Table of Contents

Random notes for operating a DBBC in the field

VSI connectors and data flow (Gino, email 20.01.2011)

What is received by the second Core2 in the 'vsi1' position is shifted by the second Core2 into the 'vsi2' place -- thus the output of the second Core2 will be {vsi1out:own data, vsi2out:previous board}

The agc_if tool

In the Tunable firmware *agc_if* settings are overridden by the Control software. One should *never* run both programs at the same time. Instead use "dbbcifa=<input#>,<agc>,<nyquistZone>", dbbcifb, dbbcifc and so on. Zone 1=500-1000 MHz, 2=0-500 MHz.

In the Polyphase firmware *agc_if* settings are overridden by the *test_poly16* software. One should *never* run both programs at the same time. Instead use "ia=<input#>,<agc>,<nyquistZone>", ib, ic and so on. Zone 0=512-1024, 1=0-512 MHz.

Using agc_if: to change IF A settings, choose 'a'. For editing IF B, choose 'b' and so on. Next you can change the signal input (example: 'a'->'i'->'2' to set IF A to use input #2), toggle the AGC on/off, choose the filter for the desired Nyquist zone (1=0..512, 2=512..1024, 3=1024..1536, example: 'a'->'l'->'2'), and set the AGC target value (example: 'a'->'c'->'22000).

Clock phase calibration with the Polyphase firmware

A test tone at 764 MHz is injected to input 1 of all IF boards. Prior to calibration, use 'p' in the test_poly16 control program to list the total powers of all channels. Verify that channel #8 has maximum power and others are near zero. Start calibration with 'C'. It runs some minutes, output is logged into a log under C:\*.txt. Finally, a list of detected optimal phase values for each of the boards is printed. These values can be applied with "f=<boardNr>,<phaseValue>" and should also be copied into the C:\dbbc_conf\*.txt file.

Note: after 'C' completes all boards remain set to the last tested phase (255 255 255 255). Use a manual "f=..." or just restart test_poly16 to apply proper values.

Clock phase calibration with the Tunable firmware (Gino, email 20.01.2011)

"There is a dbbc_config_file.txt with calibration setting. Here is reported. The goal of this calibration is to find the best phase relation between data and clock in the internal FPGA capture point. To do so a tone at an almost central frequency in the second Nyquist zone is injected and the 4 bbcs present in one Core2 are tuned equispaced in the entire band widh respect to this central tone. No bbc is tuned to see the tone. When things are correct the tone should not produce a contribution to the total power detected in all the 8 basebands 16 MHz wide (4U+4L). Indeed when phases are not correct spurious and additional components are generated and are visible in the bbcs' bands producing a total power contribution. Hope this could clarify the procedure."

Clock phases can be set manually with "f=<boardNr>,<phaseValue>".

Analog Monitor output

One channel may be viewed on the Analog Monitor output with "m=<boardNr>,<channelNr>". For the Tunable firmware the syntax is "mon=<boardNr>,<channelNr>".

The monitor output can assist in clock phase tuning. However! Modifying the clock phase of one board will affect the spectrum of monitored 'earlier' boards.

"Changing the phase in the third Core2 will change the phase relation you use to capture data from the monitor bus, with such consequences. Calibration is taking care of information produced and 'resident' only in the board under analysis, the total power. A more clever calibration process is anyway probably necessary." (Gino, email 20.01.2011)

Mark5C and mode=…

Although Mark5C DRS 0.9 in Mark5B-mode does recognize channel bitmask commands such as mode=0x0000ffff:1; (or in FieldSystem: mk5b_mode/ext,0x0000ffff,1,1), the 5C will in reality ignore the bitmask and in any case records all channels. (Status of 01/2011)

1PPS clock drift on Mark5B+

It has happened that a Mark5B+ connected via VSI to a DBBC will loose 1PPS sync. Even after a dot_set=:force; the 5B dotmon output signal will immediately and visibly drift away from the DBBC 1PPS Monitor signal. In some cases the problem disappears again for a longer time if DBBC electronics are power cycled.

The drift is due to a bad VSI data clock and may be due to two causes:
1) VSI connector not inserted properly, cleaning and re-seating should help. After a new dot_set=:force; you should see the 5B dotmon output stay in sync.
2) if the above did not help, yet power cycling the DBBC seems to fix the issue, the problem is likely a set of missing resistors on the FILA board. These are the 'enable' signals for the LVDS line drivers that drive the VSI cable. On a few FILA board the resistors were missing. Contact Michael Wunderlich for fixing.

Firmware configuration files C:\DBBC_CONF\*.txt

Tunable (DDC) example:

```
1 dbbc2.bit 684.99 16
1 dbbc2.bit 700.99 16
1 dbbc2.bit 716.99 16
1 dbbc2.bit 732.99 16
1 dbbc2.bit 134.99 16
1 dbbc2.bit 150.99 16
1 dbbc2.bit 166.99 16
1 dbbc2.bit 182.99 16
1 dbbc2.bit 597.00 16
1 dbbc2.bit 682.00 16
1 dbbc2.bit 853.00 16
1 dbbc2.bit 938.00 16
0 dbbc2.bit 597.00 16
0 dbbc2.bit 682.00 16
0 dbbc2.bit 853.00 16
0 dbbc2.bit 938.00 16
55000 55000 55000 55000
53 98 32 0
CAT1 1024
```

The first number in the tunable version is indicating whether the bbc is present or not. In the DBBC2 this works in groups of 4 (e.g. if you have bbc1 you'll have also 2--3-4 and so on). This is necessary to the software to know the hardware configuration, number and address of boards to configure have a dialogue. The tunable version still don't have this additional possibility to configure the Fila10G, as the polyphase has (even if still needs to be fixed).

Polyphase example:

```
0 poly_dbbc.bit
1 poly_dbbc.bit
2 poly_dbbc.bit
99 poly_dbbc.bit
99 ACE.bit
99 fila10g.bit
22000 22000 22000 22000
10 10 100 0
CAT2 1024
```

The first number instructs the test_poly16.exe program which firmware it should loaded onto which Core2 board. That number is not necessarily equal to the device index in the JTAG chain.

The last three rows configure the AGC target for each IF board ("22000 22000 22000 22000"), the inter-board clock vs data phase adjustment for ADC data ("10 10 100 0") and finally the version of the global clock synthesizer board and output frequency selection ("CAT2 1024").

VSI bits and channels

Polyphase firmware channel definition is the standard from Haystack (mark5 memo #016). In order:

```
ch0 (16MHz not usable)   pin 1 sign and 2 mag    VSI BS bit 0=s|1=m
ch1 32 MHz               pin 3 sign and 4 mag    VSI BS bit 2=s|3=m
ch2 32 MHz               pin 5 sign and 6 mag    VSI BS bit 4=s|5=m
..............................
```

FILA10G Setup and Configuration Commands

Supported back-ends

Currently the FILA10G has been successfully tested for VSI to 10 GbE conversion on the DBE/iBob and the DBBC back-ends (03/2012).

The FILA10G passed zero baseline tests: 1) DBE/FILA10G/Mark5C <=> RDBE/Mark5C, 2) DBBC/FILA10G/Mark5C <=> DBBC/Mark5B+. Back-ends ran 16-channel polyphase filterbank firmware, channelizing 512-1024 MHz of a common noise signal. Mark5 recording correlations with "vlbi2": 95-100% with phase slope accross channels.

Programming the FILA10G firmware (VSI to 10GbE firmware)

The standalone FILA10G can be configured with Xilinx IMPACT and the Xilinx USB JTAG cable. In impact, initialize the jtag chain, bypass the Xilinx ACE part, assign fila10g.bit to the FPGA. Then right click on the fpga symbol and choose "configure". Once successfully programmed, the FILA10G will answer on the serial port (19200,8,n,1).

Inside the DBBC, FILA10G is configured either manually with Xilinx IMPACT and bypassing all devices except the fila10g virtex4 on the jtag chain. However, it is better to use the DBBC conf.txt file and DBBC's own executables that already handle the self-programming.

Configuring the FILA10G via the console

The default FILA10G data source is the internal test vector generator. Right after programming, the FILA10G streams UDP packets with test vector data. Change to a real VSI input with the "inputselect vsi1" command.

The FILA10G defaults to sending Address Resolution Protocol (ARP) queries on the 10G interface to gather a MAC-vs-IP address list for any devices on the local subnet. This allows normal IP routing and data capture with any Linux PC such as XCube/Mark6, but does not work well with the Mark5C.

Below are the typical configuration steps, to be run over the serial RS232 console.

There are also two Python scripts at the end of this page. They perform the same configuration and time synchronization steps, but work over TCP or UDP. This assumes your FILA10G is equipped with an RS232-Ethernet converter.

regwrite regbank0 10 x4170	change the station name to "Ap"
inputselect vsi1	change to VSI input #1
reset	resyncs everything to VSI#1 clocks
(time sync)	(see separate section below)
arp off	disable ARP broadcasts from FILA10G
tengbcfg eth0 mac=ba:dc:af:e4:be:e1	change MAC of the FILA10G
regread regbank0 11	(returns UDP stream destination IP, default ends with 0x0A)
tengbarp eth0 1 01:00:5E:40:20:01	add fake non-zero MAC for the gateway ip xx.xx.xx.1
tengbarp eth0 10 01:00:5E:40:20:01	add fake non-zero MAC for UDP destination ip xx.xx.xx.10
tengbinfo eth0	(to verify correct configuration of eth0 10GbE)

FILA10G time synchronization

Time synchronization on the FILA10G is triggered from the selected input data source: either the test vector generator with internal fake 1PPS, the external VSI#1 with actual 1PPS, the external VSI#2 with actual 1PPS, or combined external VSI#1+VSI#2 triggered from VSI#1 1PPS ("inputselect").

After any "inputselect" change you should issue a "reset" to ensure the Mark5B frame at a new second starts exactly with the data of that second, with zero samples delay. The long explanation: the FILA10G UDP packetizer is resynchronized only once to the 1PPS source, after "reset". The packetizer is fed from an internal FIFO where data and 1PPS are bundled and always in sync. On a data 1pps the packetizer resets the Mark5B header frame counter to zero, effective on the next outgoing 5B frame. This 1pps does not interrupt frame assembly. If another source was chosen with "inputselect", the new 1pps might be shifted and always occur in the middle of a packet. There can then be a fixed delay of up to 2499 samples in the Mark5B frame data stream in that case. The "reset" command purges old data from internal FIFO's and forces the UDP packetizer to resync data in Mark5B frame stream to the new 1PPS.

The Mark5B data has 3-digit MJD and second-of-day timestamps. For correct timestamps, the FILA10G real time clock has to be set with "timesync <second of day> <MJD last 3 digits>".

The time configured via "timesync" is loaded into the internal real time clock on the *next* 1PPS. On all later 1PPS the real time clock simply increments by 1 second. The clock wraps at day boundaries and increments the MJD.

If the computer NTP time and the FILA10G 1PPS source (eventually, house GPS) are not within 0.5 seconds of each other, there may be a potential +-1 second offset in the Mark5B time stamps and the actual time.

Windows and Linux/Python utilities

The "timesyncFILA10G.exe <COMxx>" program uses the DBBC's NTP-tied system time. It automatically waits 0.5s from the previous computer clock seconds change and sends the future time stamp to the FILA10G over the serial port. A Windows binary and original source code are at the end of this page.

Other FILA10G default configuration steps are performed by the "prepfila10g.bat"/.zip at the end of this page. The batch script uses the Windows utilities timesyncFILA10G and "sendstr <COMxx>", below.

If the FILA10G serial console is accessed over a RS232-Ethernet converter, then the Python "timesyncFILA10G.py" script can be used instead. It behaves identically to the timesyncFILA10G.exe program and supports UDP and TCP.

The "configureFILA10G.py" is the equivalent of the prepfila10g.bat batch file.

Some Mark5C considerations and 4 Gbps

The Mark5C packet size filtering feature exists currently only on paper (03/2012). To avoid ARP packets contaminate the disk recordings, FILA10G ARP should be switched off ("arp off").

Further, the FILA10G output format is currently {4 bytes 0x00, 4 bytes Packet Sequence Number, 5008 bytes with half a Mark5B frame}. The UDP size is thus 5016 bytes per packet. The PSN increments on every packet. One Mark5B frame spans two packets. This matches the Roach RDBE format.

The FILA10G PSN insertion can not be disabled. Since the Mark5C packet sequence number inspection (psn mode 1, psn mode 2) does not actually work so far (03/2012), and since typically one uses a single network route from the FILA10G to the Mark5C so that no packet reordering occurs, the Mark5C DRS "packet=40:0:5008:0:0;" can be used to strip off the PSN prior to recording FILA10G data.

This way FILA10G recordings on a Mark5C can be used as-is for DiFX correlation.

Regarding 4 Gbps: since the Mark5B format does not support 4 Gbps (15-bit frame counter in the Mark5B header rolls over too early), proper time stamped data can be recorded only at 2 Gbps.

A method tried at NRAO is to first record the 4 Gbps Mark5B frame stream, then modify the files and replace the wrapped headers by fill pattern in each latter 0.4s portion of a second. This works fine when W. Briskens mark5access library, included in DiFX, is used to decode data.