Fast Fourier Spectrum backend at the 40 m radiotelescope

R. Bolaño, P. de Vicente, J. A. López Pérez
C. Almendros, S. Henche

Informe Técnico IT-OAN 2011-07

Revision history

Version	Date	Author	Updates
1.0	27-03-2011	P. de Vicente	First version
1.1	01-09-2011	P. de Vicente & R. Bolaño	Added sections

Contents

1 Introduction **3**

2 The FFT spectral backend **3**

3 Cabling **5**

4 Current IF setup **6**

5 The preliminary FFTS conditioning module **6**

6 FFTS software setup **8**

7 Diagnosing the FFT. Faulty behaviours **12**

8 FFTS ACS Component **16**

9 Using the FFTS component from ACS clients **17**
9.1 Regular usage from the obsEngine and Fits writer component 19

1 Introduction

We describe the Fast Fourier Spectral backend, the current instalation, the monitor and control software and the tests performed to check it.

2 The FFT spectral backend

On 2010, a Fast Fourier Spectral backend was purchased to Radiometer Physics GmbH, a german company that produces units based on a design from Max Planck Institut für Radioastronomie. The chosen model is an RPG-FFTS, a 19" crate equipped with 8 FFTS boards and one FFTS controller. The spectral backend was mounted in a rack in the backends room in September 2010. Fig. 1 shows a view of the rack from the front:

Figure 1: Front of the FFT rack with 8 modules. Only modules 1 to 4 are currently connected to cables from the IF

The FFTS was delivered with 4 different FPGA software cores which can be applied to any independent module. The cores allow the following spectral configuration per module: 100, 500, 750 MHz total bandwidth with 16384 channels, and 1500 MHz total bandwidth with 8192 channels.

Each module has got an SMA input connector for the IF signal. This signal has to be in the baseband range and hence the lower input frequency is 0 MHz and the upper depends on

the loaded software core (100, 500, 750 or 1500 MHz). The nominal total power for the whole band is -4 dBm. Each module has got 8 LEDs on the front, 6 grouped in the upper part, one tagged as "Synth" in the lower part of the front and one below the IF input tagged as "Level". The latter has a three color code (off/green/red/yellow). When the input power is too low ($<$ -23.2 dBm) it will light in yellow. If the power is too large ($>$ -3.4 dBm) it will light in red; if the power is in the correct range ([-3.4 dBm, -14.2 dBm]) it will light in green. Yellow and red may also blink, indicating extreme power level, either very low ($<$ -23.2 dBm) or very large ($>$ -1.2 dBm).

The group of 6 LEDs provides information using a two color code (off/green/red)regarding the transmit status, the temperature, the voltage and current in the device, the status of the IRIG-B signal and the overflow in the WOLA FFT or Integration Unit. Red is used for a faulty status and green for normal status. Off has different meanings. For further information one should refer to the manual. The "Synth" LED indicates wether the synthesizer is locked (green) or not (red or off).

Besides the 8 modules, there is a controller module with a front LCD display plus 3 LEDs (Blank, Sync and IRIG-B signals), a small LCD panel and three selection knobs. The 3 LEDs indicate the status of the blank, sync and IRIG-B signals. All these signals can be injected into the device at the rear side of the FFTS. The Blank and Sync signals are differential at RS485 level and are injected through a DB9 connector. The pinout is described in page 12 of the manual. The IRIG-B is injected via a BNC connector. The upper knob allows to select a channel, which is displayed in the small LCD pannel beside it. The two lower knobs "Function" and "Value" allow to select the properties we want to be displayed in the big LCD display. Apart from being turned to right and left these knobs can be pushed.

The most useful knob is the "Function" one. By turning it from right to left, 8 pages are displayed. Each turn allows to see one page. The LCD displays information on the Blank/Sync signal, generated either internally or externally, network information (IP address, mask, MAC-address and port), time and software version, voltages in the boards, temperature in different parts and IF level, and FFTS parameters (number of channels in use, Nysquist bandwidthm integrated spectra at each phase, and dump number). The last page depicts an ADC (analog to digital conversion) histogram indicating the usage of the bits for digitazing the IF signal. The device works in an optimum regime when a gaussian shape which uses channels all over the range appears in the display.

The analog to digital converter (ADC) has a resolution of 8 bits and the sample rate is 3 Gsamples per second. When a white noise signal is received, the digitized signal has a gaussian distribution around the zero. A digitation scheme of 8 bits allows to sample the signal in 64 levels. In an optimum situation the signal should have an amplitude such that it is properly "sampled" by the bits. If the signal is too weak only the levels close to 0 will be working and the histogram will display only a few central channels. If the signal is too strong all levels will show the same amplitude except those at the extremes which will be higher since an excess of signal is accumulated out of the measuring range. Figure 2 shows different examples to understand these cases.

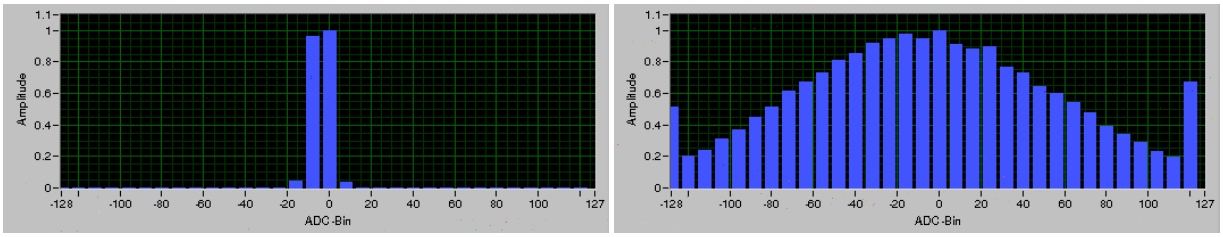

Figure 2: *Extreme cases of low and high power signals. Left: Low power digitized signal. Right: High power digitized signal. The signals are normalized. 8 bits provide 256 states which in the plot are splitted in two: 128 positive and 128 negative*

3 Cabling

An HP switch ProCurve 2810-24G was installed in the same rack at the backends room as the FFTS crate. This switch allows to connect the different modules and the controller board with the control PC. Each module has an ethernet connector in the rear part and it is assigned an IP address from the private C IP class 192.168.10.0. Section 4 provides more details on the IP assignment. The IRIG-B signal is also connected to the TimeTech IRIG-B distributor in the backends room. No connection to input reference and to the DB9 connector is used for the time being.

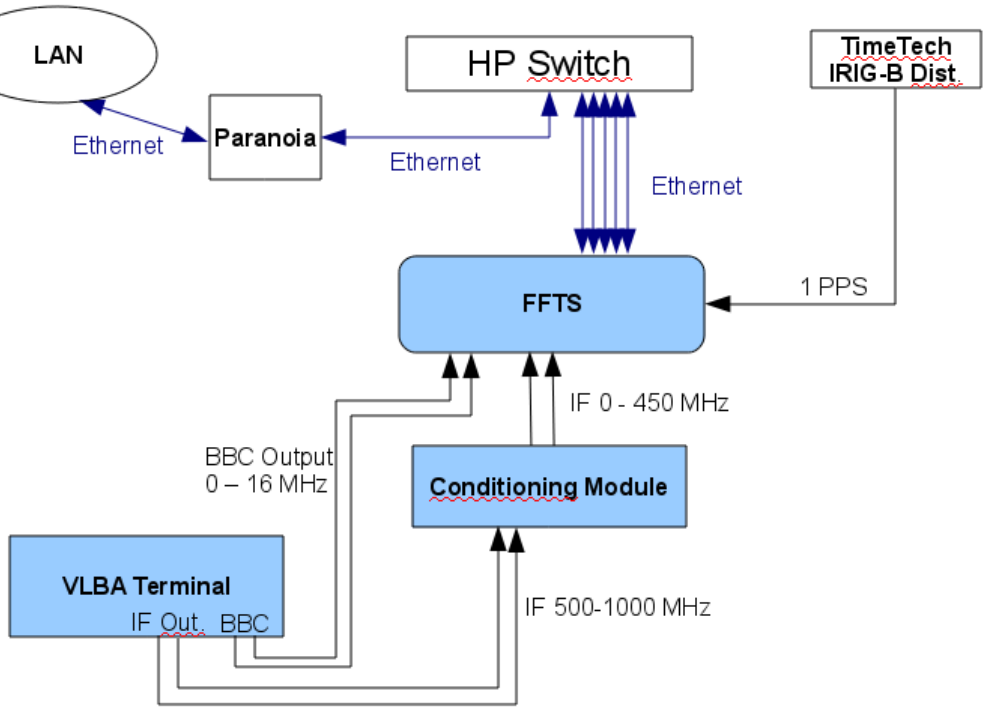

Figure 3: Scheme on the FFTS cabling.

The FFTS gets the IF signal injected at the front pannels. This signal is received from a conditioning module whose aim is to provide the correct gain and frequency band to the FFTS

modules. The IF connection is described in sections 2 and 3 below. Fig. 3 is a simple scheme on how the cabling is done.

4 Current IF setup

The current setup is as follows:

- The IF signals from the receivers come down to the patch panel with 8 N connectors in the backend room. All these signals are in the range 500 to 1000 MHz. S band is the only IF signal which uses a band smaller that 500 MHz. Its instantaneous bandwidth is 140 MHz, and the IF starts at 500 MHz.
- The connectors from the IF panel are patched to the 4 IF VLBA inputs labelled A, B, C and D.
- The Monitor Output IF BNC connectors on the front of the VLBA terminal are used to inject this signal to the FFT conditioning module. Currently 4 cables are available, one per IF output. Two of these cables are injected in the preconditioning module. The other two are not connected and may be used from the BBCs monitor output.
- The output of the conditioning module are injected in two FFTS modules, depending on the pass band filters used in the conditioning module. If the filters are 80 MHz wide they are injected in the first 2 modules, if the filters are 450 MHz wide they are injected in modules 3 and 4. Usually the 450 MHz filters are in use.
- In ocassions the Monitor Output from individual BBCs can be used to feed modules 1 and 2. BBCs can be tuned to a maximum bandwidth of 16 MHz, and hence a big part of the FFTS 100 MHz module is wasted.

This setup has several advantages:

- It allows to observe with the same receiver using the continuum PBE backends, the FFT backend and the VLBA without changing IF cables.
- It allows to use the VLBA IF 20 dB attenuators and the possibility to switch to an external input to measure the zero offset of the FFTS module.
- It allows to debug the behaviour of the BBCs when searching for RFI, pcal tones or any other feature visible in the spectral band. In this case a large percentage of the 100 MHz module is wasted, but nevertheless the big number of channels allows to obtain a spectral resolution of 6 KHz and 1638 channels in 10 MHz bandwidth.

5 The preliminary FFTS conditioning module

A conditioning module which amplifies and converts the signal to baseband was built in the labs. This module is required since the IF band goes from 500 to 1000 MHz and needs to be

converted to baseband. On the other hand the IF output is optimized for the VLBA IF whose required input power differs up to 30 dB with the FFTS backend. Fig. 4 shows a view of the module and Fig. 5 an schematics of the device. The board, as seen in the picture, uses two 450 MHz wide pass band filters, but it is possible to replace them by two 80 MHz wide filters to be used with the 100 MHz modules. This latter scheme has never been used since the 450 MHz filter band matches the current instantaneous IF bandwidth.

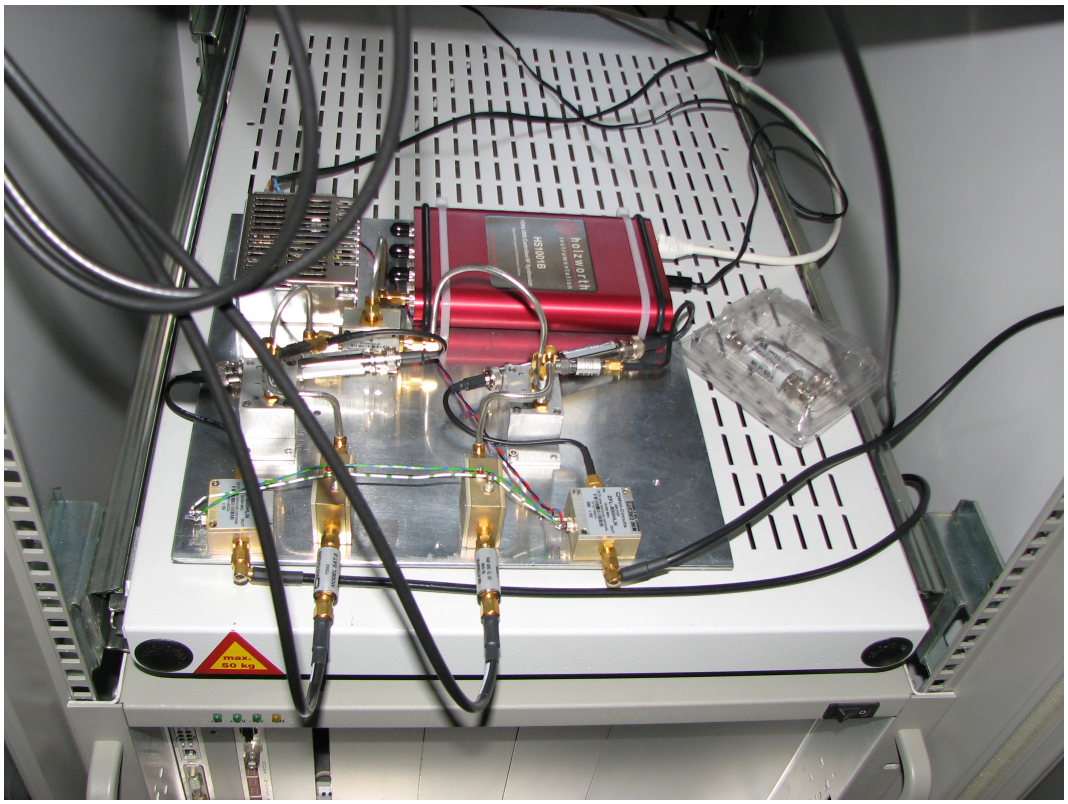

Figure 4: Picture of the preprocessor board to condition the signal prior to injecting it into the FFTS.

The final scheme depicted at figure 5 was obtained after some iterations debugging the behaviour of the FFTS. The attenuators used along the path allow to avoid reflections in the cables that cause variable signal amplitude. The amplifiers were chosen to produce the required FFTS signal level. This is due to the fact that the IF level output for the VLBA does not match the FFTS input signal level. Both levels differ up to 30 dB.

The input signal from the IF is attenuated 5 dB by Minicirtuits SAT-5 attenuator and amplified by a MiniCircuits ZFL-500HLN low noise amplifier with a gain of 24 dB. The signal is attenuated again 3 dB before entering a MiniCircuits ZLW-186MH mixer. The LO oscillator signal is obtained, after splitting it, from a tunable model (HS10001B) from Holzworth. The frequency and power are selectable via a web interface. If the oscillator is switched off, it is necessary to set the frequency and power again to 500 MHz and 13 dBm respectively. At the ouput of the mixer there is an Alan DC block which filters out any DC from the LO and a 3 dB attenuator from MiniCircuits. The signal is then filtered either by 400 MHz low pass filters or

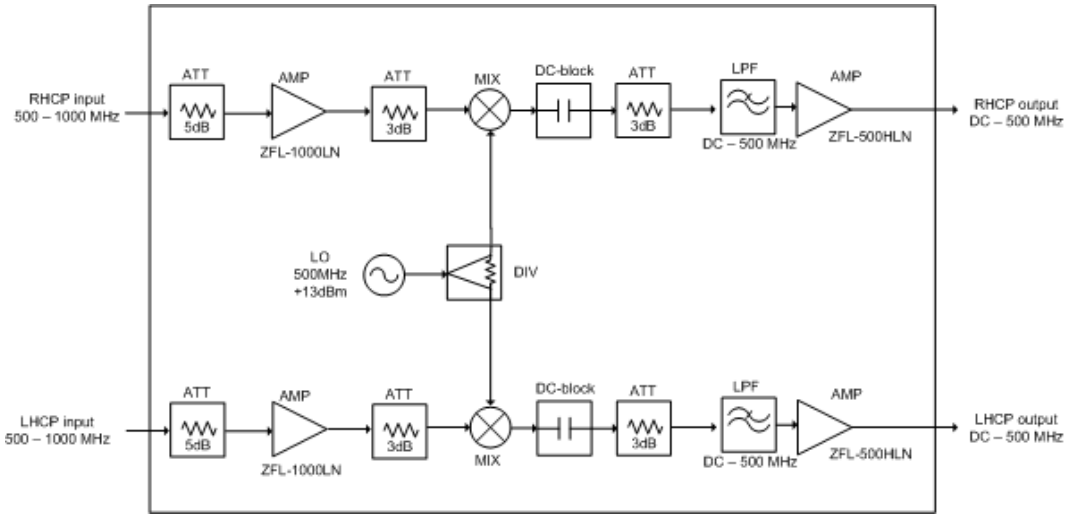

Figure 5: Schematics of the preliminary board to condition the signal prior to injecting it into the FFTS.

80 MHz low pass filters. Finally the output signal is amplified by a ZFL-1000LN MiniCircuits amplifier with a gain of 20 dB. The total amplification of the signal is 36 dB approximately. The board uses semirigid and flexible cables in between the different stages.

6 FFTS software setup

The FFTS is controlled by a multithreaded program called AFFTS available for 32 and 64 bit linux kernels ($\geq$ 2.6.18) that can handle systems with up to 32 boards and 4 crates. The AFFTS program loads startup information from a configuration file, establishes a TCP connection to the FFTS, and then uploads an FPGA core file and the polyphase filter coefficients to the boards through ethernet. Once this process is completed the FFTS is ready for measurements. The AFFTS program transmits spectral data through a TCP network connection and is controlled by an UDP/SCPI command interface. A graphical client program based on LabView called FFTS_Monitor is also supplied for monitoring. The communication between AFFTS and FFTS_Monitor programs is based on file IO. In order to boost this communication a RAM disk with a maximum size of 1 GB must be created and normal users must be allowed to access it. The ramdisk is mounted at boot time through the following script:

```
  paranoia almamgr:~ 536 > more /etc/init.d/fftsramdisk
  #!/bin/sh

  dir=/ramdisk

  if [ ! -d \${dir} ]
  then
  mkdir \${dir}
  fi

  mount -t ramfs -o maxsize=1048576 none \${dir}
  chmod 0777 \${dir}

exit
```

In order to enable the script the following instruction must be executed as root:

```
update-rc.d fftsramdisk start 90 S .
```

All the software provided by the manufacturer is installed on a dedicated PC known as Paranoia that runs Debian Lenny (kernel 2.6.26-2-686) and located under `/home/almamgr/AFFTS/` directory. In order to compile the software a symbolic link to the correct version of a library has to be made. The version of the library depends on the system architecture (32 or 64 bit), in our case a 32 bit architecture.

```
paranoia almamgr:~/AFFTS/src 554 > ls -l
total 520
-rwxr-xr-x 1 almamgr almamgr  97768 jul 20  2010 AFFTS
-rw-r--r-- 1 almamgr almamgr   3389 oct 25  2010 AFFTS.cfg
-rw-r--r-- 1 almamgr almamgr  31334 jul 20  2010 affts.h
-rw-r--r-- 1 almamgr almamgr  25766 jul 20  2010 AFFTSmain.c
-rw-r--r-- 1 almamgr almamgr  17720 jul 20  2010 AFFTSmain.o
-rw-r--r-- 1 almamgr almamgr   3387 jul 20  2010 AFFTS_MULTICORE1.cfg
-rw-r--r-- 1 almamgr almamgr   3387 jul 20  2010 AFFTS_MULTICORE2.cfg
-rw-r--r-- 1 almamgr almamgr   2329 jul 20  2010 AFFTS_SINGLECORE.cfg
-rw-r--r-- 1 almamgr almamgr   1843 jul 20  2010 datatypes.h
-rwxr-xr-x 1 almamgr almamgr  11478 oct 27  2010 FW
-rw-r--r-- 1 almamgr almamgr  11445 oct 27  2010 FW.c
-rw-r--r-- 1 almamgr almamgr   2611 jul 20  2010 jconfig.h
-rw-r--r-- 1 almamgr almamgr 113948 jul 20  2010 libAFFTS32.a
-rw-r--r-- 1 almamgr almamgr 165526 jul 20  2010 libAFFTS64.a
lrwxrwxrwx 1 almamgr almamgr     12 jul 20  2010 libAFFTS.a -> libAFFTS32.a
-rw-r--r-- 1 almamgr almamgr    936 jul 20  2010 Makefile
-rw-r--r-- 1 almamgr almamgr    876 jul 20  2010 swap.h
```

First the file libAFFTS.a must be deleted:

```
rm libAFFTS.a
```

and then the symbolic link can be created and the compilation process can be started:

```
ln -s libAFFTS32.a libAFFTS.a
make
```

Once these steps are accomplished the AFFTS program is fully operational. The file `AFFTS.cfg` contains the startup settings used by the AFFTS program and should be modified in order to meet the specific system requirements (bandwidth, number of channels...).

```
paranoia almamgr:~/AFFTS/src 555 > more AFFTS.cfg
#
# Array Fast Fourier Transform Spectrometer
#
# setup and configuration file / multi-core version
#
# 2010-05-20,  bklein@mpifr-bonn.mpg.de
#

[common]
telescopeName           = ARIES         # maximum 10 chars
backendName             = AFFTS         # maximum  8 chars
UDPportNumber           = 16210
TCPportNumber           = 25144
tempFiles               = /ramdisk/     # directory for temporary files

# SETUP:  100 MHz bandwidth, 16k spectral channels
[FPGA1]
coreFile                = ../XILINX/virtex4_16K_100_4wola_18-8_166.bin
bandwidthMHz            = 100
numberOfChannels        = 16384
specFilter              = 2
windowFile              = ../windows/FLATTOP4WOLA16K.dat
WOLA                    = 4
ADCdelayFile            = ./ADCdelay.data
```

```
# SETUP:  500 MHz bandwidth, 16k spectral channels
[FPGA2]
coreFile                = ../XILINX/virtex4_16K_500_4wola_19-0_666.bin
bandwidthMHz            = 500
numberOfChannels        = 16384
specFilter              = 2
windowFile              = ../windows/FLATTOP4WOLA16K.dat
WOLA                    = 4
ADCdelayFile            = ./ADCdelay.data

# SETUP:  750 MHz bandwidth, 16k spectral channels
[FPGA3]
coreFile                = ../XILINX/virtex4_16K_750_4wola_19-3_750.bin
bandwidthMHz            = 750
numberOfChannels        = 16384
specFilter              = 2
windowFile              = ../windows/FLATTOP4WOLA16K.dat
WOLA                    = 4
ADCdelayFile            = ./ADCdelay.data

# SETUP: 1500 MHz bandwidth, 8k spectral channels
[FPGA4]
coreFile                = ../XILINX/virtex4_8K_1500_4wola_18-7_1290.bin
bandwidthMHz            = 1500
numberOfChannels        = 8192
specFilter              = 2
windowFile              = ../windows/FLATTOP4WOLA8K.dat
WOLA                    = 4
ADCdelayFile            = ./ADCdelay.data

# configuration settings  BOARD->FPGA<num>
[BOARD_FPGA]
FFTSboard01             = FPGA1
FFTSboard02             = FPGA1
FFTSboard03             = FPGA2
FFTSboard04             = FPGA2
FFTSboard05             = FPGA3
FFTSboard06             = FPGA3
FFTSboard07             = FPGA4
FFTSboard08             = FPGA4

[FFTScrate]
usedFFTScontrollers     = 1
usedFFTSboards          = 8

[FFTScontroller]
refFreqMHz01            = 100           # INT(ernal) frequency is 100 MHz
refFreqMHz02            = 100           # reference frequency range: 5 - 125 MHz,
refFreqMHz03            = 100           # step size is: 5 MHz
refFreqMHz04            = 100
refFreqSource01         = INT
refFreqSource02         = INT
refFreqSource03         = INT
refFreqSource04         = INT

[FFTScontrollerIPs]
IPcontrollerNum01       = 192.168.10.10 # master FFTScontroller / crate 1
IPcontrollerNum02       = 192.168.10.20 # FFTScontroller / crate 2
IPcontrollerNum03       = 192.168.20.30 # FFTScontroller / crate 3
IPcontrollerNum04       = 192.168.20.40 # FFTScontroller / crate 4

[FFTSboardIPs]
IPboardNum01            = 192.168.10.11
IPboardNum02            = 192.168.10.12
IPboardNum03            = 192.168.10.13
IPboardNum04            = 192.168.10.14
```

```
IPboardNum05            = 192.168.10.15
IPboardNum06            = 192.168.10.16
IPboardNum07            = 192.168.10.17
IPboardNum08            = 192.168.10.18

IPboardNum09            = 192.168.10.21
IPboardNum10            = 192.168.10.22
IPboardNum11            = 192.168.10.23
IPboardNum12            = 192.168.10.24
IPboardNum13            = 192.168.10.25
IPboardNum14            = 192.168.10.26
IPboardNum15            = 192.168.10.27
IPboardNum16            = 192.168.10.28

IPboardNum17            = 192.168.20.31
IPboardNum18            = 192.168.20.32
IPboardNum19            = 192.168.20.33
IPboardNum20            = 192.168.20.34
IPboardNum21            = 192.168.20.35
IPboardNum22            = 192.168.20.36
IPboardNum23            = 192.168.20.37
IPboardNum24            = 192.168.20.38

IPboardNum25            = 192.168.20.41
IPboardNum26            = 192.168.20.42
IPboardNum27            = 192.168.20.43
IPboardNum28            = 192.168.20.44
IPboardNum29            = 192.168.20.45
IPboardNum30            = 192.168.20.46
IPboardNum31            = 192.168.20.47
IPboardNum32            = 192.168.20.48

#################################################################
```

The AFFTS program can be added to the system path. A simple way to do it is by adding the following lines to the `.bashrc` file:

```
# AFFTS-settings
export AFFTS_DIR=AFFTS
export PATH="\$HOME"/"\$AFFTS_DIR"/sh:\$PATH
```

The PC paranoia has two NICs, one is dynamically configured for LAN operation and the second one is statically configured to be part of the FFTS private network. The FFTS boards are connected to this PC through a network switch (HP ProCurve 2810-24G).

```
paranoia almamgr:~ 535 > more /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug eth1
iface eth1 inet dhcp

allow-hotplug eth2
iface eth2 inet static
address 192.168.10.1
netmask 255.255.255.0
broadcast 192.168.10.255
```

The file /etc/hosts must be modified in order to include the IP addresses of the FFTS boards and the FFTS controller:

```
paranoia almamgr:~ 551 > more /etc/hosts
127.0.0.1       localhost
192.168.0.134   paranoia.oan.es paranoia

# AFFTS-Extensions #############

192.168.10.10 ffts-ctrl10
192.168.10.11 ffts-board11
192.168.10.12 ffts-board12
192.168.10.13 ffts-board13
192.168.10.14 ffts-board14
192.168.10.15 ffts-board15
192.168.10.16 ffts-board16
192.168.10.17 ffts-board17
192.168.10.18 ffts-board18

#192.168.10.20 ffts-ctrl20
#192.168.10.21 ffts-board21
#192.168.10.22 ffts-board22
#192.168.10.23 ffts-board23
#192.168.10.24 ffts-board24
#192.168.10.25 ffts-board25
#192.168.10.26 ffts-board26
#192.168.10.27 ffts-board27
#192.168.10.28 ffts-board28

#192.168.10.30 ffts-ctrl30
#192.168.10.31 ffts-board31
#192.168.10.32 ffts-board32
#192.168.10.33 ffts-board33
#192.168.10.34 ffts-board34
#192.168.10.35 ffts-board35
#192.168.10.36 ffts-board36
#192.168.10.37 ffts-board37
#192.168.10.38 ffts-board38

#192.168.10.40 ffts-ctrl40
#192.168.10.41 ffts-board41
#192.168.10.42 ffts-board42
#192.168.10.43 ffts-board43
#192.168.10.44 ffts-board44
#192.168.10.45 ffts-board45
#192.168.10.46 ffts-board46
#192.168.10.47 ffts-board47
#192.168.10.48 ffts-board48

# End_of_AFFTS-Extensions #############

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

7 Diagnosing the FFT. Faulty behaviours

As already mentioned in section 2, the FFTS was supplied with a LabView client that runs under Linux and that allows to select the channel to view. It is possible to zoom the graph and check the detected power on all channels and see the distribution of bits of the digitized signal. If the

power on the channels is not adequate the bit distribution is not a gaussian and this can easily be spotted.

Fig. 6 shows two of the first spectra obtained (January 2011) with this client. The LCP channel has an adequate IF power level whereas the RCP shows a very low level. This is easily seen at the bit diagram, in the lower part of the figures. Low level signals do not produce a gaussian shape, since the central bit channels are more numerous than expected. The RCP channel also shows ripples. The IF cable from the VLBA terminal to the FFT module was found to be partially broken. Fig. 7 shows the spectra for the same configuration except that a hot load was placed in front of the horn to stop possible standing waves. This time the LCP is saturated as seen in the bit diagram. The gaussian shape is not achieved and the bit channels from both ends have bigger amplitudes than the contiguous ones. The ripples are still present in the RCP channel.

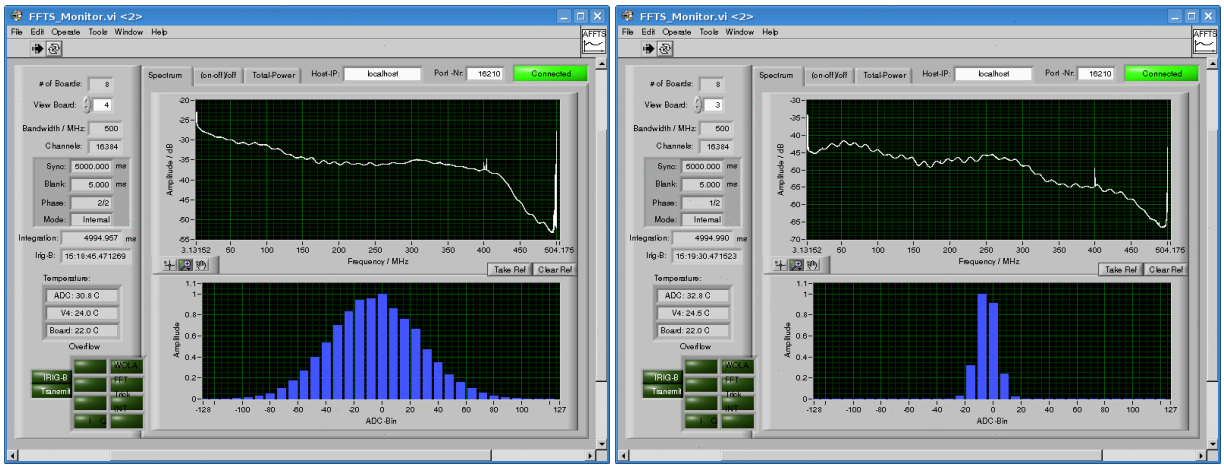

Figure 6: *Spectrum of the 22 GHz IF band while pointing to the zenith. Left: LCP. Right: RCP. The RCP channel has a low amplitude and high ripples.*

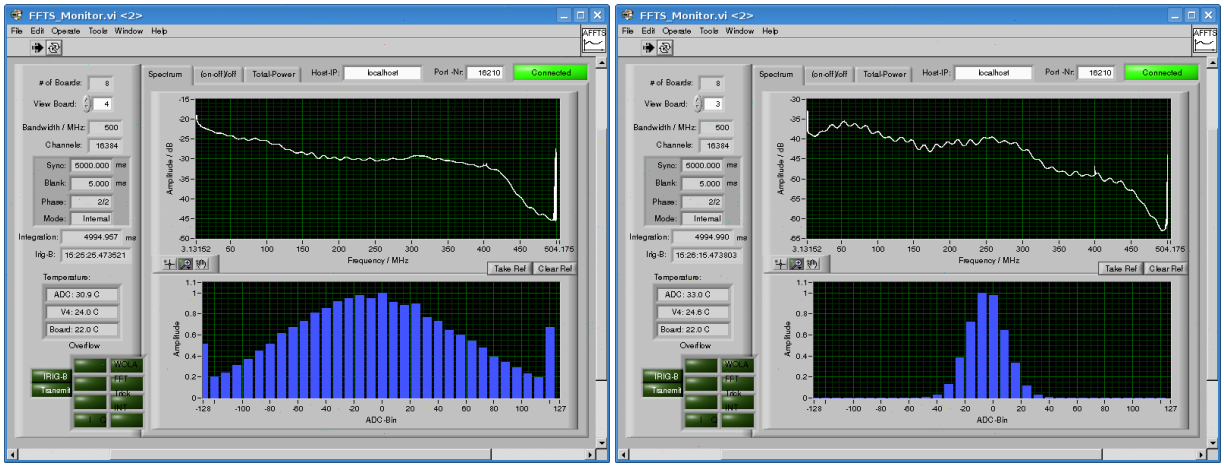

Figure 7: *Spectrum of the 22 GHz IF band. The horn was covered with a hot load. Left: LCP. Right: RCP. The LCP channel is saturated. The RCP channel shows high ripples.*

Fig. 8 shows another example after the cable was repaired (February 2011). It is an spectrum taken with the X band receiver looking at the sky. The band still shows ripples at both channels. This effect was corrected once the FFT preprocessor was modified by inserting an attenuator and amplifier prior to the mixing stage. This prevented reflections in the IF cables from the mixer in the preprocessor board.

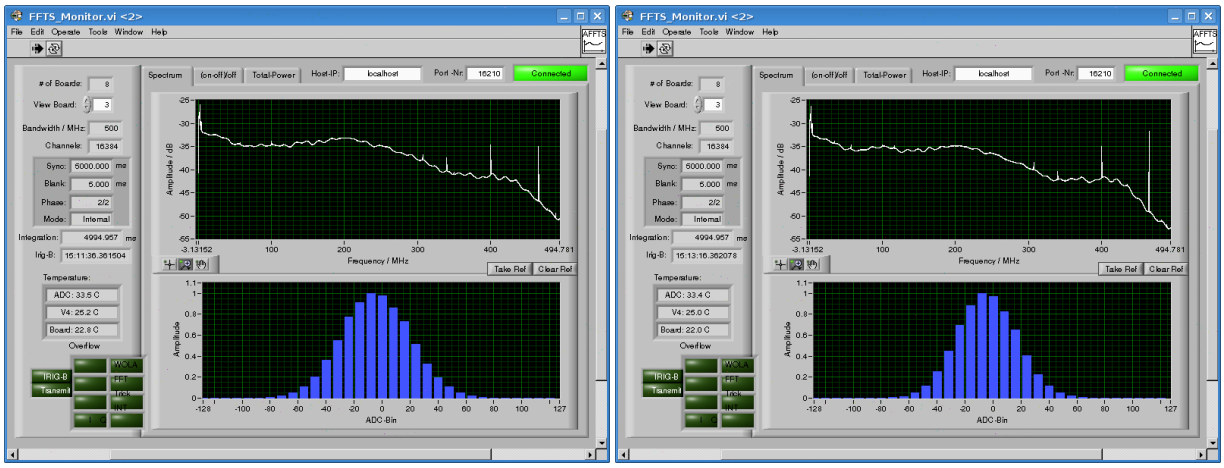

Figure 8: *Spectrum of the X band IF band at both polarizations. Both spectra were taken with the same FFT channel. This spectra were acquired before the preprocessor board was reordered to the new design displayed in Fig. 5*

After these two major hardware corrections described above, three minor effects are still present, which have not been addressed since we expect that the definitive preprocessor module currently being built at the lab will solve them.

- The spectra are not flat along the frequency range. The lack of flatness is associated to a non uniform noise in the band as shown in Fig. 9. The higher the power received in the FFTS module the larger the noise detected. The power decreases 3 dB from one end of the backend to the other. The maximum power is achieved at lower frequencies. Fig. 9 shows a clear fall at channel 14750, which is due to the edge of the 450 MHz filter in the preprocessor board.

- The LCP channel displays a strong lack of flatness due to a low frequency modulation of unknown origin, but which seems to be associated to an element in the preprocessor board.

- Big peaks are seen at both ends of the FFTS backend. This may come from the LO that leaks into the backend band. Fig. 10 shows an example. Currently the software drops 154 channels at both ends when a calibration scan is performed. However the whole spectrum is kept and written.

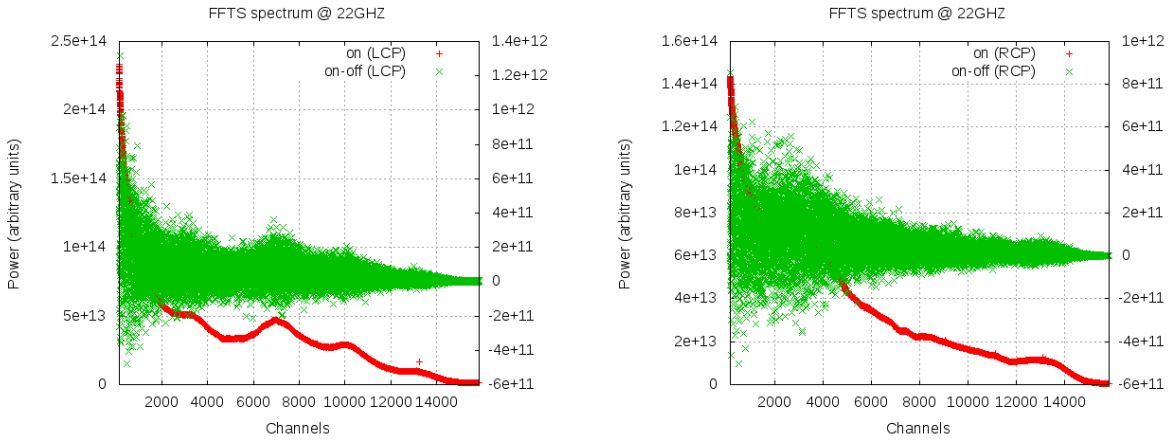

Figure 9: *Spectrum of the 22 GHz IF band at both polarizations (right: RCP, left: LCP). The power for both, the on and onoff curves, is in arbitrary units. The power fall off at channel 14750 is due to the 450 MHz bandwidth of the filter in the preprocessor board. The noise is correlated to the power of the input signal.*

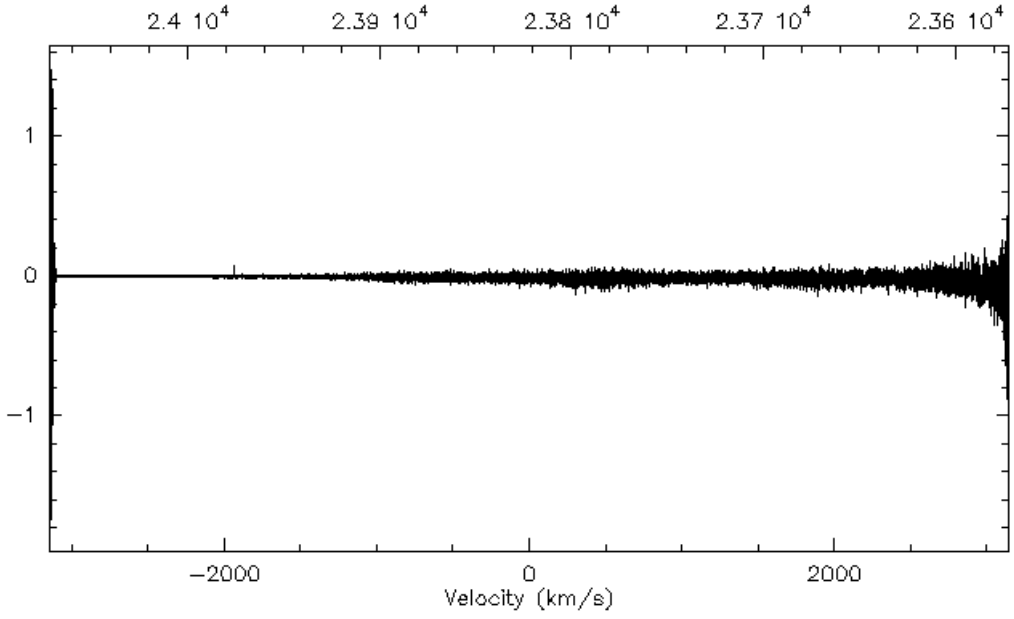

Figure 10: Complete spectrum at 22 GHz towards CepA with high levels at both ends.

8 FFTS ACS Component

The control system of the 40m radiotelescope is based on ACS. As a part of the system, the FFT Spectrometer is controlled by an ACS component written in Python. The code can be found under the module ffts of the aries21 branch of the CVS. The component has the following methods:

- start(): Starts measurement in first phase
- stop(): Stops measurement after last phase
- abort(): Aborts measurement after actual phase
- configure(): Activates all commanded settings
- initSynthesizer(): Initializes all FTTS onboard synthesizer
- getState(): Gets FFTS state
- getBlankTime(): Gets blank time in microseconds
- setBlankTime(in long useconds): Sets blank time in microseconds
- getSyncTime(): Gets sync time in microseconds
- setSyncTime(in long useconds): Sets sync time in microseconds
- getNumPhases(): Gets number of blank/sync phases
- setNumPhases(in long phases): Sets number of phases
- getBlankSyncMode(): Gets blank/sync mode
- setBlankSyncMode(in MODE mod): Sets blank/sync mode
- getUsedSections(): Gets selected sections to transmit spectra
- setUsedSections(in usedSectionsSeq sections): Selects sections to transmit spectra
- getVersion(): Gets AFFTS software version number
- getRelease(): Gets AFFTS release date
- calibrateAllADC(): Calibrates all ADCs
- getInfo(): Gets board information
- dump(): Dumps out spectra from selected boards
- saveDelays(): Saves all ADC delays
- loadDelays(): Loads ADC delays

- getNumSpecChan(in long band): Gets number of spectral channels
- setNumSpecChan(in long band, in long channels): Sets number of spectral channels
- getBandwidth(in long band): Gets section bandwidth in MHz
- setBandwidth(in long band, in long bwMHz): Sets section bandwidth in MHz
- getMirrorSpectra(in long band): Gets mirror spectra status
- setMirrorSpectra(in long band, in MIRRORSPECTRA mirror): Sets mirror spectra
- calibrateADC(in long band): Calibrates/optimizes section ADC
- setDelay(in long band, in long delay): Adjusts timing between ADC and FPGA
- getTime(in long band): Gets GPS IRIG-B time and date
- getTemperature(in long band): Gets board temperatures
- setSpecFilter(in long band, in long specfilter): Removes interference line

The ffts component obtains static information from the configuration database like the IP address of the host where the AFFTS program runs and the port numbers used for the UDP and TCP connections. Once the FFTS is configured and the measurement process is started, the ffts component reads spectral data through the TCP connection. There are two issues that have to be taken into account regarding data retrieval, first, the MSG_WAITALL socket flag must be used with the read system call because the data is sent in bunches, and second, binary data must be unpacked from the received string. Once the data is unpacked and processed it is sent through a notification channel to any client subscribed to it.

9 Using the FFTS component from ACS clients

Below we provide an example of the configuration of the FFTS from a python ACS client, which we comment below:

```
ffts = sc.getComponent("FFTS")
ffts.stop()
ffts.setBlankSyncMode(backends.ffts.INTERNAL)

tint = 5
ffts.setSyncTime(int(tint*1e6))
ffts.setBlankTime(5000)
ffts.setUsedSections(sectSequence)

# Spectral setup
ch = 16384
section = 1
ffts.setNumSpecChan(section, ch)
section = 2
ffts.setNumSpecChan(section, ch)

ffts.configure()
ffts.calibrateAllADC()
```

```
print datetime.datetime.now(), " FFTS already set"

print datetime.datetime.now(), " Start integrating ..."
ffts.start()
print datetime.datetime.now(), "Started integrating ..."
```

- `stop()`. To avoid that the FFTS gets in a un undesired setup the delivery of data is stopped
- `setBlankSyncMode()` The synchronization of the signal is done internally since no sync generator is currently available at the 40 m. These signals are devised for a chopper or wobbler that perform a fast onoff.
- `setSyncTime()` The synctime is equivalent to the integration time in microseconds. The maximum allowed integration time is 5 seconds.
- `setBlancTime()` The blank time is the time between phases during which the data is not valid. This is thought for the transition between a wobbler or chopper position. Since no phases are used no blank time would be required. However the FFTS requires this number to be larger than 0. We have set it to its minimum, 5 ms.
- `setUsedSections([])` The sections to be used are commanded as a python sequence. For example `[0, 0, 1, 1, 0, 0, 0, 0]` states that only modules 3 and 4 will be used.
- `setNumSpecChan()`. The spectral setup is achieved by stating the number of channels we are interested in. This number must always be equal or lower than the number of channels set in the core. If the number is less, only the data from these number of channels are delivered. For example if the loaded core has 16384 channels and the channels set with this command is 1638, only the first 1638 channels will be delivered.
- `configure()` loads the previous options into the FFTS. This instruction takes around 14 seconds to complete when the integration time is 5 seconds.
- `calibrateAllADC()` The analog to digital converters need to be calibrated before delivering the data.
- `start()`. Data acquisition starts from this instruction.

If the FFTS were being used before loading this script, the data may appear in the notification channel before expected. It is usually adavisable to stop the data acquisition and wait for the integration time to elapse before grabbing new data.

The data are pushed into a notification channel. The data from every module is sent in a structure which contains:

- the integration time in seconds.
- the phase number

- the backend number (or module)
- the number of channels of the module
- an array with data. The size of the array is given by the previous number in this list.

The client that suscribes to the FFTS notification channel reads the structure and discovers where the data comes from and how many channels the array has, using the information from the first 4 numbers. The phase number is ignored because we are not currently using a sync signal.

Below we include an example of the suscription to the notification channel and the definition of the method that acquires the data:

```
onsumer(backends.FFTSCHANNEL)
g.addSubscription(backends.fftsDataBlock, fftsDataHandler)
g.consumerReady()
```

Method `fftsDataHandler` needs to be redefined. For example as below:

```
def fftsDataHandler(fftsStr):
    global fOut

    fOut = open('fft_16384channels.log','w')
    integrationTime = fftsStr.integrationTime
    phaseNumber = fftsStr.phaseNumber
    backendNumber = fftsStr.backendNumber
    numChannels = fftsStr.numChannels
    data = fftsStr.data


    sn = datetime.datetime.now().strftime("%H:%M:%S.%f")
    strLine = '%s FFTS (integrationTime: %d, phaseNumber: %d,  backendNumber: %d, numberofChannels: %d, len(da
              (sn, integrationTime, phaseNumber, backendNumber, numChannels, len(data))
    print strLine

    for ch in data:
      fOut.write("%f\n" % ch)

    fOut.close()

    return
```

The previous python method prints information from the notification channel and writes the data on a file (one line per channel) whose name is passed as a global variable.

The `stop()` instruction, which is not included above, stops the generation of data from FFTS. It may take some time to complete depending on the integration time and the moment it is issued. The stop never aborts a current data acquisition and only affects the next integration period.

9.1 Regular usage from the obsEngine and Fits writer component

The FFTS is managed from the obsEngine in a similar way to the one described in the previous subsection. The FFTS is commanded only before each scan, and never in between subscans. If a scan is cancelled, a `stop()` instruction is sent to the FFTS component. The start and stop instructions are only sent from this component.

The FITS writer component uses a similar code to the one included above, but in C++. The component suscribes to the notification channel and fills some arrays which are kept in memory. The data acquisition is not stopped in between the subscans to speed up the data acquisition. Indeed this component never starts or stops the FFTs component. To avoid writing data in the FITS file while the telescope is slewing, an internal variable controls wether the telescope is in the requested position or not. If it is not the data are discarded. The typical dump time is 5 seconds and hence the subscans integration time should be a multiple of this value.

References

[RPG2010] Radiometer Physics GmbH, ”RPG-FFTS Fast Fourier Transform Spectrometer. Manual”. 2010.