1	Un-assigned$_2$	Ref Epoch$_6$	Data Frame # within second$_{24}$	
Word 2	V_3	log$_2$(#chns)$_5$	Data Frame length (units of 8 bytes)$_{24}$	
Word 3	C_1	bits/sample-1$_5$	Thread ID$_{10}$	Station ID$_{16}$
Word 4	EDV$_8$	Extended User Data$_{24}$		
Word 5	Extended User Data$_{32}$			
Word 6	Extended User Data$_{32}$			
Word 7	Extended User Data$_{32}$			

Figure 1 - VDIF Frame Header in VDIF Specification version 1.1.1. The subscripts are field lengths in bits. Bytes are relative byte addresses within 32-bit Words. All words are stored in Little Endian format. The header is either 16 bytes long (when flag bit L=1) or is 32 bytes long (when flag bit L=0). Invalid frames have flat bit I=1. The VLBI data are complex (real, imag) when C=1.

Due to historic reasons the total length of a frame (header + data) is given in 8 byte units.
That is, a value of 164 in the "Data Frame Length" field means that the frame is 1312 bytes long.

Time stamps in VDIF consist of a 30-bit count of *integer seconds* from a reference epoch ("Ref Epoch"). The *fractional seconds* are not given in the header, but can usually be determined using frame length and counter value "Data Frame #", provided that one knows the data rate (e.g., 8192 Mbit/).

At stations with multiple backends, or polarizations, or simultaneous frequency bands, etc., one can assign each of them a different VDIF "Thread ID". This produces multi-threaded VDIF.

4 Details on Mark6

Mark 6 system:

Motherboard (P9X79 series) based on X79 chipset, quad-channel DDR3 bus for ~40 GByte/s.	
Two 1 GbE interfaces	
Four 10 GbE interfaces	can be copper, or optical (SFP+ transceiver modules)
Two SATA controller cards	16 SATA disks behind each card (32 disks total)

Four modules can be attached, each with 8 disks.
Each module has 2 MiniSAS connectors.
With 6 TB disks[1] the maximum data capacity is 4 modules x 8 disks x 6 TB = 192 TB.
At 8 Gbit/s (~1 GByte/s) a 192 TB disk configuration can fit roughly 50 hours of data.

Naming:

"module"	a disk pack with up to 8 disks, module has a serial number ("VSN")
"module group"	one, two, three, or four modules combined for recording

"**Getting started with your Mark6**" describes cables, connectors, how to start Mark6, see:
http://www.haystack.mit.edu/tech/vlbi/mark6/documentation.html
https://deki.mpifr-bonn.mpg.de/@api/deki/files/5110/=Getting_started_with_your_Mark_6-1.01.pdf

The main Mark6 software is *dplane* ("data plane") and *cplane* ("control plane").
The Mark 6 dplane software can record to disk in one of two modes:

RAID0
Erase and format disks using for example:

```
mod_init=1:8:VSN00123:raid:new;
mod_init=2:8:VSN00124:raid:new;
group = new:12;
```

Good: Easy to access complete files after recording, no extra steps necessary.
Bad: If even one disk fails, the file system is gone. Possible to recover data but challenging.

Scatter-Gather ('sg')
Erase and format disks using for example:

```
mod_init=1:8:VSN00123:sg:new;
mod_init=2:8:VSN00124:sg:new;
group = new:12;
```

Good: Even with slow disks or dead disks, recording and reading is still possible.
Bad: Less convenient to read. Every recorded file is split into several fragments, one fragment per disk. Fragments contain some metadata instead of just plain VDIF, meaning these fragments cannot be directly processed in e.g. DiFX. Fragments must be combined back into a single file with a special tool (*gather*, *vdifuse, fuseMk6*).

[1] Helium filled consumer SATA drives (Ultrastar He6; 6 TB) were tested at the APEX (sub)millimeter telescope at an altitude of 5100 meters. Disks worked fine during ~60 hours of recording and playback in 01/2015 fringe tests. They did not need the pressurized housing that normal consumer SATA drives require at this altitude.

Functionality: by now (01/2015) most of the Mark6 Command Set is implemented and mostly works as in the Mark6 documents. Some features require newer software, e.g.:

> Recent *dplane/cplane* (01/2015; not public, ask G. Crew or C. Ruszczyk) correctly support
> `input_stream = add:… : <ethX> : <ip> : <port> : <groupY>;`
> In the recent version this should now correctly record data from ethX onto groupY.
> 1) If KVN 22/43/86/129 GHz are separated onto 10 GbE eth2/3/4/5, the new software version can record these bands onto their own groups, e.g., onto modules 1/2/3/4 separately.
> 2) APEX R2DBE and DBBC sample two Nyquist zones that can now be recorded to their own groups, rather than onto the same group where they are separable only by VDIF Thread ID.

4.1 Mark6 Network Setup

Each 10 GbE interface should be on its own logical network. Otherwise there can be Linux IP routing problems. One possible configuration for the 10 GbE interfaces is:

Interface	IP address range	Own IP	Network and broadcast	(Backend IP; at APEX)
eth2	10.10.1.1 -- 10.10.1.14	10.10.1.1	net .0 bcast .15	(10.10.1.5)
eth3	10.10.1.17 -- 10.10.1.30	10.10.1.17	net .16 bcast .31	(10.10.1.21)
eth4	10.10.1.33 -- 10.10.1.46	10.10.1.33	net .32 bcast .47	(10.10.1.37)
eth5	10.10.1.49 -- 10.10.1.62	10.10.1.49	net .48 bcast .63	(10.10.1.53)
netmask 255.255.255.240 (10.10.1.xx/28), mtu 9000				

Table 1 – Possible 10G network configuration on Mark6 10G, and backend 10G IPs as used at APEX.

The above specifies four networks with a 16-address ranges each. The /etc/network/interfaces is:

```
auto lo eth0 eth2 eth3 eth4 eth5
iface lo inet loopback

# eth0: APEX Ctrl network
iface eth0 inet static
  address 10.0.2.103  netmask 255.255.255.0
  gateway 10.0.2.1    dns-nameservers 10.0.6.19

# iface eth1: unused

# 10 GbE interfaces
iface eth2 inet static
   address 10.10.1.1 netmask 255.255.255.240 network 10.10.1.0
   broadcast 10.10.1.15 mtu 9000
iface eth3 inet static
   address 10.10.1.17 netmask 255.255.255.240 network 10.10.1.16
   broadcast 10.10.1.31 mtu 9000
iface eth4 inet static
   address 10.10.1.33 netmask 255.255.255.240 network 10.10.1.32
   broadcast 10.10.1.47 mtu 9000
iface eth5 inet static
   address 10.10.1.49 netmask 255.255.255.240 network 10.10.1.48
   broadcast 10.10.1.63 mtu 9000
```

Table 2 - The Mark6 network configuration (/etc/network/interfaces)

The 10G interface configuration on VLBI backends (DBBC/FILA10G, R2DBE) should match the configuration used on the Mark 6.

Of the 1G interfaces one is connected to the telescope control network. The second 1G interface is free to be used for something else. At APEX:

10.0.2.103	vlbi1.apex-telescope.org	eth0 of Mark6 recorder #1
10.0.2.104	vlbi2.apex-telescope.org	eth0 of Mark6 recorder #2

At the KVN:

1 Gbit/s links for SSH/terminal remote use, etc

TODO	?.kvn.kasi.re.kr	1G eth0 of Mark6 recorder #1 at Tamna
TODO	?.kvn.kasi.re.kr	1G eth0 of Mark6 recorder #1 at Ulsan
TODO	?.kvn.kasi.re.kr	1G eth0 of Mark6 recorder #1 at Yonsei

10 Gbit/s links to DiFX computing cluster

TODO	?.kvn.kasi.re.kr	10G eth? of Mark6 recorder #1 at Tamna
TODO	?.kvn.kasi.re.kr	10G eth? of Mark6 recorder #1 at Ulsan
TODO	?.kvn.kasi.re.kr	10G eth? of Mark6 recorder #1 at Yonsei

4.2 Recording with Mark6 Software

As user 'oper', start in this order:

```
terminal1> dplane 3     # starts C program that records from 10 GbE to disk
terminal2> cplane -l 0  # starts Python server for VSI-S and diskpack metadata
```

Manual commands can be given in a console that is started with:

```
terminal3> da-client    # allows user command input (VSI-S commands)
```

To kill programs, in this order:

```
terminal4> ecplane   # terminates cplane
```

(Note: a bugfixed version of ecplane is required, 08Dec2014 or later, and for now you can get it from https://bitbucket.org/jwagner313/kvnvdiftools/raw/master/scripts/ecplane.sh)

```
terminal4> dboss t 1 # termninates dplane
```

4.2.1 Configuration Commands before Recording

Typical VSI-S commands to configure the Mark 6 for recording are:

```
mod_init?;
group=open:12;   (or, to erase/format a new group, see commands on page 7)

<Any extra commands dependent on backend, and maybe manual test recordings>

<Example for VOA:  has no packet seq number, and fixed VDIF 1312-byte frame>
input_stream=add:voa:vdif:1312:42:0:eth2:10.10.1.5:60000:12;
input_stream=commit;
input_stream?;

<Example for R2DBE: has 8-byte PSN, adjustable VDIF size, here 8224 byte>
input_stream=add:r2dbe0:vdif:8224:50:0:eth2:10.10.1.5:4001:1;
input_stream=add:r2dbe1:vdif:8224:50:0:eth3:10.10.1.21:4001:2;
```

```
input_stream=commit;
input_stream?;

<Example for FILA10G: has 8-byte PSN, adjustable VDIF size, here 8032>
input_stream=add:fila10g:vdif:8032:50:0:eth2:10.10.1.5:46227:1;
input_stream=add:fila10g:vdif:8032:50:0:eth3:10.10.1.21:46227:2;
input_stream=commit;
input_stream?;

group=close:12;
group=unmount:12;
```

Table 3 - Mark6 configuration commands.

Undocumented features:

1) When *cplane* is started it will try to execute Mark6 commands found in `/opt/mit/mark6/etc/m6_commands`
2) The *cplane* also attempts to load an XML "configuration" (syntax undocumented) `/opt/mit/mark6/etc/m6_config.xml`

(Did not test if any of these are implemented fully or correctly in Mark6 software of 01/2015)

4.2.2 Manual Recording and "record=" Command Syntax

For the syntax of the record= command please see the Mark 6 Command Reference.
Currently (01/2015) the following syntax works:

```
record=2015y017d06h00m00s:600:300:017-0600:t15017:AP;
```

This records `600` data seconds starting at VDIF header timestamp `2015y017d06h00m00s`. The number `300` is the estimated final recording size in GByte. The name of the scan is `017-0600`, the experiment `t15017`, and the station code is `AP`. Recording produces file fragments that are called:

```
/mnt/disks/[1-4]/[0-7]/t15017_AP_017-0600.vdif
```

File fragments contain a Mark6 scatter-gather header, VDIF data, and additional metadata every 10MB. Thus these files cannot be read with VDIF tools such as the utilities in the mark5access library.

4.2.3 Manual Recording via Script

This below script (*mk6rec.sh*) tells Mark6 to start recording after 5 seconds, and record for 30 seconds.

```
#!/bin/bash
#  Usage:   mk6rec.sh [<mark 6 ip address>]

MK6_IP=$1
if [ "$1" == "" ]; then
    MK6_IP=127.0.0.1
fi

# Get UT time 5 seconds in the future, in VEX format (like
2015y017d06h00m00s)
vextime=`date -u --date='5 seconds' +%Yy%jd%Hh%Mm%Ss`
scanname=`date -u --date='5 seconds' +%j-%H%M`

# Use netcat (nc) to send Mark 6 the command to record 30 seconds
cmd="record=$vextime:30:30:$scanname:test:KT;\n"
echo "Sending $cmd to Mark6 at $MK6_IP"
echo -e $cmd | nc -w 1 $MK6_IP 14242
```

Table 4 - Script to record with Mark6 software "immediately" for 30 seconds.

4.2.4 Automated Recording using a “Field System”

The system that processes the VEX file describing the observation should send “`record=…`” commands to the Mark 6 (TCP port 2620). In fact, at the start of an observation, the system should also send the correct setup commands of 4.2.1. At the end of the observation, it could also send the `close` and `unmount` commands of 4.2.1 so that the modules can be removed.
The current APEX Field System is configured to control one Mark5C and “drudg”ing the VEX schedule file using script *drudgc.pl* will produce .prc/.snp that have recording commands in Mark5C command format. Recording on the APEX Mark 6’s is currently not driven by Field System.

4.2.5 Automated Recording based on VEX/XML without “Field System”

There are two scripts by G. Crew that conveniently provide timed recording of scans listed in an XML file, without any Field System control of the Mark 6. In case the scripts (vex2xml.py, m6cc.py) are not yet part of the Mark6 official distribution, an early version used in 01/2015 can also found at:

https://bitbucket.org/jwagner313/apex-tools/raw/master/Mark6/

The script *m6cc.py* provides timed recording. It requires an XML file that can be created manually, or that can be derived from a VEX file with *vex2xml.py*. An example XML file is shown below:

```
<experiment name="t15017" station="AR" start="2015017100500" end="2015017110000">
  <scan experiment="t15017" source="SGRA" station_code="AR" start_time="2015017100500"
   duration="300" scan_name="017-1005"/>
  <scan experiment="t15017" source="SGRA" station_code="AR" start_time="2015017101500"
   duration="300" scan_name="017-1015"/>
  <scan experiment="t15017" source="SGRA" station_code="AR" start_time="2015017102500"
   duration="300" scan_name="017-1025"/>
</experiment>
```

Table 5 – Example XML file produced by vex2xml.py and ready for recording with m6cc.py.

Note that Mark 6 programs *dplane* and *cplane* must be running, and the basic configuration as in 4.2.1 must have been completed (manually, or automatically) before starting *m6cc.py*. An example:

```
$ vex2xml.py -f t15017.vex  AP # convert schedule into XML for station “AP”

$ da-client # manually enter some preparatory commands, for example:
   group=open:12;
   mstat?
   input_stream=add:r2dbe1:vdif:8224:50:0:eth3:10.10.1.21:4001;
   input_stream=commit;
   input_stream?

$ m6cc.py -f t15017.xml -r 4096 # begin automated recording, 4096 Mbit/s
```

Table 6 – Example of automatic recording by a VLBI schedule using vex2xml.py and m6cc.py.

4.3 “Playback” with Mark6 Software

The Mark6 uses normal Linux file systems. Thus there is no actual “playback”. All data can be accessed directly (in comparison, on Mark 5A/B/B+/C the file systems were proprietary and recordings were not accessible from Linux directly, only via the StreamStor PCI-X board and buggy closed-source drivers).

To assemble a scatter-gather recording scattered over file fragments

```
/mnt/disks/[1-4]/[0-7]/t15017_AP_017-0600.vdif
```

you can use Mark6 utility *gather*. An example:

```
$ gather /mnt/disks/[1-4]/[0-7]/t15017_AP_017-0600.vdif \    # input files
                        /mnt/RAID/ t15017_AP_017-0600.vdif       # new output file
```

Another option (from 12/2014) is to use Mark6 scatter-gather FUSE filesystem(s). This avoids having essentially two copies of the same data. Current Mark6 FUSE implementations are:

1) *fuseMk6* : A very basic implementation, part of a library, also found in DiFX:
https://bitbucket.org/jwagner313/kvnvdiftools/raw/master/mark6_scattergather/
2) *vdifuse* : An optimized much faster implementation by Geoff Crew, now found in DiFX as well:
https://svn.atnf.csiro.au/difx/applications/m6support/trunk/

Both Mark6 FUSE implementations provide normal file access to the scatter-gather fragments, while the assembly of fragments is done on-demand in the background, in main memory rather than on disk.
For help on how to use *vdifuse* (preferred, as it is faster than *fuseMk6*) see the output of

```
$ vdifuse -h
```

4.4 Recording setup with jive5ab software

Harro Verkouter had some emails to summarize the setup steps. The information is collected at:
https://bitbucket.org/jwagner313/kvnvdiftools/raw/master/jive5ab/flexbuf-recording.txt

Recording uses same remote user commands as the Mark 6 (telnet to port 2620):

```
record = on : scan_name;
```

Data will be saved into file fragments like these:

```
/mnt/disk[0-31]/scan_name/scan_name.00000001
```

where the number (00000001 and increasing) means this is the Nth fragment of the entire complete file. Fragment are 256MB in size, but this can also be changed.

Unlike the Mark6 scatter-gather format, the file fragments created during *jive5ab* recording do not contain any metadata. Instead the file fragments are plain VDIF.
There are several advantages to using *jive5ab* instead of Mark6 software:

1) A complete file can be assembled without extra tools, simply by

```
$ cat /mnt/disk[0-31]/scan_name/scan_name.*   > /mnt/disk[0-31]/scan_name.vdif
```

2) A FUSE file system is included with jive5ab, thus a complete file could also be “presented” to user applications via the FUSE layer, if necessary.

3) Fragments are normal VDIF files. Thus the DiFX correlator utility *directory2filelist* can detect the start and stop times of every fragment. Similarly in the SFXC software correlator.

Note that DiFX/SFXC process all files that have time stamps in the range of the current scan. Thus they load all files in correct order. Assembling them into a single file is unnecessary!

Essentially, *jive5ab* recordings can be correlated directly, without requiring FUSE or *gather*/*cat*.

4.5 Issues Recording VOA (also OCTA) data with Mark6

The VOA system seems to produce non-standard VDIF in several ways.

Problem 1: VOA frame size too short for 4 Gbit/s or faster recording

Issue: A VDIF frame size >5000 byte is required for lossless 10 GbE capture at 4 Gbit/s and faster. This is because of CPU load and 10G NIC interrupt rate limits in current computer architectures. The VOA firmware has a fixed 1312-byte VDIF frame size, probably from the time before 10 Gbit Ethernet.

Solution: Modify VOA firmware.
However, it seems there will never be a future VOA firmware upgrade.

Problem 2: VOA "VDIF" has wrong header Endianness, wrong sample encoding

Issue: The VOA "VDIF" header has Big Endian byte order but standard VDIF Little Endian[2]. This is a problem for VOA, because Mark6 recording uses a UT start time and duration:

"record=2015y017d06h00m00s:600:600:017-0600:t15017:AP;"

With this recording command the Mark6 waits until the incoming 10 GbE VDIF data reached a timestamp of 2015y017d06h00m00s. The Mark 6 then records data until the VDIF timestamp changes to later than this UT time + 600 seconds. Because the VOA "VDIF" header has a wrong Endianness, the timestamp is incorrect. The Mark 6 then starts or stops recording VOA data at some "random" time.

Solution 1: Fix bugs in VOA firmware so that it produces correct VDIF data.
However, it seems there will never be a future VOA firmware upgrade.

Solution 2: modify Mark6 *d-plane* source code. Have it convert VDIF timestamps into Little Endian.

In file `dplane.h` add `#include <endian.h>`
In file `spooler.c` search for `glob.stream[kstr].t_vdif = pvdh->seconds;` and change it into

```
// VOA bug fix:
// fix Endiannes of first 32-bit header data (Invalid:1bit,Legacy:1bit,Seconds:30bit)
uint32_t* hdr32 = (uint32_t*)pvdh;
pvdh[0] = be32toh(pvdh[0]);
glob.stream[kstr].t_vdif = pvdh->seconds;
```

This allows the Mark6 to correctly detect the (now converted/fixed) timestamp in the VOA "VDIF" data. Recording should now start and stop at the specified UT time.

Note: The modified *d-plane* version will not recognize standard VDIF.
Note 2: You can have two *dplane*'s on the Mark 6, one for VOA and another for VDIF. Place the VOA-modified *dplane* into its own directory. For example */usr/local/bin/VOA/dplane*. Remember to start this special *dplane* program when using VOA! (Simply renaming *dplane* into *dplane-voa* does not work, because the Mark6 *cplane* software looks for a running process with exactly the name *dplane)*

[2] VDIF Specification 1.1 -- http://vlbi.org/vdif/docs/VDIF_specification_Release_1.1.1.pdf

Problem 3: Other corrupt VDIF header fields, wrongly formatted VLBI payload data.

Issue: The VOA data recorded successfully on Mark5C/Mark6/OCTA still has errors in the VDIF headers. It also has partly wrong VLBI payload data. This is because VOA data is not converted in real-time at the time of recording.

Solution 1: For Mark 6 or Mark 5C files: Run *kvnVDIF2VDIF* to convert from VOA “VDIF” into actual VDIF. If you find that some header fields are still incorrect (wrong number of channels, wrong number of bits per sample, data marked as Invalid, etc) you can adapt *kvnVDIF2VDIF.c* sources or run *modifyVDIF*.

Solution 2: For copying recordings from an OCTA disk system onto a Linux file system (OCTA output data use the VOA format): The new program *recv_octa* (11/2014) updated by Dr. DG Roh and JH Yeom can now do the required conversion during copying.

Solution 3: (Not yet done, and not recommended by people on the DiFX Users mailing list: add a new format called “VOA” into the mark5access library. This library is used by DiFX to decode many formats like VLBA, MarkIV, Mark5B, VDIF/VDIFC/VDIFL/VDIFCL, D2K, and KVN5B. A new format would help for direct correlation in DiFX – but not in SFXC, nor for processing in any VDIF utilities like spectrometers).

5 USEFUL TOOLS

5.1 INSPECT VDIF DATA FILES

Tools in the DiFX mark5access library are useful to check recordings in multiple formats, including also in VDIF format. Two useful tools to check VDIF integrity are *m5time* and *m5test*. Spectra of the recorded data can be produced with *m5spec* (it outputs spectra as a text file that can be plotted in *gnuplot*). Slightly higher level checks can be done with DiFX *countVDIFPackets* and *searchVDIF*.

For a low-level look at VDIF data, verify for example that the header information looks correct, one can use *vdifheader2.pl* found at https://bitbucket.org/jwagner313/vdifstream/raw/master/vdifheader2.pl and refer to the VDIF specifications at http://vlbi.org/vdif/

```
$ vdifheader2.pl /scratch0/n14st02c_8Gbps_fringetest/Ky/p14sl01c-No00035-
06h50m12s-VOA.vdif | less
Using VDIF (litte-endian) unpack format
Reading /scratch0/n14st02c_8Gbps_fringetest/Ky/p14sl01c-No00035-06h50m12s-
VOA.vdif

 I:0 L:0 1-ch 2-bit 1312-byte Th:4 EP:29 T:10392613 sec F#58270 06:50:13
 I:0 L:0 1-ch 2-bit 1312-byte Th:3 EP:29 T:10392613 sec F#58271 06:50:13
 I:0 L:0 1-ch 2-bit 1312-byte Th:1 EP:29 T:10392613 sec F#58271 06:50:13
 I:0 L:0 1-ch 2-bit 1312-byte Th:2 EP:29 T:10392613 sec F#58271 06:50:13
 I:0 L:0 1-ch 2-bit 1312-byte Th:4 EP:29 T:10392613 sec F#58271 06:50:13
 I:0 L:0 1-ch 2-bit 1312-byte Th:3 EP:29 T:10392613 sec F#58272 06:50:13
 I:0 L:0 1-ch 2-bit 1312-byte Th:1 EP:29 T:10392613 sec F#58272 06:50:13
 I:0 L:0 1-ch 2-bit 1312-byte Th:2 EP:29 T:10392613 sec F#58272 06:50:13
 I:0 L:0 1-ch 2-bit 1312-byte Th:4 EP:29 T:10392613 sec F#58272 06:50:13
 I:0 L:0 1-ch 2-bit 1312-byte Th:3 EP:29 T:10392613 sec F#58273 06:50:13
 I:0 L:0 1-ch 2-bit 1312-byte Th:1 EP:29 T:10392613 sec F#58273 06:50:13
 I:0 L:0 1-ch 2-bit 1312-byte Th:2 EP:29 T:10392613 sec F#58273 06:50:13
 I:0 L:0 1-ch 2-bit 1312-byte Th:4 EP:29 T:10392613 sec F#58273 06:50:13
...
```

Table 7 – Decoding VDIF headers with vdifheader2.pl.

To look for missing data frames in VDIF files one can use *vdifcontinuitycheck.py* found at https://bitbucket.org/jwagner313/vdifstream/raw/master/vdifcontinuitycheck.py:

```
$ vdifcontinuitycheck.py /scratch0/n14st02c_8Gbps_fringetest/Ky/p14sl01c-No00035-06h50m12s-VOA.vdif

Thread 3 Second 10392613 : 141720 frames : #58271--#199999 : 9 lost, 0 out-of-order, 0 invalid, of 141729
total
Thread 1 Second 10392613 : 141725 frames : #58271--#199999 : 4 lost, 0 out-of-order, 0 invalid, of 141729
total
Thread 2 Second 10392613 : 141720 frames : #58271--#199999 : 9 lost, 0 out-of-order, 0 invalid, of 141729
total
Thread 4 Second 10392613 : 141725 frames : #58270--#199999 : 5 lost, 0 out-of-order, 0 invalid, of 141730
total
Thread 3 Second 10392614 : 200000 frames : #0--#199999 : 0 lost, 0 out-of-order, 0 invalid, of 200000 total
Thread 1 Second 10392614 : 200000 frames : #0--#199999 : 0 lost, 0 out-of-order, 0 invalid, of 200000 total
Thread 2 Second 10392614 : 200000 frames : #0--#199999 : 0 lost, 0 out-of-order, 0 invalid, of 200000 total
Thread 4 Second 10392614 : 200000 frames : #0--#199999 : 0 lost, 0 out-of-order, 0 invalid, of 200000 total
Thread 3 Second 10392615 : 200000 frames : #0--#199999 : 0 lost, 0 out-of-order, 0 invalid, of 200000 total
Thread 1 Second 10392615 : 200000 frames : #0--#199999 : 0 lost, 0 out-of-order, 0 invalid, of 200000 total
Thread 2 Second 10392615 : 199999 frames : #0--#199999 : 1 lost, 0 out-of-order, 0 invalid, of 200000 total
Thread 4 Second 10392615 : 200000 frames : #0--#199999 : 0 lost, 0 out-of-order, 0 invalid, of 200000 total
Thread 3 Second 10392616 : 200000 frames : #0--#199999 : 0 lost, 0 out-of-order, 0 invalid, of 200000 total
Thread 1 Second 10392616 : 200000 frames : #0--#199999 : 0 lost, 0 out-of-order, 0 invalid, of 200000 total
Thread 2 Second 10392616 : 200000 frames : #0--#199999 : 0 lost, 0 out-of-order, 0 invalid, of 200000 total
Thread 4 Second 10392616 : 200000 frames : #0--#199999 : 0 lost, 0 out-of-order, 0 invalid, of 200000 total
…
```

Table 8 - Checking the integrity of a VDIF file with vdifcontinuitycheck.py.

5.2 Verify Backend Time Synchronization in Real-Time (*vdiftimeUDP*)

Once a backend (DBBC/FILA10G, R2DBE) has been synchronized to GPS 1PPS, the offset between the backend "UT" time (VDIF frame timestamps) and the current UT time should be small (<< 1 ms).

The UT time offset can be checked with *vdiftimeUDP* at https://bitbucket.org/jwagner313/vdifstream.

```
# R2DBE backend data stream, look at timestamps in VDIF Thread #0
oper@Mark6-4031:~$ vdiftimeUDP --cpu=2 --offset=8 -t 0 4001

# DBBC/FILA10G data stream, look at timestamps in VDIF Thread #0
oper@Mark6-4032:~$ vdiftimeUDP --cpu=2 --offset=8 -t 0 46227
```

The program runs on **Mark6** and reports the difference between computer UT time (NTP-synchronized) and the VDIF timestamps in the 10 GbE UDP stream. Other statistics such as data loss are also shown.

The program is intended to help answer the question "Does the current VDIF Data Second match the current UT second?" It is ***not*** intended for tracking the sub-second clock offset between the GPS 1PPS and the VLBI backend 1PPS ("GPS minus FMout")! For that you need a counter such as the HP 53131A.

Settings in *vdiftimeUDP*:

-c|`--cpu=<n>`	bind to CPU core n, can reduce packet loss on Mark 6 at >4 Gbit/s
-o|`--offset=<n>`	discards an n-byte PSN that is added by R2DBE and DBBC/FILA10G
-t|`--thread=<n>`	selects VDIF Thread #n for VDIF-vs-Computer UT time comparison
-s|`--saveto=<fn>`	writes the first VDIF frame of a second into the specified file 'fn'
-b|`--bigendian`	required for VOA VSI-to-10GbE converter data
`46227`	UDP port of VDIF data stream (usually: R2DBE: 4001, DBBC: 46227).

Typical output:

```
----- VDIF Time ------------------------------ Computer Time ---------Time Delta----Frames total/max-----Rate peak------Rate nominal----Rate actual---
VDIF frame#157037 2014y287doy 03:00:23 : PC 2014y287doy 03:00:23.79 :  -0.812s : 1/157038 fps : 4x1608.07 Mbps :    0.01 Mbps : 1296.00 Mbps
VDIF frame#0      2014y287doy 03:00:24 : PC 2014y287doy 03:00:24.00 :  -0.001s : 200000/200000 fps : 4x2048.00 Mbps : 2073.60 Mbps : 2073.63 Mbps
VDIF frame#0      2014y287doy 03:00:25 : PC 2014y287doy 03:00:25.00 :  -0.002s : 200000/200000 fps : 4x2048.00 Mbps : 2073.60 Mbps : 2073.63 Mbps
VDIF frame#0      2014y287doy 03:00:26 : PC 2014y287doy 03:00:26.00 :  -0.001s : 200000/200000 fps : 4x2048.00 Mbps : 2073.60 Mbps : 2073.65 Mbps
VDIF frame#0      2014y287doy 03:00:27 : PC 2014y287doy 03:00:27.00 :  -0.003s : 200000/200000 fps : 4x2048.00 Mbps : 2073.60 Mbps : 2073.55 Mbps
VDIF frame#0      2014y287doy 03:00:28 : PC 2014y287doy 03:00:28.00 :  -0.001s : 200000/200000 fps : 4x2048.00 Mbps : 2073.60 Mbps : 2073.64 Mbps
VDIF frame#0      2014y287doy 03:00:29 : PC 2014y287doy 03:00:29.00 :  -0.001s : 200000/200000 fps : 4x2048.00 Mbps : 2073.60 Mbps : 2073.61 Mbps
…
```

Fields shown in the output are

VDIF time	the most recent VDIF timestamp to 1 second accuracy
Computer Time	the computer UT time, reported by the Linux *gettimeofday()* function
Time Delta	time difference 'dT = VDIF - Computer Time' at VDIF seconds rollover
Frames total	number of frames actually received in previous VDIF data second
Frames max	number of frames expected in previous VDIF data second
Rate peak	expected goodput per VDIF thread ("Number of threads x Rate")
Rate nominal	expected throughput (overhead included, like VDIF frame headers)
Rate actual	measured data rate during the previous VDIF data second, somewhat imprecise as *gettimeofday()* is used for the receiver-side time delta

The computer must be NTP-synchronized. NTP over the Internet is accurate to a few milliseconds. Due to cumulative latencies a 'Time Delta' in *vdiftimeUDP* of ~50 milliseconds is still normal.

Troubleshooting:

<u>Issue 1:</u> *vdiftimeUDP* cannot see any UDP packets

Check that data are really incoming on the Mark6:

```
      $ sudo tcpdump -enqti eth2 -c 10   # dump 10 packets from first 10 GbE interface
      $ sudo tcpdump -enqti eth3 -c 10   # dump 10 packets from second 10 GbE
```

The UDP destination IP & MAC that *tcpdump* reports must exactly match with the IP & MAC of the Mark6 10 GbE interface. If not, fix the backend network settings, or correct the Mark6 network settings.

<u>Issue 2:</u> *vdiftimeUDP* reports time offsets of >0.05 seconds

You can re-do the 1PPS sync & UT time synchronization on the respective backend.
At APEX make sure the Mark 6 is not accidentally using the APEX NTP server that runs on TAI time. The time offset could be very large; until June 2015 the TAI is ahead of UT already by +35 leap seconds.

In case of APEX DBBC2 an approximate verification of UT time sync is also possible by comparing LEDs on the FILA10G board against UT time roll-over. One LED will light on every 10th UT second.

<u>Issue 3:</u> *vdiftimeUDP* reports time offsets and lost frame counts that vary significantly

This happens if the VLBI backend sends several VDIF Threads to the same destination IP and UDP port. This kind of multi-threaded VDIF is sometimes not handled very well by *vdiftimeUDP*. You should use *vdiftimeUDP* option `--thread=n` to select just one specific thread for the VDIF-vs-UT time comparison.

5.3 Verify VLBI Backend Data and Spectra

Several quick tests can be done on data captured with *vdifsnapshotUDP*

https://bitbucket.org/jwagner313/vdifstream/raw/master/vdifsnapshotUDP.c

Example:

```
oper@Mark6-4031:~$ vdifsnapshotUDP
Usage: vdifsnapshotUDP [--cpu=<0..31>] [--offset=<n bytes to skip in UDP>]
                       <size in Mbyte> <port> <output file.vdif>

# R2DBE backend data stream, capture 128MB, remove 8-byte PSN
oper@Mark6-4031:~$ vdifsnapshotUDP --cpu=2 --offset=8 128 4001 capture.vdif
```

The program makes a short burst-mode data capture into Mark6 main memory and writes it to a file after the capture. It does not need any disk modules nor the Mark6 *dplane/cplane/da-client* software. The captured data can be inspected in DiFX and mark5access tools like *m5spec* and *m5bstate*.

The snapshots can be checked in *m5bstate* to find potential errors such as “stuck bits” on the VSI buses. One can also produce spectra form the captured data.

Example scripts are at https://bitbucket.org/jwagner313/apex-tools/raw/master/Correlation/scripts/ :

`m5bspec_test32.sh`	capture + spectrum of DBBC VDIF, 2 x 16 x 32 MHz firmware
`m5bspec_test62.5_DBBC.sh`	capture + spectrum of DBBC VDIF, 2 x 16 x 62.5 MHz firmware
`m5bspec_test_R2DBE.sh`	capture + spectrum of R2DBE VDIF, 1 x 2048 MHz, one IF
`m5bspec_test_R2DBE_dualIF.sh`	capture + spectrum of R2DBE VDIF, 2 x 2048 MHz, two IFs

The DBBC/FILA10G VDIF data in some cases has multiple VDIF Threads. To be able to use mark5access utilities (m5spec, …) these threads have to be separated into their own files. Use *modifyVDIF* found at:

https://bitbucket.org/jwagner313/kvnvdiftools/

Example: make short 128MB recording and produce baseband spectra

```
# Make a 128MB snapshot of DBBC data, DBBC running 2 x 16x62.5 MHz PFB firmware:
vdifsnapshotUDP --cpu=2 --offset=8 128 46227 m5spec_test.vdif # capture 128 MB of data

# The DBBC produces multi-threaded VDIF (Nyquist 1: thread 0, Nyquist 2: thread 1).
# Need to extract threads to be able to use mark5access ‘m5spec’!
modifyVDIF --extract=0 m5spec_test.vdif m5spec_test_t0.vdif   # extract thread 0
modifyVDIF --extract=1 m5spec_test.vdif m5spec_test_t1.vdif   # extract thread 1

# Make spectra using the captured data:
m5spec -nopol m5spec_test_t0.vdif VDIF_8000-4000-16-2 8192 2000 m5spec_test_t0.m5spec
m5spec -nopol m5spec_test_t1.vdif VDIF_8000-4000-16-2 8192 2000 m5spec_test_t1.m5spec

# Plot the ASCII files .m5spec in Octave and write plot into a PostSCript file
echo "m5specGlueplot('m5spec_test_t0.m5spec')"> ./m5spec_test.m
octave ./m5spec_test.m
echo "m5specGlueplot('m5spec_test_t1.m5spec')"> ./m5spec_test.m
octave ./m5spec_test.m
rm ./m5spec_test.m
```

Table 9 – Example commands (script) to capture VLBI backend data and produce spectra.

The above example produces two plots, one of which is shown below.

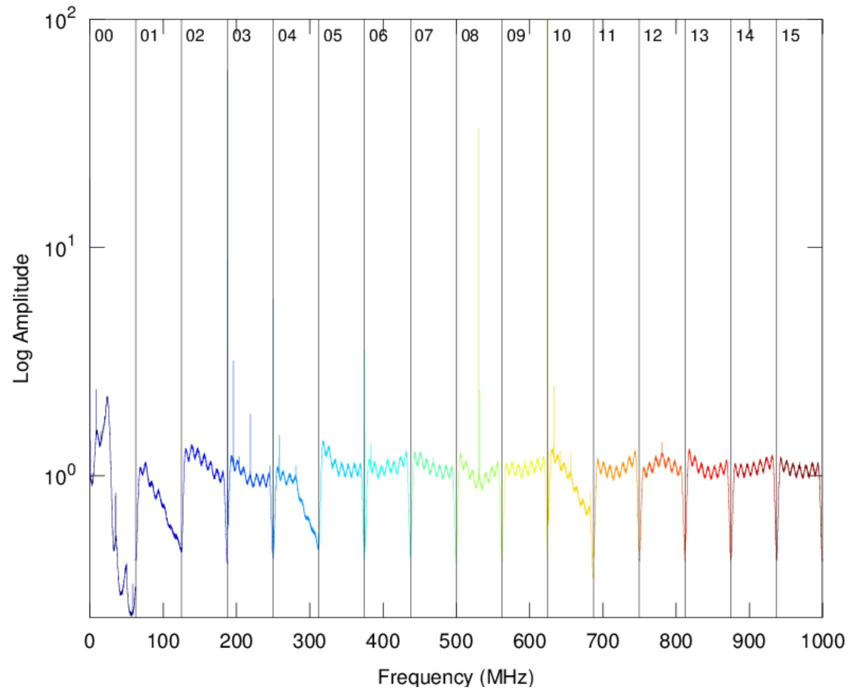


Figure 2 - Example output of vdifsnapshotUDP and m5spec based quick look at backend data and spectrum.

Such quick bust-mode VDIF captures (works also at >8 Gbit/s) are useful to check that the backend is configured correctly, look correct, and the bandpasses and tones (PCal) or spectral lines look fine.

5.4 Verify R2DBE Data using Internal ADC Samples

An R2DBE spectrometer script that uses 8-bit samples from the ADC directly, and involves no capture of packets coming off the R2DBE, can be found at https://bitbucket.org/jwagner313/apex-tools/raw/master/R2DBE/ and also https://github.com/sma-wideband/r2dbe.

```
oper@Mark6-4031:~$ /etc/r2dbe/quickspecR2DBE.py [<number of time integrations Nint>]
```

The script displays spectra of ADC input IF0 and IF1. It is useful to quickly check the bandpass shape and overall data quality. The R2DBE always returns a fixed number of ADC samples at once (256k samples?).

To increase SNR on spectral lines one can use `Nint>1` to time-average several such 256k sample blocks.

5.5 Mark5C Data Capture without using Disk Module

The program *mk5netdump* captures 10 GbE data directly into memory, rather than writing to a diskpack.

This is similar to the *vdifsnapshotUDP* program except that *mk5netdump* is only for Mark5C, as it reads data through the Amazon PCI-X card and 10G daughterboard, rather than via a real 10 GbE network card.

The program is found at https://bitbucket.org/jwagner313/apex-tools/raw/master/Mark5/mk5netdump

```
# Usage
mk5netdump <packetsize> <dataoffset> <filename> <bytes>
   packetsize   : 5008 bytes to extract
   dataoffset  : 40   bytes after start of the Ethernet frame
   filename    : file name and path to write to
   bytes       : number of bytes to capture into file

# Example for RDBE, and FILA10G Mark5B-format mode
$ mk5netdump 5008 40 test.m5c $((16*1024*1024))
```

The captured file is in whatever VLBI format the backend is producing, possibly Mark5B or VDIF. The captured data can be checked with *m5spec*, *m5bstate* and other mark5access utilities.

6 DETAILS ON FILA10G

Firmware (*fila10g_v*.bit*) and documentation (*DBBC2 FiLa10G command set v*.pdf*, *FILA10G registers v*.pdf*) and) is available from Gino Tuccari (g.tuccari@ira.inaf.it).

There are currently no First-Time User Guides or Installation Guides.

For help on "what to connect where" you could ask Gino Tuccari or Michael Wunderlich (mwunderlich@mpifr-bonn.mpg.de).

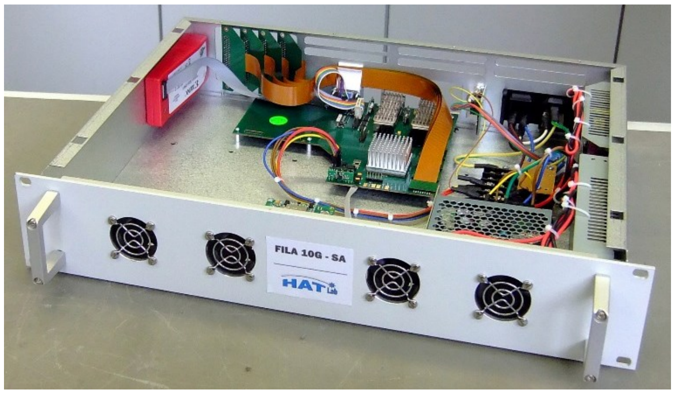


Figure 3 – The FILA10G-SA. Photo from hat-lab.com.

6.1 SETTING UP FILA10G-SA

6.1.1 Cabling

Connectors are indicated in Figure 4. Attach: Power --- Serial cable to DB9, or Ethernet cable to RJ45 if remote console is desired --- Add 10G transceivers (10GBASE-SR XFP) and SR fibers --- Attach USB --- Connect an active GPS antenna if a GPS module is installed (SMA connector) --- Connect necessary VSI cables. Note: USB and serial console must be connected to a permanently installed computer!

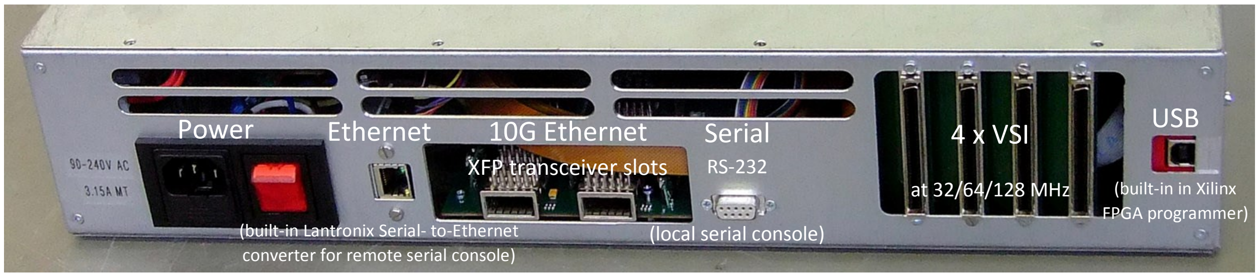


Figure 4 - Rear connectors on a FILA10G-SA without the GPS Module option. Photo from hat-lab.com.

Mark 6 issue: The Linux kernel crashes if FILA10G USB is connected to any Mark 6 USB 3.0 port. Use the legacy USB 2.0 port if you are on Mark 6!!

6.1.2 Firmware Upload after Power-On

The FILA10G uses Xilinx FPGA technology. Every time you switch on the FILA10G remember to program the firmware (.bit file) again[3].

Use the USB connector on the FILA10G. It is an USB-based FPGA programmer similar to that in Figure 5 and is built into the FILA10G-SA. You need to install a *free* software from the Xilinx support site at

http://www.xilinx.com/support/download

The software is either "Standalone Programming Tools" or "Lab Tools".

Figure 5 - The Xilinx Platform Cable USB II. This "Xilinx cable" is built into FILA10G-SA and the FILA10G USB port seen in Figure 4 is actually this programmer.

[3] Usually FPGA firmware is stored in a separate memory chip. However, FILA10G 1) lacks the necessary Xilinx Platform Flash chip, 2) does have a Xilinx ACE chip for firmware storage on CompactFlash (in theory) but even Xilinx technical support failed to get this to work. Thus FILA10G always loses its firmware when power is switched off.

The software includes *iMPACT*. Firmware programming[4] with *iMPACT* is very easy. Many tutorials are found on the Internet, and some videos:

https://www.youtube.com/results?search_query=xilinx+impact

iMPACT offers a graphical interface and a command line mode. In Linux under LabTools v14.1 (for example) you can program the FILA10G from the command line using:

```
$ . /opt/Xilinx/14.1/LabTools/settings64.sh
$ impact -batch impact.cmd
```

where file 'impact.cmd' contains the actual commands

```
# See Xilinx documentation for details on the commands
setMode -bs
setCable -port auto
Identify
IdentifyMPM
assignFile -p 1 -file /etc/fila10g/v331-eng/fila10g_v3.3.1_1_011014.bit
erase -p 1
program -p 1
quit
```

FILA10G Firmware Upload:

For FILA10G-SA:

1) Turn on FILA10G-SA power,
2) Run `impact -batch impact.cmd` similar to the above,
3) Make sure step 2 finished with a "`Programmed successfully.`"

For DBBC FILA10G:

The correct firmware bundles for Core2+FILA10G are loaded automatically at the start of DBBC software. On the APEX DBBC desktop there are some shortcuts. Current versions:

v15A = 2000 MHz PFB as 16 x 62.5 MHz ("ALMA compatible") + FILA10G 125 MHz VSI
v15F = 2048 MHz PFB as 16 x 64 MHz or full band + FILA10G 128 MHz VSI

An example `C:\DBBC_CONF\dbbc_poly_config_v15A.conf` is shown below:

```
20 dbbc2_pfb_v15A.bit   # firmware to load onto Core2 board 0
21 dbbc2_pfb_v15A.bit   # firmware to load onto Core2 board 1
99 dbbc2_pfb_v15A.bit   # 99: skip
99 dbbc2_pfb_v15A.bit   # 99: skip
99 ACE.bit              # firmware for Xilinx ACE chip
2 fila10g_v3.3.2_1.bit COM1 # firmware for FILA10G (125 MHz VSI version)
2 28000     # target counts for AGC on PFB channels of Core2 board PFB
2 28000     # target counts for AGC on PFB channels of Core2 board PFB
0 48000     # ignored
0 48000
0 48000
0 48000
0 48000
0 48000
67 69 0 0   # ADC data vs. Core2 board 0 and 1 FPGA DCM clock phasing
```

[4] Xilinx calls the task of FPGA "firmware programming" by several names: "configuration of the FPGA", "configuration download", "bitstream download", and possibly other names.

```
CAT2 2000   # Timing board v2 (CAT2), synthesize a clock of 2000 MHz
```

Table 10 – Example DBBC configuration file (.conf).

In the DBBC configuration file in Table 10 the clock phasings (67, 69) depend on the individual DBBC and are the result of DBBC "phase calibration". The process is described in the APEX Project Book and DBBC Users Guide.

6.1.3 USB Programming Cable Issues on Mark 6

There are two fixable issues when using FILA10G + Xilinx tools with Mark 6.

Issue 1: Mark 6 crashes if FILA10G USB is connected

Solution: The Linux kernel used on Mark 6 (kernel 2.6.32) is quite old and has USB 3.0 bugs that crash the system when FILA10G is connected to any of the USB 3.0 ports of the Mark 6. Connect FILA10G only to the "legacy" USB 2.0 port if you are on Mark 6!

Issue 2: FILA10G USB not detected in Xilinx LabTools / Programming Tools

Solution: Xilinx' installer for Linux is buggy. The same problem and a solution are reported under http://forums.xilinx.com/t5/Configuration/installing-platform-cable-USB-II-ubuntu/m-p/66729#M324

First, if program `/sbin/fxload` is missing on the Mark 6 then install it together with libusb :

```
    $ sudo aptitude install fxload libubusb-dev build-essential
```

Next, edit file /etc/udev/rules.d/xusbdfwu.rules and change $TEMPNODE into $tempnode, and BUS into SUBSYSTEM. The final file should be as below. Then reboot.

```
SYSFS{idVendor}=="03fd", SYSFS{idProduct}=="0008", MODE="666"

SUBSYSTEMS=="usb", ACTION=="add", SYSFS{idVendor}=="03fd", SYSFS{idProduct}=="0007",
RUN+="/sbin/fxload -v -t fx2 -I /usr/share/xusbdfwu.hex -D $tempnode"

SUBSYSTEMS=="usb", ACTION=="add", SYSFS{idVendor}=="03fd", SYSFS{idProduct}=="0009",
RUN+="/sbin/fxload -v -t fx2 -I /usr/share/xusb_xup.hex -D $tempnode"

SUBSYSTEMS=="usb", ACTION=="add", SYSFS{idVendor}=="03fd", SYSFS{idProduct}=="000d",
RUN+="/sbin/fxload -v -t fx2 -I /usr/share/xusb_emb.hex -D $tempnode"

SUBSYSTEMS=="usb", ACTION=="add", SYSFS{idVendor}=="03fd", SYSFS{idProduct}=="000f",
RUN+="/sbin/fxload -v -t fx2 -I /usr/share/xusb_xlp.hex -D $tempnode"

SUBSYSTEMS=="usb", ACTION=="add", SYSFS{idVendor}=="03fd", SYSFS{idProduct}=="0013",
RUN+="/sbin/fxload -v -t fx2 -I /usr/share/xusb_xp2.hex -D $tempnode"

SUBSYSTEMS=="usb", ACTION=="add", SYSFS{idVendor}=="03fd", SYSFS{idProduct}=="0015",
RUN+="/sbin/fxload -v -t fx2 -I /usr/share/xusb_xse.hex -D $tempnode"
```

The FILA10G USB ("Xilinx Platform Programming Cable II USB") has its *own firmware* (different from FILA10G .bit firmware files), and it has firmware files /usr/share/xusb_* that are part of the Xilinx software. Now when the FILA10G USB is attached the xusbdfwu.rule should make sure the correct USB firmware is uploaded so iMPACT can later upload FILA10G .bit firmware.
If the USB firmware upload fails, iMPACT cannot detect a Xilinx cable. In this case look at `lsusb` output,

```
oper@Mark6-4012:~$ lsusb | grep Xilinx
Bus 005 Device 003: ID 03fd:0013 Xilinx, Inc.
```

and upload USB firmware manually by these instructions:

```
# Find Xilinx device on USB bus
$ lsusb | grep Xilinx
Bus 005 Device 003: ID 03fd:0013 Xilinx, Inc.

# Upload firmware using Bus number (here:005) and Device number (here:003)
$ /sbin/fxload -v -t fx2 -I /usr/share/xusb_xp2.hex \
      -D /dev/bus/usb/005/003
```

Directly after this start iMPACT, it should now detect the Xilinx cable.

6.1.4 FILA10G Serial Console

The FILA10G accepts user commands over a serial port (19200,n,1). Commands are described in the *DBBC2 FiLa10G command set* PDF. The FILA10G has just one free serial port and it is available through one connector on the FILA10G box.

A. DB-9 connector: on-board RS232 serial is internally connected directly by a cable.
   To control FILA10G from a laptop or a computer (e.g., Mark6) without old DB9 serial ports you should attach a USB-to-Serial converter cable (e.g, cables based on PL2303) to your computer.

B. RJ-45 Ethernet connector: via a Serial-to-Ethernet converter inside the enclosure
   The first time the FILA10G is installed one must configure the Serial-to-Ethernet converter using http://www.lantronix.com/device-networking/utilities-tools/device-installer.html

Note: To change between DB9 and RJ45, you need to do some re-cabling inside the FILA10G.

Serial port is at:

For DBBC/WinXP: serial console (A) is 'COM1'
For Linux: serial console (A) is probably at /dev/ttyUSB1
Network: remote console (B) is at the IP and port configured via Lantronix DeviceInstaller

The remote serial console (B) can be accessed with for example *telnet*.

The real serial console (A) can be accessed with for example *minicom*, or *screen*, but it is more comfortable to use the Python script consoleFILA10G.py.

```
$ consoleFILA10G.py --cmd=/etc/fila10g/cmds/DBBC_ALMA_2000MHz_2if.cmd /dev/ttyUSB0
```

TODO: consoleFILA10G.py console runs on Windows and Linux but currently works only with local serial ports. Could modify it a bit to allow remote TCP e.g. *tcp://10.0.2.99:1234* as well.

6.1.5 Configuration over Serial Console

On the APEX DBBC: the WinXP desktop on APEX DBBC has some .bat scripts that you can run to configure FILA10G to the correct settings. There is also a shortcut to start consoleFILA10G.py that both provides a console as well as executes a ready .cmd file with FILA10G configuration commands.

Typical commands in a configuration file for FILA10G are:

```
# This file can be executed by the consoleFILA10G.py program
# that offers both command upload and console.
```

```
#
# The commands set up the DBBC2-internal FILA10G. Settings include
# VDIF format, time sync using the DBBC2-internal GPS module,
# and the IP addresses (source, destination) for streaming
# the VDIF data over 10GbE to the Mark6.
#
# These settings are for an ALMA-style 2-IF x 16-channel PFB x 62.5 MHz
# wide frequency band allocation. The DBBC2 PFB channels however do
# not have the 50% overlap of the ALMA channel layout (N. Pradel)

reboot

# Switching to VSI1/VSI2(2Gbps) or VSI1-2(4Gbp)
# APEX DBBC with FILA10G firmware v2.0 takes clock from VSI2
# so we cannot use this FILA10G to stream VSI1-only data

inputselect vsi1-2
vsi_samplerate 125000000
splitmode on
reset

# The following line (with two # comment markers!)
# ## auto_timesync_fila10g
# would be interpreted by consoleFILA10G.py itself (not FILA10G) and causes
# program consoleFILA10G.py to handle FILA10G synchronization to the computers
# current UT time. It does this by waiting and sending some commands to FILA10G.
#
# However, for the APEX DBBC/FILA10G we have connected the GPS module and a GPS antenna.
# In this case the GPS time sync command on the FILA10G can be used:
timesync

# Wait a bit longer since ALMA-compat. firmware plus GPS timesync seems a bit slower...
wait
wait

# APEX Mark6 4 x 10GbE have been configured to have four subnets on 10.10.1.x,
# namely 10.10.1.1/28 and 10.10.1.16/28 and 10.10.1.32/28 and 10.10.1.48/28
# For DBBC just the first two will be used.
# Because the ARP handling in FILA10G is not yet able to properly split
# the last IP number into subnets, we just use /27 to span across both subnets...

tengbcfg eth0 ip=10.10.1.5 nm=27
destination 0 10.10.1.1

tengbcfg eth1 ip=10.10.1.21 nm=27
destination 1 10.10.1.17

vdif_station ap
vdif_frame 2 16 8000
start vdif

tengbinfo eth0
tengbinfo eth1
sysstat

# For Mark5C recording ARP should be off, but for Mark6 it is
# okay to have APRs enabled - and is actually quite helpful!
arp on
```

6.2 Issues with the ADS sampler

Issue 1: The ADS 2-bit samples are in VLBA/K5/MarkIV encoding, but VDIF has a different encoding.

VLBA	{-3.3359 V, +1.0 V, -1.0 V, +3.3359 V} for 2-bit samples {0b00, 0b01, 0b10, 0b11}
VDIF	{-3.3359 V, -1.0 V, +1.0 V, +3.3359 V} for 2-bit samples {0b00, 0b01, 0b10, 0b11}

The DBBC backend has a command ('`enc=vdif`' (?)) to change to the VDIF 2-bit encoding before writing samples onto the VSI bus. These VSI data are ready to be sent over 10 GbE with the FILA10G.

The ADS cannot do such an internal conversion. However, the only difference in the encodings is that the 2 bits are mirrored, a task that can be performed on the FILA10G via the `chan_perm` register bank. Details are given in section 6.3.

Issue 2: Potential FILA10G issue when using several ADS samplers

Usually FILA10G receives VSI data directly from the DBBC over 2…4 VSI buses. The VSI data clocks and 1PPS in this case are mutually well synchronized. Cable delay differences are <<1 nanosecond.

If the FILA10G's VSI inputs are fed by 4 different ADS samplers, there might be relative offsets in VSI data clocks and 1PPS's that are large enough (>1 ns) to require some alignment logic in the FILA10G. It is not clear whether such alignment logic *actually* exists in current FILA10G firmware. TODO: Check!

6.3 Data Permutation on FILA10G (Sample Encoding Conversion, IF Interleaving)

The FILA10G has a register bank (`chan_perm`) that changes the order of VSI input data before. This is useful when 4 VSI inputs carry single-channel wideband data, and you want to combine all 4 VSI inputs into a 4-channel VDIF output. Multi-threaded single-channel VDIF causes some problems in DiFX, but rearranging the input data via `chan_perm` allows using more common and problem free single-threaded multi-channel VDIF. In addition `chan_perm` can be used to convert the 2-bit sample encoding from a VLBA/Mark5B encoding into the correct VDIF encoding.

The command syntax is

```
        regwrite chan_perm <N> <0x(idx3)(idx2)(idx1)(idx0) e.g. 0x03020100>
```

The permutator inputs one 128-bit data word (4 x 32-bit VSI). It outputs rearranged 128-bit data. Each register `N` chooses four output bits out of the 128 input bits, with indices as follows:

VSI Input:	VSI In #1	VSI In #2	VSI In #3	VSI In #4
Bit numbers: Indices:	0-31 0x00-0x1F	32-64 0x20-0x3F	64-95 0x40-0x5F	96-127 0x60-0x7F

Output bits:	0-3	4-7	… (4N to 4N+3) …	124-127
Selected by register `N`:	N=0	N=1	… (N) …	N=31

For example: `regwrite chan_perm 1 0x07060504` copies input bits 0x04 to 0x07 (=VSI1 bits 4 to 7) into output bits 4 to 7 (N=1) in original order. Using `0x04050607` reverses the bit order. Example 2: `regwrite chan_perm 0 0x61412101` packs the second bit from each VSI input into output bits 0 to 3 (N=0).

You can use the Octave/Matlab script `chanpermCalc.m` to generate `regwrite` commands.
https://bitbucket.org/jwagner313/fila10gtools/raw/master/chanpermCalc.m
Some examples FILA10G commands can be found in .cmd files under
https://bitbucket.org/jwagner313/fila10gtools/raw/master/cmds/v331-eng
A few additional `chan_perm` examples are given below.

6.3.1 Example **`chan_perm`**: Direct 128->128 bit copy with no permutation (default)

This are the default FILA10G `chan_perm` register values. They keep VSI input data in original order.

```
regwrite chan_perm 0 0x03020100
regwrite chan_perm 1 0x07060504
regwrite chan_perm 2 0x0B0A0908
regwrite chan_perm 3 0x0F0E0D0C
regwrite chan_perm 4 0x13121110
regwrite chan_perm 5 0x17161514
regwrite chan_perm 6 0x1B1A1918
…
regwrite chan_perm 29 0x77767574
regwrite chan_perm 30 0x7B7A7978
regwrite chan_perm 31 0x7F7E7D7C
```

6.3.2 Example `chan_perm`: Change 2-bit sample encoding from VLBA/K5/Mark5B into VDIF

The VDIF specification uses 2-bit sample encoding that is different from earlier VLBI formats.

VLBA	{-3.3359 V, +1.0 V, -1.0 V, +3.3359 V} for 2-bit samples {0b00, 0b01, 0b10, 0b11}
VDIF	{-3.3359 V, -1.0 V, +1.0 V, +3.3359 V} for 2-bit samples {0b00, 0b01, 0b10, 0b11}

The following swaps every 2 bits and accomplishes a conversion to VDIF 2-bit sample encoding:

```
regwrite chan_perm 0 0x02030001
regwrite chan_perm 1 0x06070405
regwrite chan_perm 2 0x0A0B0809
regwrite chan_perm 3 0x0E0F0C0D
regwrite chan_perm 4 0x12131011
regwrite chan_perm 5 0x16171415
regwrite chan_perm 6 0x1A1B1819
regwrite chan_perm 7 0x1E1F1C1D
…
```

Table 11 – Example FILA10G bit permutation settings to convert to VDIF 2-bit sample encoding.

6.4 Configuration File for KVN (K/Q/W/D)

This FILA10G configuration file is an example for 4-band FILA10G output at 8 Gbps.

```
# Configuration for Four-IF input with single-channel IF's on each VSI.
# The configuration will group the IF samples to produce 4-IF VDIF data.
# This configuration also applies a 2-bit sample conversion.
# Produces VDIF with 8032 byte/frame, 8000-byte payload.

# Configure
stop
reset
arp on

# Mark6 4 x 10GbE have been configured to have four subnets on 10.10.1.x/240 (nm=28)
# Because the ARP handling in FILA10G is not yet able to properly split the
# last IP number into subnets, we just use nm=26 to span across all subnets
tengbcfg eth0 ip=10.10.1.5 nm=26
tengbcfg eth1 ip=10.10.1.21 nm=26

# Stream IP address destination
destination 0 10.10.1.1:46227
destination 1 10.10.1.17:46227

inputselect vsi1-2-3-4
vsi_samplerate 64000000
splitmode off
reset

# Input:
#  VSI1 carries 16 samples of IF1, VSI2 carries 16 samples of IF2,
#  VSI3 carries 16 samples of IF3, VSI4 carries 16 samples of IF4
#  = 128 bit that are not in {t0/IF1,t0/IF2,t0/IF3,t0/IF4, t1/IF1,T1/IF2, ...} order
#    but rather in {t0/IF1,...,t15/IF1, t0/IF2,..,t15/IF2, ...} order
#
# Interleave the data to group together the samples of the 4 IFs:
#  Input 128-bit data
#  VSI1: bits 0-31, VSI2: bits 32-63, VSI3: bits 64-95, VSI4: bits 96-127
#       0x00-0x1F         0x20-0x3F        0x40-0x5F         0x60-0x7F
#  Output 128-bit data
#
# Also do 2-bit sample encoding conversion (exchange sign, mag bits)
regwrite chan_perm 0 0x20210001
regwrite chan_perm 1 0x60614041
regwrite chan_perm 2 0x22230203
regwrite chan_perm 3 0x62634243
regwrite chan_perm 4 0x24250405
regwrite chan_perm 5 0x64654445
regwrite chan_perm 6 0x26270607
regwrite chan_perm 7 0x66674647
regwrite chan_perm 8 0x28290809
regwrite chan_perm 9 0x68694849
regwrite chan_perm 10 0x2A2B0A0B
regwrite chan_perm 11 0x6A6B4A4B
regwrite chan_perm 12 0x2C2D0C0D
regwrite chan_perm 13 0x6C6D4C4D
regwrite chan_perm 14 0x2E2F0E0F
regwrite chan_perm 15 0x6E6F4E4F
regwrite chan_perm 16 0x30311011
regwrite chan_perm 17 0x70715051
regwrite chan_perm 18 0x32331213
regwrite chan_perm 19 0x72735253
regwrite chan_perm 20 0x34351415
regwrite chan_perm 21 0x74755455
regwrite chan_perm 22 0x36371617
regwrite chan_perm 23 0x76775657
regwrite chan_perm 24 0x38391819
regwrite chan_perm 25 0x78795859
regwrite chan_perm 26 0x3A3B1A1B
regwrite chan_perm 27 0x7A7B5A5B
regwrite chan_perm 28 0x3C3D1C1D
regwrite chan_perm 29 0x7C7D5C5D
regwrite chan_perm 30 0x3E3F1E1F
regwrite chan_perm 31 0x7E7F5E5F
reset

vdif_station KT
vdif_frame 2 4 8192 ct=off
regupdate vdif_header 2 0x02000000 0x1F000000
regupdate vdif_header 3 0x04000000 0x7F000000

# No GPS module is installed for UTC timing so have to use
# the consoleFILA10G.py Computer-to-FILA10G timesync feature.
## auto_timesync_fila10g

start vdif
```

Figure 6 - FILA10G configuration file example for KVN 4-band data output at 8 Gbps.

6.5 Future developments required for FILA10G integration

The FILA10G should be automatically configured based on the VEX file. Mainly,

1) How many and which VSI inputs are active (FILA10G: `input_select <vsi*>`),
2) How many channels (AF's) there are in total and if corner-turning is necessary (`vdif_frame <bit/sample> <nchan> <payloadlen> <ct=on|off>`),
3) What the VSI input clock rate is (`vsi_samplerate <n Hz>`),
   vsi_samplerate = (AF bandwidth (Hz) * 2 * 2 bit/sample * number of AFs) / 32 bit
4) In case of VSI with single-channel wideband data: how the data should be converted and interleaved (`regwrite chan_perm <n> <perm>`) to get a single-threaded multi-channel VDIF output – at least until DiFX is updated to better handle multi-threaded VDIF.

One can create *.cmd* files with the complete FILA10G configuration for specific cases. The correct *.cmd* file can be executed manually on the FILA10G (copy&paste), or via `consoleFILA10G.py` or similar.

Unfortunately command files are quite specific. For example one *.cmd* can be specific to "firmware 3.3.1 + single-channel 512 MHz wideband 2-bit sampled data on each VSI + use VSI 1 & 2 (K&Q band) only + interleave single-channel VSI 1 & 2 into standard VDIF multi-channel order + convert 2-bit VLBA encoded samples produced by ADS sampler into VDIF 2-bit encoded samples".

It would be easier if the VEX file were used to derive the correct FILA10G commands for a particular observation. For example, perhaps not all bands need to be recorded.

7 DETAILS ON R2DBE

Steps for setting up an R2DBE environment from scratch can be found on the EHT wiki:
http://eht-wiki.haystack.mit.edu/Event_Horizon_Telescope_Home/Technical_Development/R2DBE/Setup_of_fresh_Roach2_as_R2DBE
A number of diagnostic tools, self-zero-baseline test, and ADC time offset alignment are described in:
http://eht-wiki.haystack.mit.edu/Event_Horizon_Telescope_Home/Technical_Development/R2DBE/R2DBE_Usage
Also see the APEX Project Book, in particular on the SMA input connector assignments.

7.1 BOOT PROCESS

When the R2DBE powers up, it starts a built-in bootloader (uBoot) that can boot from SD card (not installed at APEX), from a USB stick (not used), or from the 1 GbE network connected to the "PPC" network port. Network boot is used at APEX. The Mark6-4031 serves the root file system.

The Mark6 settings to supply R2DBE with things required for network booting (DHCP/BOOTP, NFS server) can be found on the EHT wiki page in general (see link above). Configuration files adapted for APEX are found on both Mark6's and there is a copy under
https://bitbucket.org/jwagner313/apex-tools/raw/master/R2DBE/

The R2DBE accepts SSH login as 'root', without a password. However, usually it is not necessary to run programs on the R2DBE itself. Instead, a server process on the R2DBE (tcpborphserver3) listens for remote commands. These commands are sent to the R2DBE over the KATCP protocol.

7.2 CONFIGURATION AFTER BOOTING

Every time after successfully booting the R2DBE, the actual FPGA firmware has to be loaded, and the ADC, 10GbE, real-time clock, and VDIF packetizer have to be configured. A configuration script found in the SMA correlator repository (https://github.com/sma-wideband/r2dbe) has been adapted for APEX. To configure the APEX R2DBE, run:

```
oper@Mark6-4031:~$ /etc/r2dbe/APEX_config.py
```

The same APEX_config.py script performs 1PPS and UT time synchronization as part of the configuration. Currently there is no separate light-weight script to renew just the time synchronization.

Before and during an observation it is necessary to readjust 8-bit to 2-bit quantization threshold settings to get an optimal (Gaussian) 2-bit sample distribution (e.g., 16%:34%:34%:16%). The adjustment can be done automatically (without recording and looking at VDIF data) by running:

```
oper@Mark6-4031:~$ /etc/r2dbe/requantizeR2DBE.py
```

8 PHASE CALIBRATION TONE

8.1 DETAILS

Electronic variations in telescope instrumentation cause time-varying delays to the astronomical signal. In some systems there are also variations in the frequency response of the analog signal paths (mainly in older systems), or even in digital processing (e.g., phase flip mistakes like in the DBE digital backend).

To calibrate those variations, Phase Calibration (PCal) tones are injected into the analog IF, or into the receiver at sky frequency. They fall into the VLBI recorded bands. Tone amplitudes and phases are usually extracted by DiFX during every integration period. There are also simple extraction utilities that work without DiFX. There are roughly two kinds of VLBI PCal:

single-tone	only one tone per AF, or only one tone in the entire IF
multi-tone	several tones in one AF; important for Geodetic VLBI bandwidth synthesis

An example spectrum of one VLBI 16 MHz wide AF is show in Figure 7 and contains multi-tone PCal.

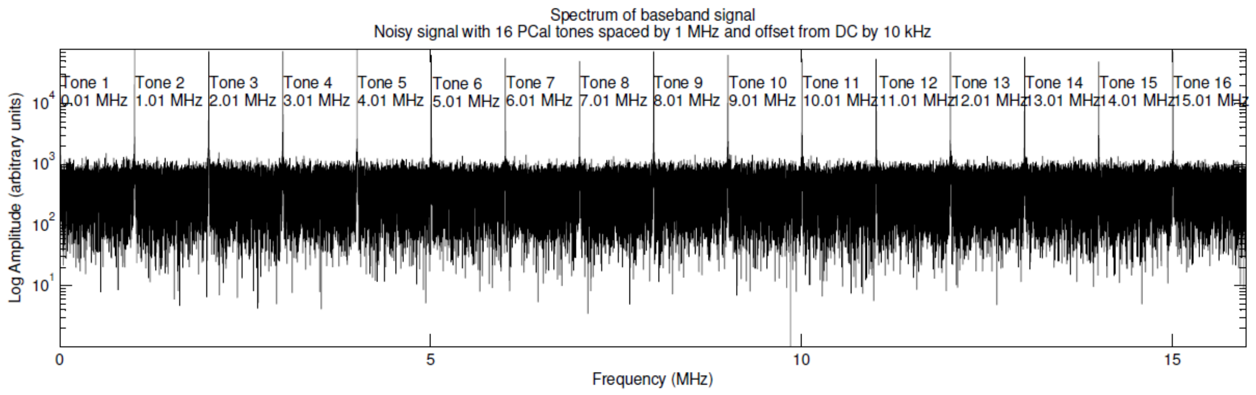


Figure 7 - Example 16 MHz wide AF with 16 PCal tones. Tones have an offset of 10 kHz and spacing of 1 MHz as in Geodetic VLBI. The tones have a different amplitude, a phase offset relative to H-maser 1PPS, and a phase slope across the band.

There are several methods to extract PCal amplitude and phase.

For multi-tone PCal one can use PCal tone periodicity in the time and frequency domain.

The multi-tone PCal tones are evenly spaced in frequency at e.g. 1 MHz intervals. The VLBI tuning is usually such that the first tone in an AF is offset from the 0 Hz band edge by some integer fraction of the tone spacing, e.g., the first tone is at an offset of 1 MHz (= 1 MHz spacing/1) or perhaps 10 kHz (= 1 MHz spacing / 100). Usually for VLBI data this offset evenly divides the bandwidth.

The above means there is a simple time domain periodicity for all PCal tones in VLBI data. An illustration is shown in Figure 8.

The shortest segment of samples that contains full periods (1 or more) of all tones is

$$N = 2\,B / \gcd(2\,B, f_1)$$

where B is the signal bandwidth (in Hz), and f1 is the offset (in Hz) of the first tone from 0 Hz. In the 8 MHz bandwidth case with tones spaced by 1 MHz and the first tone at 1 MHz, the shortest segment is N = 16 samples. If the first tone is at 10 kHz the shortest segment is N = 1600 samples.

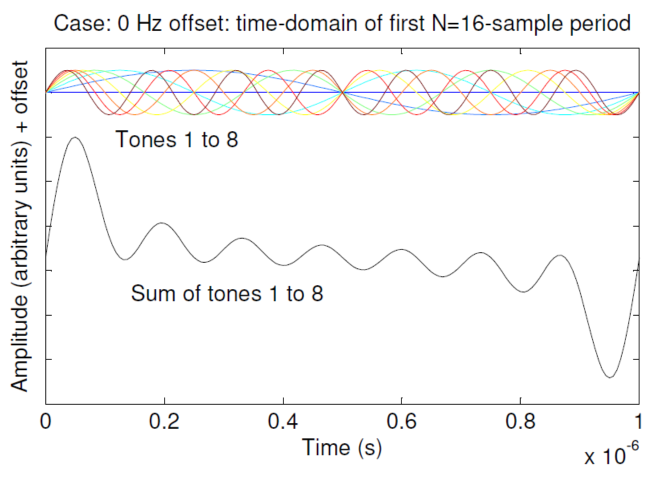


Figure 8 – Multi-tone PCal shown in the time domain. The first 16 samples of the digitized AF are shown. Top: Waveforms of individual tones located at 1 MHz, …, 8 MHz in a 8 MHz wide band, note the periodicity. Bottom: The sum of the individual tones.

A quite short Fourier transform can be used to extract the amplitude and phase information of all tones at once. To increase SNR, the VLBI AF sample data are folded in N-sample segments and time averaged (as in "pulsar folding"), and are Fourier transformed after time averaging. Tones can then be collected from certain bins in the output of the Fourier transform. An example with the first tone at 10 kHz is shown in Figure 9.

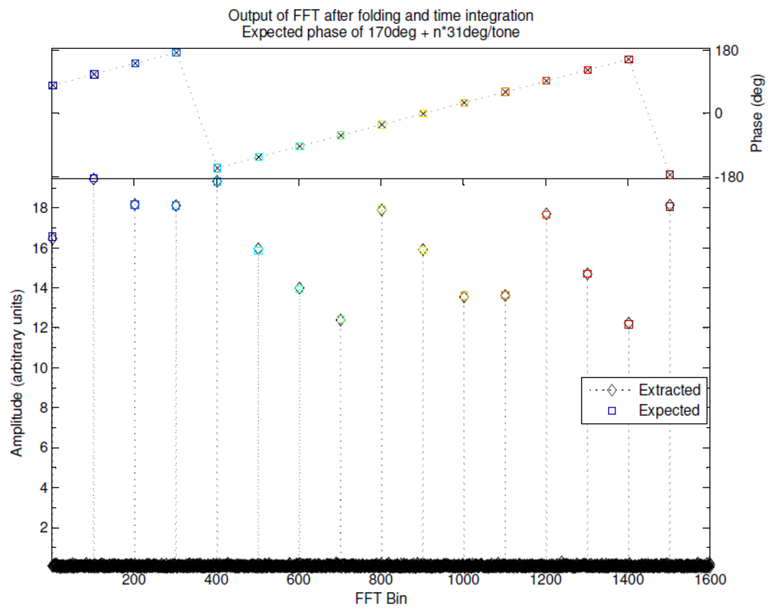


Figure 9 – The Fourier transform of folded (N=1600) and time integrated AF data during multi-tone PCal extraction as in DiFX. Test data are synthetic with PCal sinusoids and Gaussian noise. The original amplitudes and phases of synthetic tones are shown in color, the tone information extracted from synthetic data with added noise are shown in black.

8.2 PCAL EXTRACTION IN DIFX

The DiFX PCal extraction requires certain entries in the VEX and v2d files. The VEX file needs to have a $PHASE_CAL_DETECT section and references to it, for example:

```
$MODE;
def EUR2-SX.SX;
     ref $FREQ = EUR2-SX-SX01:Sm:Eb:Ny:Wz:Ys;
     ref $PHASE_CAL_DETECT = StdPCal:Sm:6a:Eb:Ny:Wz:Ys;
enddef;

$FREQ;
def EUR2-SX-SX01;
    sample_rate =  8.0 Ms/sec;
    chan_def = &X :  8210.99 MHz : U :  4.000 MHz : &CH01 : &BBC01 : &U_cal;
    chan_def = &S :  2292.99 MHz : U :  4.000 MHz : &CH16 : &BBC14 : &U_cal;
enddef;

$PHASE_CAL_DETECT;
def StdPCal;
    ** Also see http://www.vlbi.org/vex/docs/vex%20definition%2015b1.pdf
    phase_cal_detect = &U_cal : 1 : -1;  * detect lowest and highest tone in band
enddef;
```

Table 12 - VEX file entries for PCal extraction.

Then in the DiFX v2d each antenna with PCal needs to have a non-zero `phaseCalInt` setting:

```
ANTENNA Eb
{
    phaseCalInt = 1  # PCal spacing (=interval) in MHz
}
```

Table 13 - DiFX v2d file entry for PCal extraction.

DiFX outputs both interferometric visibility data (e.g., ./expt_01.difx/DIFX_*) as well as extracted single-dish PCal data with one file per antenna (./expt_01.difx/175-233000_Eb_PCAL_*). The DiFX converter *difx2fits* includes these PCal data in the output FITS file in a 'PC' table (e.g., AIPS PCCOR). Similarly *difx2mark4* includes the PCal information during conversion to MarkIV output (HOPS, fourfit).

8.3 Multi-Tone PCal Extraction from Files (m5pcal)

The mark5access library in DiFX offers *m5pcal* for extracting multi-tone PCal from VLBI recordings in the usual formats supported by mark5access (VDIF, Mark5B, KVN5B, VLBA, …). A slightly more flexible version is under https://bitbucket.org/jwagner313/kvnvdiftools/raw/master/phasecal and perhaps will be merged into DiFX later.

The program outputs ASCII format PCal data. The output is easy to load into Octave/Matlab/Python for station diagnostics. The ASCII file is not in the format expected by AIPS PCLOD, though.

```
$ m5pcal -n 500 -N 1 \                      # integration 'time', 1 iteration
       test.vdif VDIF_1280-512-8-2 \        # input file (Mark5A/B/C/VDIF)
       0.01 0.01 0.01 ... 0.01 \            # first tone (MHz) in each IF
       out.pcal                             # output PCal in text format

Sub-band 0 = 0.010000-16.010000 MHz
Sample   0  Tone  0  Freq=1.010 MHz  Amp=0.0246  Phase=-25.25 deg
Sample   0  Tone  1  Freq=2.010 MHz  Amp=0.0170  Phase=-24.22 deg
Sample   0  Tone  2  Freq=3.010 MHz  Amp=0.0210  Phase=-24.94 deg
…
Sample   0  Tone 13  Freq=14.010 MHz  Amp=0.0129  Phase=-24.88 deg
Sample   0  Tone 14  Freq=15.010 MHz  Amp=0.0114  Phase=-25.44 deg
t1=1080.00000 s  t2=1080.01000 s  Freq=8.010 MHz  Delay=2.7809 ns
Sub-band 1 = 0.010000-16.010000 MHz
…

Sub-band 16 = 0.010000-16.010000 MHz
…
```

Table 14 - Multi-tone (or single-tone) PCal extraction with m5pcal.

In principle m5pcal can also extract single-tone PCal.

8.4 Single-Tone PCal Extraction from Files (m5tone)

The extractor used for APEX receiver coherence testing is found in DiFX under https://svn.atnf.csiro.au/difx/sites/MPIfR/mark5/m5tone/. Currently the VLBI format, tone frequency, FFT resolution, and integration time are hard-coded. It extracts one tone per file (rather than per AF). Coherence is most readily measured in the analog domain, using the coherence test unit and a 10 kHz off-tuned reference tone from a synthesizer. Coherence during VLBI is better checked with *m5tone*.

Depending how *m5tone* is compiled, the output ASCII file should be usable for *difx2mark4* (HOPS, fourfit) or for WVR-like corrections (*fourfit*).

9 Details on DiFX Correlation

For general DiFX documentation see http://cira.ivec.org/dokuwiki/doku.php/difx/documentation

A few DiFX examples from tests at the KVN and APEX are at

https://bitbucket.org/jwagner313/apex-tools/raw/master/Correlation/DiFX/

Examples include

zerob2512x512.v2d (.vex)	: DBBC 16 x 32.0 MHz PFB x DBB 16x 32.0 MHz PFB
zerob1000x2048.v2d (.vex)	: DBBC 16 x 62.5 MHz PFB x R2DBE 2048 MHz
zerob2048x2048.v2d (.vex)	: R2DBE 1 x 2048 MHz x R2DBE 1 x 2048 MHz
zerob2048x2048-zoom.v2d	: as above, but DiFX configured to split 2048 MHz/16
kvn_VOAxFILA10G_8gbps.v2d (.vex)	: ADS->VOA 4 x 512 MHz x ADS->FILA10G 4 x 512 MHz

9.1 DiFX Zoom Band Setting

The zoom band processing in DiFX does a Fourier transform on a recorded wideband signal to split it into one or more sub-bands. These new sub-bands can be correlated against actual recorded narrowband signals of the other telescope or backend.

For DiFX "zoom band" mode, see example files *zerob1000x2048.v2d* and *zerob2048x2048-zoom.v2d*.

Example DiFX correlator output for a wideband signal, without and with zoom band, is in Figure 10.

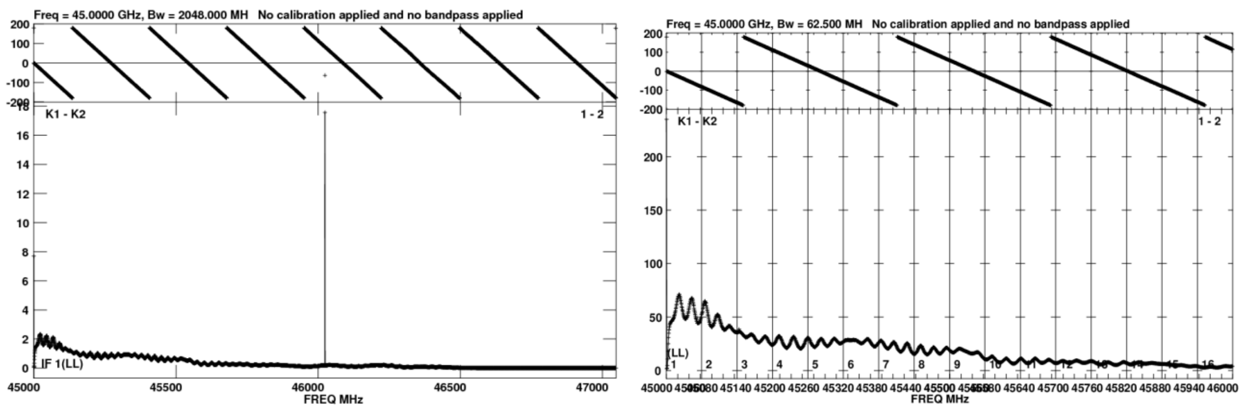


Figure 10 – DiFX zoom band example. Left: 2048 MHz wide R2DBE IF0 correlated against IF1 (zerob2048x2048.v2d). Right: 2048 MHz wide R2DBE IF0 correlated against lower 1000 MHz of IF1 split into 16 x 62.5 MHz using zoom band mode.

In DiFX .v2d SETUP section must set:

```
# Required for ZOOM to work:
guardNS = 150
strideLength = 1
xmacLength = 1

# To correlate 16 x 62.5 MHz against 1 x 2048 MHz, DiFX spectral
# resolution must be set to a fraction of
#   gcd(62.5e6, 2048e6) = 0.5 MHz
# so specRes = 0.5 MHz, or 0.25 MHz, or 0.125 MHz, or smaller.
specRes = 0.5
```

9.2 DiFX FORMAT=VDIF OR FORMAT=INTERLACEDVDIF

The VDIF data can be single-threaded or multi-threaded VDIF. In DiFX the multi-threaded VDIF is for some reason called “interlaced VDIF”. Settings in DiFX v2d are format=VDIF for single-threaded, or format=INTERLACEDVDIF for multi-threaded VDIF. The number of threads and the frame size can be found by inspecting some recordings of the current experiment. Two examples:

```
$ vdifheader2.pl /scratch0/n14st02c_8Gbps_fringetest/Ku/p14sl01c-No00035-06h50m12s-
FILA10G.vdif | less
KU I:0 L:0 4-ch 2-bit 1312-byte Th:0 EP:29 T:10392613 sec F#117541 06:50:13
KU I:0 L:0 4-ch 2-bit 1312-byte Th:0 EP:29 T:10392613 sec F#117542 06:50:13
KU I:0 L:0 4-ch 2-bit 1312-byte Th:0 EP:29 T:10392613 sec F#117543 06:50:13
KU I:0 L:0 4-ch 2-bit 1312-byte Th:0 EP:29 T:10392613 sec F#117544 06:50:13
…
# 1312-byte frames, only one thread (Th #0)
#   ---> DiFX .v2d :  STATION KU { format=VDIF/1312 }
```

```
$ vdifheader2.pl /home/oper/correl/zbt/r2dbe/r2dbe_t0_short.vdif | less
AR I:0 L:0 1-ch 2-bit 8224-byte Th:0 EP:30 T:616662 sec F#54586 03:17:42
AR I:0 L:0 1-ch 2-bit 8224-byte Th:0 EP:30 T:616662 sec F#54587 03:17:42
AR I:0 L:0 1-ch 2-bit 8224-byte Th:0 EP:30 T:616662 sec F#54588 03:17:42
AR I:0 L:0 1-ch 2-bit 8224-byte Th:0 EP:30 T:616662 sec F#54589 03:17:42
…
# 8224-byte frames, only one thread (Th #0)
#   ---> DiFX .v2d :  STATION AR { format=VDIF/8224 }
```

An example for multi-threaded VDIF, here produced by VOA and later converted to VDIF:

```
$ vdifheader2.pl /scratch0/n14st02c_8Gbps_fringetest/Ky/p14sl01c-No00035-06h50m12s-
VOA.vdif | less
 I:0 L:0 1-ch 2-bit 1312-byte Th:4 EP:29 T:10392613 sec F#58270 06:50:13
 I:0 L:0 1-ch 2-bit 1312-byte Th:3 EP:29 T:10392613 sec F#58271 06:50:13
 I:0 L:0 1-ch 2-bit 1312-byte Th:1 EP:29 T:10392613 sec F#58271 06:50:13
 I:0 L:0 1-ch 2-bit 1312-byte Th:2 EP:29 T:10392613 sec F#58271 06:50:13
 I:0 L:0 1-ch 2-bit 1312-byte Th:4 EP:29 T:10392613 sec F#58271 06:50:13
 I:0 L:0 1-ch 2-bit 1312-byte Th:3 EP:29 T:10392613 sec F#58272 06:50:13
…
# 1312-byte frames, multi-threaded because four threads (Th #1 to Th #4)
#   ---> DiFX .v2d :  STATION KY { format=INTERLACEDVDIF/1:2:3:4/1312/2 }
```

See http://cira.ivec.org/dokuwiki/doku.php/difx/vex2difx for a description of format=INTERLACEDVDIF.

9.3 CORRELATING MULTI-THREADED VDIF DATA

DiFX can handle ‘interleaved’ VDIF recordings only if *all* Threads in the file start exactly at the same timestamp and same frame number. One can use *dd* to remove leftover VDIF frames:

```

$ vdifheader2.pl p14sl01c-No00038-07h17m19s-VOA.vdif | head -8
 I:0 L:0 1-ch 2-bit 1312-byte th:2 EP:29 T:10394240 sec F#46050 07:17:20
 I:0 L:0 1-ch 2-bit 1312-byte th:4 EP:29 T:10394240 sec F#46050 07:17:20
 I:0 L:0 1-ch 2-bit 1312-byte th:3 EP:29 T:10394240 sec F#46051 07:17:20
 I:0 L:0 1-ch 2-bit 1312-byte th:1 EP:29 T:10394240 sec F#46051 07:17:20
 I:0 L:0 1-ch 2-bit 1312-byte th:2 EP:29 T:10394240 sec F#46051 07:17:20
# --> Must use 'dd' to remove first 2 frames (#46050) since they are
# found for threads 2 and 4 only and DiFX would search forever for thread 1, 3.

# Using dd to copy, with block size parameter "bs=1312" byte because the VOA
# frame size is 1312 byte, and skip=2 to remove the first 2 frames.
```

```
$ dd if=p14sl01c-No00038-07h17m19s-VOA.vdif of=p14sl01c-No00038-07h17m19s-
VOA.dd.vdif bs=1312  skip=2
```

At the moment DiFX .v2d format= and file= settings do not contain any byte ‘offset’ parameter like the mark5access tools (e.g., m5spec). That is why the entire file has to be copied with *dd* just to remove a few frames from the start…

9.4 Fixing VOA Issues

For recording VOA on Mark6, a modified Haystack software is required (modified *dplane*, see §4.5).

After recording, data must be converted into VDIF format. See *kvnVDIF2VDIF* in section 4.5.
After these steps data can probably be correlated in DiFX.

10 Details on H-Maser Rate Monitoring

At APEX a HP 53131A counter is measuring the offset between GPS 1PPS and the H-maser 1PPS. The counter is queried either by APECS (with logging into APECS databases) or “manually” with *pollCounters* that writes a text log file (see APEX Project Book).

The *pollCounters* source code of 01/2015, and current Matlab/Octave scripts *pollCounters_fit_rate.m* and *pollCounters_glue_logs.m* can be found in

https://bitbucket.org/jwagner313/apex-tools/raw/master/GPIB-tools/

The *pollCounters* log format was updated for 01/2015 to include the UT time of the program start.

When *pollCounters* is stopped, rename *pollCounters.log* before starting *pollCounters* again!

Logs from several *pollCounter* runs may be combined into a new file using *pollCounters_glue_logs.m*:

```
oper@mark6-4031 $ cd gps-maser
oper@mark6-4031 $ octave
>> pollCounters_glue_logs('gps-maser-2015y008d03h51m25s.log', 'gps-maser-
2015y013d01h22m47s.log', 'gps-maser-2015y015d03h52m38s.log')
Loading 'gps-maser-2015y008d03h51m25s.log'…
Loading 'gps-maser-2015y013d01h22m47s.log'…
Loading 'gps-maser-2015y015d03h52m38s.log'…

>> edit pollCounters.log.glued
```

Example of fitting a rate into an output log from *pollCounters*:

```
oper@mark6-4031 $ cd gps-maser
oper@mark6-4031 $ octave
>> pollCounters_fit_rate('gps-maser-2015y008d03h51m25s.log')
Rate by York regression : -2.711677501500e-012
Wrote plot to file  pollCounters_fit_rate.pdf
```

The linear fit uses York regression since the fit residuals are non-Gaussian and temporally correlated. The script shows a plot (as in Figure 11) and also writes it to a PDF file.

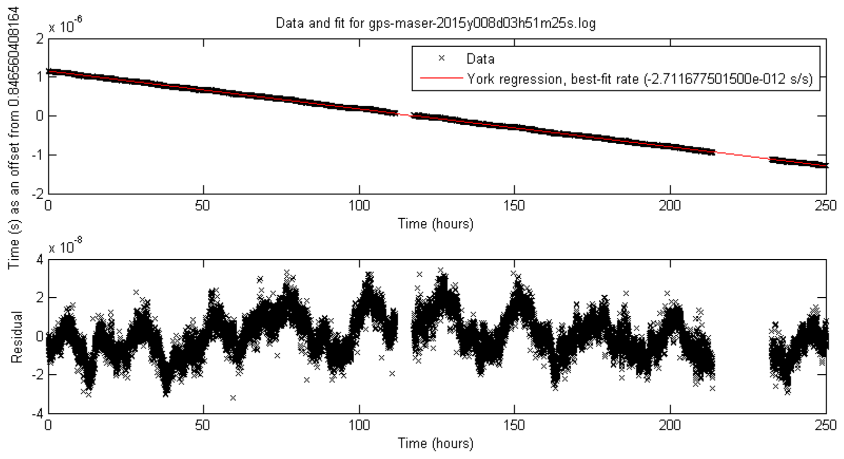


Figure 11 – Example H-maser rate fit into GPS 1PPS vs. H-maser 1PPS data logs.

11 APECS-BASED ALTERNATIVE TO FIELD SYSTEM

The official Field System (FS) distribution is available at ftp://atri.gsfc.nasa.gov/. It is necessary to register a new IP/host with Ed Himwich to "unblock" that IP and allow it access to the FTP site.

The FS version 9.10.4 is installed on Mark5c1 that runs 32-bit Debian. Mark6 has 64-bit Debian.

The FS at APEX contains some station-specific additions (*antcn.c*) that send source coordinates and tracking commands to the APEX telescope control system (APECS). They also trigger Tsys measurements. On the APECS side a Python script listens for measurement events (e.g., the completion of a Tsys task in APECS), grabs the new measurements from the APECS database, and sends them back to FS via *ssh* execution of the FS *inject_snap* program. The FS also controls recording on a single recorder (did not manage to get multi-recorder handling to work). For the two Mark 6's during 01/2015 another method was easier, i.e., VEX-derived timed recording with m6cc.py.

A copy FS 9.10.4 in its APEX variant is in the git repository, at
https://bitbucket.org/jwagner313/apex-tools/raw/master/FieldSystem
It requires certain Python scripts to be present on the VLBI account on APECS system, recently
https://bitbucket.org/jwagner313/apex-tools/raw/master/APECS/2015/

Current issues:

1) Ed Himwich confirmed FS is currently 32-bit (01/2015). He is considering to make FS 32/64-bit clean but is unsure what this involves. Looking at the C and Fortran code (and seeing many errors of http://www.viva64.com/en/a/0065/), a cleanup would require a tedious source code changes and testing. Attempts at APEX to compile a 32-bit Field System on Mark6 using Debian multiarch libraries (32+64 bit) failed because some libraries are not by default available as 32-bit.
   Options:
   - More persistence, perhaps missing Debian libraries can be compiled "easily" to 32-bit?
   - Run FS on a virtual machine with 32-bit Linux? Will timekeeping be good enough?
   - Install a dedicated computer to run Field System?
2) APEX operators did not like things going on "behind the scenes" triggered by Field System and with APECS in "remote command" mode -- despite remote mode being an official APECS mode. The main issue was the operators do not see what will happen next: in what time they have to complete manual calibration tasks, whether APECS and the 230 GHz receiver will be back to the correct state for VLBI recording after these manual tasks, etc. The operators were also versed in APECS commands but had no VLBI observing experience.
3) Field System requires extra preparatory work and schedules do no always *drudg*'e, especially when they contain wideband frequency setups. In the end, FS primarily sends VLBI source coordinates to APECS. Sources need to be entered into an APECS catalog file prior to observing, anyway. FS does not control the backends (manual). FS also does not handle APEX receiver tuning (done by VLBI scripts on APECS, an APECS catalog, and occasionally manually on the respective receiver GUI).

The main benefit of Field System at APEX is that it produces a log file in a VLBI standard format. Such a log file could also be created from APECS database entries since all information is contained in APECS, and in Mark6 recording logs, apart from VLBI operator comments.

A first version of a "Field System"-like observing system completely in APECS can be found at
https://bitbucket.org/jwagner313/apex-tools/raw/master/APECS/apecsVLBI/

Usage:

```
# Set up Python enviroment
t-091.f-0006-2013@observer3:~/testing>  . ./env.sh

# Convert a VEX file into an APEX source catalog (*.cat)
# and into an observing file (*.obs):
t-091.f-0006-2013@observer3:~/testing> ./vex2apecs.py <vexfile> <siteID>

# Run the timed commands of an .obs file on APECS
t-091.f-0006-2013@observer3:~/testing> ./apecsVLBI.py <obsfile>
```

During a run, apecsVLBI.py produces a Field System –like log file.

As of 01/2015 this 'apecsVLBI' system has not yet been tested in the APECS simulator.

Further development is still necessary.

In particular, a user console should be added that allows comment entry, a simple graphical display could be added that lists the VLBI schedule -related APECS tasks upcoming in the next minutes, next to a timer, and also have a chance to halt/pause the VLBI schedule.

There could also be further fine tuning to what commands vex2apecs.py inserts into the .obs file. An example of the current typical contents of an .obs file is:

```
#
# APEX observing script for station APEX (AP), experiment t15017
# Experiment starts 2015y017d06h00m00s, ends 2015y017d10h00m00s.
# File created on 2015.017.05:12:16 UT
#
# Columns: 1) Start time 2) Duration in seconds 3) APECS command
# Details on the columns:
#    1) @always, or UT date-time in a 2015.016.06:42:40 format (yyyy.doy.hh:mm:ss)
#         modifiers: !2015.016.06:42:40 to not skip command even if start time already is past
#    2) estimated duration of the command in seconds
#    3) APECS command and parameters to execute. If the command includes whitespace.
#       Commands include, e.g.: tsys(), interactive("message"), tracksource("sourcename"), ...
#
# Time                  Duration   Command
@always                 2          execfile('vlbi_commands_def.apecs')
@always                 2          sourcecats('vlbi-sources-t15017.cat')
@always                 2          linecats('vlbi-freqs-t15017.lin')
@always                 2          exec_apecs_script('shfi_commands')
@always                 2          setup_shfi(fename='het230',linename='vlbifreq',
sideband='',mode='cont', cats='user')
@always                 2          het230.configure(doppler='off')
@always                 2          tp()
@always                 2          offset(0,0)
@always                 2          reference(0,0)
@always                 2          use_ref('off')
#### No0001/J0522-363/eht-1mm-drudg ############################################
2015.017.05:59:50       5          doppler('off')
2015.017.05:59:55       5          source('J0522-363',cats='user')
2015.017.06:00:00       600        track()
2015.017.06:10:05       50         calibrate()
2015.017.06:11:00       15         readMeters()
#     225 seconds until next scan
#### No0002/J0522-363/eht-1mm-drudg ############################################
2015.017.06:14:50       5          doppler('off')
2015.017.06:14:55       5          source('J0522-363',cats='user')
2015.017.06:15:00       300        track()
2015.017.06:20:05       50         calibrate()
2015.017.06:21:00       15         readMeters()
#     225 seconds until next scan
#### No0003/J0522-363/eht-1mm-drudg ############################################
```

```
2015.017.06:24:50      5          doppler('off')
2015.017.06:24:55      5          source('J0522-363',cats='user')
2015.017.06:25:00      300        track()
2015.017.06:30:05      50         calibrate()
2015.017.06:31:00      15         readMeters()
2015.017.06:31:15      0          interactive('About 825 seconds available for
pointing/focusing/other')
# ...
#### No0021/SGRA/eht-1mm-drudg ##################################################
2015.017.09:54:50      5          doppler('off')
2015.017.09:54:55      5          source('SGRA',cats='user')
2015.017.09:55:00      300        track()
2015.017.10:00:05      50         calibrate()
2015.017.10:01:00      15         readMeters()
2015.017.10:01:15      1          remote_control('off')
################################################################################
### end of schedule
################################################################################
```