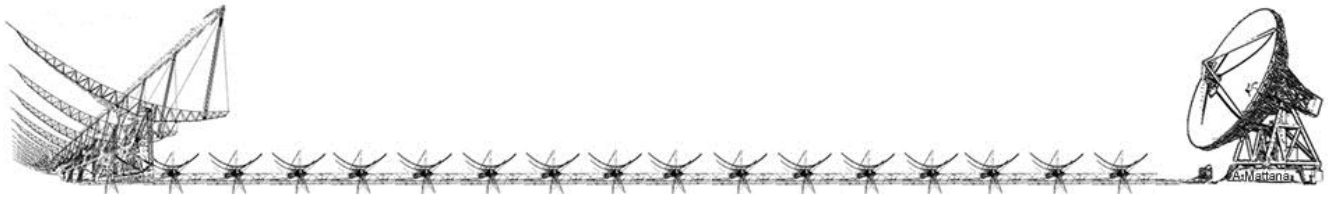
Space Debris Digital Beamformer based on CASPER Hardware

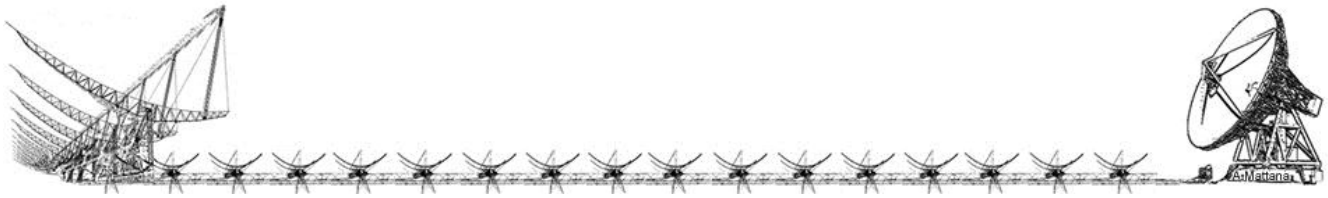
A. Mattana[1], *G. Naldi*[1], *G. Pupillo*[1]

IRA 462/12

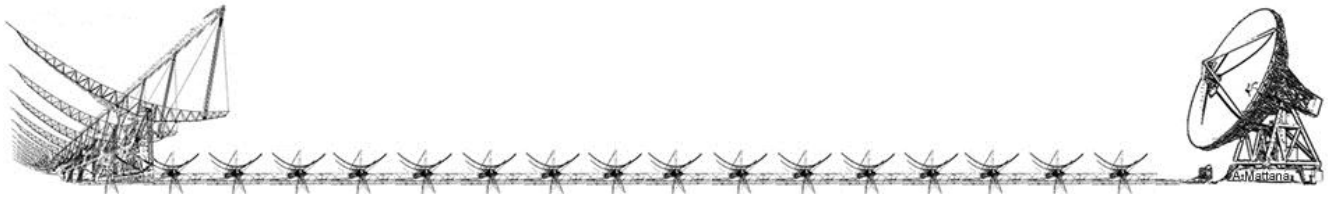
1) *Istituto di Radio Astronomia, Bologna, INAF*



Referee: Monari Jader



Contents



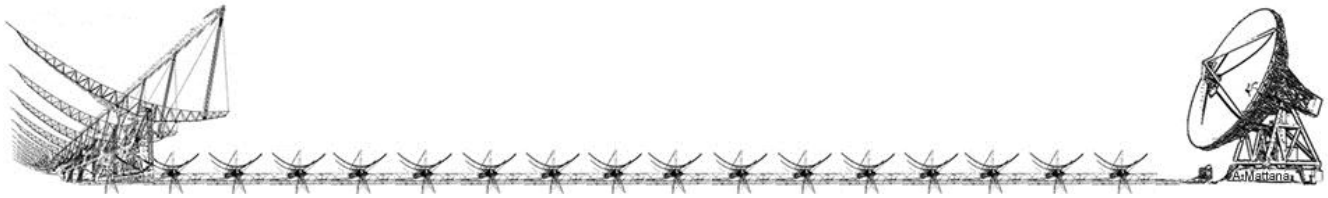
Preface

This document will describe deeply the Beamformer system starting from the Hardware Architecture, trough the operational phases (scheduling, observing), to the data process till to obtain the beam of the observation target which can be a well-known radio source, useful to equalize system parameters, or as for this specific application, a space debris.

This Beam former system takes the input signals coming from single Northern-Cross Radio Telescope antenna receivers, aligned in East-West, and “form” a unique beam describing the power of the observed target. Depending of the type of the observation the processed beam can show a total power of a radio source or even a doppler shift of a debris transiting over the field of view of the antenna whilst illuminated by a transmitter.

The beamformer system has been mainly developed for radar measurements of space debris orbiting around Earth. In these observations different sections of the Northern Cross array are used as the receiving part of a bistatic radar system operating in UHF band in conjunction with a proper transmitter. Due to the particular features of the received radio echoes, the beamformer has been optimized in order to fulfill some specific requirements, such a narrow receiving band and a high time accuracy.

We would like to thanks Marco Schiaffino and Marco Bartolini for their useful contributes.



System Requirements

Northern Cross antenna has been used as receiving part of a bistatic radar for Space Debris detection. This radar configuration (Fig. 1) uses transmitting and receiving antennas at different locations allowing the transmission of CW signals. The antenna beams are kept in a fixed direction with respect to the Earth (beam-park technique) so that, when an object passes through the common volume at the beams intersection, it produces a radio echo.
In CW unmodulated trasmissions, the echoes are expected to be quasi-monochromatic and, consequently, signal post-processing is mainly performed in the frequency domain.

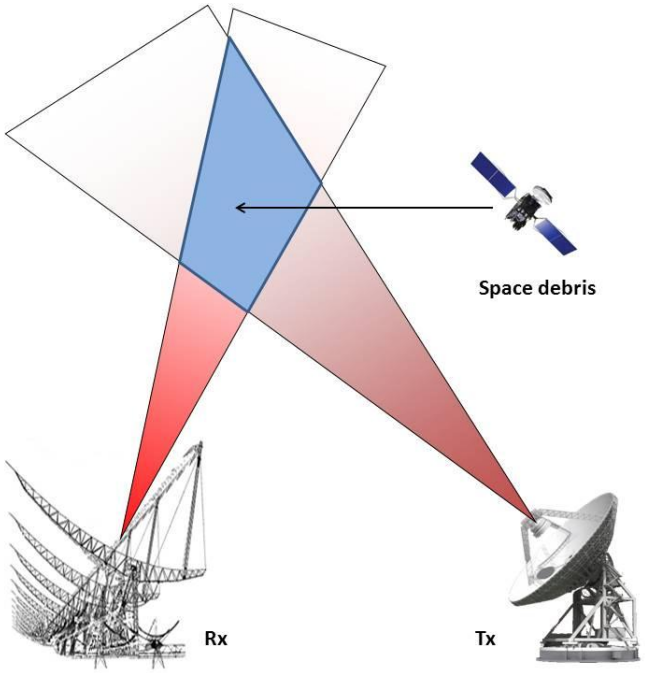
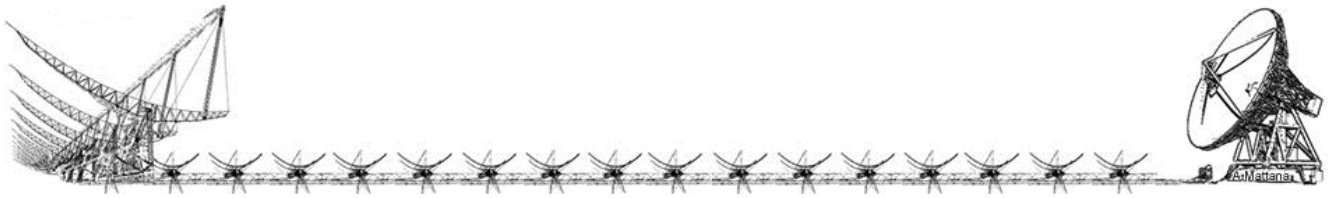


Fig. 1: Bistatic radar configuration geometry.

The most part of our potential radar targets are in low Earth orbit (LEO), i.e. below an altitude of about 2000 km from the Earth surface. This is the most polluted orbital region in which radars are very sensitive and outperform all other types of sensors (optical telescopes, etc.).



The Northern Cross space debris bistatic radar observations are carried out in beam-park mode, without tracking the target. Due to the high angular speed of the objects in LEO, the typical duration of a space debris echo is of the order of the second.

In a bistatic configuration the frequency Doppler shift, Δf, of the received signal is given by:

$$\Delta f = -\frac{1}{\lambda}\left(\dot{R}_{Tx} + \dot{R}_{Rx}\right) \qquad [\mathrm{Hz}]$$

where

λ = transmitted wavelength [m]
$\dot{R}_{Tx}$ = target radial velocity respect to the transmitter [m/s]
$\dot{R}_{Rx}$ = target radial velocity respect to the receiver [m/s]

Considering a maximum speed for a debris in LEO of 7.8×10^3 m/s (assuming a circular orbit) and a signal wavelength of 0.735 m, the bistatic Doppler shift doesn't exceed $\pm$22 KHz in frequency.

The beamformer receiving band of 100 KHz is optimized for space debris observations because it permits to detect any possible radar echo coming from a LEO target and, at the same time, prevents the storage of large amounts of data acquired in the time domain.

Hardware

CASPER group

The term CASPER means “Collaboration for Astronomy Signal Processing and Electronics Research”. The CASPER was born at the Berkeley University of California, with a collaborations of several institute and laboratories. The primary goal of CASPER is to streamline and simplify the design flow of radio astronomy instrumentation by promoting design reuse through the development of platform-independent, open-source hardware and software.

The CASPER group aim is to couple the real-time streaming performance of application-specific hardware with the design simplicity of general-purpose software. By providing parameterized, platform independent gateware libraries that run on reconfigurable, modular hardware building blocks, they abstract away low-level implementation details and allow astronomers to rapidly design and deploy new instruments.

IBOB and BEE2 boards

The Casper researcher group has been develop more than one powerful board based on Xilinx FPGAs, the boards used for this specific application are the IBOB and the BEE2. Every CASPER boards are sold without firmware but with a lot of primitives which allow to develop your own application.

The IBOB (Interconnect Break Out Board) is the very first product of the CASPER, it has been developed on the 2005, that means, the libraries we are using today to implement acquisition system over this hardware have been carefully checked and tested from many engineers working in the astronomical world since seven years.

IBOB board main features are:

- a Xilinx Virtex-II Pro FPGA programmable via a JTAG port (setting specific jumpers, it can load the firmware at the switch on from an on-board EEPROM);

- 2x CX4 10Gbps serial connectors;
- 1x RJ45 Ethernet interface;
- 1x RS232 interface
- 2x ZDOK connector where plug lots of custom A/D board up to 1GSample
- 1x MDR 40 differential pair connector;
- 80x GPIO headers with selectable IO voltage;
- 2x SMA IO;
- 2x 512k x 36-bit SRAMs

The Board can be schematically represented with the block diagram of Fig. 2.

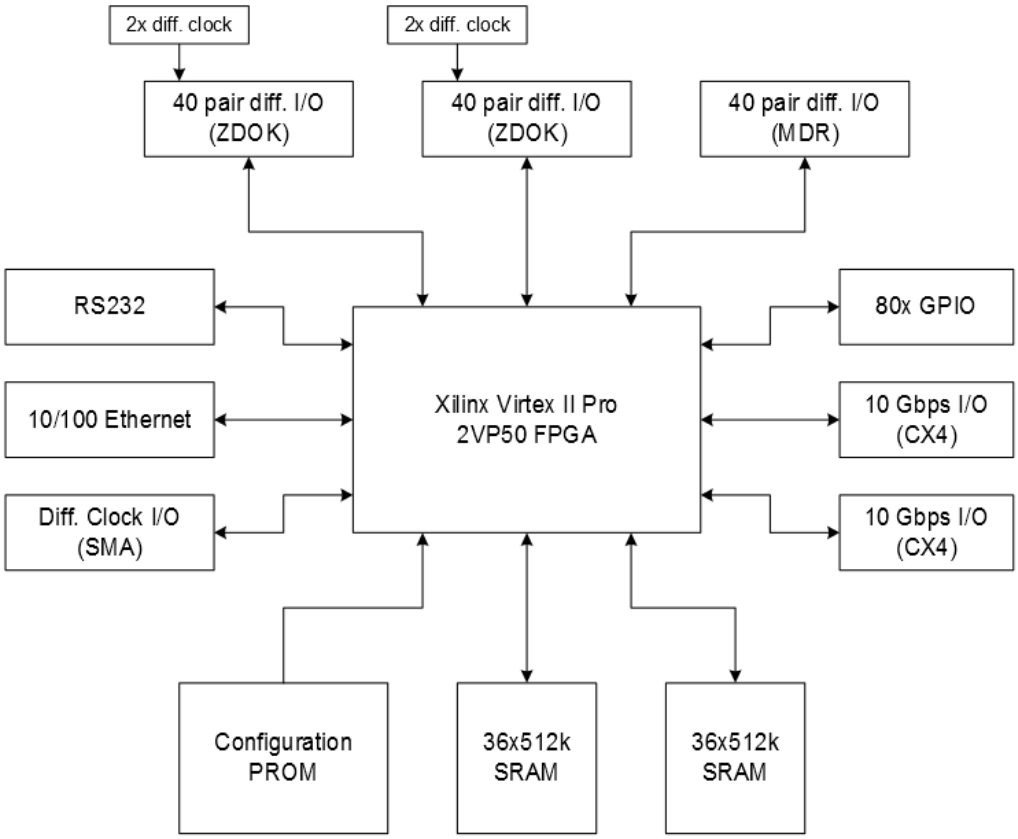


Fig. 2: Block diagram of the Ibob board general architecture

Every CASPER boards have the same kind of connector for the A/D board in order to re-use AD boards even with the next CASPER hardware generation.

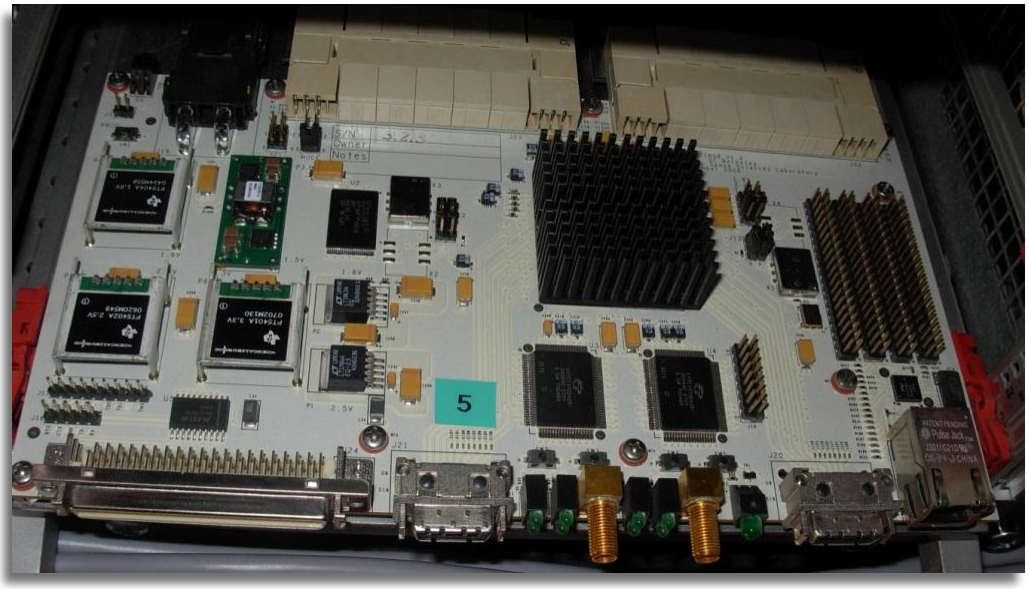


Fig. 3: The IBOB board, you can see on the top the 2 ZDOKs for A/D boards, two CX4 connectors below, JTAG pins on the left, while the Xilinx Virtex 2 Pro is behind the cooler

IBOB can mount 2 A/D board, we are using the custom “iADC”, which takes in input:

- 8 bit Dual 1Gsps;
- A ref Clock: 10MHz-1GHz, 50Ω, 0dBm;
- A pulse per second reference (PPS);

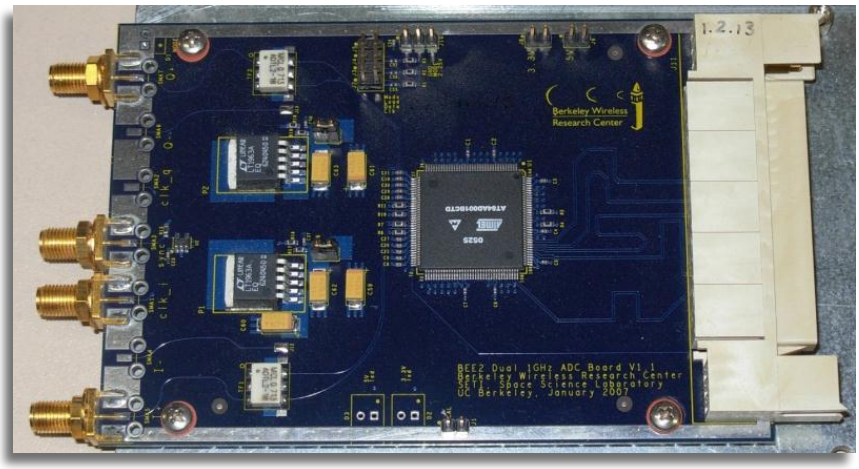


Fig. 4: CASPER iADC

Therefore the IBOB can acquire signals coming from 4 antenna receivers (Northern Cross Radio telescope has antennas only with one polarization). Data collected and processed can be sent via the 10 Gbit Ethernet link to a workstation (via UDP protocol) or even to another CASPER board (via XAUI that is a easier point to point protocol) for additional processing.

The BEE2 is a module, which is a single printed circuit board containing five FPGAs arranged within a very high-speed inter-chip and inter-board communication structure. Additional peripherals are present to enhance usability and permit a fully functional free-standing computing system. Each FPGA has direct access to four DDR2 memory modules for program and intermediate result storage.

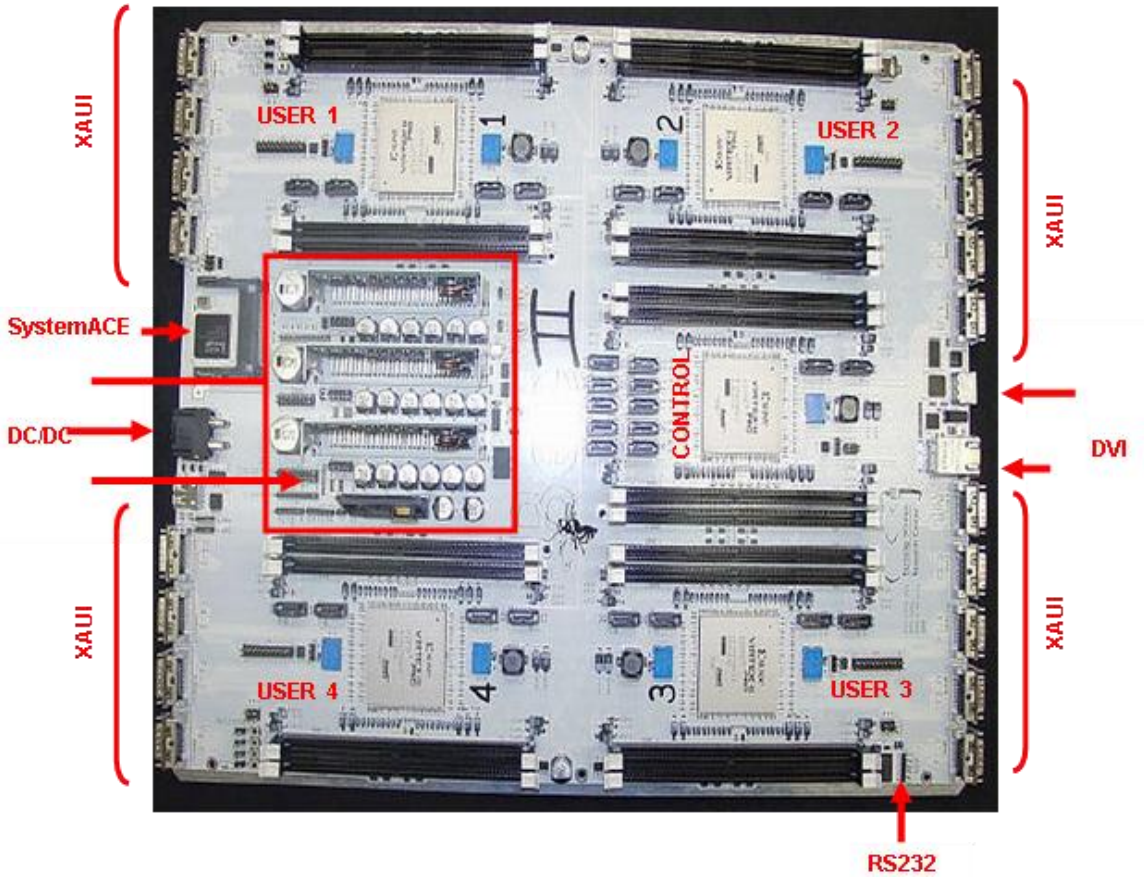
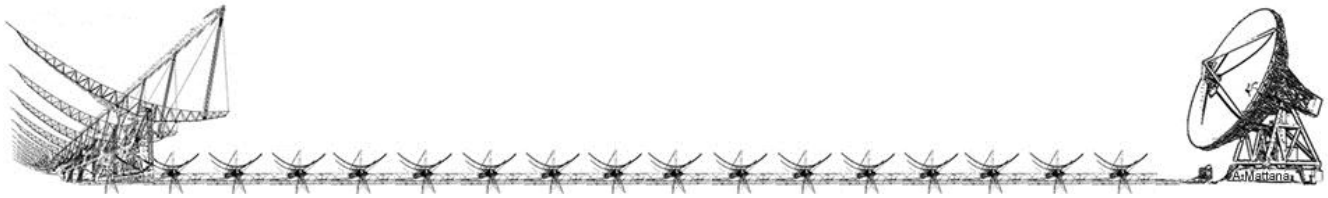


Fig. 5: A picture of the BEE2 board whose principal components are pointed out.

- 5 Xilinx Virtex-II Pro 70 FPGAs, Speed Grade 6
- Up to 40 GB DDR2 SDRAM
- SystemACE controller and Type II CompactFlashTM for configuration and data storage
- USB port
- DVI port
- SelectMAP configuration from control FPGA
- 10/100 Ethernet PHY
- RS-232 serial port (control FPGA)
- Status LEDs
- Serial ATA ports
- 18 AUI links (8 or 10 Gbps)
- Voltage and temperature monitoring
- Real Time Clock
- On-board power supplies
- Panel display
- Internal or external system clock and user programmable clock

The basic compute elements on the module are five Xilinx Virtex 2 Pro 70K FPGAs. The center FPGA, termed the control FPGA, has primary responsibility for system management. Each of the remaining four user FPGAs are tasked with computional workloads and can communicate with the control FPGA at a rate of 2.5GB/s. The user FPGAs can also inter-communicate at a speed of 5GB/s via a ring configuration. These Intra-module links are implemented using parallel buses utilizing general purpose I/O signals of the Xilinx FPGA's high-speed serial links.

The selected product family has an additional 20 high speed serial RocketIO blocks implemented directly on chip which can be used of up to 3.125Gb/s, and in this instance, specificed to meet the IB4X description common to both by the Infiniband and the 10Gb-Ethernet-CX4 standards. It groups four 2.5Gb/s (Infiniband) or 3.125Gb/s (10Gb-CX4) differential pairs in a single unit to reach an effective maximum bit-rate of either 8Gb/s or 10Gb/s per port.



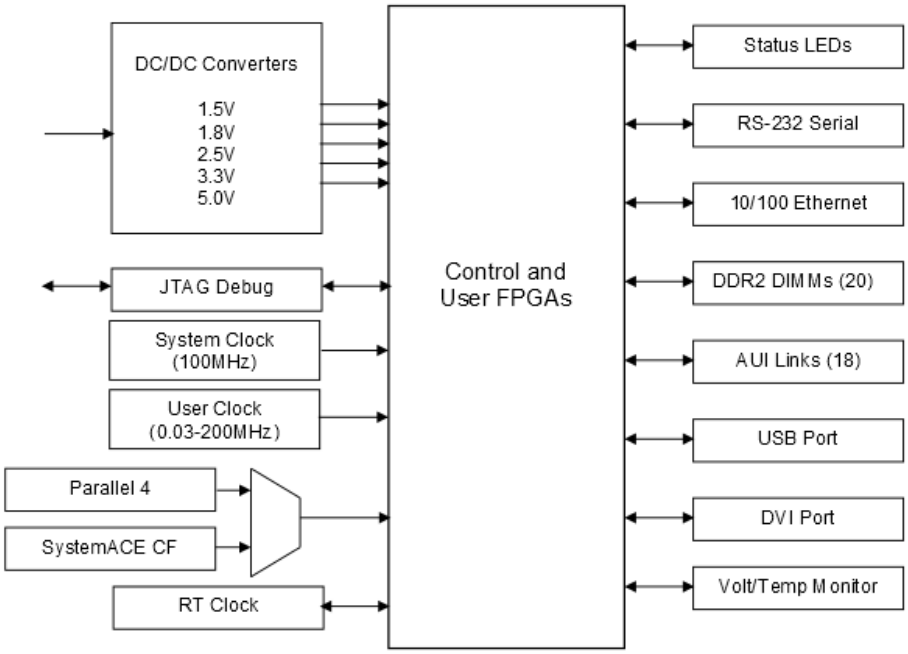


Fig. 6: Block diagram of the BEE2 board general architecture

Each user FPGA has four IB4X ports, whereas the control FPGA has only two. Each individual port requires four RocketIOs, which leaves four transceivers unused on the user FPGAs, and twelve on the control FPGA. Up to four DDR2 DIMMs can be attached to each FPGA. Individual 72-bit wide DRAM modules can be accessed at speeds of up to 400Mb/s/wire, corresponding to an aggregate throughput of 3.4GB/s per DIMM.

A brief schematic representation of the data flow can be the follow:

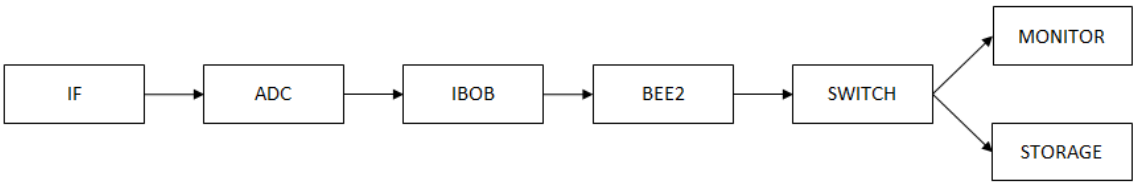


Fig. 7: Data Flow scheme

Signals coming from 4 receivers (that have to be calibrated in phase and equalized in amplitude before) will be digitalized and acquired from the IBOB that apply a first stage of filtering and processing. Then a second stage of filtering will applied on the BEE2 which makes the beam by summing the time domain data and sends the result to workstations for the analysis.

The time accuracy is demanded to the IBOB firmware which synthetized an internal clock synchronized “indirectly” via a NTP server placed on the Medicina Dish Control Room where time accuracy is guaranteed by a Hydrogen Maser Atomic Clock. A 10 MHz Sinewave locked to the Maser is distributed to every backends. The iADC sampler is locked to this 10Mhz, and the control software during the system initialization perform a time update with the IBOB internal clock (*see chapter “Firmware/IBOB”*).

Communication between the BEE2 and workstations for control, monitor and storage must be supported by a 10Gbit Ethernet Switch with CX4 ports. We have used the Fujitsu XG700 12 Ports Layer 2 Switch which provides a throughput 240Gbit/s.

Fig. 8: Fujitsu XG700 12CX4 Ports 10Gb Switch

Connection Scheme

The Beamformer Digital Backend has been realized in the Medicina “Receiver Room”. It is placed in the South-East room corner where three 19’’ inch racks hosts most of the devices used for this project. The overall project scheme is as follow:

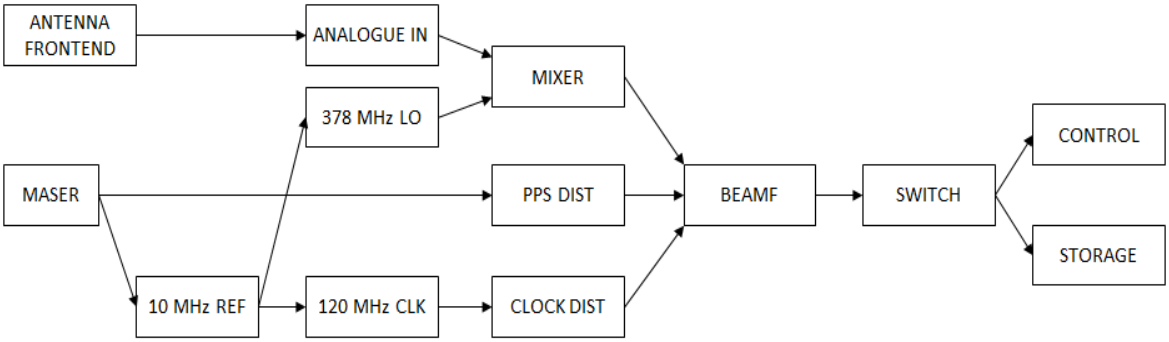


Fig. 9: Overall scheme

Synchronization and is guaranteed by the distribution of a 10MHz synewave derived by the hydrogen maser atomic clock that stabilizes the local oscillator used by a mixer that converts the RF input signals to 30MHz Ifs. The Beamformer sends the generated time domain beam to workstations trough the 10GbE switch.

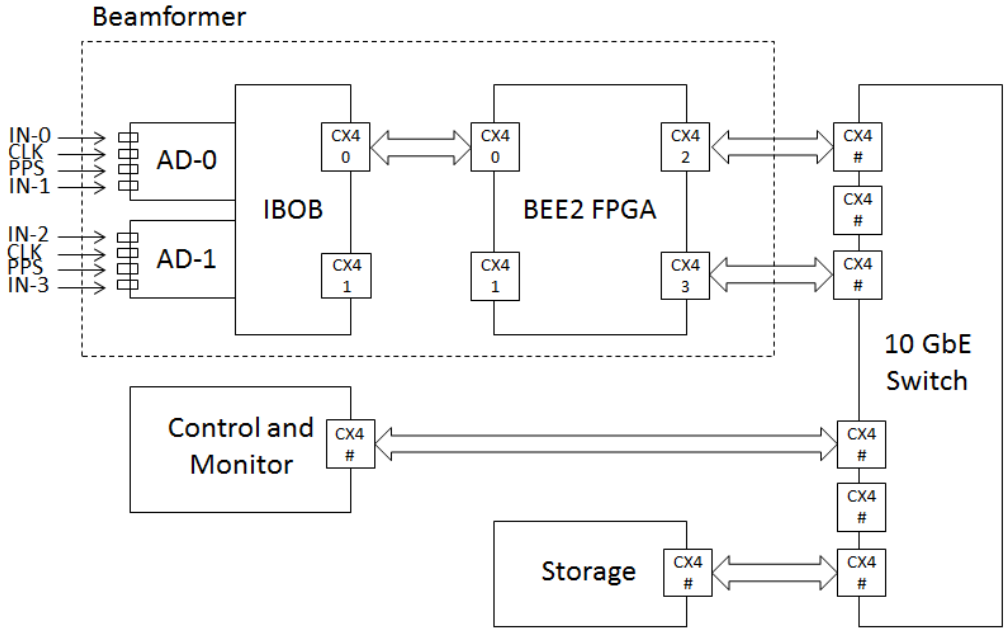


Fig. 10: Basic connection scheme

Each IBOB is able to manage 4 IF, that for the Medicina's Northern Cross Radiotelescope means 4 antenna receivers (Northern Crosso does not have the dual polarization). After the first stage of filters the data reachs the BEE2 via the 10GbE link (CX4 port 0) and the synthetized beam will leave the beamformer on the BEE2 CX4 port 3 with UPD packet destinated to the storage machine. A copy of the same data will reach the "control and monitor" machine (always trough the switch) using the CX4 port 2.

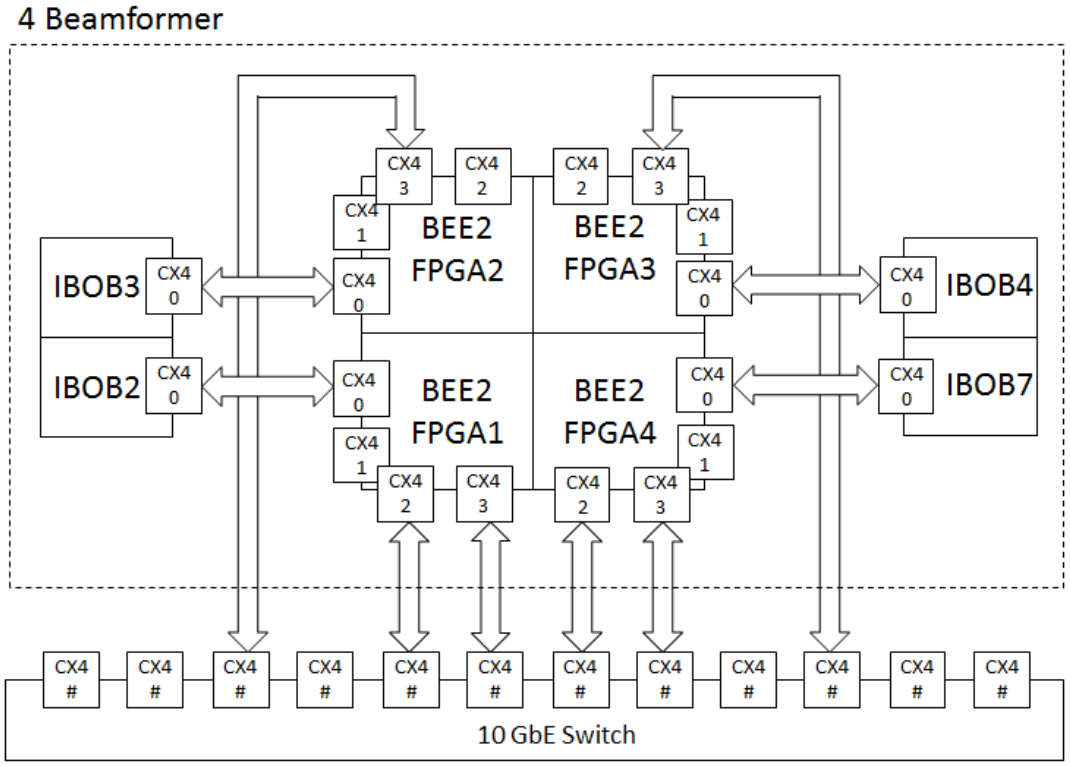


Fig. 11: 4 Parallel Beamformer System Connection Scheme
(control and storage must be connected as in the previous scheme)

One beamformer uses just one BEE2 FPGA, it is possible to run till four beamformer at the same time using 4 IBOB and just one BEE2 having redundancy. The number of the connections increase, but it is not mandatory to have the real time monitor, so you can disclaim to connect the BEE2 FPGA CX4 port 2 where you believe is not necessary.

We found very useful in our tests to make observations using different configuration of the input signals at the same time, because if an antenna receivers has trouble, due for example to weather conditions during observation, the redundant system has most likely given however good results.

Fig. 12: Picture of the Local Oscillator set to 378MHz in input to the MIXER to generate a IF of 30MHz

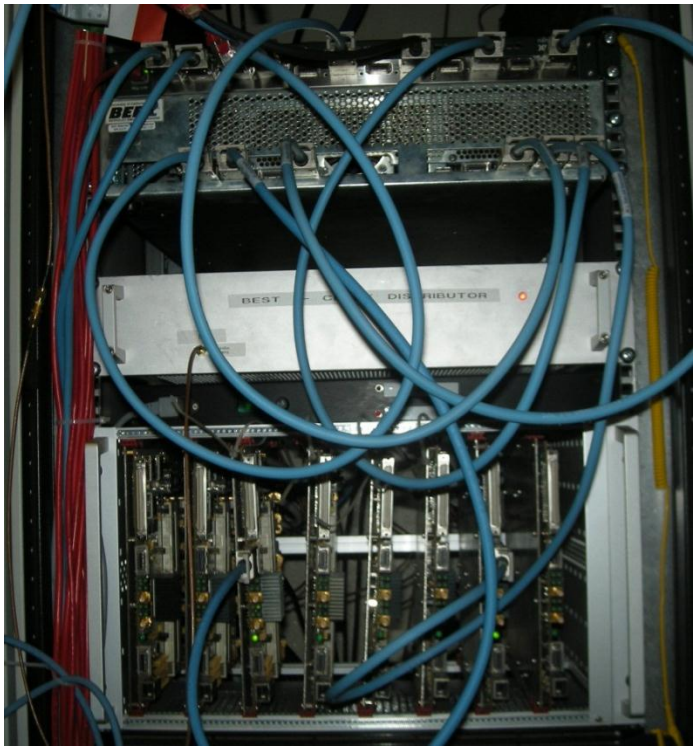


Fig. 13: Picture of the main Beamformer Digital Backend modules

The above picture shows the Beamformer, you can recognize from the top:

a. The Fujitsu XG700 12CX4 Ports 10Gb Switch.
b. The BEE2 front side with the CX4 of the FPGA 1 and 4.
c. The Clock distributor, front side is a -19db input clock, on the rear there are 32 splitted output.
d. The PPS distributor (in/out on the rear).
e. 8 IBOB boards, input signals are in the rear side while in the front there are the CX4 port.

Firmware

IBOB

IBOB firmware has been written by using the Xilinx standard library combined with CASPER IBOB customized libraries (highlighted in yellow) which help to address board resources on your synthetized project. The below picture is the Matlab Simulink model file that the system generator will use to generate the HDL code. Going into details the firmware can be explained in the next schemes.

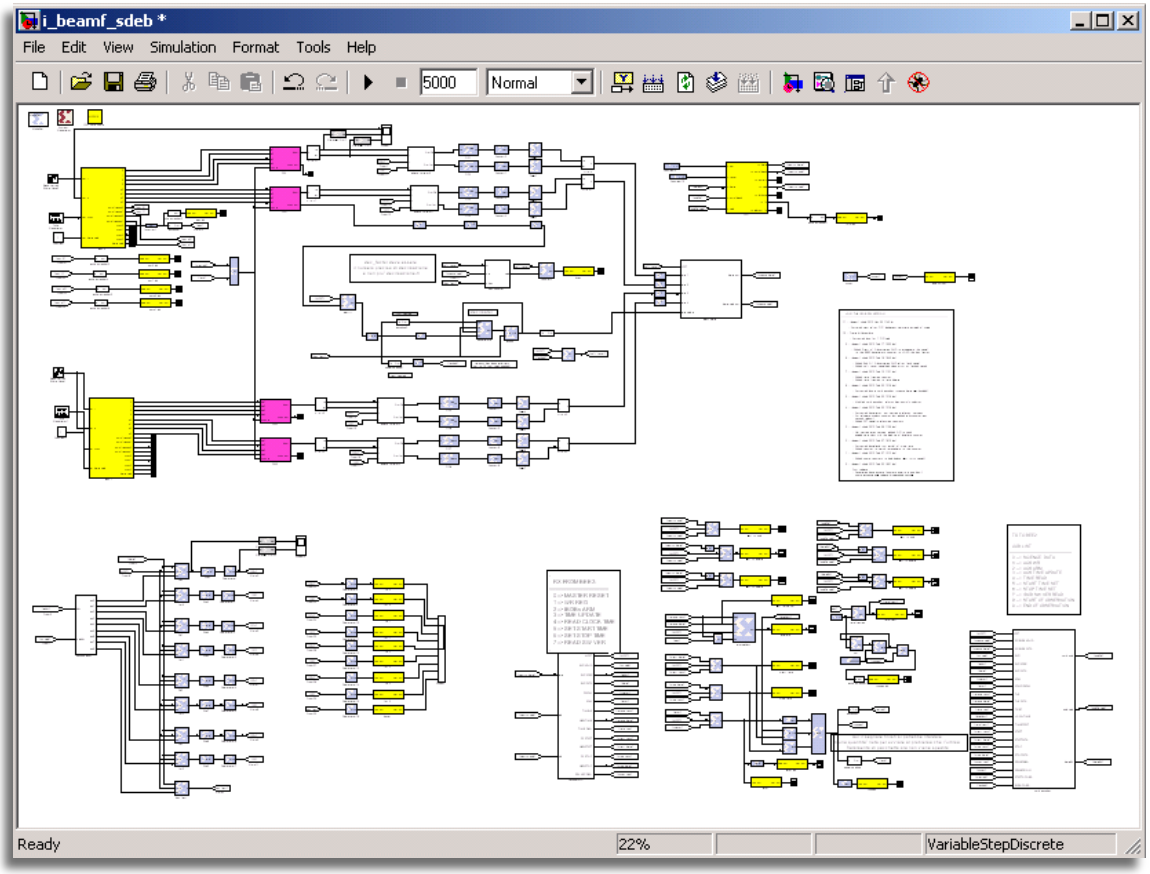


Fig. 14: Matlab Simulink Screenshot of the IBOB project

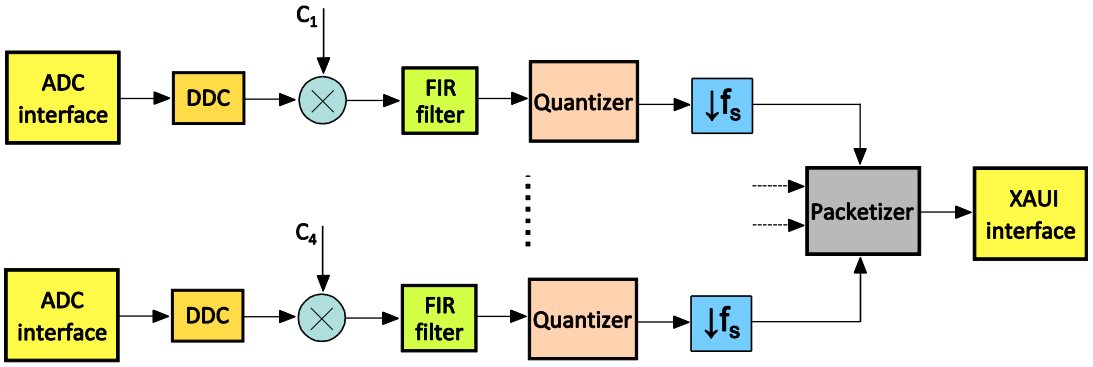


Fig. 15: IBOB firmware architecture

The four IF real analog signals are digitized by the ADC cards with 8 bits precision and a sampling rate of 120 MSample/sec. Then they are converted to a base-banded complex signals centered at zero frequency by a DDC block (Fig. 16).

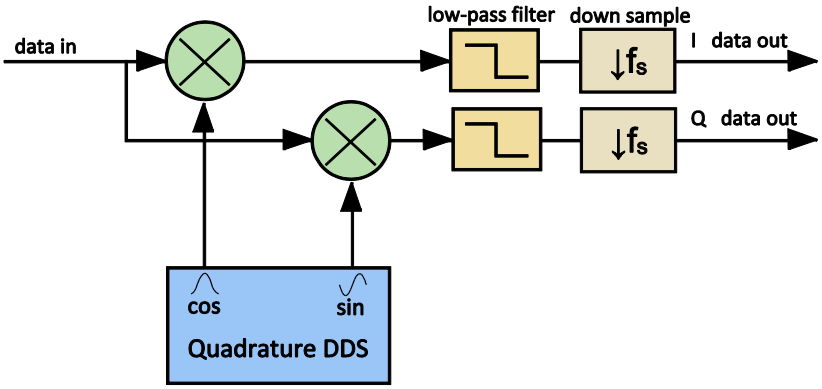


Fig. 16: DDC schematic

The DDS generates a complex sinusoid ($e^{j2\pi f} = \cos(2\pi f) + j\sin(2\pi f)$) at the intermediate frequency. Multiplication of the intermediate frequency with the input signal creates images centered at the sum and difference frequency (which follows from the frequency shifting properties of the Fourier transform). Properly designed lowpass filters (in relation to the IF bandwidth), placed after this multiplication, pass the difference (i.e. baseband) frequency while rejecting the sum frequency image, resulting in a complex baseband representation mathematically equivalent to the original signal. Thanks to this property, the complex baseband signal can be appropriately down-sampled without losing any information.

In the case of the project the IF frequency is 30 MHz, the IF analog bandwidth is variable and it can be 2.7 MHz, 5 MHz or 16 MHz according to the receiver system connected to the ADC. During Space Debris observational campaigns the most used bandwidth is the first: 2.7 MHz.

The lowpass filter of the DDC has been designed using the Filter Design and Analysis Tool of MATLAB. The main characteristic parameters of the FIR filter are shown in the picture below.

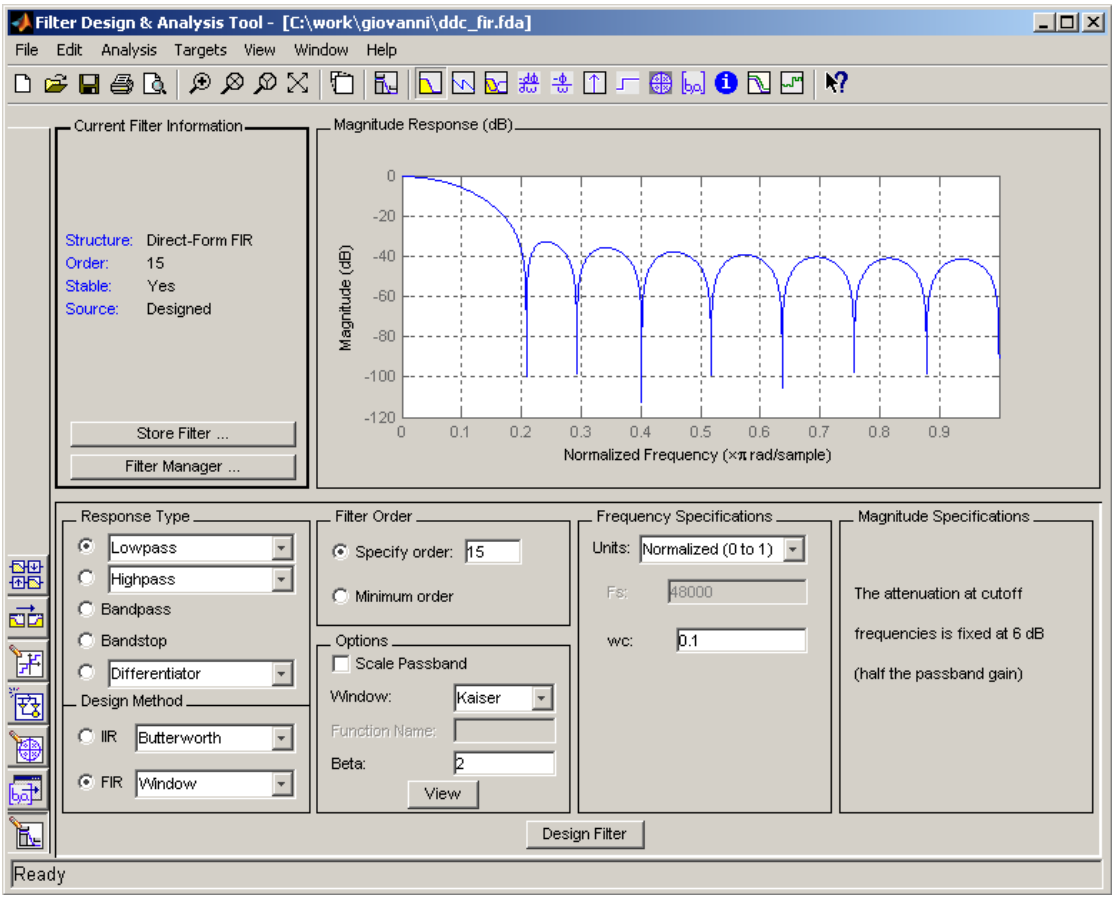


Fig. 17: design parameters of the FIR filter inside the DDC block.

The normalized cutoff frequency (wc) of the FIR filter is 0.1 of the real processed bandwidth which is 60 MHz (since the ADC sampling rate is 120 MSample/sec). So the filter cuts at 6 MHz.

The DDC block synthesized in the FPGA is optimized (in terms of resource usage) to have a down sample factor of 4. Nevertheless with this filter it would be allowed a down sample factor even higher (up to 10); however oversampling the signals prevents from any undesired effects of aliasing.

In general when mathematical operations (e.g. addition or multiplication) are applied to signals their binary representation grows in terms of number of bits. Obviously the greater the number of bits, the higher the percentage of hardware resources occupation in FPGA. This is the case of the DDC in which there are several multiplications and one addition. For this reason a re-quantization of data is accomplished just after the decimation stage: in particular we pass from 39 bits to 8 bits binary representation of signals (8 bits for real part and 8 bits for imaginary part), of course allowing a certain but tolerable loss of precision.

After being down converted by the DDC, the signals are multiplied by 16 bits complex coefficients with unitary amplitudes in order to equalize the phases of all the signal chains. These coefficients are calculated using an astronomical calibration procedure that will be the subject of a future technical report.

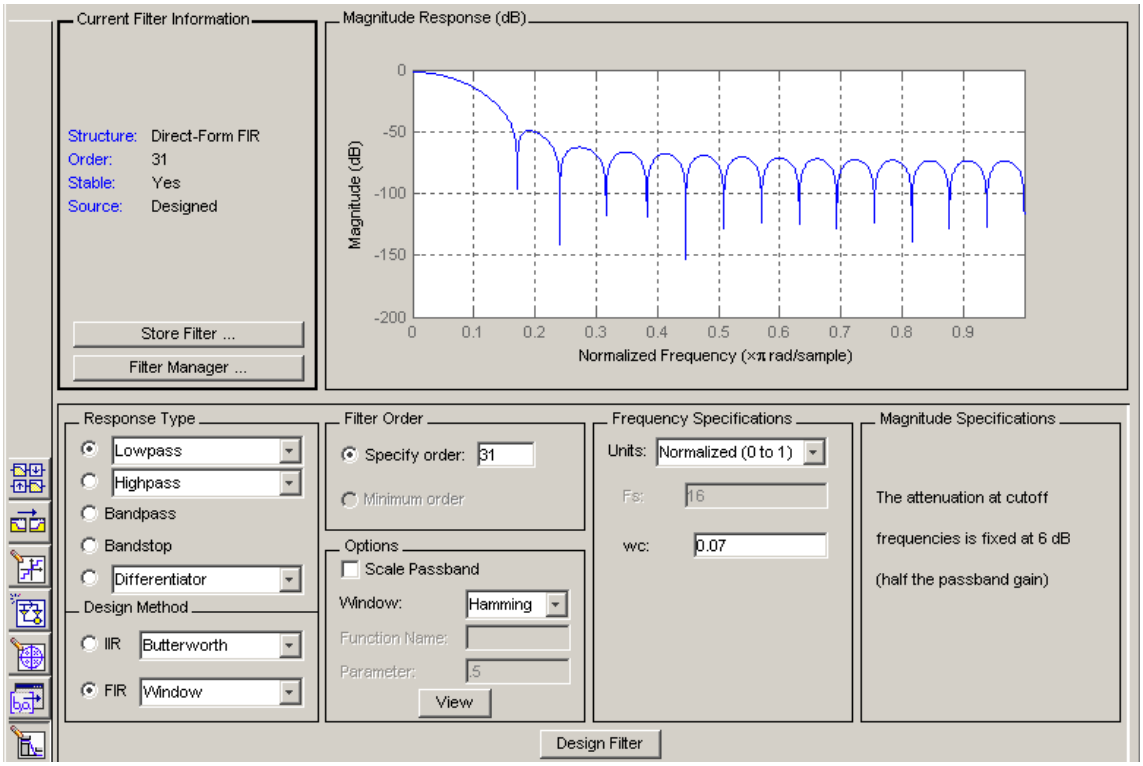


Fig. 18: Design parameters of the FIR filter synthesized in the IBOB.

Given the requirement to have at the output of the system raw data with about 100 KHz of complex bandwidth, another lowpass FIR filter is applied to data stream. This filter has been splitted in 2 stages: one synthesized in the IBOB and the other in the BEE2 board. Unfortunately it is not possible to realize a unique FIR filter stage because of the limited number of available resources in the IBOB.

A good trade-off between size (n. of taps) and performance of the filter has been found with the design parameters shown in Fig. 18.

With 30 MSample/sec of sampling rate and 0.07 of filter wc, it follows that the filter cuts at $1.05\ \mathrm{MHz}\ (15\ \mathrm{MHz} * 0.07)$. The data can be decimated setting a down sample factor of 13 still remaining in the Nyquist sampling zone. In fact 13 is a proper integer divisor of the sampling rate to have at least twice the cutoff frequency of the filter.

$$\frac{30\ \mathrm{MSample/sec}}{13} = 2.307692\ \mathrm{MSample/sec} > 2 * 1.05\ \mathrm{MHz}$$

The problem of data dimension recurs again because at this point signals are represented by 32 bits words (32 bits for real part and 32 bits for imaginary part). For the same reason mentioned before both real part and imaginary part of signals are re-quantized with 8 bits precision.

Then the resulting 16 bits complex signals of the 4 parallel chains are packetized together at the same time into 64 bits words and sent over XAUI interface. The 64 bits word is structured as schematically represented in the following table.

Position of bits from the most to the least significant

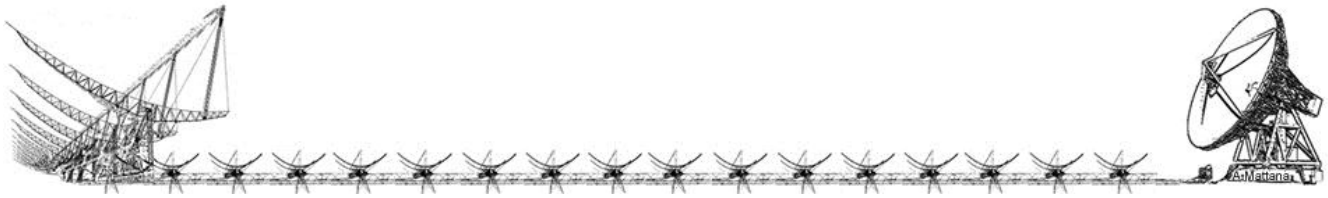
64 ... 1

Re #1	Im #1	Re #2	Im #2	Re #3	Im #3	Re #4	Im #4

Tab. 1: Structure of the 64 bits word transmitted over XAUI.

Data transfer using XAUI consists of a point-to-point regular stream of 64 bits words without overhead on CX4 connectors. So there is a direct link between Ibob and BEE2 CX4 ports.

Time accuracy is the main constraint in this kind of projects, an internal clock has been synthetized into the IBOB in order to send data to the BEE2 at a specific time window. The internal clock is locked to the Medicina station Maser Hydrogen Atomic Clock by using the PPS (Pulse Per Second) and the 10MHz reference input.



During an initialization phase the workstation that is synchronize via NTP to the Maser Clock perform a "time update" operation that consists on:

1. Wait for a "new" second (t_0), that means HH:MM:SS.0000000
2. Send the UT Timestamp to the IBOB which will certainly arrives in less than one second
3. Wait for the next new second (t_1) (in the meantime a PPS has incremented the IBOB clock)
4. Ask for the IBOB local time
5. If the Received Timestamp is equal to the workstation local time, Success.

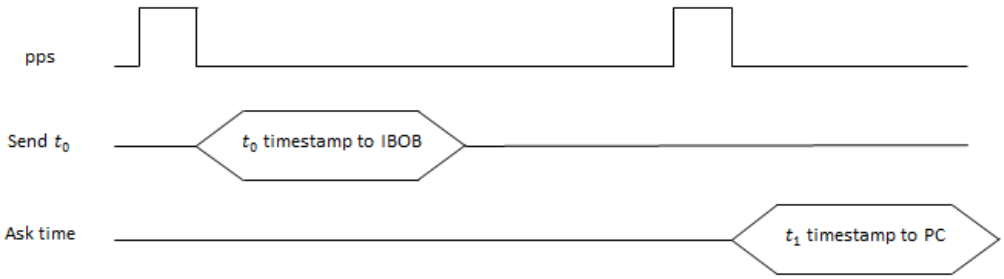
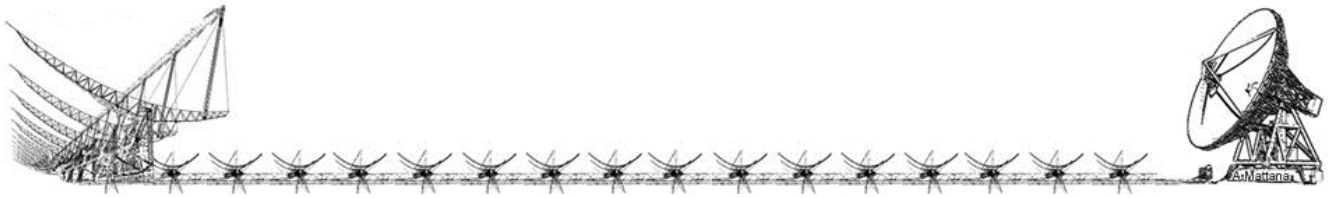


Fig. 19: Time diagram of the time update procedure

We assume that the time on the workstation is very accurate, anyway latency time to send timestamp to the IBOB (which means from PC to BEE2 over a private 10/100 Mbit Ethernet network and from BEE2 to IBOB over XAUI point to point 10Gbit link) will be very shortly and always less than 1 second. However just after a second there is a cross check asking again the time of the IBOB and it is expected to be equal. In worst case, if the previous latency time was greater than 1sec the new pps has not increased it and there is a time mismatch message reported.

Now the IBOB is up to date, it knows what time is it and what day is today. Even if not needed a counter increasing with the IBOB clock (30 MHz) provides fraction of seconds. When performing a measurement just upload to dedicated IBOB registers timestamps of the start time and end time, and, if the system has been armed data will be generated.



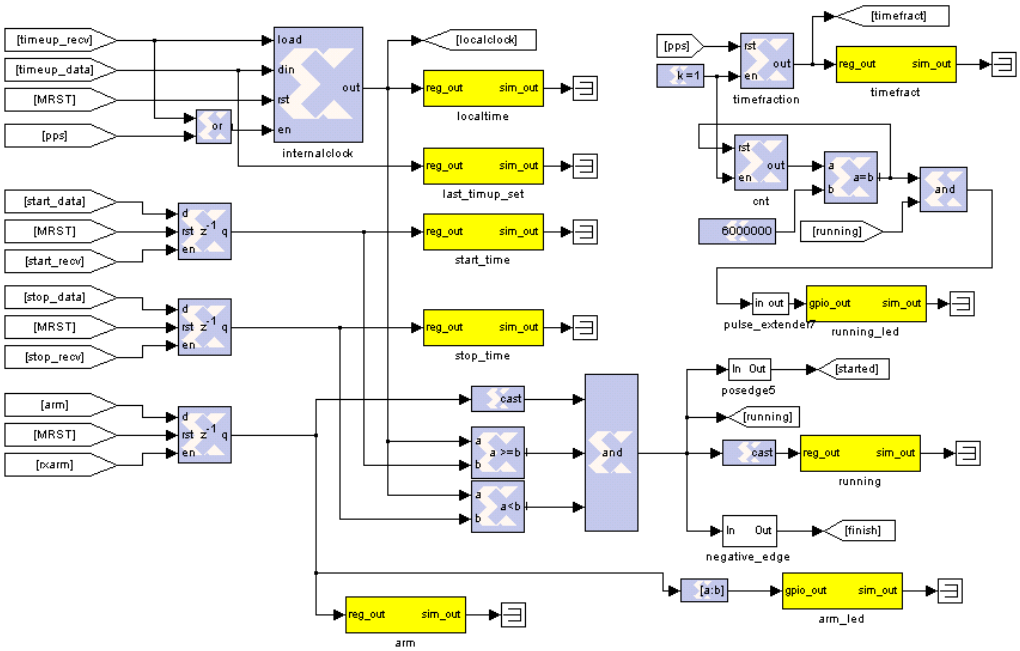


Fig. 20: IBOB Timing registers

An internal clock is necessary but is not enough for time accuracy. IBOB must guarantee that the first sample leaving the IBOB is really the first sample acquired from the A/D and then processed into the DDC and the FIR block set. A sync signal activated at the start time has been propagate on the entire chain taking into account each block latency and it indicates to the data packetizer when start.

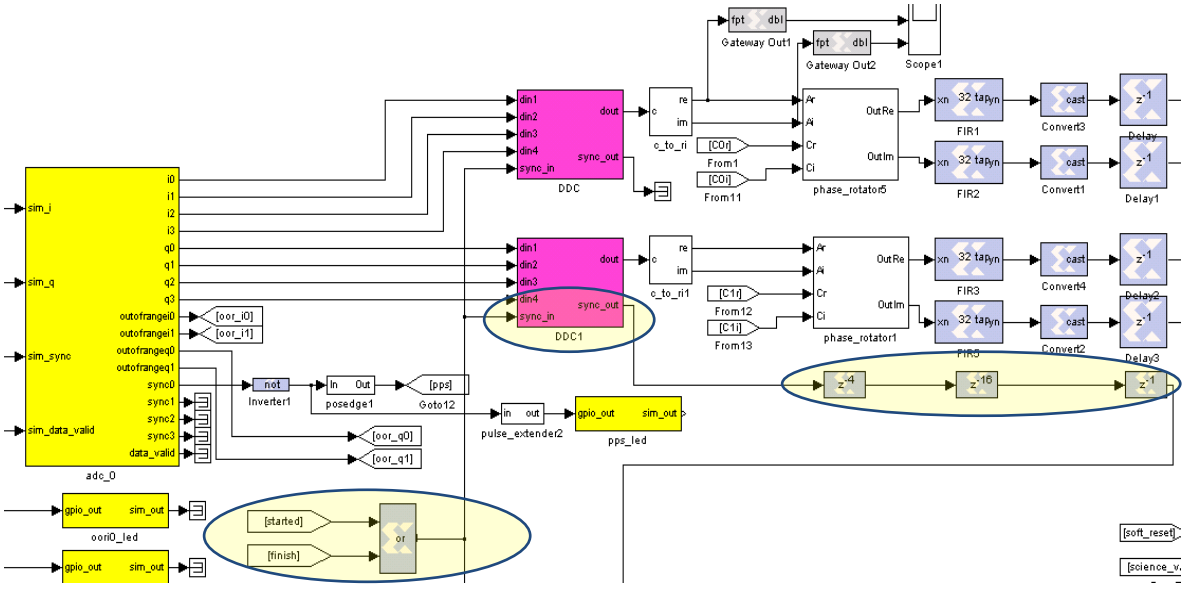


Fig. 21: Propagating the sync signal with intermediate block latencies

There is also a constraint for the down sampler, this decimator must be initialized to the decimation value in order to enable the first sample as shown on the next picture: time between the start and the first sync is the latency due to the DDC - Phase Rotator - FIR block set, the very first sample will pass and the next sync will be down sampled.

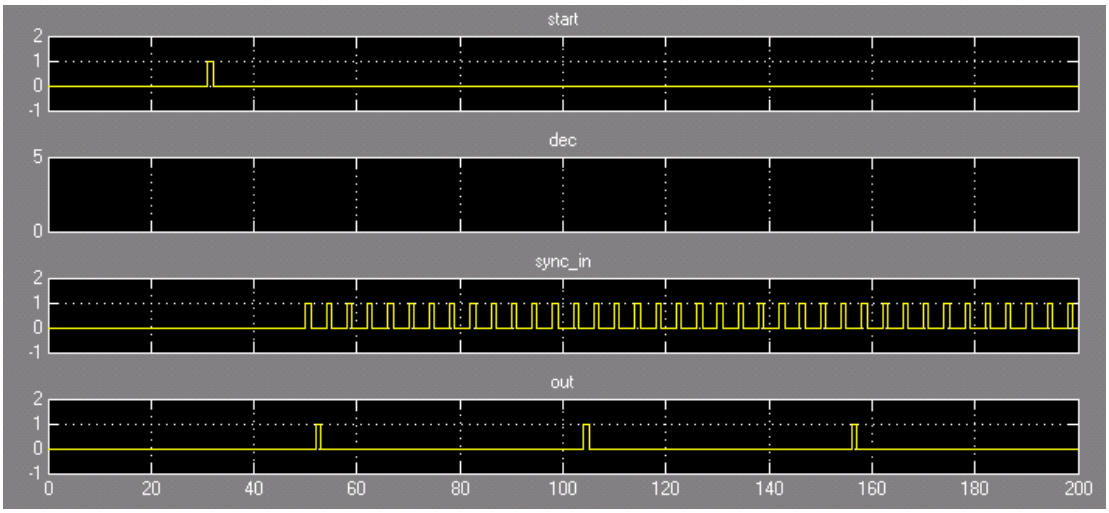


Fig. 22: Simulation of the IBOB decimator performed using Simulink.

BEE2

The BEE2 FPGA uses an internal oscillator as a 205MHz clock, the firmware can be splitted in 2 main parts, one dedicated to the data processing and the other one to route commands between the workstation and the IBOB.

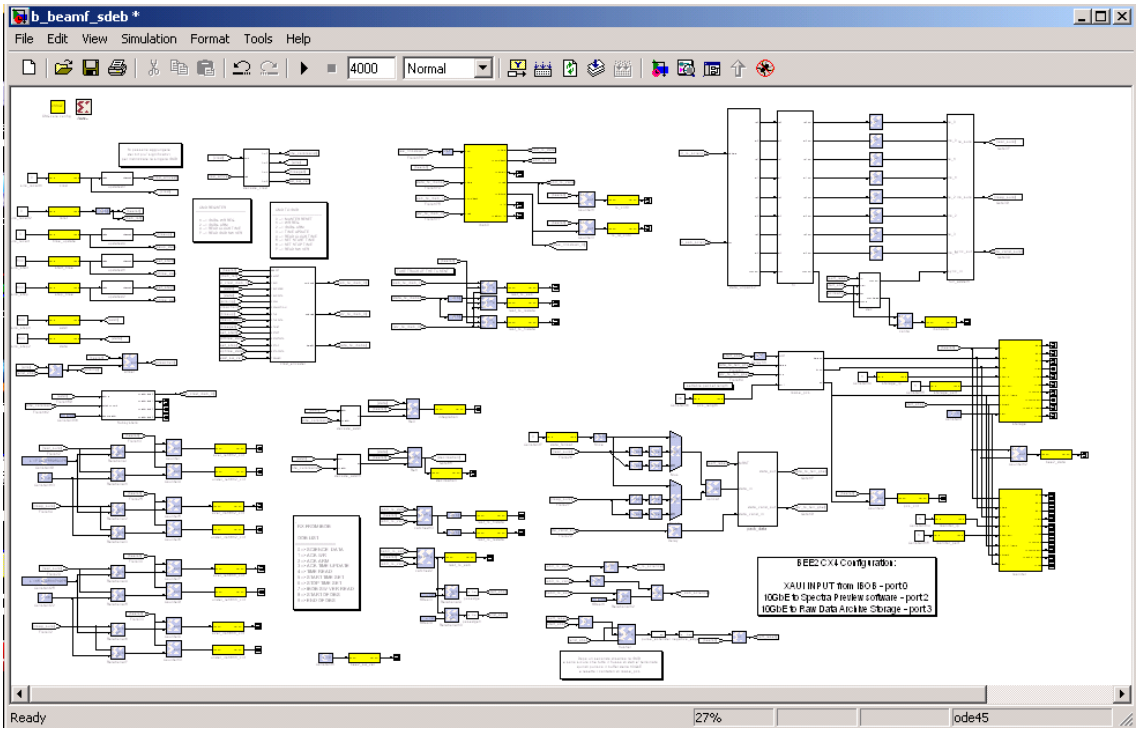


Fig. 23: BEE2 MATLAB Model file

The following scheme represents the architecture of the firmware.

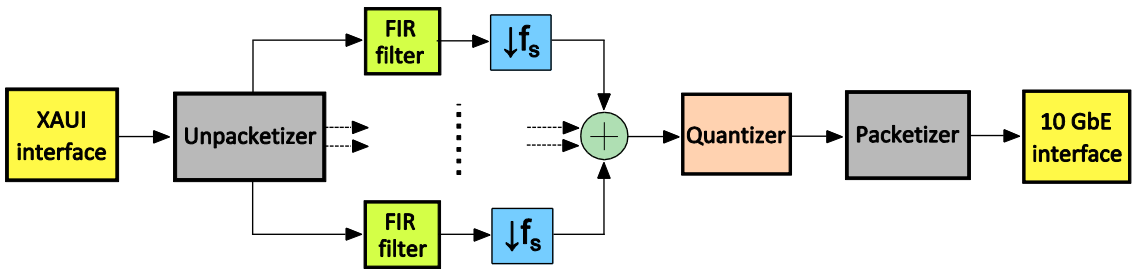


Fig. 24: BEE2 firmware architecture

Data received from XAUI interface are splitted in 4 parallel lines adopting the same criterion used in the transmission side. So at the output of the unpacketizer block we have complex signals with 16 bits precision to which we apply lowpass FIR filters. The target complex bandwidth of about 100 KHz is achieved thanks to a FIR filter with a Hamming window, 110 taps and a normalized cutoff frequency of 0.04 as shown in the next figure.

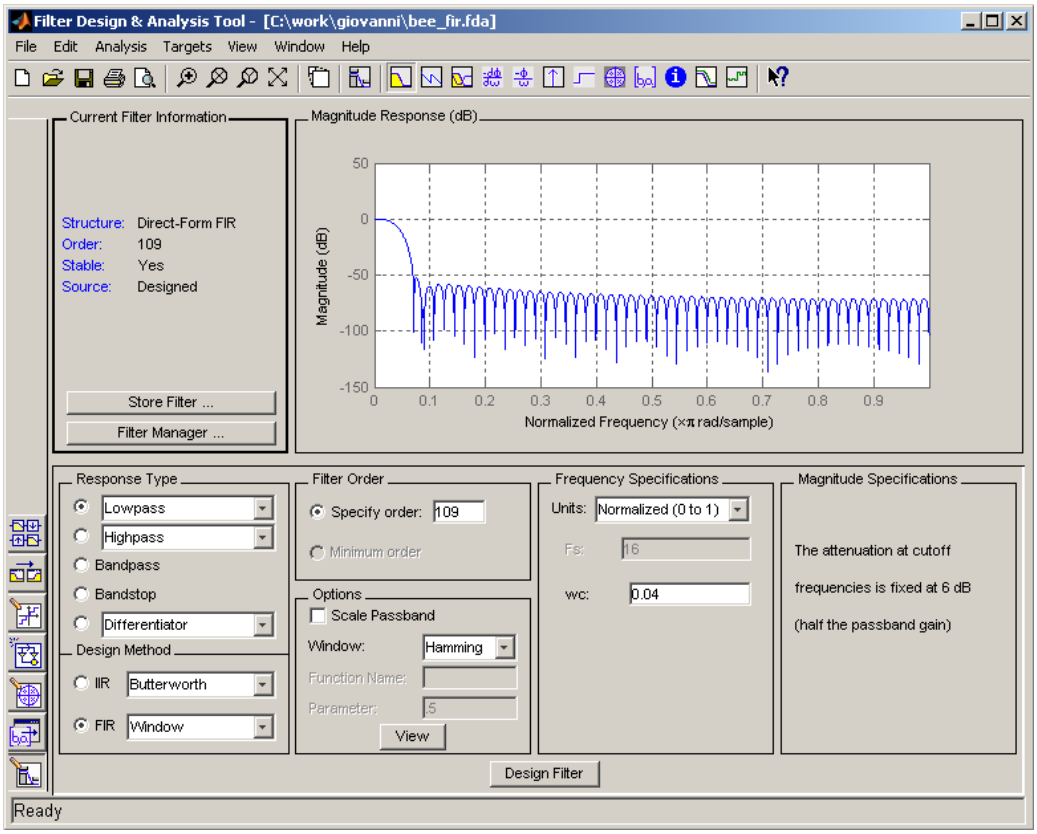


Fig. 25: Design parameters of the FIR filter synthesized in the bee2.

Since the sample rate is about 2.307692 MSample/sec, it follows that the filter cuts at 46.153846 KHz ($1.153846\ \text{MHz} * 0.04$). In this way we obtain a total complex bandwidth of 92.3076923 MHz and 23 results to be a suitable sample factor that can be used in order to respect the Nyquist-Shannon sampling theorem.

$$\frac{2.307692\ \text{MSample/sec}}{23} = 100.334448\ \text{KSample/sec} > 2 * 46.153846\ \text{KHz}.$$

Finally the 4 parallel data stream, that correspond to the signals coming from the 4 antennas of the array, are summed up together in phase forming the antenna array beam. Again, after FIR filtering and summing operations, the resulting complex signal is now represented by a greater number of bits, in particular 38 (19 for real part + 19 for imaginary part). So, before making packets and sending them through 10 Gbit Ethernet link, data are re-quantized in order to have the signal with 32 bit precision (16 for real part + 16 for imaginary part).

The packetizer block produces time domain complex data in UDP packets of 1280 bytes over the 10 Gbit link. The format of the packet is very simple as follow:

Field	**Offset (byte)**	**Length (byte)**
Counter	0	8
Data	8	1272

Tab. 2: UDP packet format

Counter: The Counter field is simply a unsigned 64 bit integer counter of the packet, it starts from zero and it is useful for the storage script to understand if there are lost packets. Unfortunately, using UDP protocol in case of data loss the packet lost will be not transmitted again. This field will be not stored in the output file.

Data: The Data field contains complex pairs of 32bit (Real part highest 16 bit and Imaginary part the lowest 16 bit) and their representation is a signed fixed point 16.11 (or 16.12, it depends on the configuration chosen during the initialization). The MSB of 16's represent the sign. When negative the number has to be handled with 2's complement.

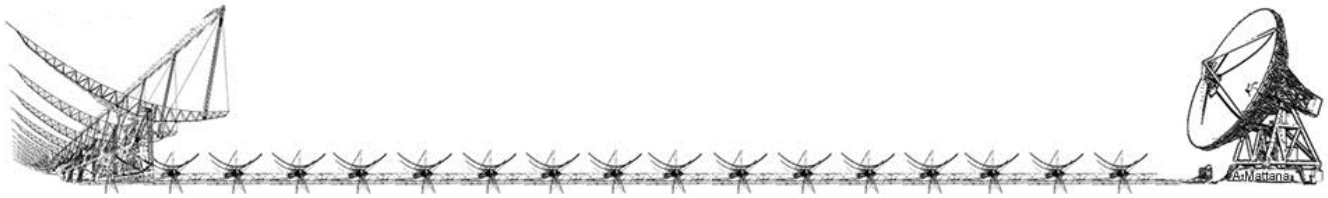
s	i	i	i	i	d	d	d	d	d	d	d	d	d	d	d
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

The weight of each bit is as follow (generic form):

$$\sum_{bit=0}^{n} 2^{bit} * bit\_value$$

where the bit value can assume only values 0 or 1. Using a numbering system centered to the binary point position (Fix 16.11) as follow:

s	i	i	i	i	d	d	d	d	d	d	d	d	d	d	d
±	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11



Therefore, for the specific case Fix 16.11 is:

$$\sum_{bit=-11}^{3} 2^{bit} * bit\_value$$

If the bit value number 15 is 1 the 2's complement is obtained by inverting the value of every bits and then adding 1. The same result can be easily obtained by subtracting the minimum number representable (in case of a signed fixed point 16.11 is -8) to the absolute value of the number (as it is a unsigned fixed point 16.11).

The possibility to convert this data in real time from fixed point to real value is under investigation. There is a package called fixreal developed by Marco Bartolini that converts fixed point data to floating point and vice versa but we should study before the feasibility because it takes lot of CPU/DISK time making this operation in real time while acquiring and storing data to the disk and we do not want to risk to lose data.

The order of the samples in the data field is simply as follow:

Data Field	**Offset (byte)**	**Length (byte)**
T_0 Im	0	2
T_0 Re	2	2
T_1 Im	4	2
T_1 Re	6	2
…	…	…
T_{317} Im	1268	2
T_{317} Re	1270	2

Tab. 3: Data field in UDP packets

Where the T index means time relative just to this packet. There is no time marker on packet header, it is important to refer to the output file name to associate each sampling time to a sample.

A hardware communication protocol has been developed on the BEE2 and IBOB firmwares to allow to exchange data, set-points, commands, acknowledges and messages. A data field called oob ("out of band") will contain different values depending on the meaning of the transmitted message. The OOB field is an extra 8 bit line (physically really extra 8 wires) available over XAUI chip to chip protocol. Those 8 extra bit do not use the 64 bit bandwidth of the data field.

OOB	BEE2 -> IBOB	IBOB -> BEE2
0	MRST - Send Master RESET	DATA - Science Data
1	WR - Write Register	WR ACK - Write Acknowledge
2	ARM - Arm IBOB	ARM ACK - Arm Acknowledge
3	TUP - Send Time Update	TUP ACK - TUp Acknowledge
4	TR - Ask for IBOB local time	TRA - Send Time Read
5	START - Set START Time	STACK - Start Time set
6	STOP - Set STOP Time	STOCK - Stop Time set
7	SW - Ask for SW version	SWVER - IBOB Software Version
8	Not Used	STARTED - Start Obs Signal
9	Not Used	FINISHED - End of Obs Signal

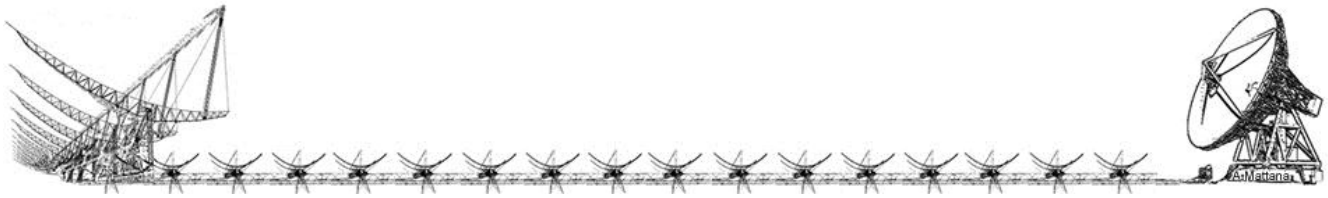
Tab. 4: OOB list for IBOB-BEE2 internal communication protocol.

When a command is sent to the IBOB via the BEE2 you can find the acknowledge few clock cycles later on the BEE2 registers, this is very useful to understand if the command sent to the IBOB has been receveid and applied.

Eg 1: Arming the System.

1. Workstation sends the command to ARM the IBOB at t_0
2. BEE2 sends to IBOB: OOB=2, DATA=1 at t_1
3. IBOB receives OOB=2, DATA=1, updates the ARM register value to 1 and sends an acknowledge to the BEE2 at t_2 sending OOB=2, DATA=1
4. BEE2 receives the ack and updates its own registers “last_rx_oob” and “last_rx_data”.
5. Workstation can wait polling those registers to know if the command has been received correctly from the IBOB, and at t_3 can start to send the next commands.

The workstation does not have to wait for a specific time between commands but can wait polling the BEE2 registers to know when the IBOB is ready to accept the next command. This protocol is not only used for communication between the wokstation and the system but also to exchange information between IBOB and BEE2 implementing a unique state machine as you can see on the next two examples.



Eg 2: Starting an observation

Starting condition: Start time and Stop Time is set, System is Armed (Arm led on IBOB switched on).

1. At t_0 the local timestamp is equal to the start time, a STARTED signal is sent to the BEE2 from the IBOB using OOB=8, DATA=0 (in case of a signal the data field does not make sense)
2. BEE2 send a "soft reset" to the system which prepares the packetizer (reset a counter) and the 10 Gbit buffer (empting the buffer) to start.

You will see the running led starting to blink.

Eg 3: Ending an observation

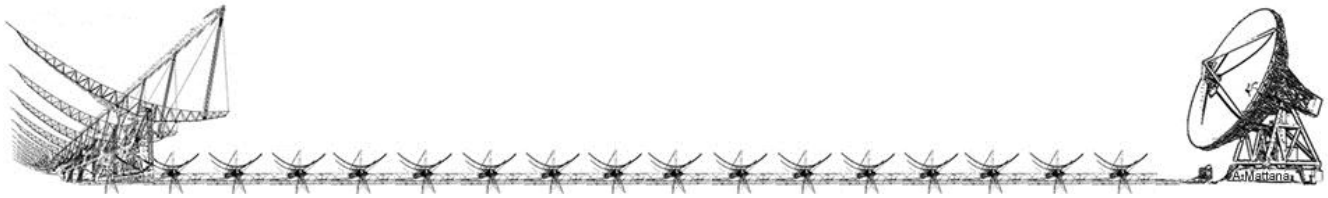
Starting condition: Start time and Stop Time is set, System is Armed (Arm led on IBOB switched on).

1. At t_0 the local timestamp is equal to the stop time, a FINISHED signal is sent to the BEE2 from the IBOB using OOB=9, DATA=0 (in case of a signal the data field does not make sense)
2. BEE2 answers to the IBOB disarming the system sending a OOB=2, DATA=0.

In this case you will see the running led stopping blinking and the arm led switching off.

If at the end of an observation you will see the running led switched of and the arm led switched on there is a failure condition due to possible causes such:

- BEE2 has unexpectedly stopped to work, the storage should reports an error on the amount of data dumped that differ than expected.
- The XAUI physical link connecting the IBOB and BEE2 has been disconnected.



Networking

Using the UDP protocol each BEE2 FPGA Ethernet device must have a proper network configuration in order to avoid conflicts between devices of the same network.

We have identified a possible scheme on assigning IPs and MACs that unless than a common part each address differs by only two ID digits as follow:

MAC	00:12:6D:AE:0B:XY
IP	192.168.11.XY
GATEWAY	192.168.11.10
PORT	6X00Y

Tab. 5: 10Gb Eth interface configurations

Where

X = FPGA ID, can assume value from 1 to 4
Y = SERVICE ID, can assume value from 1 to 5

The PORT scheme is valid also for the Ethernet communication over the interface 10/100Mbit between the Control PC and the FPGA hardware registers. Here shown in the next table the Service ID list:

INITIALIZATION	6X001
CALIBRATION(*)	6X002
MONITOR	6X003
STORAGE	6X004
RECORDER(**)	6X005

Tab. 6: Service ID List

(*) For the Calibration service see the "Space Debris Calibration" internal report.
(**) Used only between the Control and Storage machines.

While the INITIALIZATION and CALIBRATION services use the private 100Mb network (192.168.10.X) the MONITOR and STORAGE packets travel on the 10GbE Network and the RECORDER commands over the public network.

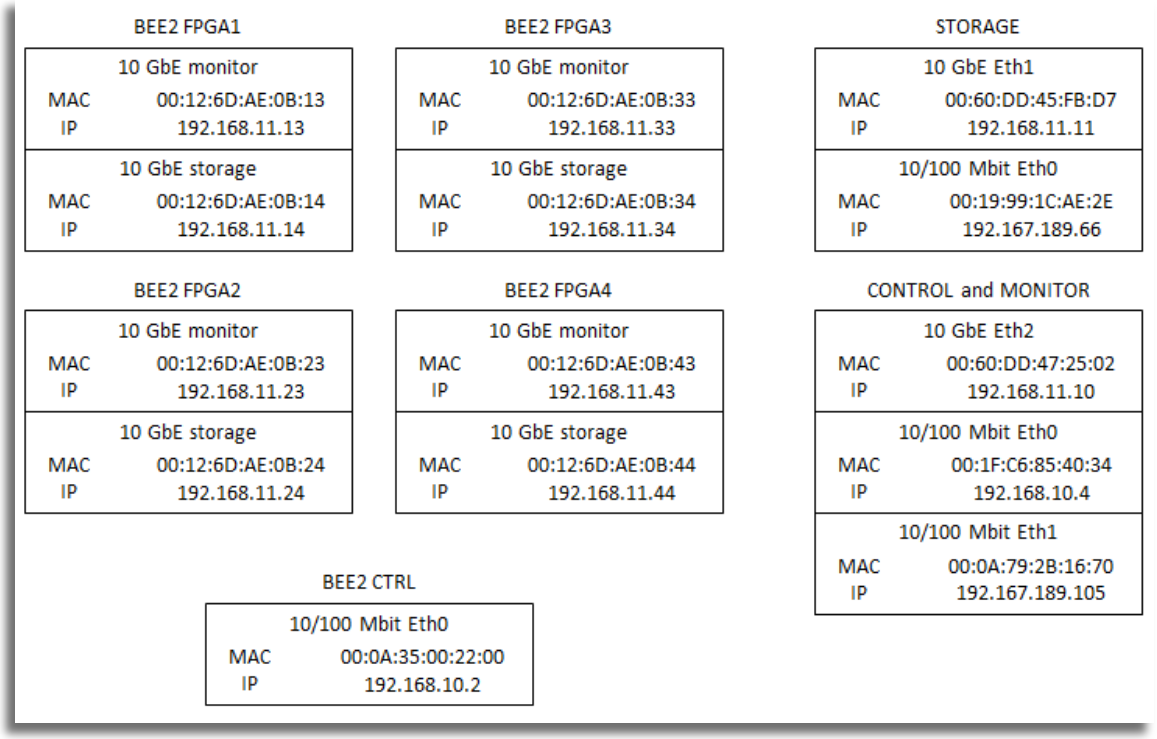
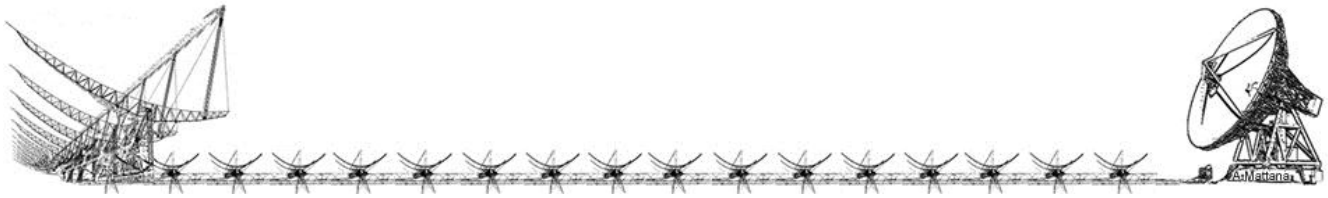


Fig. 26: IPs an MACs table

The above picture lists the interfaces configuration for the 3 network:

1) 192.168.11.X the 10GbE Network
2) 192.168.10.X 10/100Mbit Private Network
3) 192.167.189.X 10/100Mbit Public Network



Control Software

Communicate with hardware

The workstation can communicate directly only with the BEE2 board which will be also a bridge interface between the IBOB and us. It is possible to interact with hardware registers of the project loaded on FPGA thanks to the BORPH.

BORPH is an extended Linux kernel that treats FPGA resources as native computational resources on reconfigurable computers such as BEE2. As such, it is more than just a way to configure an FPGA. It also provides integral operating system supports for FPGA designs, such as the ability for an FPGA design to read/write to the standard Linux file system. A user process in BORPH, can therefore either be a software program running on a processor, or a hardware design running on a FPGA. A hardware design that is running on a FPGA is called a hardware process.

BORPH uses regions of FPGA fabric as computation units to spawn hardware processes. Each reconfigurable region is defined as a hardware region (hwr). Logically, it is the smallest unit of a RC that is managed by BORPH. Physically, it can be implemented as an entire FPGA in a multi-FPGA system, or a partially reconfigurable region within a FPGA. On a BEE2 module, there is only one hwr type defined, the b2fpga, which corresponds to one user FPGA.

The hardware process is seen as a software process running on Linux and it is located on the directory “/proc” identified by the PID (Process Identifier). Subdirectories on that folder contain lot of resources information, the most important are in “hw/ioreg” that contains files for the input/output registers defined on the Matlab model file. Reading a file content means read part of hardware region of that fpga. Depending of the configuration assigned to a registers (“from/to processor”) at development level you are allowed to read or even to write values.

Preliminary operations

IBOB bit file can be loaded via JTAG protocol (which require a hardware serial link) into the FPGA. This is a volatile operation because after a power off the configuration will be lost. IBOB board mounts an EEPROM that can be written statically and can contains firmwares, and setting specific pin jumpers, the firmware will be automatically loaded on FPGA at the power on.

Xilinx Impact is a software which provides JTAG support, you can just debug a jtag chain or even read/write bit files from/to a target chip of the chain. Switching the software in advanced mode you are allowed to generate a eeprom file ("mcs") starting from a bit file to a specific type of chip. IBOB EEPROM has been written with the mcs file of the beamformer while, thanks to the BORPH, the BEE2 does not require this operation.

BEE2 file system can be accessed using secure shell protocols. The architecture file synthetized (BOF file) for a BEE2 User FPGA must be copied on the BEE2 file system (in any directory) using a sftp client. BOF file must have executable permission ('x') to all.

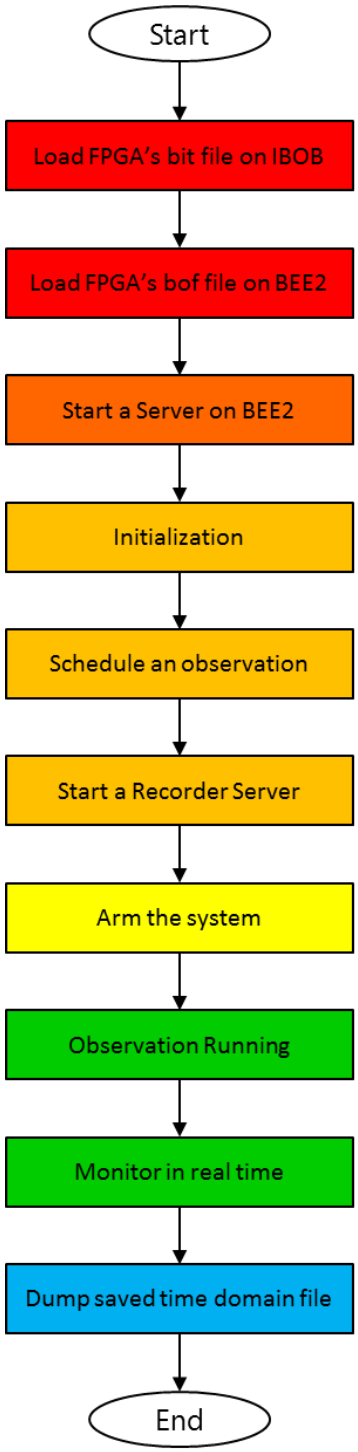
The flow chart on the following page summarize step by step the few operations needed to perform an observation of a beam.

Booting up the system

Power on the IBOB board that will automatically load the firmware from the EEPROM, you will recognize a led (bottom left) blinking with the PPS.

Open a ssh client and connect to the BEE2 (beecool host aliases) from the workstation (called bee2, please do not confuse bee2 host with the BEE2 board):

```
oper@bee2:~$ ssh obs@beecool
Password:  ####### (confidential)
Linux (BORPH) beecool 2.4.30-pre1 #1 Thu Nov 9 12:06:49 PST 2006 ppc

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Apr 30 19:43:58 1970 from 192.168.10.4
obs@beecool:~$
```



The up to date version of the latest bit file has been written into the EEPROM, therefore, just switch on the IBOB

Thanks to the BORPH, loading a bit file on BEE2 FPGAs means run a BOF (Borph Object File) file on a Linux operating system

The BOF will be seen as a Linux process and each hardware registers on FPGA as files, therefore running a listening server on the BORPH allows to interact with the hardware registers

It is possible now to write parameters on these registers by instantiating a client on the workstation that sends commands over the 10/100 Mbit Ethernet

Depending on the target just write a file giving the start_time and the stop_time of the observation. The antenna pointing at that time has to point the specific declination.

A server on the storage machine has to be started before to load the schedule.

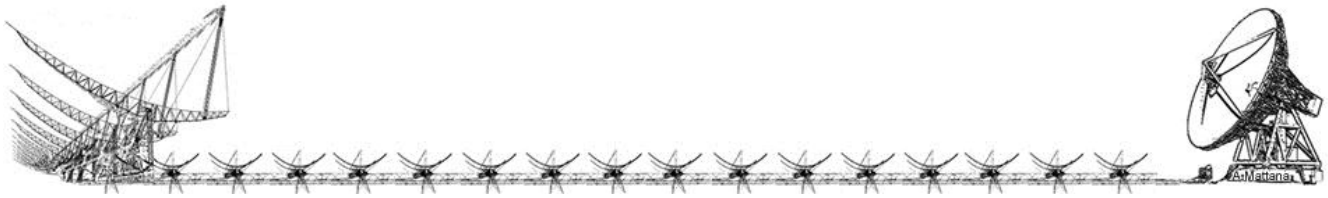
Load the schedule on the IBOB means load the stat and stop time when data will be generated and ARM the entire system (IBOB, BEE2 and the storage).

A blinking led on the IBOB indicates the observation is running.

At that time you can have a real time plot on the workstation.

At the end you can download the file saved on the storage machine recognizable by the file name reporting time and target information.

Fig. 27: State Machine Flow Chart



BEE2 BOF files for this project has been located on “sdeb/bof” directory, go there and run the BOF file as executable, if success you will recognize a new command promt.

```
obs@beecool:~$ cd sdeb
obs@beecool:~/sdeb$ cd bof
obs@beecool:~/sdeb/bof$ ./b_beamf_sdeb_fpga4_2012_Nov_09_1706.bof

****************************
* TinySH lightweight shell *
****************************
Design name : b_beamf_sdeb
Compiled on : 09-Nov-2012 17:06:29

DON'T PANIC ;-)

Type 'help' for help
Type '?' for a list of available commands


BEE2 %
```

Possible Failures: If a bof file is already running you get this error:

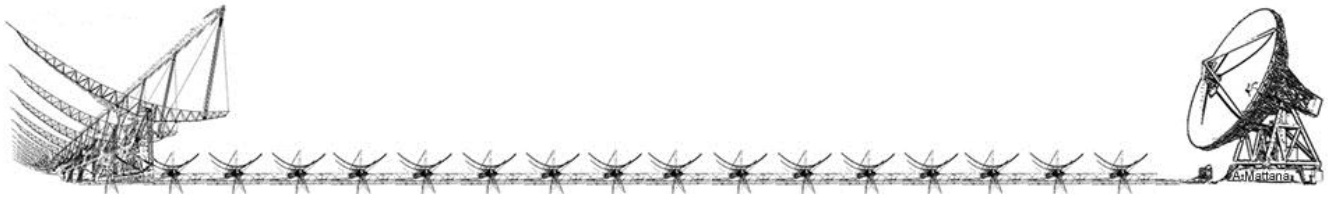
```
obs@beecool:~/sdeb/bof$ ./b_beamf_sdeb_fpga4_2012_Nov_09_1706.bof
-bash: ./b_beamf_sdeb_fpga4_2012_Nov_09_1706.bof: Device or resource busy
obs@beecool:~/sdeb/bof$
```

If the already running bof file is not related to this project just kill him and run the bof again.

```
obs@beecool:~/sdeb/bof$ ps -ef | grep ./b_beamf_sdeb_fpga4_2012_Nov_09_1706.bof
obs      26558 26553  0 19:52 pts/8    00:00:00 ./b_beamf_sdeb……9_1706.bof
obs      26574 26566  0 19:58 pts/9    00:00:00 grep ./b_beamf_sde……9_1706.bof
obs@beecool:~/sdeb/bof$ kill 26558
```

Using that new command prompt you can interact with the hardware registers. These operations have been simplified writing a python script that instantiate a listening server that receive read/write commands over TCP and it has direct access to the registers files on “/proc/PID/hw/ioreg” directory. Since the beecool shell is busy with the bof you have to open a new one making a new ssh connection as before.

```
obs@beecool:~/sdeb$ ./start_server.py

##########################################
Beamformer Server Launch Wizard
Please follow the instruction...

[1]  26576 ./b_beamf_sdeb_fpga4_2012_Nov_09_1706.bof

Select the PID index:  1
```

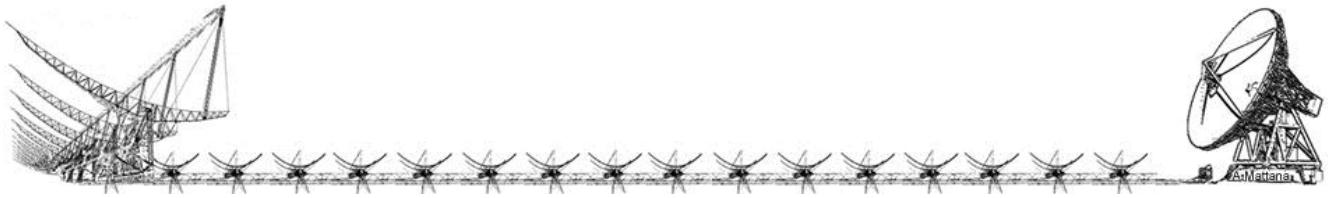


Answer this question: if you see only one bof file type 1, if you see many bof file running detect the bof you just run and type its index (the number on the left within the brackets). Usually, should be the last of the list, but, if another user is running another bof at the same time is not guaranteed. Several bof file name differ only of one digit, the number of the fpga used, therefore pay attention on the targeted fpga.

```
[1] FPGA1 (usually IBOB2-BEE1)
[2] FPGA2 (usually IBOB3-BEE2)
[3] FPGA3 (usually IBOB4-BEE3)
[4] FPGA4 (usually IBOB7-BEE4)

Select the branch:  2
```

Answer this question: if the antenna signals are connected on the IBOB n. 7 and there is a 10Gb link between that IBOB and BEE2 FPGA4 (front-right) you have to answer 2. Please verify always the hardware connections before to start.

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

Starting server for
  PID: 26576
  BOF: ./b_beamf_sdeb_fpga4_2012_Nov_09_1706.bof

server listening on port: 64001
```

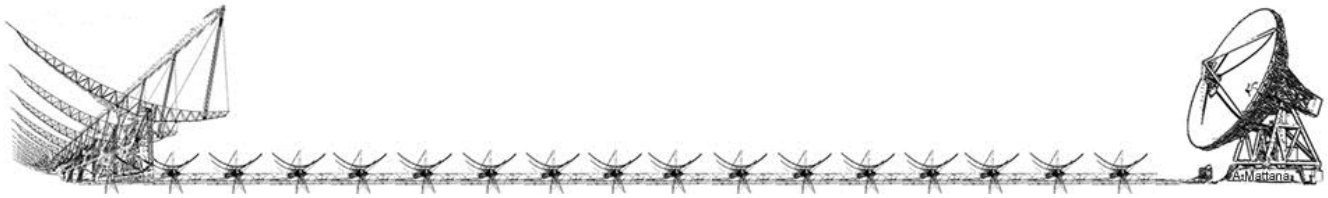


Initialization

The initialization phase customizes the system for a specific observation. There are common parameters always valid such Ethernet configurations and values to reach the desired bandwidth starting from a higher sampling frequency, and, parameters changing day by day such antenna phase corrections in case we want to have a unique beam of multiple antenna.

An initialization python script instantiates a TCP client which will send these configuration parameters that have been previously written in a text file to the listening server running on the BEE2. This text file can be manually modified. A python script has been developed to automatically generate the configuration file to avoid (or at least reduce) typos, it is an interactive scripts that require to answer to some questions.

```
Last login: Thu Nov 22 11:41:42 2012 from 192.167.189.65
oper@bee2:~$ cd /media/data/sdeb/
oper@bee2:/media/data/sdeb$ python config_wizard_test.py

##########################################

Beamformer Configuration file Wizard

Please follow the instruction...

+-----------+-----------+
|    2      |     3     |
+-----------+-----------+    BEE2 FPGAs
|    1      |     4     |
+-----------+-----------+

[1] BEE FPGA-1 (front - left)
[2] BEE FPGA-2 (rear  - left)
[3] BEE FPGA-3 (rear  - right)
[4] BEE FPGA-4 (front - right)

Which BEE2 FPGA are you going to use [1/2/3/4]?  4

[1] Fix 16.11
[2] Fix 16.12
Which data cast are you going to use [1/2]?  1


[1] LOFAR antennas
[2] NS receivers
[3] EW channels

Which antennas are connected to the system [1/2/3]? 3
```

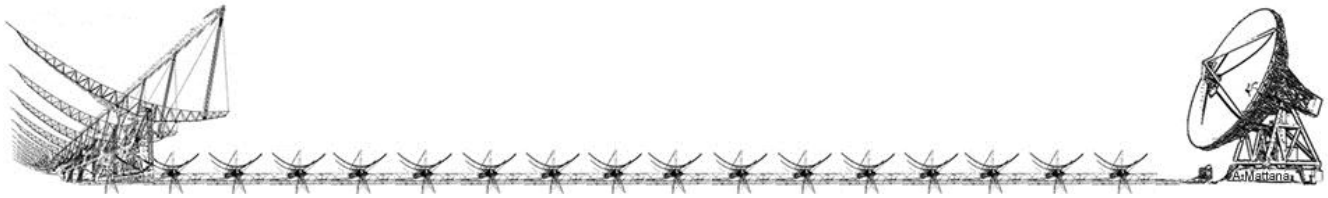


```
Lastest phase calibrations have been done
on 14/11/2012 at 00:00:00 using radiosource 3C123

[1]     4E = +154.70863
[2]     5E = -1.2245701
[3]     2E = +00.000000
[4]     3E = +55.895791

"none" means channel off

Edit a channel by typing the index in the brackets
    or type zero [0] to confirm the configuration:
```

The answers provided in this example will generate a file for a beamformer running on the FPGA-4 of the BEE2 that talk with an IBOB having connected to its A/D input lines the signals coming from the East-West arm of the Northern Cross Radio Telescope, channels 2E, 3E, 4E and 5E.

The script loads automatically the latest calibration of the East – West channels. If the observation we are going to do does not need to use 4 E/W channels it is possible to disable a channels by editing the phase writing a “none”. When the phase corrections are set the script ask to show the file generated and ask if save it.

```
The configuration file is ready to be saved.


Do you want to check before to save [y/n]?  y
##################################################
#
# Configuration file automatically generated by using
# the Wizard on 22/11/2012 10:45:57 UT
# NO WHITESPACE ALLOWED BETWEEN TEXT DELIMITERS!

# design global parameters
[global]
dec_factor = 13
bee_dec_factor = 23

# integration for on board data accumulation
integration = 100334

# Common network parameters
monitor_ip = 3232238346
storage_ip = 3232238347
pck_length = 160
gbe1_name = monitor
gbe2_name = storage
```



```
# Parameters for the FPGA4 system
monitor_port = 64003
storage_port = 64004
gbe1_filename = gbe_fpga4_monitor.conf
gbe2_filename = gbe_fpga4_storage.conf
# Data cast 16.11
data_format = 1
[main_fpga]
server = beecool:64001
#Phase shift in degrees for each single antenna
#none: mute antenna
d0 = +154.70863   # 4E
d1 = -1.2245701   # 5E
d2 = +00.000000   # 2E
d3 = +55.895791   # 3E
d4 = none   #
d5 = none   #
d6 = none   #
d7 = none   #
d8 = none   #


d9 = none   #
d10 = none   #
d11 = none   #
d12 = none   #
d13 = none   #
d14 = none   #
d15 = none   #
############################################################


Do you want to save as fpga4.conf [y/n]?  y
oper@bee2:/media/data/sdeb$
```

The configuration file is ready to handle up to 16 input signals coming from more than one IBOB. Finally it is possible to send this information to the BEE2 by running the python script “sdeb_int.py” (there is a –L in lower case in the middle):

```
oper@bee2:/media/data/sdeb$ ./sdeb_init.py -l fpga4.conf
2012-11-22 15:08:36,556 - ****************************************
2012-11-22 15:08:36,556 - ******* INITIALIZATION PROCESS *********
2012-11-22 15:08:36,556 - ****************************************
2012-11-22 15:08:36,557 - parsing configuration file fpga4.conf
2012-11-22 15:08:36,557 - Initializing the Space Debris system
2012-11-22 15:08:36,557 - Sending Master Reset to Bee2 and IBOBs
2012-11-22 15:08:36,884 - Setting equalization on main fpga
2012-11-22 15:08:36,885 - Setting (RE_0, IM_0) to: (140, 55)
```

```
2012-11-22 15:08:36,977 - Setting (RE_1, IM_1) to: (127, 253)
2012-11-22 15:08:37,071 - Setting (RE_2, IM_2) to: (127, 0)
2012-11-22 15:08:37,165 - Setting (RE_3, IM_3) to: (72, 106)
2012-11-22 15:08:37,259 - Setting (RE_4, IM_4) to: (0, 0)
2012-11-22 15:08:37,353 - Setting (RE_5, IM_5) to: (0, 0)
2012-11-22 15:08:37,447 - Setting (RE_6, IM_6) to: (0, 0)
2012-11-22 15:08:37,541 - Setting (RE_7, IM_7) to: (0, 0)
2012-11-22 15:08:37,634 - Setting (RE_8, IM_8) to: (0, 0)
2012-11-22 15:08:37,728 - Setting (RE_9, IM_9) to: (0, 0)
2012-11-22 15:08:37,822 - Setting (RE_10, IM_10) to: (0, 0)
2012-11-22 15:08:37,917 - Setting (RE_11, IM_11) to: (0, 0)
2012-11-22 15:08:38,011 - Setting (RE_12, IM_12) to: (0, 0)
2012-11-22 15:08:38,104 - Setting (RE_13, IM_13) to: (0, 0)
2012-11-22 15:08:38,198 - Setting (RE_14, IM_14) to: (0, 0)
2012-11-22 15:08:38,292 - Setting (RE_15, IM_15) to: (0, 0)
2012-11-22 15:08:38,387 - Setting decimation factor to 13
2012-11-22 15:08:40,361 - Setting Data Format to 16.11
2012-11-22 15:08:40,473 - Setting Packet Length to 160
2012-11-22 15:08:40,583 - Setting Monitor IP to 192.168.11.10
2012-11-22 15:08:40,694 - Setting Monitor Port to 64003
2012-11-22 15:08:40,804 - Setting Storage IP to 192.168.11.11
2012-11-22 15:08:40,914 - Setting Storage Port to 64004



2012-11-22 15:08:41,024 - Starting 10GbE interface: monitor
2012-11-22 15:08:41,302 - Starting 10GbE interface: storage
2012-11-22 15:08:41,386 - Performing time update for sync...
2012-11-22 15:08:42,092 - Sending: 2012/11/22 14:08:42 UT
2012-11-22 15:08:43,190 - Received: 2012/11/22 14:08:43 UT after 1 sec.
2012-11-22 15:08:43,190 - Time updated successfully

Initialization Process Successfully Completed!

oper@bee2:/media/data/sdeb$
```

It is extremely important that the last line reports a success message about the time update, a failure may indicate that the running IBOB bit file might be wrong or that the bee2 bof file has an issue. In any case if the time update fails restart from the beginning.

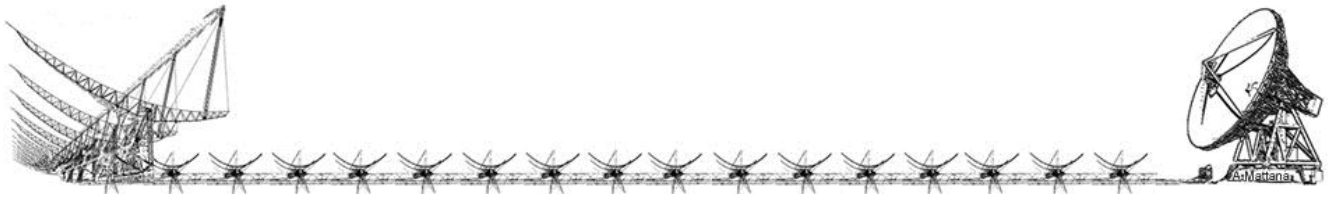
Scheduling

Scheduling an observation means to write a “start time” and a “stop time” in the IBOB registers. This is done by another python script that instantiates a client socket and send to the BEE2 (over the same port as for the init) the content of a text file, located just for convention in the “sched” directory, as the following:

```
oper@bee2:/media/data/sdeb$ more sched/dirac_test_ew.conf
[Obs]
System = fpga4
Start_time = 2012/11/13_09:27:00
Stop_time = 2012/11/13_09:28:00
Target = targetname
```

Remember that all times are in UT. The system will produce data only between the Start_time and Stop_time parameter. Target parameter it is a string that do not have whitespace because will be a part of the output filename, therefore if you really need please use only characters allowed, in any case a underscore character will precede and follow this string. All parameters are case sensitive, do not edit the parameter name but only the value. The output file name will always be saved starting whit the Start_time date and time as the following example:

```
20121113_092700_targetname.dat
```



Storage

Before to arm the system and load the schedule it is mandatory to start the recorder server on the storage machine that is the 192.167.189.66 (host alias batman) by opening a ssh connection (or if possible open a terminal shell in local) and run a script located on "/media/data/sdeb/" directory.

```
oper@bee2:~$ ssh -l oper 192.167.189.66
Password:
Last login: Tue Nov 13 10:24:12 2012 from bee2desktop
Have a lot of fun...
oper@batman:~> cd /media/data/sdeb/
```

Depending on the system you are using launch the recorder server (they differ only by the number of fpga):

```
oper@batman:/media/data/sdeb> ./fpga4_recorder_server.py
server listening on port: 64005
```

If you got a failure message it means that a server is already running. To be sure to do not lost the observation please kill the other instance of the server and start it again as in the following example.

```
oper@batman:/media/data/sdeb> ./ fpga4_recorder_server.py
Traceback (most recent call last):
  File "./ fpga4_recorder_server.py", line 74, in <module>
    server = SdebTCPServer(("", 64005))
  File "./ fpga4_recorder_server.py", line 29, in __init__
    SocketServer.TCPServer.__init__(self, addr, SdebTCPHandler)
  File "/usr/lib64/python2.6/SocketServer.py", line 400, in __init__
    self.server_bind()
  File "/usr/lib64/python2.6/SocketServer.py", line 411, in server_bind
    self.socket.bind(self.server_address)
  File "<string>", line 1, in bind
socket.error: [Errno 98] Address already in use

oper@batman:/media/data/sdeb> ps -ef | grep fpga4_recorder_server.py
oper  21620 21380  0 12:07    00:00:00 python ./ fpga4_recorder_server.py
oper  21623 21567  0 12:07    00:00:00 grep fpga4_recorder_server.py

oper@batman:/media/data/sdeb> kill 21620

oper@batman:/media/data/sdeb> ./ fpga4_recorder_server.py
server listening on port: 64005
```

Run a schedule

Load the schedule and arm the system using the python script “sdeb_run” giving as parameter the schedule file (there is a –L in lower case in the middle):

```
oper@bee2:/media/data/sdeb$ ./sdeb_run.py -l  sched/dirac_test_ew.conf
2012-11-13 11:45:21,073 - ********************************************
2012-11-13 11:45:21,073 - ************** ARM A NEW OBSERVETION *********
2012-11-13 11:45:21,073 - ********************************************

2012-11-13 11:45:21,073 - Parsing conf. file sched/dirac_test_ew.conf
2012-11-13 11:45:21,074 - Loading observation parameters...
2012-11-13 11:45:21,088 - Loaded START time 2012/11/13 09:27:00 UT
2012-11-13 11:45:21,099 - Loaded STOP time 2012/11/13 09:28:00 UT
2012-11-13 11:45:21,691 - System EW armed!
2012-11-13 11:45:21,711 - Data expected: about 22 MB (24080232 bytes)
2012-11-13 11:45:21,712 - Starting Recording data on 192.167.189.66:64005
2012-11-13 11:45:21,717 - Observation "targetname" Loaded Successfully!
```

As you can see the recording server has received a message with the observation info and it is ready to save a file. As a confirmation on the storage terminal you should see messages on that.

```
oper@batman:/media/data/sdeb> ./fpga4_recorder_server.py
server listening on port: 64005
Command received: record 1353598020 targetname 1353598080 24080232

################################################

Executing:  ./fpga4recorder.py  -o  targetname  -s  2012/11/13_09:27:00  -t
2012/11/13_09:28:00 -e 24080232

################################################

318_09:23:00 - INFO: Running with options:
318_09:23:00 - INFO: port: 64004
318_09:23:00 - INFO: pkg length: 8000
318_09:23:00 - INFO: fmt: >Q
318_09:23:00 - INFO: target name: targetname
318_09:23:00 - INFO: start time: 2012/11/13_09:27:00
318_09:23:00 - INFO: stop time: 2012/11/13_09:28:00
318_09:23:00 - INFO: output: data/20121113_092700_fpga4_targetname.dat
318_09:26:59 - INFO: server listening
318_09:27:00 - INFO: recording...
318_09:27:59 - INFO: closing communication
318_09:28:00 - INFO: received up to package: 18931
318_09:28:00 - INFO: closing recorder
################################################
```

Real time monitor

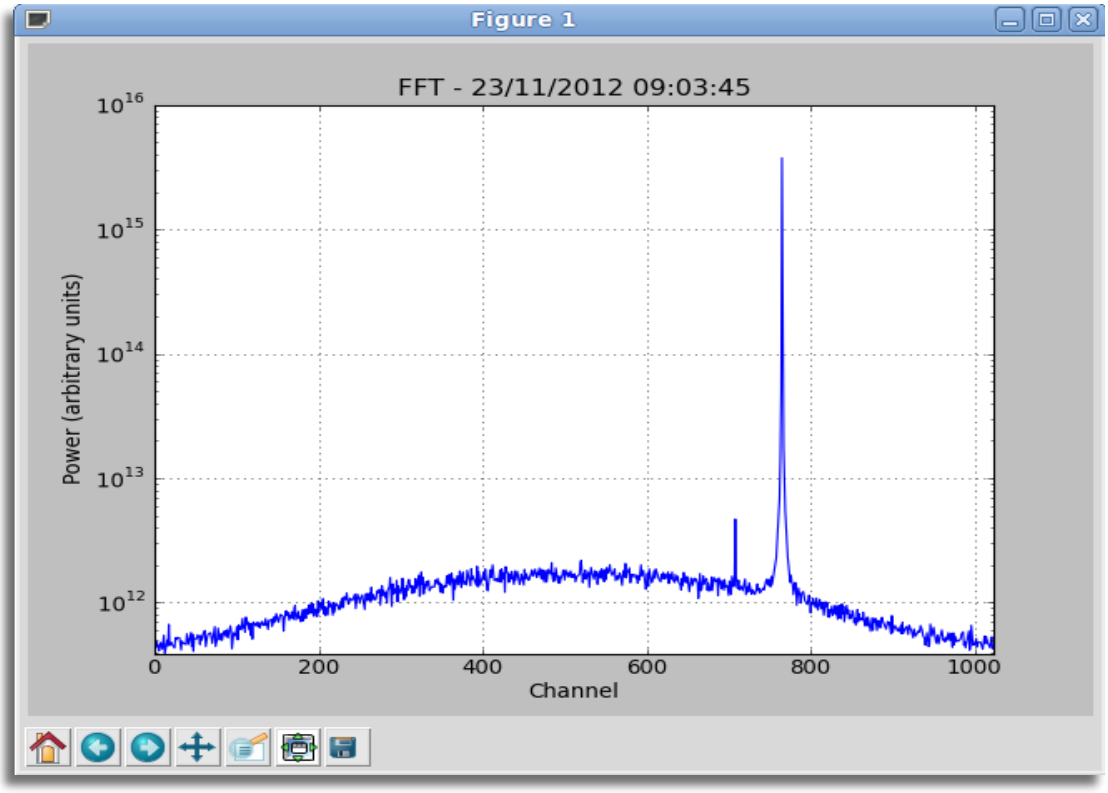


Fig. 28: A real time FFT plot observing a debris in a bistatic radar configuration

During the observation there is the possibility to see a real time fft of the data generated in the monitor pc running the python script “realtimefft.py”.

```
oper@bee2:~/andrea/bin$ python ./realtimespectra.py --help
Usage: realtimespectra.py [options]

Options:
  -h, --help            show this help message and exit
  -p PORT, --port=PORT
  -k PKG_LEN, --pkg_len=PKG_LEN
                        package length expressed in 64b
  -c FFTSIZE, --fftsize=FFTSIZE
                        number of fft channels
  -i INTEGR, --intgr_time=INTEGR
                        number of integrations
  -w WINDOW, --window=WINDOW
                        type of window, default=no window, possible value:
                        hamming, hanning, bartlett, kaiser(default shape 10%)
```

This script takes as parameters the number of the FFT channels, the integration time and if needed also a window to be applied to the FFT.

Here is a screenshot of the observation of a debris (ID 18096) transiting the 23/11/2012 and detected by the system (left peak).

```
oper@bee2:~/andrea/bin$ python ./realtimespectra.py -c 1024 -i 100
```

The FFT of this picture is a 1024 channel FFT of 100 spectra integrated. In this configuration you will see a refresh on the screen every about 1 second (100*1024 = 100K), but due to the low channel resolution (100 Hz) there are many cases in which the information can be hidden with the noise, a post process of the same data using higher resolution can show not only clear information in terms of frequencies but also the shape of the passage (doppler shift). The following picture is a plot of the same data with channel resolution of about 7 Hz without integrating any spectra.

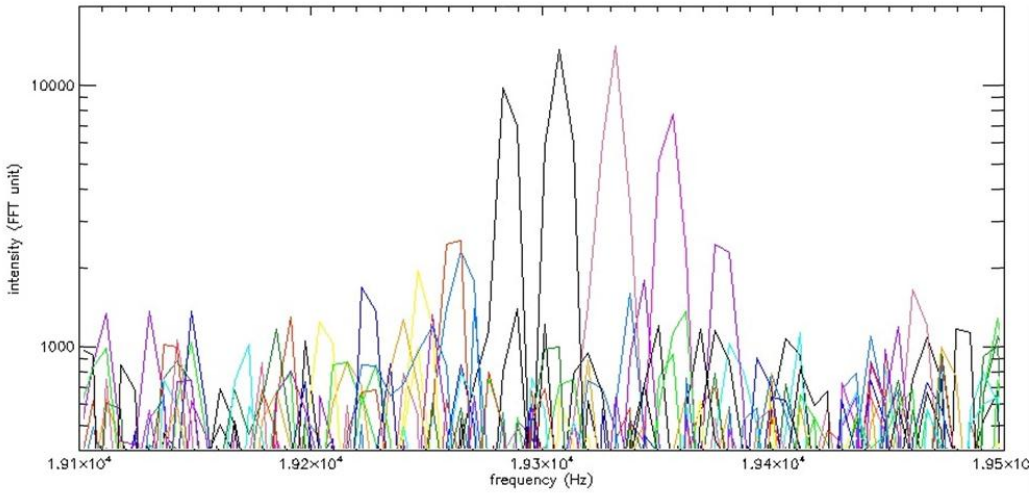
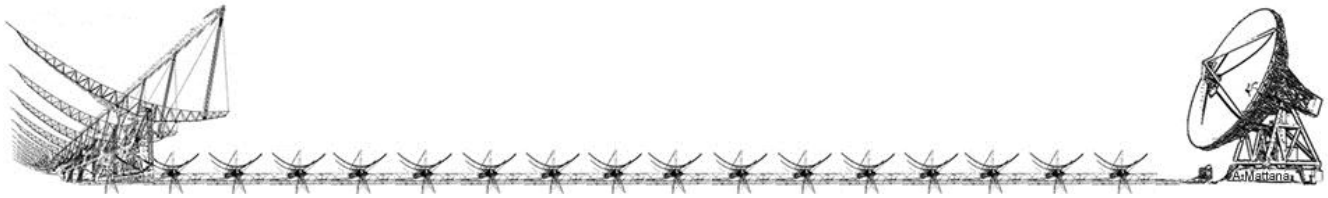


Fig. 29: The doppler shift of the debris ID 18096. Animating this plot the peak moves from right to left

There are several consideration of how convenient is to use higher spectral resolution or integrate more spectra but they are strictly case dependent (power of the transmitter used, speed of the debris and so on…), but making real time FFT while observing just as a preview does not really make sense to increase the channel resolution stressing the CPU especially if there are more than one system going on at the same time.



Validation

Debug tools

In order to validate the system it has been necessary to develop software and make some specific basic tests (like transmitting a signal and analyzing it in frequency) that helped to solve few hidden bugs. The aim of this chapter is also focused on keeping memory of the encountered problems and how they have been solved: this can be very useful and can save time for future developments.

Fig. 30: Omnidirectional Antenna

One of the useful debug tools is an omnidirectional antenna placed on the roof of the Medicina building that we have used to inject tones on the secondary lobes of the array radiation pattern. Its RF coax cable is directly connected to a signal generator, placed in the receiver room, that provides sinusoidal monochromatic signals with programmable power and frequency.

This antenna works in the frequency range between 25MHz to 1300MHz without adding gain.

The signal generators we have used are: a) the HP 8657B with an output frequency range of 100kHz to 2.6GHz (1Hz resolution) and amplitude from +13dBm to –143.5dBm into (0.1db resolution); b) the Rohde&Schwarz SMX Signal Generator with an output frequency range of 100kHz to 1000MHz (10Hz best resolution, it depends on the output frequency set) and output level from +13dbm to -137dbm. Both the signal generators have an internal oscillator lockable to the station's 10MHz distributed from the Maser. One of them has been configured and used to transmit a monochromatic tone with the omnidirectional antenna to a frequency very close (~ 25KHz) to the center of the RF bandwidth (408MHz); the other one has been used for providing the A/D clock.

Fig. 31: HP 8657B

The frequency analysis has been performed using a software that processes stored data because we do not have in our labs a spectrum analyzer able to reach the 1 Hertz resolution.

Fig. 32: Rohde&Schwarz SMX

This software provides only results useful for debug and it has been written in IDL which is a scientific language. It is a project based on widget running on a microsoft windows platform (even if the IDL code is portable on every platform as it is if you do not use absolute file path), very easy to use and user friendly (messages will help you to set parameters).

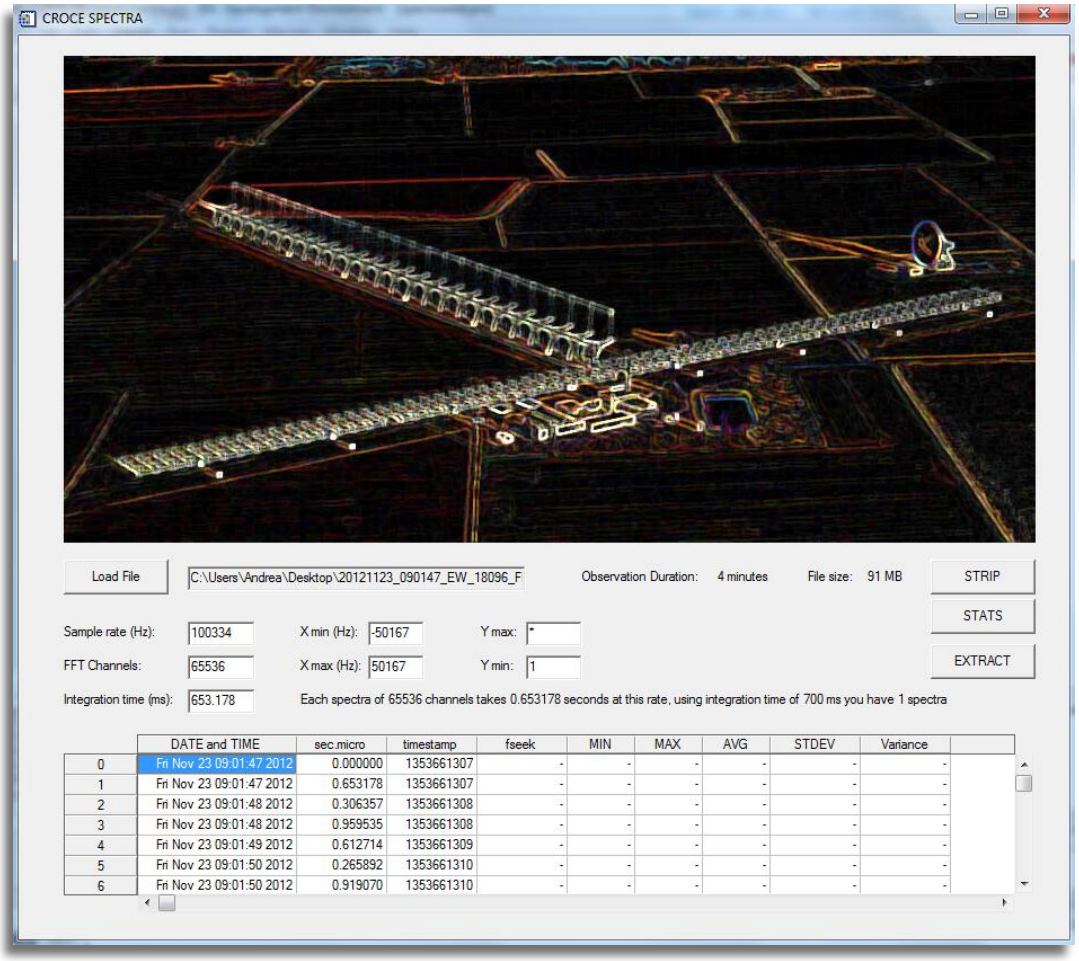


Fig. 33: The main widget of the IDL spectrometer

Running the IDL project it is shown the main widget as in the picture above. The upper part where it is shown an artistic view of the Medicina station and its radio telescopes it is the plot drawn area.

By clicking on the LOAD push button you can select the input file on a dialog window. This input file is the "dat" file saved by the beamformer storage. The size of the file and the observation time will be reported taking into account the value written into the "sample rate" text box, by default, this value has been set to 100334Hz which is the sample rate of the beamformer.

The FFT channel and the integration time needs to be set depending on what we are going to investigate. If we want to check if the monochromatic tone has been detected at the expected frequency we need to have a channel resolution of 1 Hz, therefore, if we set 65536 number of channel over a sample rate of 100KHz the resulting resolution is 1.53Hz, remembering that the FFT it is an algorithm optimized to work to 2 power numbers. However, increasing the number of channels means increase the number of samples needed to compute the FFT, that means you are increasing the time window. If we want to see a fast debris transiting over the antenna beam using many samples to have a high accuracy can be a wrong approach. The following table shows some elementary configurations with very interesting numbers:

Num. of Channels	Chan. Resolution (Hz)	Time (s)
1024	97.98242	0.010205912
2048	48.99121	0.020411825
4096	24.49561	0.040823649
8192	12.2478	0.081647298
16384	6.123901	0.163294596
32768	3.061951	0.326589192
65536	1.530975	0.653178384
131072	0.765488	1.306356768

Tab. 7: Channel Resolution and Time Window over Number of Channels

This table is useful but incomplete, because increasing the spectra resolution (and consequently the time window) the measured power of the detected debris will decrease but we have no measured how long in terms of db. On the other hands, if you are looking to a constant tone, which is not a pulse, increasing the time window and the channel resolution also the frequency line of the tone will increase because the power of the signal will be not affected by the "lower" frequencies contained on the same channel.

Sampling

The samples must be acquired regularly, this is the requirements of the Fourier Transform. If the samples are not equally spaced in time the resulting frequencies will be wrong. To test if the sampler is working well it is necessary to analyze carefully the spectra using the higher channel resolution, possibly 1 Hz per channel.

The test consists to transmit a tones of a certain frequencies with the omnidirectional antenna and look at the frequencies line in the spectra, one at the time, they have to match.

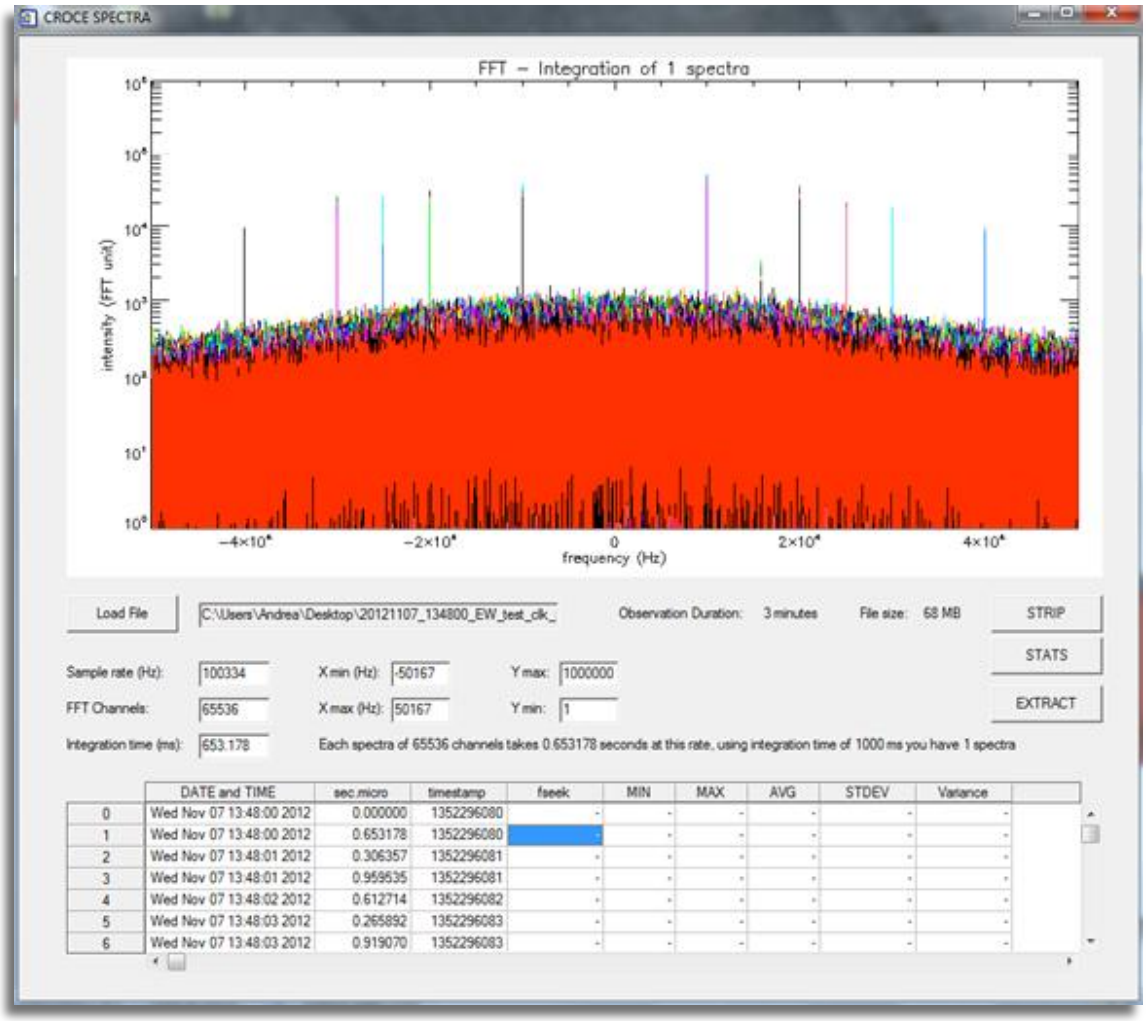


Fig. 34: Overplotting FFTs of the entire file you can recognize the frequencies used for the test.

Making this test we have found a misbehavior of one signal generator, the HP 8657B that probably has begun to work bad. We have scan many frequencies as seen on the previous picture and analyzing the tones at Hertz resolution and comparing the signal generators we have found very little difference as shown on the following pictures:

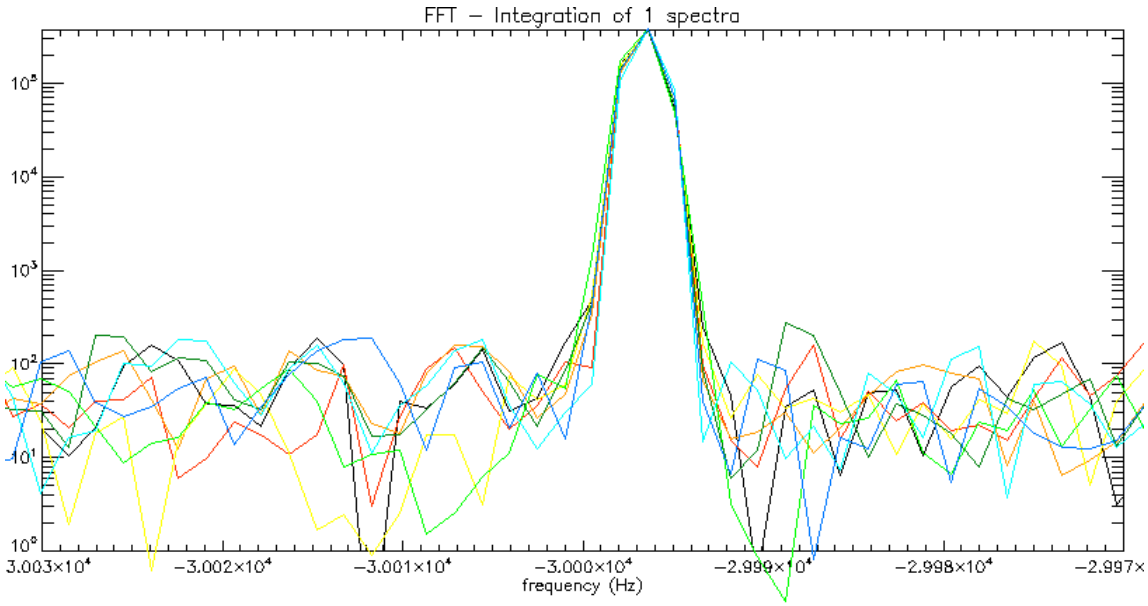


Fig. 35: Frequency 407.970kHz sampled using the HP 8657B as clock sampler.
The read frequency results 407.974kHz, 4Hz difference

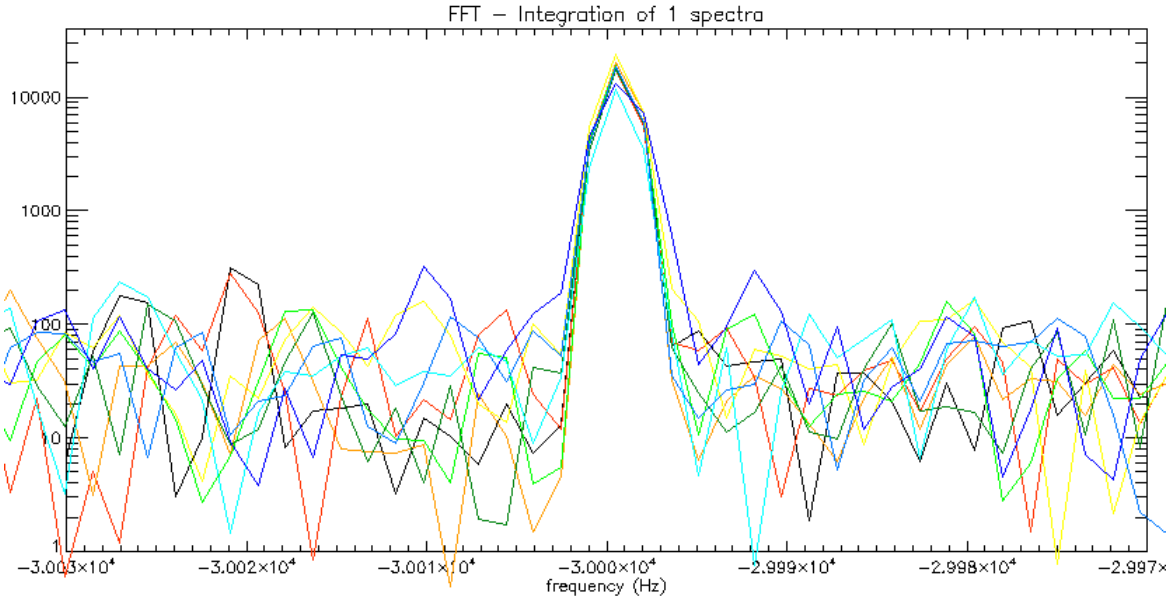


Fig. 36: Frequency 407.970kHz sampled using the Rohde&Schwarz SMX as clock sampler.
The read frequency results exactly 407.970kHz

Fast Fourier Transform Considerations

There is an important consideration to be done talking of channelization and accuracy of FFT. As you can see on the previous pictures also the peak observed with the Rohde&Schwarz is not exactly centered to the 407.970kHz frequency. This is due by the definition of the FFT algorithm which is an optimized extension of the Fourier Transform, computational complexity is reduced from N^2 to $N \cdot \log_2(N)$. Regarding complex input data, frequency lines occur at intervals Δf, where:

$$\Delta f = \frac{\text{Total Bandwidth}}{\text{Number of Channels}} = \frac{F_s}{N}$$

and because sample rate and number of channels are not necessarily multiple, most of the time Δf is not an integer number. The FFT algorithm works efficiently if the number of channels is a 2's power number and the total bandwidth has not been chosen taking into account this aspect. Each channel can be referred to as frequency bins (or FFT bins) because you can think of an FFT as a set of parallel filters of bandwidth Δf centered at each frequency increment of Δf and it contains a power contribute of frequencies contained on the same channel.

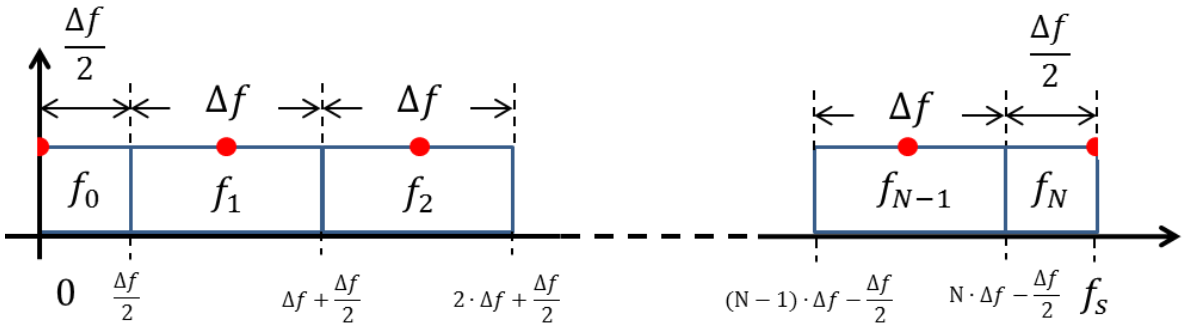
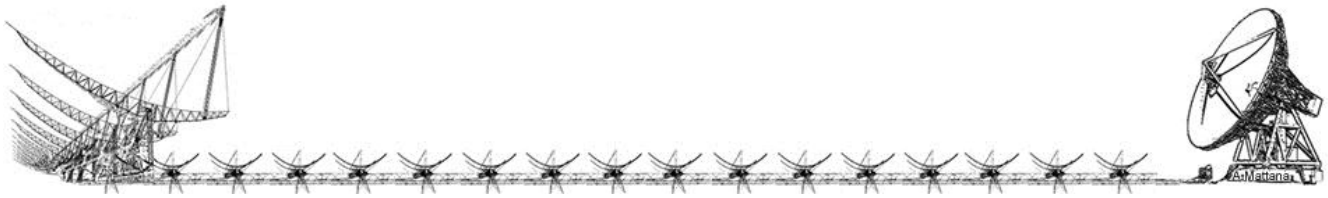


Fig. 37: FFT bins

The above picture shows how the red points (FFT lines) are the center of the FFT bins (or FFT channels) of Δf width (unless the first and the last channel that have half width) bounded within

$$N * \Delta f - \frac{\Delta f}{2} \quad \leq \quad f_N \quad < \quad N * \Delta f + \frac{\Delta f}{2}$$

Looking at our application, in the specific configuration of using 65536 FFT channels



$$\Delta f = \frac{F_s}{N} = \frac{100334}{65536} = 1.531$$

where the central frequency 408MHz is placed on the channel number 32768 and represents the power contribution between

$$408\text{MHz} - \frac{1.531}{2} \leq f_{\frac{N}{2}} < 408\text{MHz} + \frac{1.531}{2}$$

That means, if we want to check a specific frequency we should use frequencies which correspond to the center of that frequency channel. For instance the center of the frequency channel drawn on the previous pictures, the 407.970MHz, should have been 407970000.404Hz, and it contains the power measured within the limit:

$$[407969999.639, \quad 407970001.170]$$

which includes the 407.970kHz frequency line, but its contributed has been drowned in the 407970000.404Hz line. As a confirmation, the peak of the measured tone on the second picture (Rohde&Schwarz) is just to the right of the -30kHz (relative to the center of 408MHz, therefore 407.970MHz) of about half Hertz confirming the calculated center (407970000.404Hz), but is representing also the tone transmitted to exactly 407.970MHz.

Even if the frequency to be transmitted should match with a center of a bin, the frequency resolution of the signal generators (1Hz in case of the HP and 10Hz the Rohde&Schwarz) does not allowed to fit exactly the requirements, therefore frequencies for the test have been chosen arbitrary.

Signal Generation Report

Here shown in the next table the results of the scan test:

Frequency (MHz)	HP 8657B Δ (Hz)	Rohde&Schwarz SMX Δ (Hz)
407.960	0	0
407.970	+4.5	0
407.975	0	0
407.980	+21	0
407.990	+21	0
408.010	+22	0
408.020	+22	0
408.025	+25	0
408.030	+24	0
408.040	+20	0

Tab. 8: Delta Frequencies of the two signal generators

Both the signal generators have been locked to the same 10MHz distributed from the Maser, we have investigated also if the level of the reference input for the HP was too high and saturated but it was within the specifics, therefore we conclude that the HP is not suitable to be used as a A/D clock generator for spectrometry at that frequencies, maybe the PLL of the synthesizer has started to have troubles.

Results

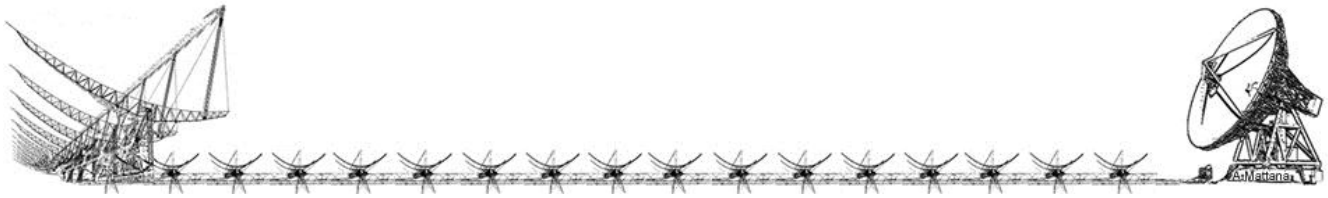
Here we examine the results of some observational tests performed in order to check the beamformer system. The aim of these measurements was the comparison between the simulated and the observed beams.

The Northern Cross incapability to track objects and the narrow bandwidth (100 KHz) of the Space Debris acquisition system imposed the observation of very strong radio sources at 408 MHz (Tab. 1) in order to have a good SNR.

Source	**RA J2000 [hh mm ss.s]**	**Dec J2000 [dd pp ss.ss]**	**Flux density @ 408 MHz [Jy]**
Cyg-A	19 59 28.4	+40 44 02.10	4862
Tau-A	05 34 31.9	+22 00 52.2	1215
Vir-A	12 30 49.4	+12 23 28.04	486

Tab. 9: The three radio-sources observed for the beamformer test.

The total power was obtained with 1 second of time integration from the data acquired in the time domain. It allowed us to describe the beam shape for both a single receiver and a beam synthetized from 4 receivers. The receiver phases were previously calibrated utilizing the interferometric fringes recorded with 2 MHz band.



The theoretical E-plane primary beams of the two configurations are shown in Fig.38.

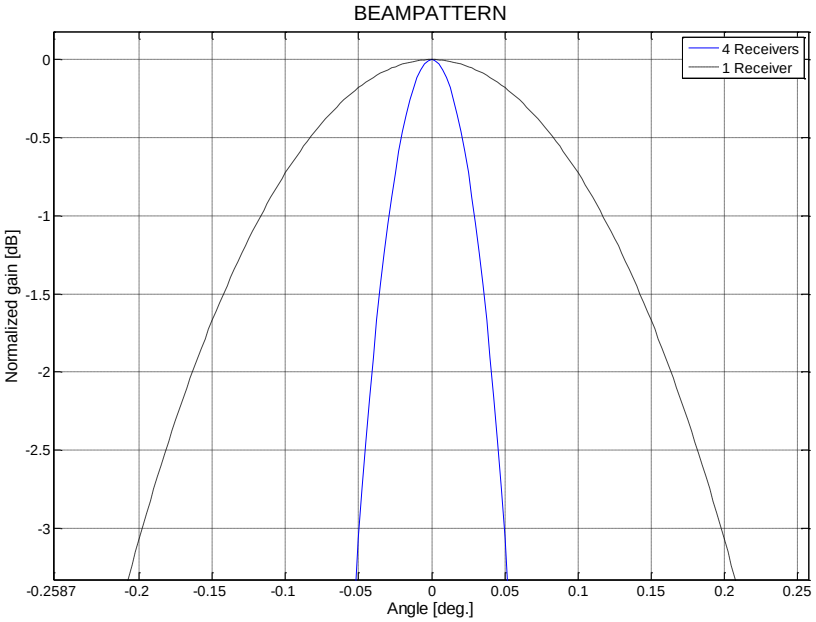


Fig. 38: Normalized E-plane power patterns calculated for a single E-W receiver (dotted black line) and for a synthetized beam of 4 receivers (continuous blue line).

The theoretical -3dB beam width of a single E-W channel is about 0.4°, while the sum of the 4 channels yields a value of about 0.1°. These values were compared to those obtained from the signal total power recorded during the transit of Cygnus-A (Fig. 39), Virgo-A (Fig. 40) and Taurus-A (Fig. 41) within the antenna beam. A synthetized beam of 2 receivers was also produced for the Taurus-A observation.

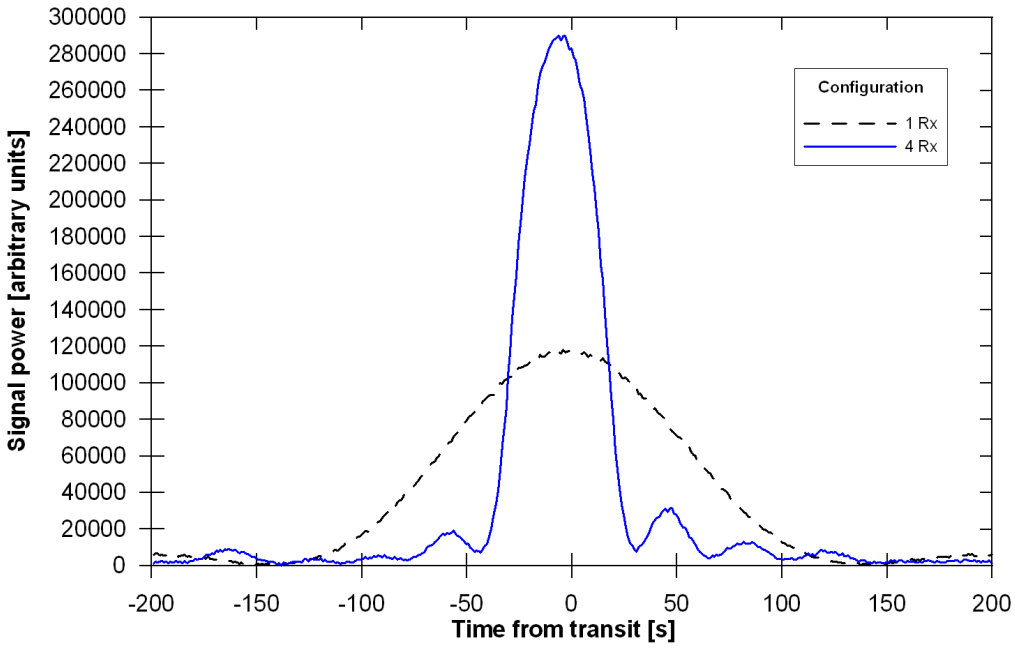


Fig. 39: Transit of Cygnus-A observed by a single E-W receiver (dashed black line) and by a synthetized beam of 4 receivers (continuous blue line).

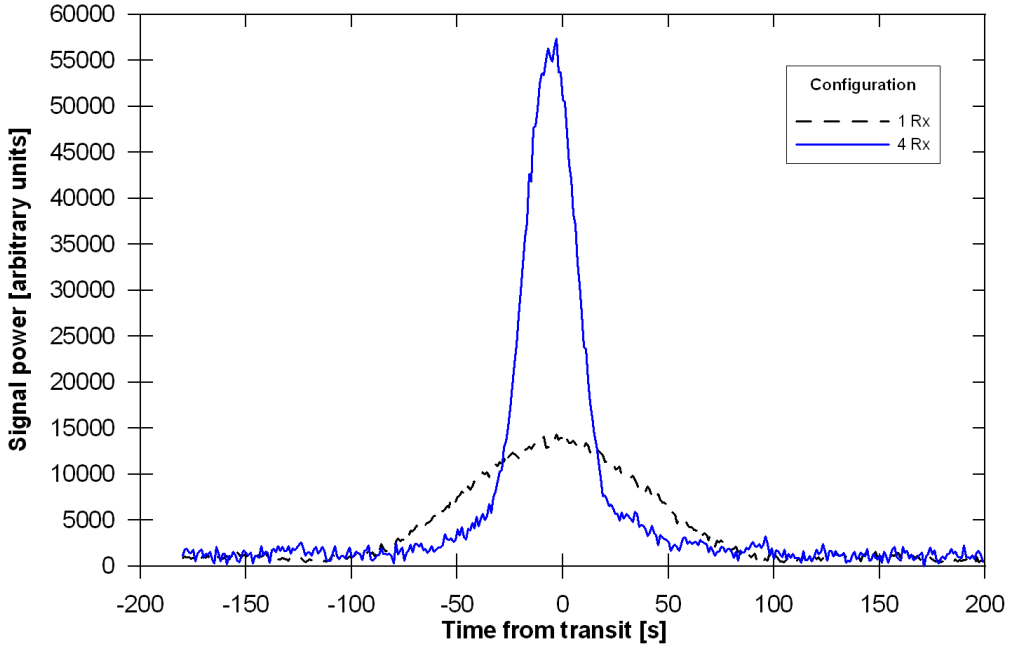


Fig. 40: Transit of Virgo-A observed by a single E-W receiver (dashed black line) and by a synthetized beam of 4 receivers (continuous blue line).

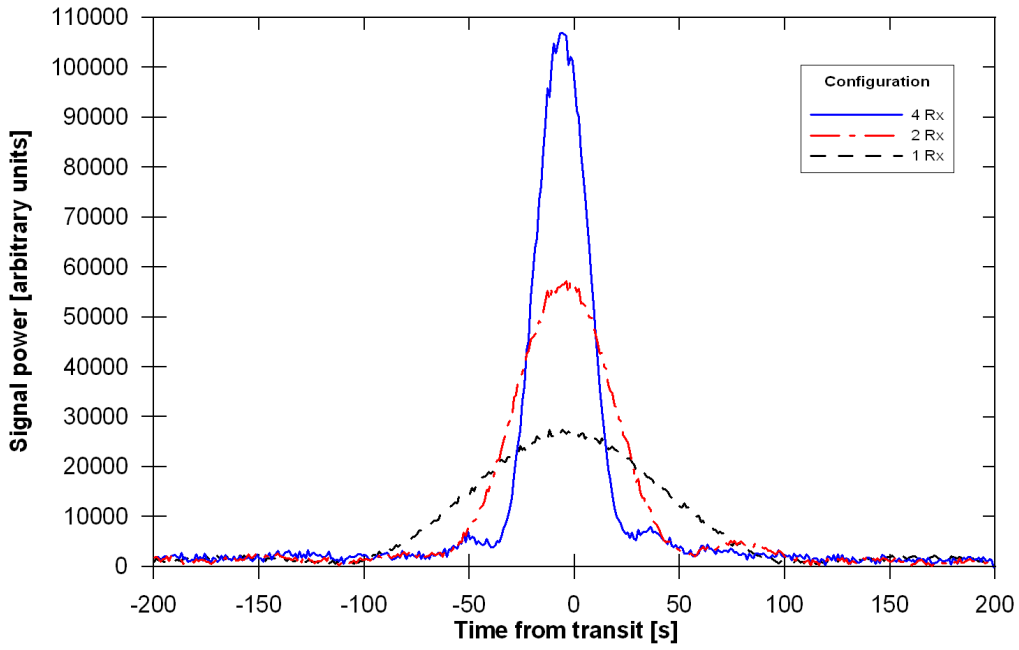


Fig. 41: Transit of Taurus-A observed by a single E-W receiver (dashed black line), by a synthetized beam of 2 receivers (dashed dotted red line) and of 4 receivers (continuous blue line)

The half power beamwidth, $\vartheta_{\mathrm{HPBW}}$, can be calculated according to the following equation:

$$\vartheta_{\mathrm{HPBW}}(^{\circ}) = \frac{15^{\circ}\cos(\delta)\Delta t_{\mathrm{HPBW}}}{3600}$$

where

$\delta =$ source declination
$\Delta t_{\mathrm{HPBW}} =$ transit time, expressed in s, at half power signal

The calculated beamwidths as well as the signal amplitude ratios acquired by the different antenna configurations are in good agreement with the theoretical values for Virgo-A and Taurus-A. Whereas the Cygnus-A expected beamwidth and signal amplitude significantly differ from the observed ones. The disagreement was probably due to the analogue receivers saturation caused by the extremely high flux of this source.

Moreover the maximum of the signal, corresponding to the transit of the source at the local meridian, happens some seconds earlier than expected. This time difference could be caused by an antenna mechanical misalignment of about one arcmin in the East direction.

Finally, the system has been tested in a real observational scenario during the space debris radar campaigns carried out by the Northern Cross array on July and December 2012. Several targets orbiting in LEO were successfully acquired by this beamformer system with a very high SNR.

An example of space debris radar detection is the observation of the first passage of the NEXTSAT satellite during the 2012 December 17 radar session. This target was a deactivated satellite (USSTRATCOM catalogue n. 30774) orbiting in LEO. The echo of NEXTSAT (Fig. 42), having a mean extimated Radar Cross Section (RCS) of 3.679 m^2, was detected in the frequency domain by a FFT analysis with 16384 channels, corresponding to a spectral resolution of 6.1 Hz/ch.

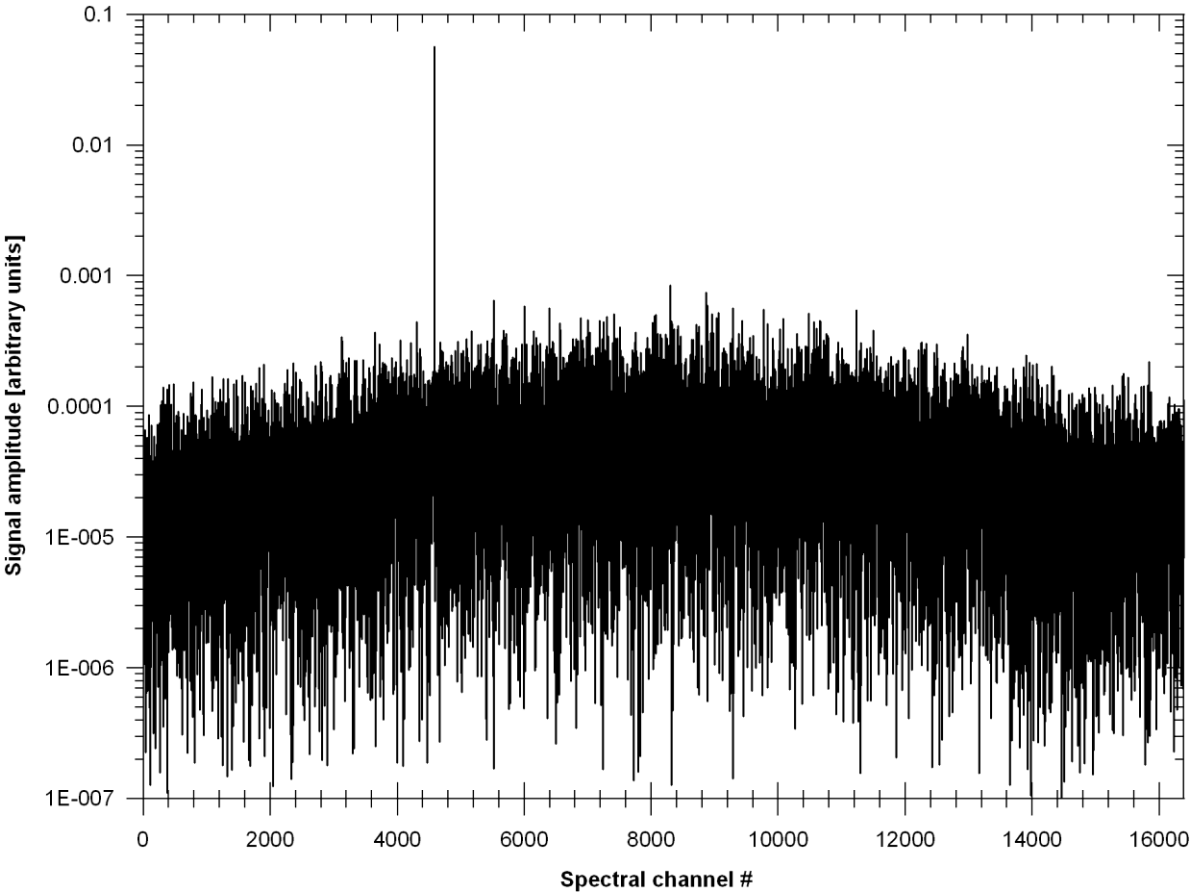
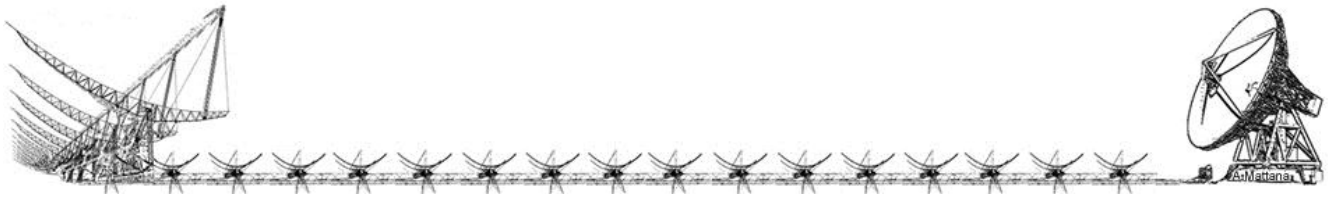


Fig. 42: Spectrum of the echo from the target NEXTSAT detected during on 2012 December 17 at 09:01:06.25 UT. The spectral window is centred at the transmitting frequency. Due to the extremely high SNR of the echo, the signal amplitude is plotted in logarithmic scale.

The target was orbiting at an altitude of 510.49 km with a bistatic slant range (signal total path length) of about 1057 km. The observed transit time and bistatic frequency Doppler shift were slightly different to those calculated from the TLEs by the orbital numeric propagators (e.g. SGP4). These differences between observed and calculated values are of extremely importance to improve the knowledge of the target orbit that is one of the most important goal of the space debris investigation.



Python scripts

Beecool

Readme

Copy all those files in the same bee2 directory.

Requirements:

- At least one bof file running

Run “start_server.py” and follow the instructions.

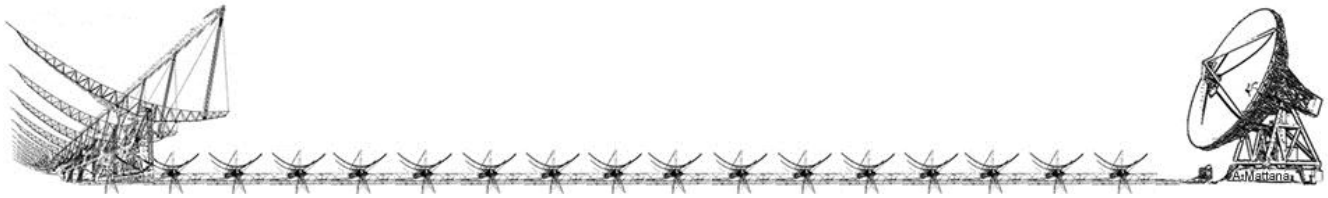
start_server.py

```
#! /usr/bin/env python
"""
Search the bof PID.

eg usage:

"""
import os, time
branches=['61001', '62001', '63001','64001']
os.system("ps -ef | grep bof | grep fpga > pid/pids.txt")
print "\n##########################################\n"
print "Beamformer Server Launch Wizard"
print "\nPlease follow the instruction...\n"
data = []
for line in open("pid/pids.txt",'r').readlines():
    #print line.split()[1], line.split()[7]
    data.append([line.split()[1],line.split()[7]])
c=0
data.pop()
for i in data:
    print '['+str(c+1)+'] ', data[c][0], data[c][1]
    c = c+1
print "\nSelect the PID index: ",
pid = raw_input()

print "\n\n[1] NS (usually IBOB2-BEE1)"
print "[2] EW (usually IBOB7-BEE4)\n"
print "Select the branch: ",
branch = raw_input()
```

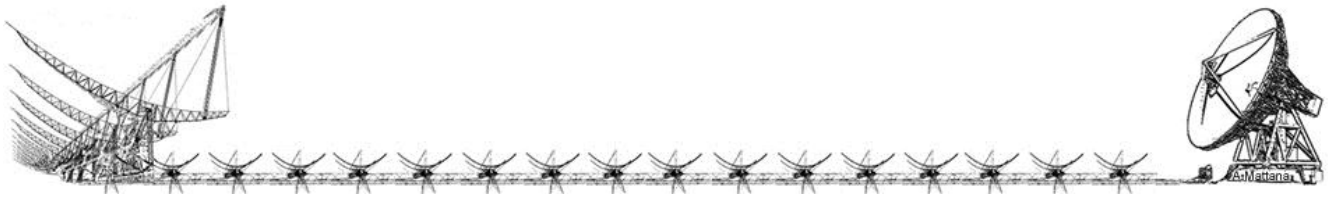
```
time.sleep(0.2)
print "\n########################################"
print "\nStarting server for"
print "  PID: "+data[c-1][0]
print "  BOF: "+data[c-1][1]+"\n"
time.sleep(0.2)
cmd="./bee_sdeb_server.py -p "+branches[int(branch)-1]+" -i "+data[c-1][0]
#print cmd
os.system(cmd)
```

bee_sdeb_server.py

```
#!/usr/bin/env python
"""
Server running on bee2 waiting for nitialization commands.
"""
import struct
import sys
import os
import SocketServer
from optparse import OptionParser

class PIDException(Exception):
    def __init__(self, msg):
        Exception.__init__(self, msg)

class CommandException(Exception):
    def __init__(self, msg):
        Exception.__init__(self, msg)

class LofarTCPHandler(SocketServer.StreamRequestHandler):
    def handle(self):
        self.data = self.rfile.readline().strip()
        print "received: " + self.data
        args = self.data.split()
        try:
            res = self.server.execute(args)
            self.wfile.write(str(res))
        except CommandException, ce:
            self.wfile.write(ce.args[0])

class LofarTCPServer(SocketServer.TCPServer):
    def __init__(self, addr, pid):
        SocketServer.TCPServer.__init__(self, addr, LofarTCPHandler)
        self.pid = pid
        self.base_path = "/proc/%s/hw/ioreg/"%(pid,)
        self.devs = os.listdir(self.base_path)
        self.commands = {
                         "listdev": self.listdev,
                         "write_int": self.write_int,
                         "read_int": self.read_int,
                         "read_bram": self.read_bram,
                         "ask_pid": self.ask_pid,
                         "start_gbe": self.start_gbe,
                        }

    def listdev(self):
        return " ".join(self.devs)
```



```
    def ask_pid(self):
        return self.pid

    def _valid_register(self, name):
        if not name in self.devs:
            raise CommandException("Register %s not found."%(name,))

    def read_int(self, reg_name):
        self._valid_register(reg_name)
        f = open(os.path.join(self.base_path, reg_name), "rb")
        bs = f.read(4)
        f.close()
        res = struct.unpack(">L", bs)[0]
        return res

    def write_int(self, reg_name, reg_value):
        self._valid_register(reg_name)
        try:
            reg_value = int(reg_value)
            print "writing: " + str(reg_value)
        except:
            raise  CommandException("write_int  second  argument  should  be  an
integer")
        f = open(os.path.join(self.base_path, reg_name), "wb")
        f.write(struct.pack(">L", reg_value))
        f.flush()
        f.close()
        return "SUCCESS"

    def start_gbe(self, gbe_conf_file, gbe_name):
        print "Executing: cp "+gbe_conf_file+" "+self.base_path+gbe_name
        a = os.system("cp "+gbe_conf_file+" "+self.base_path+gbe_name)
        return a

    def read_bram(self, bram_name, bram_len):
        self._valid_register(bram_name)
        f = open(os.path.join(self.base_path, bram_name), "rb")
        bs = f.read(bram_len)
        f.close()
        return bs


    def execute(self, args):
        if not self.commands.has_key(args[0]):
            raise CommandException("Command %s not found."%(args[0],))
        if len(args) > 1:
            try:
                res = self.commands[args[0]](*(args[1:]))
            except TypeError, te:
                raise CommandException(te.args[0])
        else:
            res = self.commands[args[0]]()
        return res

def verifyPID(pid):
    pid = str(pid)
    pipe = os.popen("ps a | grep %s"%(pid,))
    pss = pipe.readlines()
    pipe.close()
```



```
    pss = [ps[:-1].split() for ps in pss if ps.split()[0] == pid]
    if not pss:
        raise PIDException("Process %s not found."%(pid,))
    if not pss[0][-1].endswith(".bof"):
        raise PIDException("Process %s is not a running bof file design"%(pid,))

if __name__ == "__main__":
    op = OptionParser()
    op.add_option('-p', '--port', dest='port', type='int', default=60001,
                help='The listening port')
    op.add_option('-i', '--pid', dest='pid',
                help='The process id (pid) of the running bof file')
    opts, args = op.parse_args(sys.argv[1:])
    pid = str(opts.pid)
    try:
        verifyPID(pid)
    except PIDException, pe:
        print pe.args[0]
        sys.exit(0)
    #server = SocketServer.TCPServer(("localhost", opts.port), LofarTCPHandler)
    server = LofarTCPServer(("", opts.port), pid)
    try:
        print "server listening on port: " + str(opts.port)
        server.serve_forever()
    except KeyboardInterrupt:
        print "closing communication"
        del(server)
        sys.exit(0)
```

gbe_fpga1_monitor.conf

```
begin
       mac = 00:12:6D:AE:0B:13
       ip = 192.168.11.13
       gateway = 192.168.11.10
       port = 61003
end
```

gbe_fpga1_storage.conf

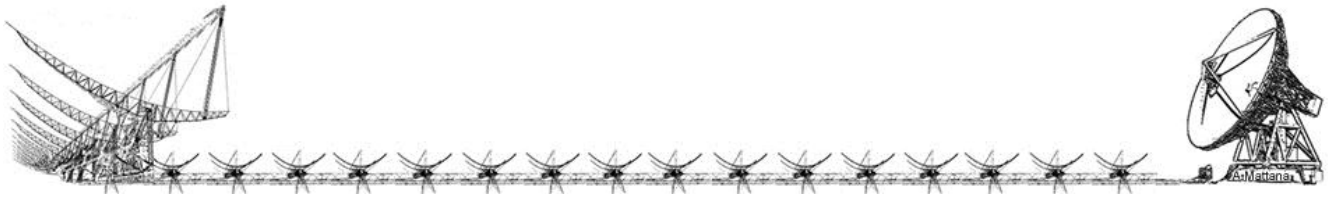
```
begin
       mac = 00:12:6D:AE:0B:14
       ip = 192.168.11.14
       gateway = 192.168.11.10
       port = 61004
end
```

gbe_fpga2_monitor.conf

```
begin
       mac = 00:12:6D:AE:0B:23
       ip = 192.168.11.23
       gateway = 192.168.11.10
       port = 62003
end
```

gbe_fpga2_storage.conf

```
begin
       mac = 00:12:6D:AE:0B:24
       ip = 192.168.11.24
       gateway = 192.168.11.10
       port = 62004
end
```



gbe_fpga3_monitor.conf

```
begin
       mac = 00:12:6D:AE:0B:33
       ip = 192.168.11.33
       gateway = 192.168.11.10
       port = 63003
end
```

gbe_fpga3_storage.conf

```
begin
       mac = 00:12:6D:AE:0B:34
       ip = 192.168.11.34
       gateway = 192.168.11.10
       port = 63004
end
```

gbe_fpga4_monitor.conf

```
begin
       mac = 00:12:6D:AE:0B:43
       ip = 192.168.11.43
       gateway = 192.168.11.10
       port = 64003
end
```

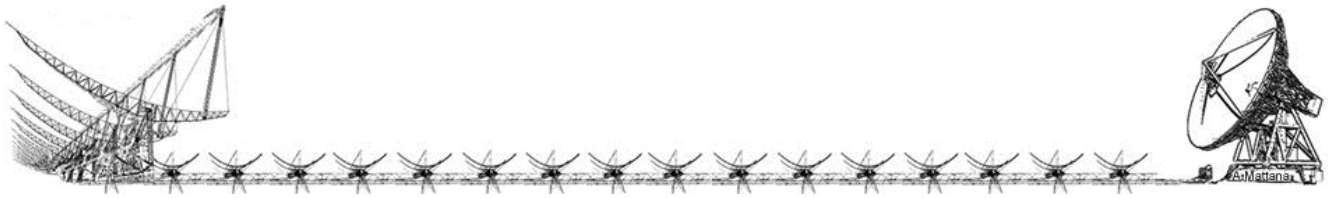
gbe_fpga4_storage.conf

```
begin
       mac = 00:12:6D:AE:0B:44
       ip = 192.168.11.44
       gateway = 192.168.11.10
       port = 64004
end
```

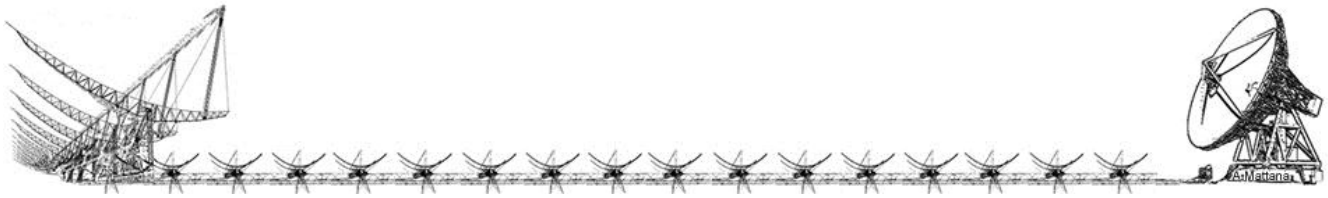
Control

Readme

```
Decide how many receivers are you going to use and type the phase corrections
on the ew_beamf.conf (or ns_beamf.conf) file.

Requirements:

   - a bof file running on the bee2 (beecool)
   - a server running on the bee2 (beecool)
   - a recorder server running on 192.167.189.66 (batman)

1. Initialization

   Run: ./sdeb_init.py -l ew_beamf.conf

Example:

Last login: Tue Nov 13 10:16:11 2012 from 192.167.189.65
oper@bee2:~$ cd /media/data/sdeb/
oper@bee2:/media/data/sdeb$ ./sdeb_init.py -l ew_beamf.conf
2012-11-13 10:19:16,294 - ************************************
2012-11-13 10:19:16,294 - ******* INITIALIZATION PROCESS ********
2012-11-13 10:19:16,294 - ************************************
2012-11-13 10:19:16,294 - parsing configuration file ew_beamf.conf
2012-11-13 10:19:16,295 - Initializing the Space Debris system
2012-11-13 10:19:16,295 - Sending Master Reset to Bee2 and IBOBs
2012-11-13 10:19:16,630 - Setting equalization on main fpga
2012-11-13 10:19:16,630 - Setting (RE_0, IM_0) to: (127, 0)
2012-11-13 10:19:16,722 - Setting (RE_1, IM_1) to: (127, 0)
2012-11-13 10:19:16,816 - Setting (RE_2, IM_2) to: (127, 0)
```



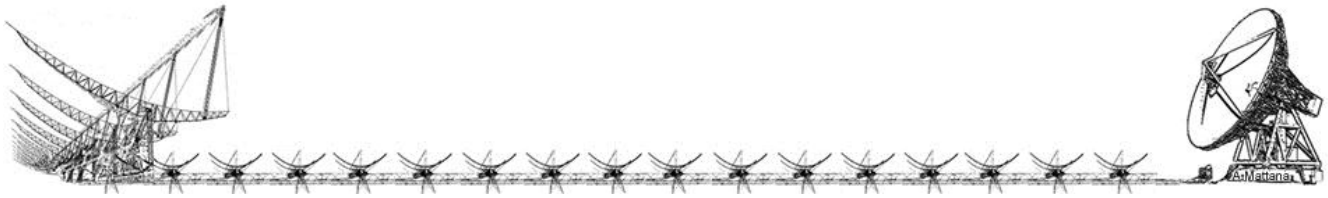
```
2012-11-13 10:19:16,909 - Setting (RE_3, IM_3) to: (127, 0)
2012-11-13 10:19:17,003 - Setting (RE_4, IM_4) to: (0, 0)
2012-11-13 10:19:17,096 - Setting (RE_5, IM_5) to: (0, 0)
2012-11-13 10:19:17,190 - Setting (RE_6, IM_6) to: (0, 0)
2012-11-13 10:19:17,283 - Setting (RE_7, IM_7) to: (0, 0)
2012-11-13 10:19:17,377 - Setting (RE_8, IM_8) to: (0, 0)
2012-11-13 10:19:17,471 - Setting (RE_9, IM_9) to: (0, 0)
2012-11-13 10:19:17,564 - Setting (RE_10, IM_10) to: (0, 0)
2012-11-13 10:19:17,658 - Setting (RE_11, IM_11) to: (0, 0)
2012-11-13 10:19:17,751 - Setting (RE_12, IM_12) to: (0, 0)
2012-11-13 10:19:17,845 - Setting (RE_13, IM_13) to: (0, 0)
2012-11-13 10:19:17,939 - Setting (RE_14, IM_14) to: (0, 0)
2012-11-13 10:19:18,032 - Setting (RE_15, IM_15) to: (0, 0)
2012-11-13 10:19:18,126 - Setting decimation factor to 13
2012-11-13 10:19:20,099 - Setting Data Format to 16.11
2012-11-13 10:19:20,211 - Setting Packet Length to 160
2012-11-13 10:19:20,321 - Setting Monitor IP to 192.168.11.10 (3232238346)
2012-11-13 10:19:20,432 - Setting Monitor Port to 64003
2012-11-13 10:19:20,542 - Setting Storage IP to 192.168.11.11 (3232238347)
2012-11-13 10:19:20,651 - Setting Storage Port to 64004
2012-11-13 10:19:20,761 - Starting 10GbE interface: monitor
2012-11-13 10:19:20,834 - Starting 10GbE interface: storage
2012-11-13 10:19:20,909 - Performing time update for sync...
2012-11-13 10:19:21,014 - Sending: 2012/11/13 09:19:21 UT (timestamp: 1352798361)
2012-11-13  10:19:22,155  -  Received:  2012/11/13  09:19:22  UT  (timestamp:
1352798362)
2012-11-13 10:19:22,155 - Time updated successfully

Initialization Process Successfully Completed!

2. Prepare an observation file

   - Edit a sched file on 'sched' directory.

Example:

oper@bee2:/media/data/sdeb$ more sched/dirac_test_ew.conf
[Obs]
System = EW
Start_time = 2012/11/13_09:27:00
Stop_time = 2012/11/13_09:28:00
Target = targetname

3. Load observation file and arm the system

  - Run "./sdeb_run.py -l sched/dirac_test_ew.conf" with the sched file

oper@bee2:/media/data/sdeb$ ./sdeb_run.py -l sched/dirac_test_ew.conf
2012-11-13                              10:26:23,423                              -
************************************************************
2012-11-13  10:26:23,423  -  ******************  ARM  A  NEW  OBSERVETION
******************
2012-11-13                              10:26:23,423                              -
************************************************************
2012-11-13 10:26:23,423 - Parsing configuration file sched/dirac_test_ew.conf
2012-11-13 10:26:23,423 - Loading observation parameters...
2012-11-13 10:26:23,439 - Loaded START time 2012/11/13 09:27:00 UT (timestamp:
1352798820)
```



```
2012-11-13 10:26:23,451 - Loaded STOP time 2012/11/13 09:28:00 UT (timestamp:
1352798880)
2012-11-13 10:26:24,042 - System EW armed!
2012-11-13 10:26:24,060 - Data expected: about 22 MB (24080232 bytes)
2012-11-13 10:26:24,061 - Starting Recording data on 192.167.189.66:64005
2012-11-13 10:26:24,067 - Observation "targetname" Loaded Successfully!

oper@bee2:/media/data/sdeb$
```

dataconversion.py *(author Marco Bartolini)*

```
"""Module dataconversion:

functions to convert numbers to and from simulink representation.
Permits conversion of 8, 16 and 32 bit representation of signed and unsigned
decimal numbers, with or without binary point.

classes:
    ConversionError: bit mismatch and format incopatibility

functions:
    get_conversion_t: factory function of conversion specs
    unsigned2real:    convert from simluink representation
    bytes2real:       convert binary data from simulink representation
    stream2real:      convert array of binary data
    real2unsigned:    convert a real number into simluink representation
"""

import struct

class ConversionError(Exception):
    def __init__(self, msg):
        Exception.__init__(self, msg)

def get_conversion_t(bits, bin_point, signed=False, scaling=1.0):
    """
    bits = the number of bits
    bin_poin = simulink binary point position
    signed = True if Fix, Flase if UFix
    scaling = optional scaling to be applied after the conversion

    returns a conversion structure that can be applied in both directions of
    conversion for the given specs.
    """
    conversion_t = {}
    conversion_t["bits"] = bits
    conversion_t["bin_point"] = bin_point
    conversion_t["signed"] = signed
    conversion_t["scaling"] = scaling
    conversion_t["dec_step"] = 1.0 / (2 ** bin_point)
    #dec_max = dec_mask * dec_step
    conversion_t["dec_mask"] = sum([2 ** i for i in range(bin_point)])
    if bits == 8:
        conversion_t["fmt"] = "B"
    elif bits == 16:
        conversion_t["fmt"] = "H"
    elif bits == 32:
```

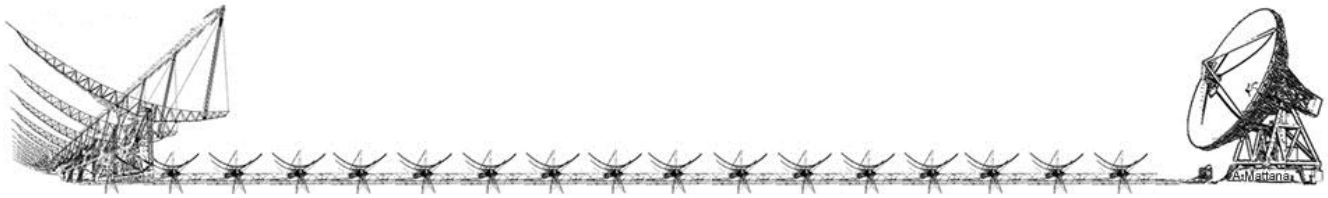


```
        conversion_t["fmt"] = "I"
    else:
        raise ConversionError("numer of bits not supported: " + str(bits))
    if signed:
        _get_signed_params(conversion_t)
    else:
        _get_unsigned_params(conversion_t)
    return conversion_t

def _get_unsigned_params(conv):
    conv["sign_mask"] = 0
    conv["int_min"] = 0
    conv["int_mask"]   =   sum([2   **   i   for   i   in   range(conv["bin_point"],
conv["bits"])])
    conv["int_max"] = sum([2 ** i for i in range(conv["bits"] -
        conv["bin_point"])])

def _get_signed_params(conv):
    conv["sign_mask"] = 2 ** (conv["bits"] - 1)
    conv["int_min"] = -1 * (2 ** (conv["bits"] - 1 - conv["bin_point"]))
    conv["int_mask"] = sum([2 ** i for i in range(conv["bin_point"], conv["bits"]
- 1)])
    conv["int_max"] = sum([2 ** i for i in range(conv["bits"] -
        conv["bin_point"] - 1)])

def unsigned2real(uval, conv):
    """
    uval = the numeric unsigned value in simulink representation
    conv = conv structure with conversion specs

    returns the real number represented
    """
    res = 0
    int_val =  ((uval & conv["int_mask"]) >> conv["bin_point"])
    dec_val = conv["dec_step"] * (uval & conv["dec_mask"])
    sign = uval & conv["sign_mask"]
    res = conv["int_min"] + int_val + dec_val
    return (res / conv["scaling"])

def bytes2real(ub, conv, endianness="@"):
    """
    ub = binary number in simulink representation
    conv = conv structure with conversion specs, dimensions must match
    endianness = optionally specify bytes endianness for unpacking

    return the real number represented
    """
    data = struct.unpack(endianness + conv["fmt"], ub)[0]
    return convert(data, conv)

def stream2real(stream, conv, endianness="@"):
    size = len(stream) // (conv["bits"] // 8)
    fmt = endianness + str(size) + conv["fmt"]
    data = struct.unpack(fmt, stream)
    data = [convert(d, conv) for d in data]
    return data

def real2unsigned(real, conv):
```

```
    """
    real = the real number to be converted into simulink representation
    conv = conv structre with conversion specs

    return the unsigned int for the simulink representation. Raise a
    ConverisonError if conv structre can't handle the number.
    """
    if not conv["signed"] and real < 0:
        raise ConversionError("cannot convert " + str(real) + " to unsigned
representation")
    if real < 0:
        sign = 1
        real = real - conv["int_min"]
    else:
        sign = 0
    int_val, dec_val = divmod(abs(real), 1)
    int_val = int(conv["int_min"] + int_val)
    int_val = int_val & (conv["int_mask"] >> conv["bin_point"])
    val = int_val
    dec = 0
    while val < real and dec < conv["dec_mask"]:
        val += conv["dec_step"]
        dec += 1
    #Adjusting rounding error
    if (val - real) > (real - val + conv["dec_step"]):
        dec -= 1
    if sign == 1:
        return conv["sign_mask"] + ((int_val << conv["bin_point"]) &
                conv["int_mask"]) + dec
    else:
        return ((int_val << conv["bin_point"]) & conv["int_mask"]) + dec
```

pack_sdeb_pars_conf.py *(author Marco Bartolini)*

```
import ConfigParser

def get_fpga_conf(conf, section):
    fpga = {}
    fpga['server'] = {'addr': conf.get(section, "server").split(":")[0],
                      'port' : int(conf.get(section, "server").split(":")[1])
                     }
    #fpga['equalization']       =       [(conf.getint(section,       "RE_%i"%(i,)),
conf.getint(section, "IM_%i"%(i,))) for i in range(8)]
    fpga['equalization'] = []
    for i in range(16):
        try:
            fpga['equalization'].append(get_float_comment(conf,           section,
"d%i"%(i,)))
        except ValueError:
            fpga['equalization'].append(None)
    return fpga

def get_float_comment(conf, section, option):
    return float(conf.get(section, option).split()[0])

def parse_settings(filename):
    conf = ConfigParser.SafeConfigParser()
```



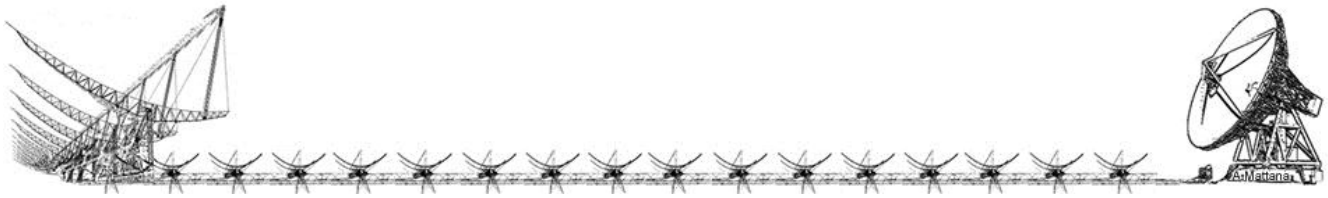
```
    #print "Parsing config file " + filename
    res = {}
    if not [filename] == conf.read(filename):
        print conf.read(filename)
        print "Cannot parse file " + filename
        return res
    res["dec_factor"] = conf.getint("global", "dec_factor")
    res["bee_dec_factor"] = conf.getint("global", "bee_dec_factor")
    res["integration"] = conf.getint("global", "integration")
    res["main_fpga"] = get_fpga_conf(conf, "main_fpga")

    res["data_format"] = conf.getint("global", "data_format")
    res["pck_length"] = conf.getint("global", "pck_length")

    res["gbe1_filename"] = conf.get("global", "gbe1_filename")
    res["gbe1_name"] = conf.get("global", "gbe1_name")
    res["monitor_ip"] = conf.getint("global", "monitor_ip")
    res["monitor_port"] = conf.getint("global", "monitor_port")

    res["gbe2_filename"] = conf.get("global", "gbe2_filename")
    res["gbe2_name"] = conf.get("global", "gbe2_name")
    res["storage_ip"] = conf.getint("global", "storage_ip")
    res["storage_port"] = conf.getint("global", "storage_port")

    return res

if __name__ == "__main__":
    filename = "ns_beamf_time.conf"
    options = parse_settings(filename)
    print options
```

pack_sdeb_pars_obs.py

```
import ConfigParser

def parse_settings(filename):
    conf = ConfigParser.SafeConfigParser()
    res = {}
    if not [filename] == conf.read(filename):
        print conf.read(filename)
        print "Cannot parse file " + filename
        return res

    res["System"] = conf.get("Obs", "System")
    res["Start_time"] = conf.get("Obs", "Start_time")
    res["Stop_time"] = conf.get("Obs", "Stop_time")
    res["Target"] = conf.get("Obs", "Target")

    return res
```

pack_sdeb_pars_systems.py

```
import ConfigParser
```

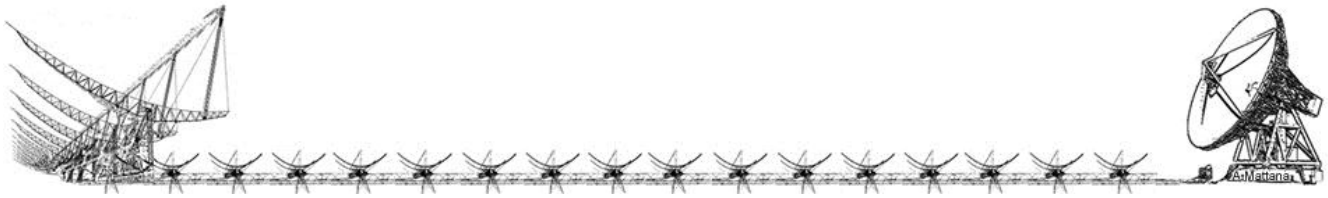


```
def parse_settings(filename,obs_system):
    conf = ConfigParser.SafeConfigParser()
    res = {}
    if not [filename] == conf.read(filename):
        print conf.read(filename)
        print "Cannot parse file " + filename
        return res

    res['server'] = {'addr': conf.get("Systems", obs_system).split(":")[0],
                     'port' : int(conf.get("Systems", obs_system).split(":")[1])
                    }

    return res
```

config_wizard.py

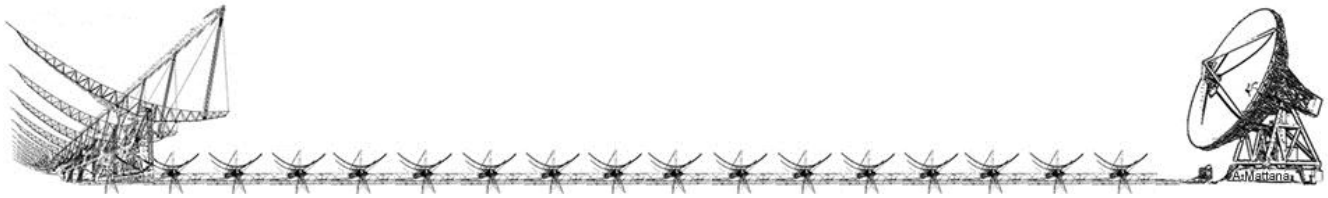
```
#! /usr/bin/env python
"""
Search the bof PID.

eg usage:

"""
import os
import datetime,time

tempfile = "config.temp"
rx_names = ['LOFAR antennas', 'NS receivers', 'EW channels']
rx_name  = [['2B', '1N-1-3', '4E'],
            ['3A', '1N-1-4', '5E'],
            ['1B', '1N-1-1', '2E'],
            ['2A', '1N-1-2', '3E'],
            ['4B', '2N-1-3', ''],
            ['5A', '2N-1-4', ''],
            ['3B', '2N-1-1', ''],
            ['4A', '2N-1-2', ''],
            ['6B', '3N-1-3', ''],
            ['7A', '3N-1-4', ''],
            ['5B', '3N-1-1', ''],
            ['6A', '3N-1-2', ''],
            ['8B', '4N-1-3', ''],
            ['9A', '4N-1-4', ''],
            ['7B', '4N-1-1', ''],
            ['8A', '4N-1-2', '']]

dataora = datetime.datetime.utcnow()
calib_date = "(date n/a)"
calib_time = "(time n/a)"
calib_source = "(source name n/a)"

print "\n##########################################\n"
print "Beamformer Configuration file Wizard"
print "\nPlease follow the instruction...\n"
f = open(tempfile,'w');
text  = "##################################################\n#\n"
text += "# Configuration file automatically generated by using\n# the Wizard on "
text += dataora.strftime("%d/%m/%Y %H:%M:%S")+" UT"
```

```
f.write(text)

text = "\n\n# NO WHITESPACE ALLOWED BETWEEN TEXT DELIMITERS! "
f.write(text)

text = "\n\n# design global parameters "
text += "\n[global] "
text += "\ndec_factor = 13  "
text += "\nbee_dec_factor = 23 "
text += "\n\n# integration for on board data accumulation "
text += "\nintegration = 100334"
f.write(text)

text = "\n\n# Common network parameters"
text += "\nmonitor_ip = 3232238346"
text += "\nstorage_ip = 3232238347"
text += "\npck_length = 160"
text += "\ngbe1_name = monitor"
text += "\ngbe2_name = storage"
f.write(text)

time.sleep(0.3)
print "+-----------+-----------+  "
print "|    2      |     3     |  "
print "+-----------+-----------+    BEE2 FPGAs"
print "|    1      |     4     |  "
print "+-----------+-----------+  \n"

print "[1] BEE FPGA-1 (front - left)  "
print "[2] BEE FPGA-2 (rear  - left)  "
print "[3] BEE FPGA-3 (rear  - right) "
print "[4] BEE FPGA-4 (front - right) "
print "\nWhich BEE2 FPGA are you going to use [1/2/3/4]? ",
branch = raw_input()

if branch == '1':
    text  = "\n\n# Parameters for the FPGA1 system"
    text += "\nmonitor_port = 61003"
    text += "\nstorage_port = 61004"
    text += "\ngbe1_filename = gbe_fpga1_monitor.conf"
    text += "\ngbe2_filename = gbe_fpga1_storage.conf"

if branch == '2':
    text  = "\n\n# Parameters for the FPGA2 system"
    text += "\nmonitor_port = 62003"
    text += "\nstorage_port = 62004"
    text += "\ngbe1_filename = gbe_fpga2_monitor.conf"
    text += "\ngbe2_filename = gbe_fpga2_storage.conf"

if branch == '3':
    text  = "\n\n# Parameters for the FPGA3 system"
    text += "\nmonitor_port = 63003"
    text += "\nstorage_port = 63004"
    text += "\ngbe1_filename = gbe_fpga3_monitor.conf"
    text += "\ngbe2_filename = gbe_fpga3_storage.conf"

if branch == '4':
    text  = "\n\n# Parameters for the FPGA4 system"
```



```
    text += "\nmonitor_port = 64003"
    text += "\nstorage_port = 64004"
    text += "\ngbe1_filename = gbe_fpga4_monitor.conf"
    text += "\ngbe2_filename = gbe_fpga4_storage.conf"

f.write(text)

print "\n[1] Fix 16.11"
print "[2] Fix 16.12"
print "Which data cast are you going to use [1/2]? ",
datacast = raw_input()

if datacast == '1':
    text  = "\n# Data cast 16.11"
    text += "\ndata_format = 1"
else:
    text  = "\n# Data cast 16.12"
    text += "\ndata_format = 0"
f.write(text)


text  = "\n[main_fpga]"
if branch == '1':
    text += "\nserver = beecool:61001"
else:
    if branch == '2':
        text += "\nserver = beecool:62001"
    else:
        if branch == '3':
            text += "\nserver = beecool:63001"
        else:
            if branch == '4':
                text += "\nserver = beecool:64001"
f.write(text)

print "\n"
i=0
while i<3:
    print "["+str(i+1)+"] "+rx_names[i]
    i = i + 1

print "\nWhich antennas are connected to the system [1/2/3]?",
rxnames = raw_input()
names = int(rxnames)

if names == 1:
    calib_file = "last_calib_lofar.conf"
else:
    if names == 2:
        calib_file = "last_calib_ns.conf"
    else:
        calib_file = "last_calib_ew.conf"

i=0
try:
    for line in open(calib_file,'r').readlines():
        if line[:4] == 'data':
            calib_date = line.split()[2]
```

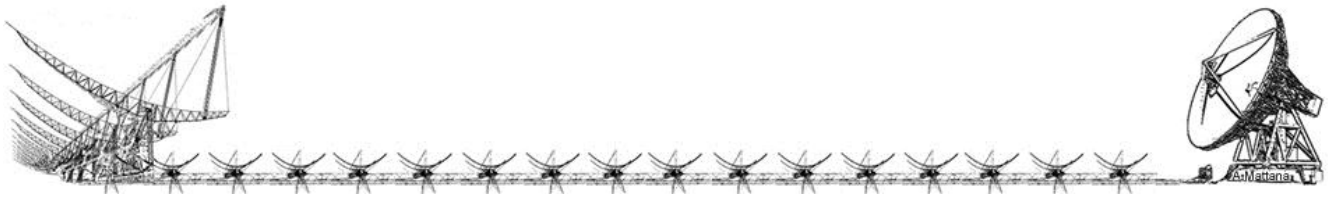
```
            calib_time = line.split()[3]
        else:
            if line[:6] == 'source':
                calib_source = line.split()[2]
            else:
                if not line[0] == '#':
                    rx_name[i][names-1] = [rx_name[i][names-1] , line.split()[2]]
                    i=i+1
except:
    pass

print "\nLastest phase calibrations have been done \non "+calib_date+" at
"+calib_time+" using radiosource "+calib_source+"\n"

confermi = 'n'
while not confermi == '0':
    if not confermi == 'n':
        print "\nThe \""+rx_name[int(confermi)-1][names-1][0]+"\" was set to
\""+rx_name[int(confermi)-1][names-1][1]+"\", type the new value: ",
        rx_name[int(confermi)-1][names-1][1] = raw_input()
        print "\nThe value for \""+rx_name[int(confermi)-1][names-1][0]+"\" has
been changed to \""+rx_name[int(confermi)-1][names-1][1]+"\"\n\n"
    i = 0
    while i<16:
        print    "["+str(i+1)+"]\t"+    rx_name[i][names-1][0]    +"    =    "+
rx_name[i][names-1][1]
        i = i+1
        if ((i>3) and (names == 3)):
            i = 16
    if rxnames == '1':
        print "\n\"none\" means antenna off"
        print "\nEdit an antenna by typing the index in the brackets\n      or
type zero [0] to confirm the configuration: ",
    else:
        if rxnames == '2':
            print "\n\"none\" means receiver off"
            print "\nEdit a receiver by typing the index in the brackets\n     or
type zero [0] to confirm the configuration: ",
        else:
            print "\n\"none\" means channel off"
            print "\nEdit a channel by typing the index in the brackets\n     or
type zero [0] to confirm the configuration: ",
    confermi = raw_input()


i = 0
text = "\n#Phase shift in degrees for each single antenna\n#none: mute antenna"
f.write(text)
text = ""
while i<16:
    text += "\nd"+str(i)+" = "+rx_name[i][names-1][1]+"    # "+rx_name[i][names-
1][0]
    i = i+1

f.write(text)

text = "\n############################################################\n"
f.write(text)
```



```
f.close()

print "\n\nThe configuration file is ready to be saved."
print "\n\nDo you want to check before to save [y/n]? ",
confermi = raw_input()
if confermi == 'y':
    for line in open("config.temp",'r').readlines():
        print line,

if branch == '1':
    print "\n\nDo you want to save as fpga1.conf [y/n]? ",
    confermi = raw_input()
    if confermi == 'y':
        os.system("mv config.temp fpga1.conf")
    else:
        print "\n\nSave aborted. You can still find these settings in the
temporary file config.temp\n"
if branch == '2':
    print "\n\nDo you want to save as fpga2.conf [y/n]? ",
    confermi = raw_input()
    if confermi == 'y':
        os.system("mv config.temp fpga2.conf")
    else:
        print "\n\nSave aborted. You can still find these settings in the
temporary file config.temp\n"
if branch == '3':
    print "\n\nDo you want to save as fpga3.conf [y/n]? ",
    confermi = raw_input()
    if confermi == 'y':
        os.system("mv config.temp fpga3.conf")
    else:
        print "\n\nSave aborted. You can still find these settings in the
temporary file config.temp\n"
if branch == '4':
    print "\n\nDo you want to save as fpga4.conf [y/n]? ",
    confermi = raw_input()
    if confermi == 'y':
        os.system("mv config.temp fpga4.conf")
    else:
        print "\n\nSave aborted. You can still find these settings in the
temporary file config.temp\n"
```

sdeb_init.py

```
#! /usr/bin/env python
"""
Script for initialising Space Debris beamformer.

eg usage: ./sdeb_init.py l configfile.conf

"""
#STD imports
import sys, os, time, math
from socket import *

#Project imports
#import lofarconf
import pack_sdeb_pars_conf
```



```
import dataconversion
import pack_sdeb_pars_systems

def get_write_func(addr, port):
    def write_func(line):
        sock = socket(AF_INET, SOCK_STREAM)
        sock.connect((addr, int(port)))
        sock.send(line + "\n")
        res = sock.recv(1024)
        sock.close()
        return res.strip()
    return write_func

def write_addr(val, addr, client):
    client("write_int addr %i"%(addr,))
    #time.sleep(0.5)
    client("write_int data %i"%(val,))
    #time.sleep(0.5)
    client("write_int cmd 1")
    #time.sleep(0.5)
    client("write_int cmd 0")
    #time.sleep(0.5)

def equalize(fpga, dec_factor, bee_dec_factor, integration, client, logger):
    conv = dataconversion.get_conversion_t(8, 7, True)
    ibob_offset = 2 ** 16
    fpga_offset = 2 ** 24
    for i,e in enumerate(fpga['equalization']):
        if not e is None:
            re = dataconversion.real2unsigned(math.cos(math.radians(e)), conv)
            im = dataconversion.real2unsigned(math.sin(math.radians(e)), conv)
        else:
            re = 0
            im = 0
        logger.info("Setting (RE_%i, IM_%i) to: (%i, %i)"%(i, i, re, im))
        if 0 <= i <= 3:
            offset = 0
        elif 4 <= i <= 7:
            offset = ibob_offset
        elif 8 <= i <= 11:
            offset = fpga_offset
        elif 12 <= i <= 15:
            offset = fpga_offset + ibob_offset
        write_addr(re, 2*(i%4) + offset, client)
        write_addr(im, 2*(i%4) + 1 + offset, client)
    logger.info("Setting decimation factor to %i"%(dec_factor,))
    write_addr(dec_factor, 8, client)
    time.sleep(0.2)
    write_addr(dec_factor, 8 + fpga_offset, client)
    time.sleep(0.2)
    write_addr(dec_factor, 8 + ibob_offset, client)
    time.sleep(0.2)
    write_addr(dec_factor, 8 + ibob_offset + fpga_offset, client)
    time.sleep(0.2)
    write_addr(bee_dec_factor, 10, client)
    time.sleep(0.2)
    write_addr(bee_dec_factor, 10 + fpga_offset, client)
    time.sleep(0.2)
```

```
    write_addr(integration, 9, client)
    time.sleep(0.2)
    write_addr(integration, 9 + fpga_offset, client)
    time.sleep(0.2)


def log_params(logger, client):
    ld0 = int(client("read_int rx_ld_cnt0"))
    dv0 = int(client("read_int rx_cnt0"))
    pck = int(client("read_int pck_cnt"))
    msg = "  ".join([str(ld0), str(dv0), str(pck)])
    logger.info(msg)

if __name__ == '__main__':
    from optparse import OptionParser
    import datetime
    import os
    import logging
    import logging.handlers

    p = OptionParser()
    p.set_usage('sdeb_init.py [options] CONFIG_FILE')
    p.set_description(__doc__)
    p.add_option('-r',   '--master_reset',   dest='reset',   action='store_false',
default=True,
                 help='If set does not send a Master Reset signal before init')
    p.add_option('-e', dest='eq', action='store_false', default=True,
                 help='If set skips the equalization process')
    p.add_option('-s', '--start_time', dest='start_time', default='now',
                 help='set the start time as dd_mm_YYYY_HH:MM:SS')
    p.add_option('-l', dest='log', action='store_true', default=False,
                 help='log data to file')

    ############ PARSING OPTIONS ########################
    opts, args = p.parse_args(sys.argv[1:])

    if args==[]:
        print 'Please specify a configuration file! \nExiting.\n'
        exit()

    start_time = opts.start_time
    if start_time == "now":
        start_time = datetime.datetime.now()
    else:
        start_time = datetime.datetime.strptime(start_time, "%d_%m_%Y_%H:%M:%S")
    log = opts.log

    start_cam = datetime.timedelta(seconds=10)
    start_delta = datetime.timedelta(seconds=1)

    #Init logging
    if log:
        if not os.path.exists("log"):
            os.makedirs("log")
        logfile = os.path.join("log", "sdeb_init.log")
        if os.path.exists(logfile):
            os.remove(logfile)
        logger = logging.getLogger("sdeb_logger")
```

```
        logger.setLevel(logging.DEBUG)
        ch = logging.StreamHandler()
        ch.setLevel(logging.DEBUG)
        ch_formatter = logging.Formatter("%(asctime)s - %(message)s")
        ch.setFormatter(ch_formatter)
        fh   =   logging.handlers.RotatingFileHandler(logfile,   maxBytes=10485760,
backupCount=5)
        fh.setLevel(logging.DEBUG)
        fh_formatter   =   logging.Formatter("%(asctime)s   -   %(levelname)s   -
%(message)s")
        fh.setFormatter(fh_formatter)
        logger.addHandler(ch)
        logger.addHandler(fh)

    ############ PARSING CONFIGURATION FILE ################
    if log:

logger.info("************************************************************")
        logger.info("****************           INITIALIZATION           PROCESS
******************")

logger.info("************************************************************")
        logger.info("parsing configuration file " + args[0])
    conf = pack_sdeb_pars_conf.parse_settings(args[0])
    fpga = conf['main_fpga']
    client = get_write_func(fpga['server']['addr'], fpga['server']['port'])

    ########## INIT SYSTEM #########
    if log:
        logger.info("Initializing the Space Debris system")
    if opts.reset:
        if log:
            logger.info("Sending Master Reset to Bee2 and IBOBs")
        client("write_int mrst 0")
        time.sleep(0.1)
        client("write_int mrst 1")
        time.sleep(0.1)
        client("write_int mrst 0")
        time.sleep(0.1)
    if opts.eq:
        if log:
            logger.info("Setting equalization on main fpga")
        equalize(fpga,         conf['dec_factor'],         conf['bee_dec_factor'],
conf['integration'], client, logger)
    client("write_int cmd 0")

    if conf['data_format'] == 0:
        logger.info("Setting Data Format to 16.12")
    else:
        logger.info("Setting Data Format to 16.11")
    client("write_int data_format "+str(conf['data_format']))
    time.sleep(0.1)

    logger.info("Setting Packet Length to "+str(conf['pck_length']))
    client("write_int pck_length "+str(conf['pck_length']))
    time.sleep(0.1)

    ipconv = str(int((conf['monitor_ip'] & 255*256*256*256) >> 24))
```

```
    ipconv += "."+str(int((conf['monitor_ip'] & 255*256*256) >> 16))
    ipconv += "."+str(int((conf['monitor_ip'] & 255*256) >> 8))
    ipconv += "."+str(int(conf['monitor_ip'] & 255))
    logger.info("Setting Monitor IP to "+ipconv+" ("+str(conf['monitor_ip'])+")")
    client("write_int monitor_ip "+str(conf['monitor_ip']))
    time.sleep(0.1)
    logger.info("Setting Monitor Port to "+str(conf['monitor_port']))
    client("write_int monitor_port "+str(conf['monitor_port']))
    time.sleep(0.1)

    ipconv = str(int((conf['storage_ip'] & 255*256*256*256) >> 24))
    ipconv += "."+str(int((conf['storage_ip'] & 255*256*256) >> 16))
    ipconv += "."+str(int((conf['storage_ip'] & 255*256) >> 8))
    ipconv += "."+str(int(conf['storage_ip'] & 255))
    logger.info("Setting Storage IP to "+ipconv+" ("+str(conf['storage_ip'])+")")
    client("write_int storage_ip "+str(conf['storage_ip']))
    time.sleep(0.1)
    logger.info("Setting Storage Port to "+str(conf['storage_port']))
    client("write_int storage_port "+str(conf['storage_port']))
    time.sleep(0.1)

    logger.info("Starting 10GbE interface: "+str(conf['gbe1_name']))
    client("start_gbe "+str(conf['gbe1_filename'])+" "+str(conf['gbe1_name']))
    logger.info("Starting 10GbE interface: "+str(conf['gbe2_name']))
    client("start_gbe "+str(conf['gbe2_filename'])+" "+str(conf['gbe2_name']))

    ######## TIME UPDATE ##########
    logger.info("Performing time update for sync...")
    while datetime.datetime.now().microsecond > 100000:
        time.sleep(0.1)
    t_now=datetime.datetime.utcnow()

    t_zero  = datetime.datetime.strptime("1/1/1970 00:00:00","%d/%m/%Y %H:%M:%S")
    t_delta = t_now - t_zero
    t_stamp_now   = t_delta.days*(60*60*24) + t_delta.seconds
    logger.info("Sending:    "+datetime.datetime.strftime(t_now,        "%Y/%m/%d
%H:%M:%S")+" UT (timestamp: "+str(t_stamp_now)+")")
    client("write_int time_update "+str(t_stamp_now))
    time.sleep(0.1)

    while datetime.datetime.now().microsecond > 100000:
        time.sleep(0.1)
    client("write_int cmd 4") # cmd Read Ibob Localtime (UT)
    time.sleep(0.1)
    client("write_int cmd 0")
    iboblocaltime = int(client("read_int last_rx_hidata"))
    logger.info("Received:
"+datetime.datetime.strftime(datetime.datetime.utcfromtimestamp(iboblocaltime),
"%Y/%m/%d %H:%M:%S")+" UT (timestamp: "+str(iboblocaltime)+")")
    if iboblocaltime == (t_stamp_now+1):
        logger.info("Time updated successfully")
    else:
        logger.info("ERROR: Time NOT updated!!!")

    #logger.info("logging order:")
    #logger.info("ld0 dv0 pck_cnt")
    #ld0 = int(client("read_int rx_ld_cnt0"))
    #dv0 = int(client("read_int rx_cnt0"))
```

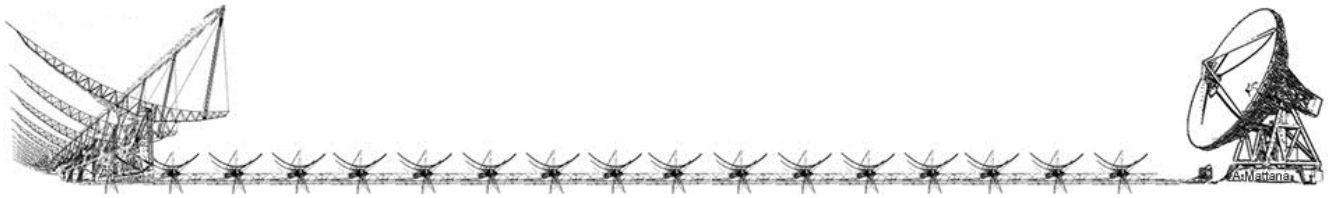
```
    #pck = int(client("read_int pck_cnt"))
    #msg = "  ".join([str(ld0), str(dv0), str(pck)])
    #logger.info(msg)

    print "\nInitialization Process Successfully Completed!\n"
```

sched/3C123_fpga1.conf

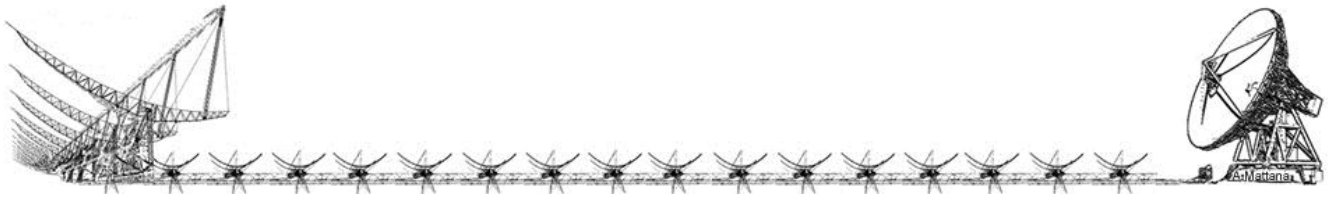
```
[Obs]
System = fpga1
Start_time = 2012/11/01_00:50:00
Stop_time = 2012/11/01_01:30:00
Target = 3C123_1RX
```

sdeb_run.py

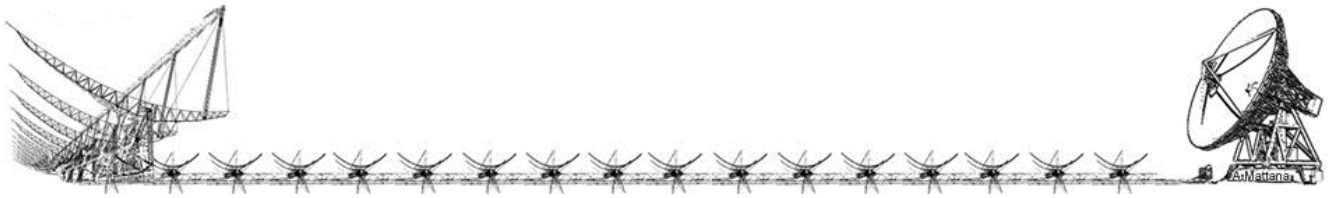
```
#! /usr/bin/env python
"""
Script to load the observation parameters.

eg usage: ./sdeb_run.py -l observable.conf

"""
#STD imports
import sys, os, time, math
from socket import *

#Project imports

import pack_sdeb_pars_obs
import dataconversion
import pack_sdeb_pars_systems

def get_write_func(addr, port):
    def write_func(line):
        sock = socket(AF_INET, SOCK_STREAM)
        sock.connect((addr, int(port)))
        sock.send(line + "\n")
        res = sock.recv(1024)
        sock.close()
        return res.strip()
    return write_func


if __name__ == '__main__':
    from optparse import OptionParser
    import datetime
    import os
    import logging
    import logging.handlers

    p = OptionParser()
    p.set_usage('sdeb_run.py [options] CONFIG_FILE')
    p.set_description(__doc__)
    p.add_option('-l', dest='log', action='store_true', default=False,
```



```
                help='log data to file')

    ########### PARSING OPTIONS ######################
    opts, args = p.parse_args(sys.argv[1:])
    log = opts.log

    if args==[]:
        print 'Please specify a configuration file! \nExiting.\n'
        exit()


    #Init logging
    if log:
        if not os.path.exists("log"):
            os.makedirs("log")
        logfile = os.path.join("log", "sdeb_run.log")
        if os.path.exists(logfile):
            os.remove(logfile)
        logger = logging.getLogger("sdeb_logger")
        logger.setLevel(logging.DEBUG)
        ch = logging.StreamHandler()
        ch.setLevel(logging.DEBUG)
        ch_formatter = logging.Formatter("%(asctime)s - %(message)s")
        ch.setFormatter(ch_formatter)
        fh   =   logging.handlers.RotatingFileHandler(logfile,   maxBytes=10485760,
backupCount=5)
        fh.setLevel(logging.DEBUG)
        fh_formatter   =   logging.Formatter("%(asctime)s   -   %(levelname)s   -
%(message)s")
        fh.setFormatter(fh_formatter)
        logger.addHandler(ch)
        logger.addHandler(fh)

    ########### PARSING CONFIGURATION FILE ################
    if log:

logger.info("************************************************************")
        logger.info("******************      ARM      A      NEW      OBSERVETION
*****************")

logger.info("************************************************************")
        logger.info("Parsing configuration file " + args[0])
    conf = pack_sdeb_pars_obs.parse_settings(args[0])
    sysconf                                                                    =
pack_sdeb_pars_systems.parse_settings("systems.conf",conf['System'])

    #client = get_write_func(conf['ip_addr'], conf['ip_port'])
    #beesys = sysconf['server']
    client = get_write_func(sysconf['server']['addr'], sysconf['server']['port'])

    if log:
        logger.info("Loading observation parameters...")

    t_start                                                                    =
datetime.datetime.strptime(conf['Start_time'],"%Y/%m/%d_%H:%M:%S")
    t_stop                                                                     =
datetime.datetime.strptime(conf['Stop_time'],"%Y/%m/%d_%H:%M:%S")
    target_name = conf['Target']
```



```
    t_nok = 0

    if t_start < datetime.datetime.utcnow():
        t_nok = 1
        logger.info("*** ERROR: Start time "+datetime.datetime.strftime(t_start,
"%Y/%m/%d %H:%M:%S")+" UT must be greater ")
        logger.info("***                                                    than     now
("+datetime.datetime.strftime(datetime.datetime.utcnow(),  "%Y/%m/%d  %H:%M:%S")+"
UT)")

    if t_nok == 0 and t_stop < datetime.datetime.utcnow():
        t_nok = 1
        logger.info("***  ERROR:  Stop  time  "+datetime.datetime.strftime(t_stop,
"%Y/%m/%d %H:%M:%S")+" UT must be greater")
        logger.info("***                                                  than     now
("+datetime.datetime.strftime(datetime.datetime.utcnow(),  "%Y/%m/%d  %H:%M:%S")+"
UT)")

    if t_nok == 0 and t_stop < t_start:
        t_nok = 1
        logger.info("***  ERROR:  Stop  time  "+datetime.datetime.strftime(t_stop,
"%Y/%m/%d %H:%M:%S")+" UT must be greater ")
        logger.info("***   than  Start  time  "+datetime.datetime.strftime(t_start,
"%Y/%m/%d %H:%M:%S")+" UT")

    if not t_nok:
        t_zero      =   datetime.datetime.strptime("1/1/1970   00:00:00","%d/%m/%Y
%H:%M:%S")
        d_start = t_start - t_zero
        start   = d_start.days*(60*60*24) + d_start.seconds
        d_stop = t_stop - t_zero
        stop   = d_stop.days*(60*60*24) + d_stop.seconds
        t_total = d_stop - d_start
        t_totsec = t_total.days*(60*60*24) + t_total.seconds

        client("write_int start_time "+str(start))
        logger.info("Loaded                         START                         time
"+datetime.datetime.strftime(datetime.datetime.utcfromtimestamp(start),  "%Y/%m/%d
%H:%M:%S")+" UT (timestamp: "+str(start)+")")
        client("write_int stop_time "+str(stop))
        logger.info("Loaded                          STOP                         time
"+datetime.datetime.strftime(datetime.datetime.utcfromtimestamp(stop),   "%Y/%m/%d
%H:%M:%S")+" UT (timestamp: "+str(stop)+")")

        # ARMING IBOB
        client("write_int cmd 0")
        time.sleep(0.1)
        client("write_int cmd 2")
        time.sleep(0.1)
        client("write_int cmd 0")
        time.sleep(0.1)
        while     (int(client("read_int     last_rx_lodata"))     ==     0)     and
(int(client("read_int last_rx_oob")) == 2):
            client("write_int cmd 0")
            time.sleep(0.1)
            client("write_int cmd 2")
            time.sleep(0.1)
            client("write_int cmd 0")
```



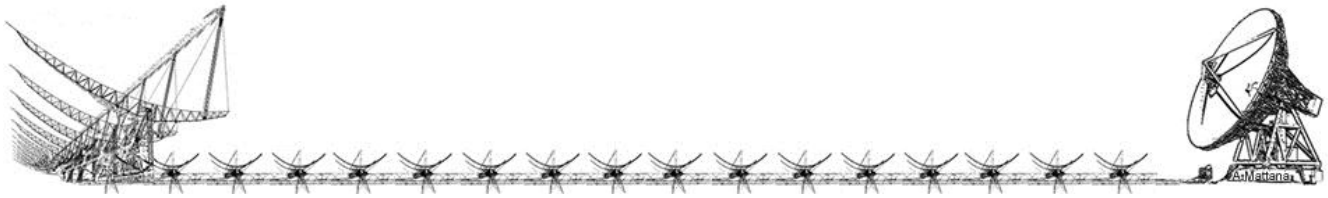
```
        logger.info("System "+conf['System']+" armed!")
        #d_volume = int(int(client("read_int integration")) * t_totsec * 4)
        d_volume = int(30000000./299 * t_totsec * 4)
        unit = ['bytes', 'KB', 'MB', 'GB']
        n_unit = 0

        d_volume /= ((int(client("read_int pck_length"))-1) * 8)
        d_volume = int(((int(client("read_int pck_length"))-1) * 8)*d_volume)

        volume=d_volume
        while n_unit <= 3 and volume > 1024:
            volume /= 1024
            n_unit += 1

        logger.info("Data expected: about "+str(int(volume))+" "+unit[n_unit]+"
("+str(d_volume)+" bytes)")

        logger.info("Starting            Recording            data            on
192.167.189.66:"+str(int(sysconf['server']['port'])+4))
        storage_client             =             get_write_func("192.167.189.66",
str(int(sysconf['server']['port'])+4))
        storage_client("record   "+str(start)+"   "+target_name+"   "+str(stop)+"
"+str(d_volume))


        logger.info("Observation \""+target_name+"\" Loaded Successfully!\n")


    else:
        logger.info("Observation \""+target_name+"\" not loaded!!!")
        logger.info("ARM aborted!\n")
```

last_calib_ew.conf

```
data = 14/11/2012 00:00:00
source = 3C123
d0  = +154.70863  # 4E
d1  = -1.2245701  # 5E
d2  = +00.0  # 2E
d3  = +55.895791  # 3E
d4  = none  #
d5  = none  #
d6  = none  #
d7  = none  #
d8  = none  #
d9  = none  #
d10 = none  #
d11 = none  #
d12 = none  #
d13 = none  #
d14 = none  #
d15 = none  #
```

systems.conf

```
[Systems]
fpga1 = beecool:61001
```



```
fpga2 = beecool:62001
fpga3 = beecool:63001
fpga4 = beecool:64001
```

fpga1.conf

```
#####################################################
#
# Configuration file automatically generated by using
# the Wizard on 15/11/2012 15:09:48 UT

# NO WHITESPACE ALLOWED BETWEEN TEXT DELIMITERS!

# design global parameters
[global]
dec_factor = 13
bee_dec_factor = 23

# integration for on board data accumulation
integration = 100334

# Common network parameters
monitor_ip = 3232238346
storage_ip = 3232238347
pck_length = 160
gbe1_name = monitor
gbe2_name = storage

# Parameters for the FPGA1 system
monitor_port = 61003
storage_port = 61004
gbe1_filename = gbe_fpga1_monitor.conf
gbe2_filename = gbe_fpga1_storage.conf
# Data cast 16.11
data_format = 1
[main_fpga]
server = beecool:61001
#Phase shift in degrees for each single antenna
#none: mute antenna
d0 = none   # 4E
d1 = none   # 5E
d2 = +00.0   # 2E
d3 = +55.895791   # 3E
d4 = none   #
d5 = none   #
d6 = none   #
d7 = none   #
d8 = none   #
d9 = none   #
d10 = none   #
d11 = none   #
d12 = none   #
d13 = none   #
d14 = none   #
d15 = none   #
#########################################################
```

Storage

README.txt

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

1. First of all connect via ssh or open a terminal window
   on 192.167.189.66 (batman), login as oper

  *  Last login: Sat Nov 10 14:03:31 2012 from bee2desktop
  *  Have a lot of fun...

2. Storage scripts are in "/media/data/sdeb"

  *  oper@batman:~> cd /media/data/sdeb

3. Depending on the branch launch 'fpga1/2/3/4' recorder server

  *  oper@batman:/media/data/sdeb> ./fpga1_recorder_server.py
  *  server listening on port: 64005

4. Server is listening, when observing you should see this:

  *  Command received: record 1352725860 scan_hp 1352725920 24080232
  *
  *  ################################################
  *
  *     Executing:  ./record_fpga1.py  -o  scan_hp  -s  2012/11/12_13:11:00  -t
2012/11/12_13:12:00 -e 24080232
  *
  *  ################################################
  *
  *  317_14:10:28 - INFO: Running with options:
  *  317_14:10:28 - INFO: port: 64004
  *  317_14:10:28 - INFO: pkg length: 8000
  *  317_14:10:28 - INFO: fmt: >Q
  *  317_14:10:28 - INFO: target name: scan_hp
  *  317_14:10:28 - INFO: start time: 2012/11/12_13:11:00
  *  317_14:10:28 - INFO: stop time: 2012/11/12_13:12:00
  *  317_14:10:28 - INFO: output filename: data/20121112_131100_EW_scan_hp.dat
  *  317_14:10:59 - INFO: server listening
  *  317_14:11:00 - INFO: recording...
  *  317_14:12:00 - INFO: closing communication
  *  317_14:12:00 - INFO: received up to package: 18931
  *  317_14:12:00 - INFO: closing recorder
  *
  *  ################################################
```

fpga1_recorder_server.py

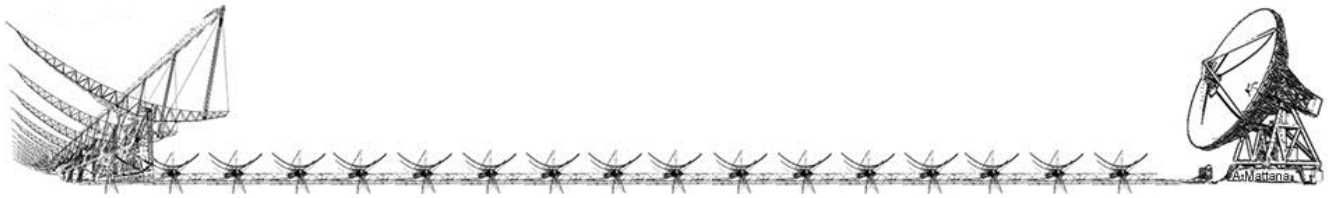
```
#!/usr/bin/env python

import SocketServer
```

```
import socket
import struct
import datetime
import time
import logging
import logging.handlers
import os

class CommandException(Exception):
    def __init__(self, msg):
        Exception.__init__(self, msg)

class SdebTCPHandler(SocketServer.StreamRequestHandler):
    def handle(self):
        self.data = self.rfile.readline().strip()
        print "Command received: " + self.data
        args = self.data.split()
        try:
            res = self.server.execute(args)
            self.wfile.write(str(res))
        except CommandException, ce:
            self.wfile.write(ce.args[0])

class SdebTCPServer(SocketServer.TCPServer):
    def __init__(self, addr):
        SocketServer.TCPServer.__init__(self, addr, SdebTCPHandler)
        self.rec = 0
        #self.base_path = "/proc/%s/hw/ioreg/"%(pid,)
        #self.devs = os.listdir(self.base_path)
        self.commands = {
                         "record": self.record,
                         "abort": self.abort,
                        }
    def record(self, t_start, target, t_stop, expsize):
        self.rec = 1
        print "\n##################################################\n"
        cmd       =       "./record_fpga1.py       -o       "+target+"       -s
"+datetime.datetime.strftime(datetime.datetime.utcfromtimestamp(int(t_start)),
"%Y/%m/%d_%H:%M:%S")+"                                                    -t
"+datetime.datetime.strftime(datetime.datetime.utcfromtimestamp(int(t_stop)),
"%Y/%m/%d_%H:%M:%S")
        cmd += " -e "
        cmd += expsize
        print "Executing: "+cmd
        #a = os.system(cmd+"&")
        os.system(cmd+"&")
        #os.system(cmd)
        print "\n##################################################\n"
        #return a

    def abort(self, target):
        print "End of recording: "+target
        self.rec = 0
        return 0

    def execute(self, args):
        if not self.commands.has_key(args[0]):
            raise CommandException("Command %s not found."%(args[0],))
```

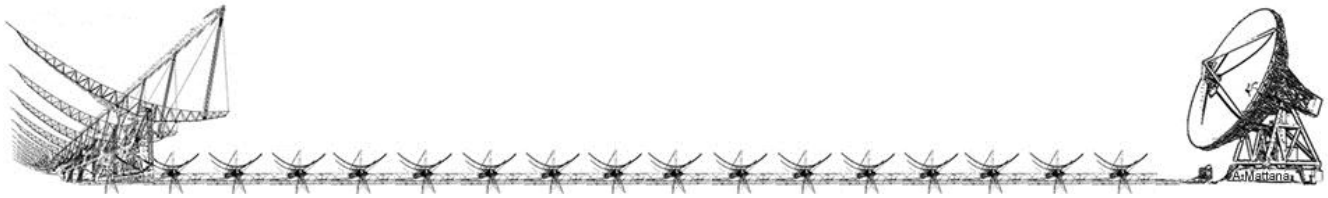
```
        if len(args) > 1:
            try:
                res = self.commands[args[0]](*(args[1:]))
            except TypeError, te:
                raise CommandException(te.args[0])
        else:
            res = self.commands[args[0]]()
        return res


if __name__=="__main__":
    from optparse import OptionParser
    import sys


    #server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server = SdebTCPServer(("", 61005))
    try:
        print "server listening on port: " + str(61005)
        server.serve_forever()
    except KeyboardInterrupt:
        print "closing communication"
        del(server)
        sys.exit(0)
```

record_fpga1.py

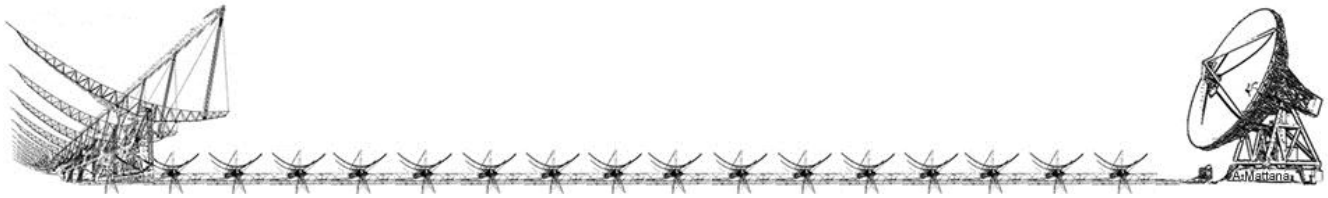
```
#!/usr/bin/env python

import SocketServer
import socket
import struct
import datetime
import time
import logging
import logging.handlers

#Setting up logging
log_filename = "log/dataserver.log"
logger = logging.getLogger('DataLogger')
logger.setLevel(logging.INFO)
console_log = logging.StreamHandler()
file_log  =  logging.handlers.RotatingFileHandler(log_filename,  maxBytes=8388608,
backupCount=5)
formatter   =   logging.Formatter("%(asctime)s   -   %(levelname)s:   %(message)s",
"%j_%H:%M:%S")
console_log.setFormatter(formatter)
file_log.setFormatter(formatter)
logger.addHandler(console_log)
logger.addHandler(file_log)

def unscram_data(data, l):
    data_len = len(data)
    for i in xrange(data_len // l):
        yield data[i*l:i*l+l]

class LofarDataUDPHandler(SocketServer.DatagramRequestHandler):
    def handle(self):
```



```
        while server.acquiring:
            buf = self.rfile.read(self.server.pkg_len)
            self.handle_pkg(buf)

    def handle_pkg(self, pkg):
        count = struct.unpack(self.server.fmt, pkg[:8])[0]
        if count != self.server.count + 1:
            logger.error("jumping from packet " + str(self.server.count) + " to "
+ str(count))
        self.server.count = count
        self.server.outfile.write(pkg[8:])

class LofarDataUDPServer(SocketServer.UDPServer):
    def __init__(self, addr=("localhost", 9999), fmt=">L",
outfilename="data/lofar.dat", pkg_len=20):
        self.outfile = open(outfilename, "wb")
        self.fmt = fmt
        self.pkg_len = pkg_len
        self.count = 0
        self.acquiring = True
        SocketServer.UDPServer.__init__(self, addr, LofarDataUDPHandler)

if __name__=="__main__":
    from optparse import OptionParser
    import sys

    #command line parsing
    op = OptionParser()
    op.add_option("-p", "--port", type="int", dest="port", default=61004)
    op.add_option("-f", "--fmt", dest="fmt", default=">Q")
    op.add_option("-o", "--outfile", dest="outfile", default="", help="target
name")
    op.add_option("-e", "--expsize", type="int", dest="expsize", default=0,
help="expected size in bytes")
    op.add_option("-k", "--pkg_len", type="int", dest="pkg_len", default=1000,
help="gets multiplied by 8")
    op.add_option("-s", "--start_time", dest="start", default="now",
help="YYYY/MM/DD_HH:MM:SS")
    op.add_option("-t", "--stop_time", dest="stop", default="never",
help="YYYY/MM/DD_HH:MM:SS")
    op.add_option("-n", "--no_record", dest="no_rec", action="store_true",
default=False, help="don\'t write data to disk")
    opts, args = op.parse_args(sys.argv[1:])
    port = opts.port
    fmt = opts.fmt
    pkg_len = opts.pkg_len * 8
    start = opts.start
    stop = opts.stop
    rec = not opts.no_rec
    expectedsize = opts.expsize
    outfile = opts.outfile

    #file_log.doRollover()
    logger.info("Running with options:")
    logger.info("port: " + str(port))
    logger.info("pkg length: " + str(pkg_len))
    logger.info("fmt: " + str(fmt))
    logger.info("target name: " + outfile)
```



```
    logger.info("start time: " + start)
    logger.info("stop time: " + stop)
    #logger.info("recording: " + str(rec))
    if outfile != "":
        outfile = "_" + outfile

    t_zero  = datetime.datetime.strptime("1/1/1970 00:00:00","%d/%m/%Y %H:%M:%S")

    if start == "now":
        start_time = datetime.datetime.utcnow()
    else:
        start_time = datetime.datetime.strptime(start, "%Y/%m/%d_%H:%M:%S")
        d_start = start_time - t_zero
        lstart     =  datetime.datetime.utcfromtimestamp(d_start.days*(60*60*24)  +
d_start.seconds -1)

    if stop == "never":
        delta = datetime.timedelta(days=365)
        stop_time = start_time + delta
    else:
        stop_time = datetime.datetime.strptime(stop, "%Y/%m/%d_%H:%M:%S")
        d_stop = stop_time - t_zero
        lstop      =  datetime.datetime.utcfromtimestamp(d_stop.days*(60*60*24)  +
d_stop.seconds +1)

    #logger.info("start time converted to: " + str(start))
    #logger.info("stop time converted to:  " + str(stop))
    #server  =  LofarDataUDPServer(addr=("",  port),  outfilename=outfile,  fmt=fmt,
pkg_len=pkg_len)
    server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    if rec:

outfilename="data/"+start_time.strftime("%Y%m%d_%H%M%S")+"_NS"+outfile+".dat"
        logger.info("output filename: "+ outfilename)
        out = open(outfilename, "wb")
    count = -1

    while datetime.datetime.utcnow() < lstart:
        time.sleep(0.3)

    logger.info("server listening")
    server.bind(("192.168.11.11", port))

    while lstop > datetime.datetime.utcnow() and out.tell() < expectedsize:
        #print expectedsize, out.tell(),
        #if out.tell() < expectedsize:
        #print "...rileggo"
        buf = server.recv(pkg_len)
        new_count = struct.unpack(fmt, buf[:8])[0]
        if not new_count == count + 1:
            logger.error("jumping from " + str(count) + " to " + str(new_count))
        if new_count == 0:
            logger.info("recording...")
        count = new_count
        out.write(buf[8:])
        out.flush()
    logger.info("closing communication")
```



```
    logger.info("received up to package: " + str(count+1))
    out.close()
    logger.info("closing recorder")
    print "\n##################################################\n"
```

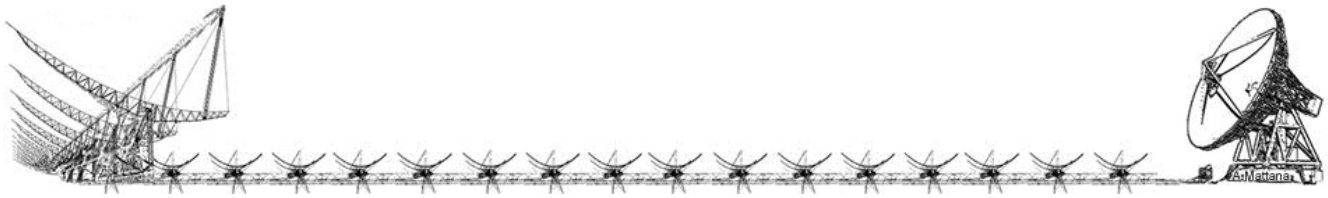
Monitor

README.txt

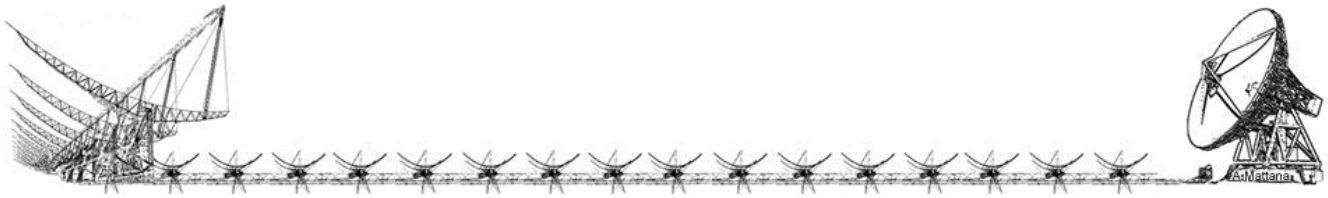
```
Plot observation in realtime

Example of usage:
oper@bee2:~/andrea/bin$ python realtimespectra.py -c 1024 -i 100 -p 61003

Help for parameters allowed:
oper@bee2:~/andrea/bin$ python realtimespectra.py --help
Usage: realtimespectra.py [options]

Options:
  -h, --help            show this help message and exit
  -p PORT, --port=PORT
  -k PKG_LEN, --pkg_len=PKG_LEN
                        package length expressed in 64b
  -c FFTSIZE, --fftsize=FFTSIZE
                        number of fft channels
  -i INTEGR, --intgr_time=INTEGR
                        number of integrations
  -w WINDOW, --window=WINDOW
                        type of window, default=no window, possible value:
                        hamming, hanning, bartlett, kaiser(default shape 10%)
```

realtimespectra.py

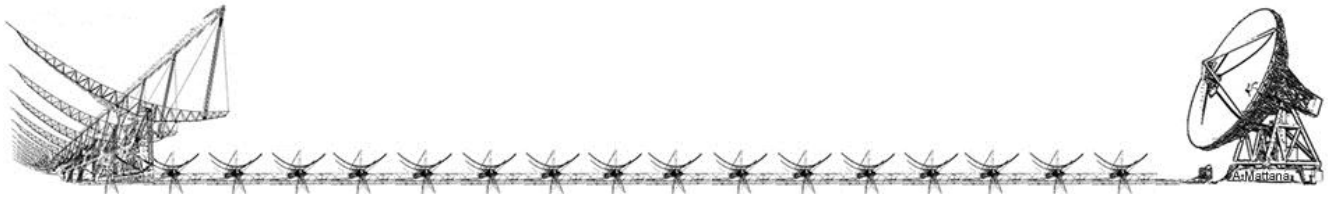
```
#!/usr/bin/env python
# ensure sysctl -w net.core.rmem_max=8388608 is set

import socket
import struct
import datetime
import time

import numpy,sys
import matplotlib
matplotlib.use('TkAgg')
from matplotlib import pylab


cnt = 0

def exit_fail():
    print 'ERROR: '
    try:
        print 'Programma terminato'
```



```
        except: pass
        raise
        exit()

    def exit_clean():
        try:
            print 'Programma terminato'
        except: pass
        exit()

    def safe_recv(socket, size):
        global cnt
        msg = ''
        while len(msg) < (size):
            chunk = socket.recv(size-len(msg)+8);
            if chunk == '':
                raise RuntimeError, "socket connection broken"
            msg = msg + chunk[8:]
        return msg


    def cplxswaporder(n):
        a = numpy.zeros(len(n)/2,'complex')
        i = 0
        q = 0
        while q < len(n):
            #a[i]=complex(n[q+2],n[q+3])
            #a[i+1]=complex(n[q],n[q+1])
            a[i]=complex(n[q],n[q+1])
            a[i+1]=complex(n[q+2],n[q+3])
            i = i+2
            q = q+4
        return a

    def applywindow(data,window):
        if window == 'hamming':
            data *= numpy.hamming(len(data))
        elif window == 'hanning':
            data *= numpy.hanning(len(data))
        elif window == 'bartlett':
            data *= numpy.bartlett(len(data))
        elif window == 'kaiser':
            data *= numpy.kaiser(len(data),len(data)/10)
        return data

    # Read header (64bit) and
    # the first 1024 pair re-im
    # of samples (blocking, re 16b, im 16b)
    def get_spectrum(socket,pkg_len,fftsize):
        #rawdatareal = numpy.zeros(fftsize,dtype=numpy.float64)
        buf = safe_recv(socket,fftsize*4)
        fmt='<'+str(fftsize*2)+'h'
        vettore = struct.unpack(fmt, buf)
        rawdata = cplxswaporder(vettore)
        if window != '':
            rawdata = applywindow(rawdata,window)
        fftdata = numpy.fft.fft(rawdata)
        #rawdatareal = numpy.array([c.real ** 2 + c.imag ** 2 for c in rawdata])
```

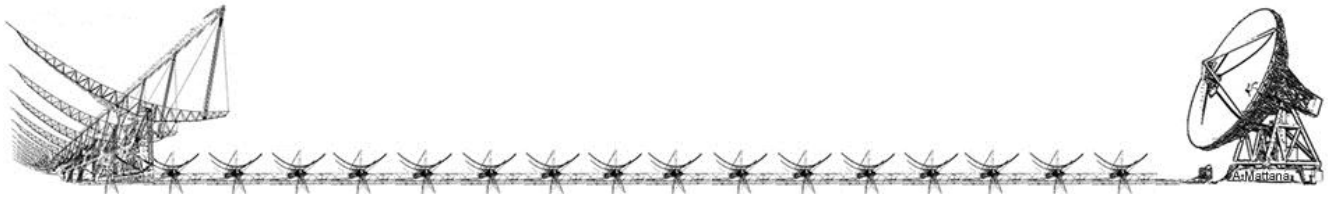


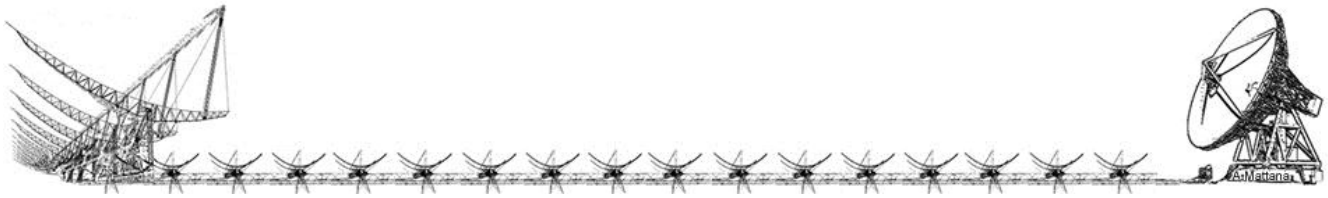
```
    #fftdata = numpy.fft.fft(rawdatareal)
    fftdata = numpy.array([c.real ** 2 + c.imag ** 2 for c in fftdata])
    #fftdata = numpy.fft.helper.fftshift(fftdata) #Frequency reordering, funziona
anche con fft complesse?!
    #fftdata = 10*numpy.log10(fftdata)
    return fftdata



if __name__=="__main__":
    from optparse import OptionParser
    import sys

    #command line parsing
    op = OptionParser()
    op.add_option("-p", "--port", type="int", dest="port", default=64003)
    #op.add_option("-f", "--fmt", dest="fmt", default="<Q")
    op.add_option("-k",  "--pkg_len",  type="int",  dest="pkg_len",  default=20,
help="package length expressed in 64b")
    op.add_option("-c",  "--fftsize",  type="int",  dest="fftsize",  default=1024,
help="number of fft channels")
    op.add_option("-i",  "--intgr_time",  type="int",  dest="integr",  default=100,
help="number of integrations")
    op.add_option("-w",  "--window",  dest="window",  default='',  help="type  of
window,  default=no  window,  possible  value:  hamming,  hanning,  bartlett,
kaiser(default shape 10%)")
    #op.add_option("-t",     "--stop_time",     dest="stop",     default="never",
help="HH:MM:SS_dd/mm/yyyy")
    opts, args = op.parse_args(sys.argv[1:])
    instrument_port = opts.port
#   fmt = opts.fmt
    pkg_len = opts.pkg_len*8
    fftsize = opts.fftsize
    integration = opts.integr
    window = opts.window

try:
    pylab.ion()

    #UDP Socket stuff
    instrument_addr   = '192.168.11.10'; #'127.0.0.1'
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM);
    s.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, 1048576*8)
    print 'Listening on port ',instrument_port, ' (',instrument_addr,')'
    s.bind((instrument_addr, instrument_port));

    try:
        print 'Plot started'
        pylab.figure()

        if instrument_port == 64003:
            rtelescope = 'EW'
        if instrument_port == 61003:
            rtelescope = 'NS'

        while 1:
            #continuous_acq()
            i=0
```



```
            spettri = numpy.zeros(fftsize,dtype=numpy.float64)
            while i < integration:
                spettro = get_spectrum(s,pkg_len,fftsize)
                spettri += spettro
                i += 1
            matplotlib.pyplot.clf()

            intspettro                                                    =
numpy.concatenate((spettri[fftsize/2:],spettri[:fftsize/2]))

            pylab.semilogy(intspettro)
            pylab.title(rtelescope+' - FFT (integration of '+str(i)+ ' spectra)
'+time.strftime('%d/%m/%Y %X'))
            pylab.ylabel('Power (arbitrary units)')

            pylab.grid()
            pylab.xlabel('Channel')
            pylab.xlim(0,len(intspettro))
            pylab.draw()

    except:
        print '\n'

except KeyboardInterrupt:
    exit_clean()
except:
    exit_fail()

exit_clean()
```

Index of Figures





Index of Tables

Acronyms

ADC	Analog to Digital Converter
CASPER	Collaboration for Astronomy Signal Processing and Electronics Research
CW	Continuous Wave
DDC	Digital Down Converter
DDS	Direct Digital Synthesizer
EDK	Embedded Development Kit
E/W	East West arm of the Northern Cross radiotelescope
FFT	Fast Fourier Transform
FIR	Finite Input Response
FPGA	Field Programmable Gate Array
IDE	Integrated Development Environment
IDL	Interactive Data Language
IF	Intermediate Frequency
IP	Internet Protocol
LEO	Low Earth Orbit
LOFAR	Low Frequency Array
MAC	Media Access Control
MSB	Most Significant Bit
N/S	North South arm of the Northern Cross radiotelescope
PFB	Poliphase Filter Bank
PLL	Phase Locked Loop
RF	Radio Frequency
ROACH	Reconfigurable Open Architecture Computing Hardware
SNR	Signal to Noise Ratio
TLE	Two Line Elements
USSTRATCOM	United States Strategic Command