A Digital Backend for the Medicina Array Demonstrator

A. Mattana

IRA 508/17

Istituto di Radio Astronomia, Bologna, INAF

Referee: Marco Poloni

Index

INDEX **3**

INTRODUCTION **5**

HARDWARE **6**

System Requirements 6
Real Time Power Computing 7
ADC 8

FIRMWARE **10**

Digitalization 10
Synchronization 11
Frequency channels 12
Amplitude Equalization 13
Instrumental Phase Calibration 14
The 64 bit system 16

CONTROL SOFTWARE **28**

System Start Up 28
Check Antenna Power 32
Load Coefficients 34
Plot Instantaneous Spectra 36
Save Raw Data 37
Plot Raw Data 37

UDP PACKETS DATA FORMAT **40**

The Data Set 40

MAPPING ANTENNA/RX/ADC **43**

Firmware Antenna Map 43
BEST Receivers Map 43
MAD Receivers Map 44
Network Receivers Map 44

ACRONYMS **45**

Page intentionally left blank

Introduction

The Medicina Array Demonstrator Project (MAD) consists in one of the first SKA precursor. It is a 3x3 regular Vivaldi dual polarization antenna array. The aim of the project is to measure the array beam pattern by using an artificial radio source, a sinewave transmitter (TX) installed on a remote controlled hexacopter (UAV) flying over the array. The MAD backend acquires the signal received from the antennas and provides in output preprocessed data. These are the course frequency bin containing the TX signal. The rest of the frequency bins are just throw to save data rate.

The MAD project and its results are described in details on other the technical reports:

- *The 2nd measurement campaign of the Medicina Array Demonstrator*, G. Pupillo, G. Naldi, A. Mattana, J. Monari, F. Perini, M. Schiaffino, P. Bolli, G. Virone, A. Lingua, IRA 479/14

- *Medicina Array Demonstrator: Overview and Results of the third campaign,* G. Pupillo, G. Naldi, A. Mattana, J. Monari, M. Poloni, F. Perini, M. Schiaffino, G. Bianchi, P. Bolli, A. Lingua, I. Aicardi, H. Bendea, P. Maschio, M. Piras, G. Virone, F. Paonessa, Z. Farooqui, R. Tascone, A. Tibaldi, IRA 482/14

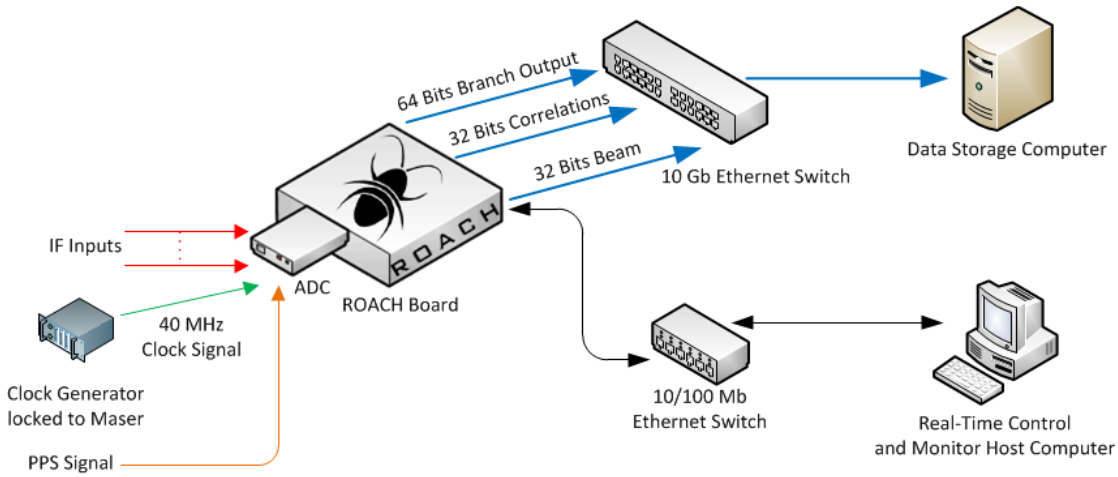
The MAD Project ended in late 2014. The Backend Firmware and all the software have been maintained afterwards to be able to complete some scientific test such the phase closure measurements done many months later.

I started writing this document in 2015 but I finished writing only in 2017 because of the numerous work commitments.

Hardware

System Requirements

The digital backend for MAD experiment (Medicina Array Demonstrator) has been developed using the ROACH-1 CASPER board (**R**econfigurable **O**pen **A**rchitecture **C**omputing **H**ardware, see IRA 462/12 and https://casper.berkeley.edu/wiki/Hardware) which is a VIRTEX 5 FPGA populated of many peripherals such 4x CX4 10Gbps high-speed serial connectors, 10/100Mbit RJ45 Ethernet, DDR2 DRAM DIMM, 2x 2M x 18-bit QDRII+ SRAMs and much more.



The overall backend architecture is represented above. The IF analog signals arriving from the MAD optical receivers are sampled by the ROACH's ADC, then an onboard processing logic computes the FFT and makes auto correlations and cross correlation products of the interested frequency channel and the beam formed. These data arrives to the storage system via high speed 10 Gbit/s links, while the management is driven by a separate Ethernet link.

The clock generator as well as the computers (through a NTP server) are synchronized using a ultra-stable 10 MHz reference sine wave and PPS signal provided by the MASER Atomic Clock.

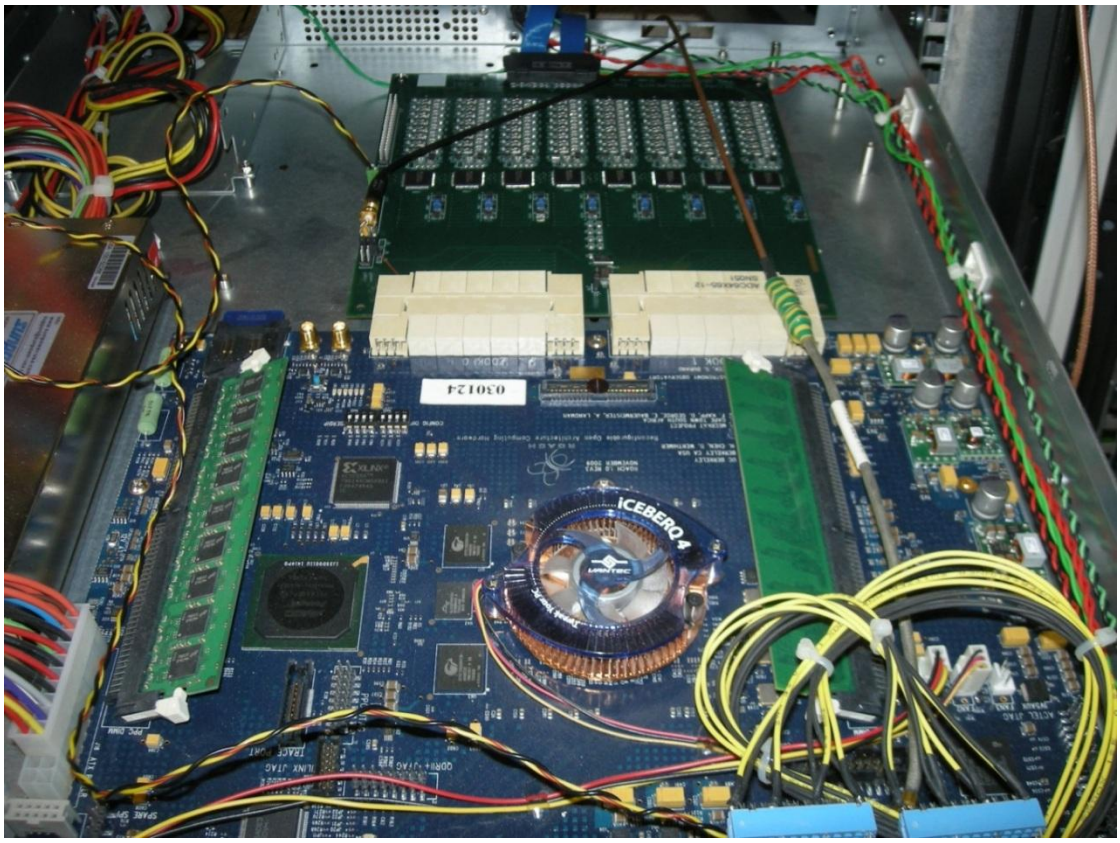
The management computer is a NFS server for the ROACH 1 that expects to boot its operating system via network, it is a UBUNTU LINUX OS 12.04 equipped with 2 networks card adapter, one for the public network and the other one for the ROACHs private network. It provides also IP numbers as DHCP for the ROACHs. The communication with the ROACH has been developed with the PYTHON programming language using a set of Python modules (primitive) provided by the CASPER community that allows to interact with the ROACH FPGA internal registers.

The storage system is a workstation equipped with a set of disk configured in RAID-0.

The Clock generator is the Rohde & Schwarz SMX Signal Generator 100kHz-1000MHz.

The 10 Gbit Ethernet Switch is the Fujitsu XG 700 with 12 CX4 ports.

Real Time Power Computing



The aim of the Digital Backend is to acquire analog signals coming from the Vivaldi antenna array on the field and to produce in output the result of the Beamformer and the FX-Correlator computed aboard. Two independent parts of the firmware will produce output of 32 bit and 64 bit to allow the study of some quantization effect over the digital chains. The firmware has been designed starting by an already tested part which is a F-engine working as follow:

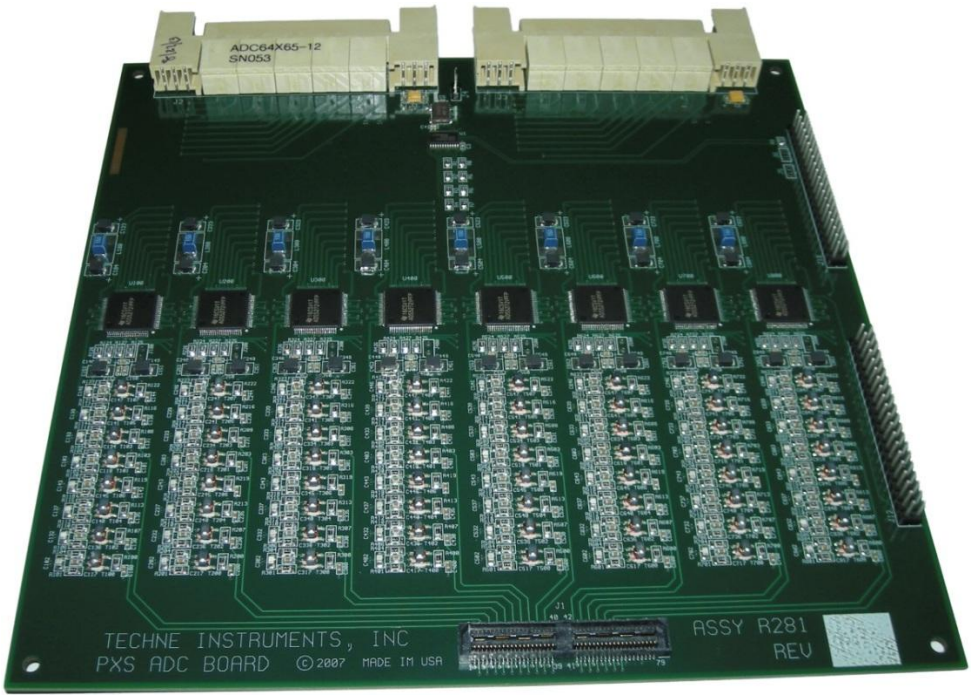
1. The 64ADCx64-12 board (https://casper.berkeley.edu/wiki/64ADCx64-12) samples 64 analog signals at the same time and gives the 12 bits values to the FPGA in 4 clock cycles (16 per clock cycles in serial).
2. The FPGA, that is at least 4 times faster than the ADC clock rate, takes 32 time domain antenna samples and implements a digital Polyphase Filters Bank and the FFT to produce frequency channels.
3. Each frequency channel can be equalized in amplitude with a gain factor and calibrated in phase.

Every computations of each stage (PFB, FFT, AMP, PHASE)* produce values that need to be represented with a small number of bits to save FPGA resources. There are two independent branches of the same project taking these values: a 32 bit system customized for the MAD case (9 dual pol antennas), and, a 64 bit programmable system.

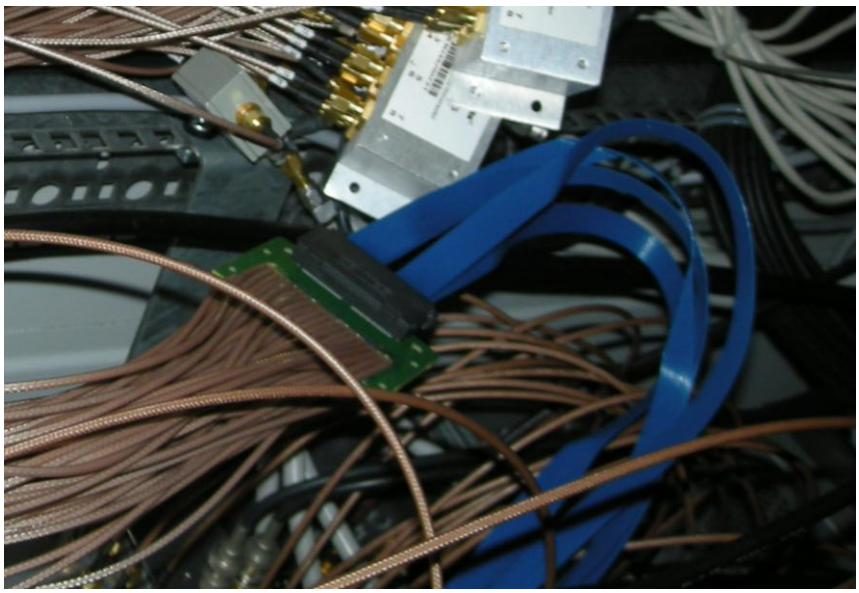
* PFB=Polyphase Filter Bank, FFT=Fast Fourier Transform, AMP=Amplitude equalization, PHASE=Phase calibration

ADC

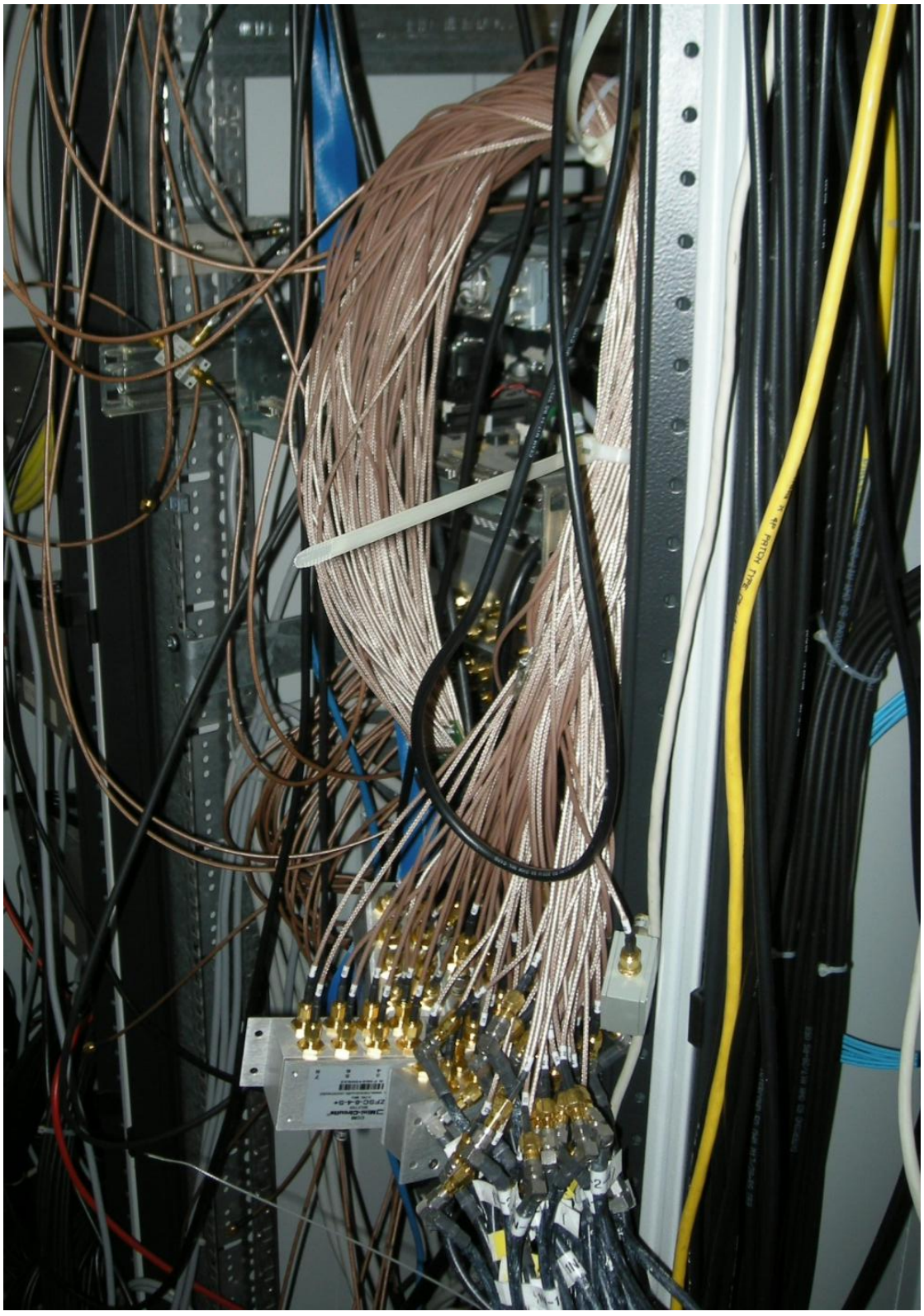
The ADC is the 64ADCx64-12, 64 inputs, 64 Msps, 12 bit, double wide board (it uses both the 2 ZDOK connectors) developed by Rick Raffanti. The next picture shows the ADC board, you can easily distinguish the 8 ADC chips (8 input each), the two ZDOK connectors at the top, few input ports on the right used to get differential sync signal (PPS) and a small connector at the bottom which links the analog inputs through a flat cable (see also the previous picture).



The next picture shows the rear of the ROACH boards rack where you can see the blue flat cable in details ending in a small printed circuit boards whit about 80 coaxial cables with SMA connectors for the analog inputs (antennas and sync signals).



The next picture shows the cabling, the black coaxial cables are the input signals coming from the optical receivers (16 MHz of bandwidth centered at 30 MHz), the metallic boxes are some splitter used to close to 50 Ohm a set of unused ADC inputs.



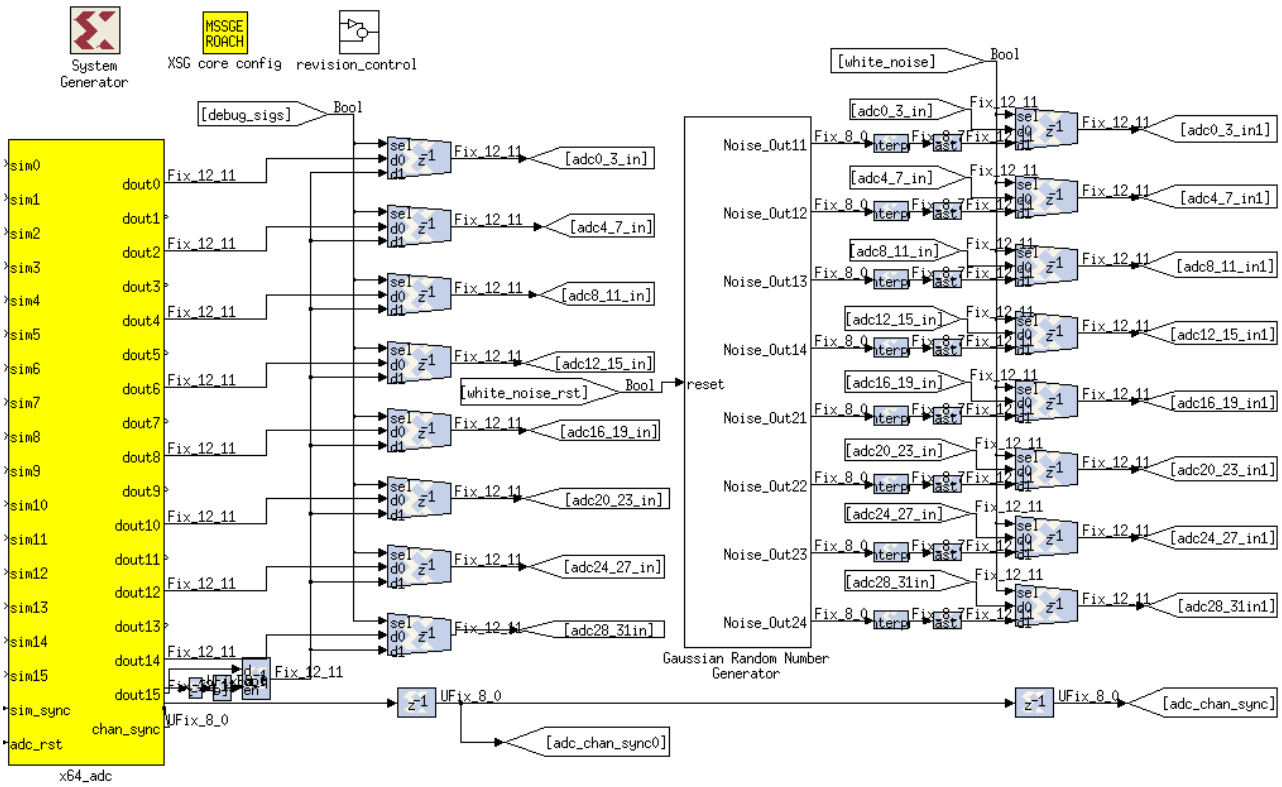
Firmware

Digitalization

The MAD3 firmware has been developed using the Xilinx System Generator® running in Mathworks MATLAB® that allows to convert a Simulink® model file (which consist of simply block diagrams) in VHDL code. Thanks to the CASPER engineering group that has developed the ROACH board there are many custom library block sets to address all the Xilinx Virtex5® peripherals available, and some software to compile the project file directly via the MATLAB command line that calls the XILINX ISE Suite® and produces the FPGA bitfile.

We have already produced a firmware for the MAD2 test campaign that was customized for a minimal set of correlations and for just one polarization because the transmitter aboard the UAV is a dipole transmitting in one polarization. That firmware allows a limited set of data analysis but was very efficient in terms of data rate and data volume produced. The idea of the MAD3 firmware is to improve the capabilities such acquire the two polarization at the same time which allow imaging analysis, and improve the bit quantization to investigate better some feature of the data, unfortunately all the changes increase seriously both the data rate and the data volume.

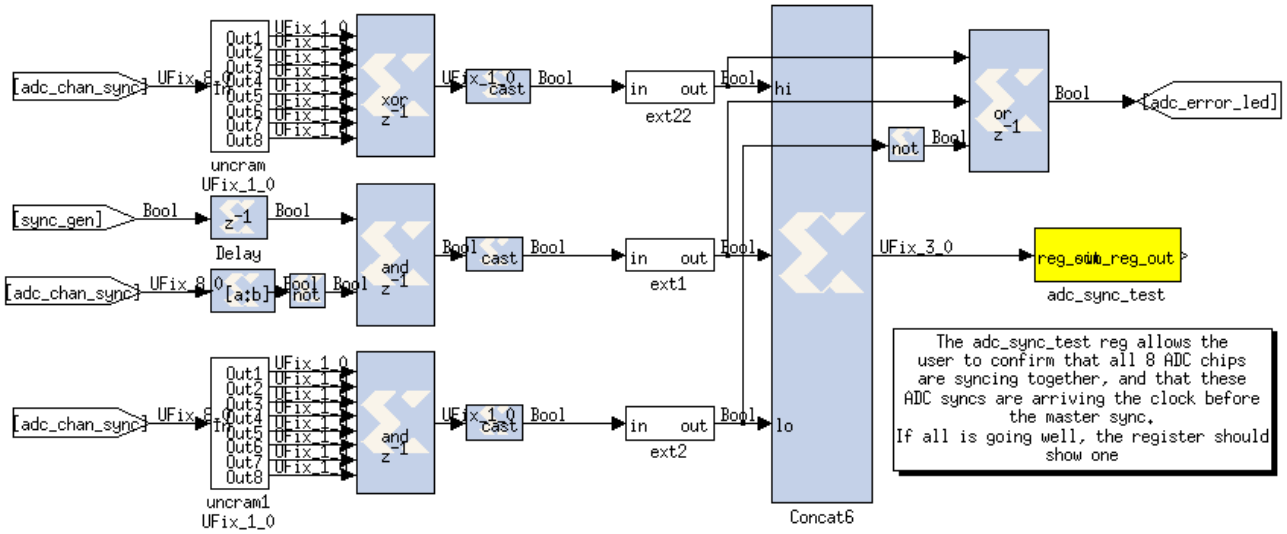
There are common parts between the old and the new firmware: the digitalization of the analog inputs, the channelization from time domain data to frequency channels, the amplitude equalization and the phase calibration. Even the backend control registers are shared between firmwares.



The ADC yellow block is a Simulink block written by the CASPER group that allows to address the ADC64in board. It produces 16 digital output in a clock cycle and the 64 parallel outputs in 4 clock cycles. Running the FPGA at least 4 times faster than the ADC board (which will have a dedicated synthetized clock signal) you will be able to manage up to 64 different input. The format of the data is Fix_12_11 which means a fixed point number of 12 bits with 11 bit of decimals.

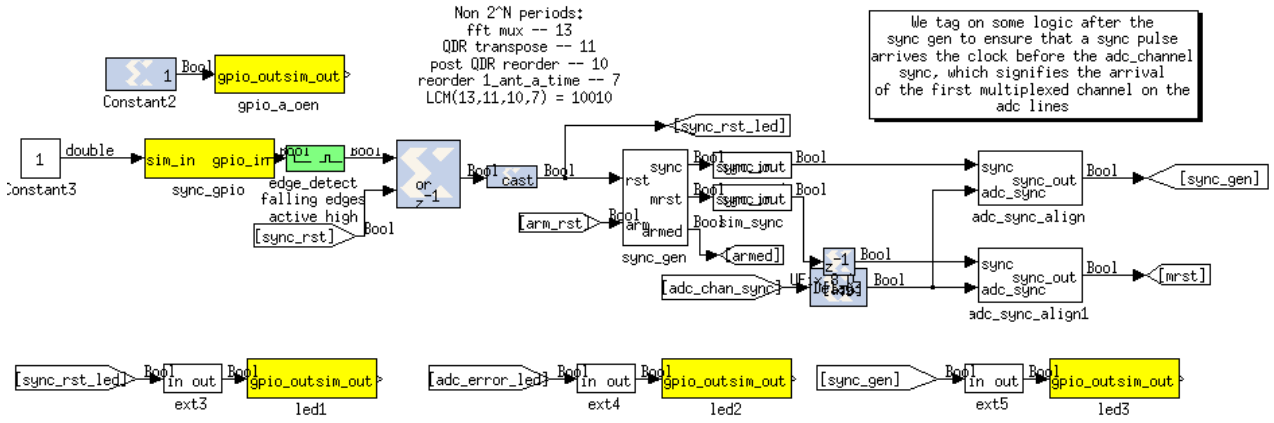
Synchronization

The ADC block provides also the sync signal useful for the synchronization. A separate logic tests the syncs of the 8 ADC chips on the ADC board and a python control scripts running during the initialization phase test if the ADCs are aligned with the sync.



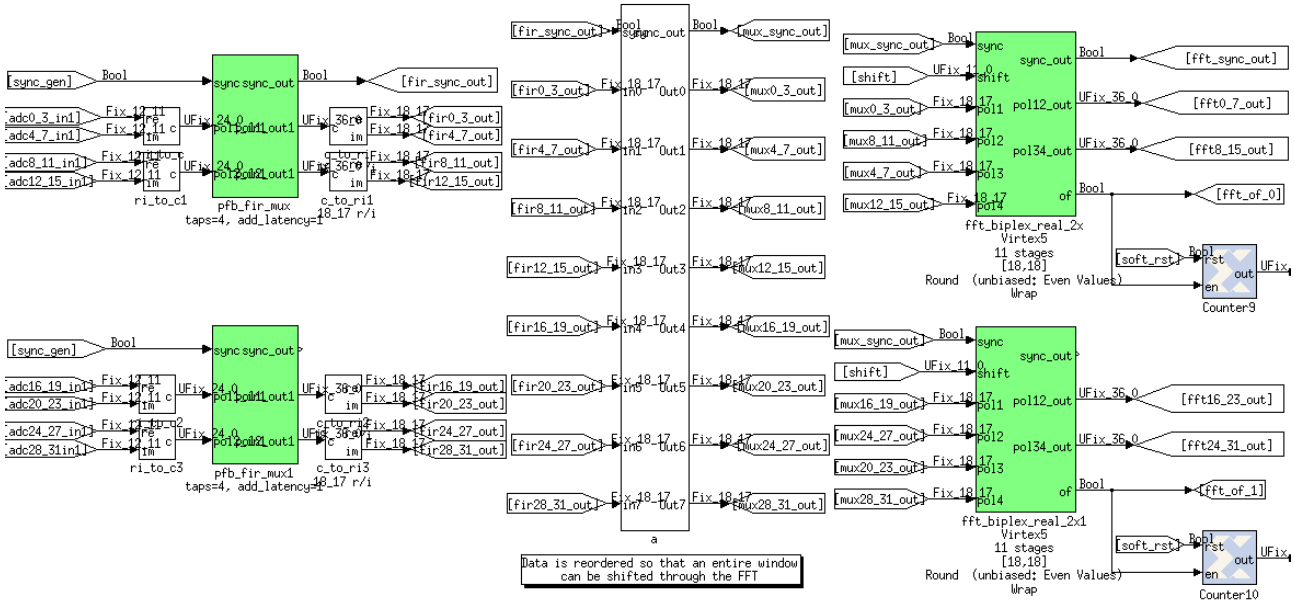

```
oper@bee2:~/andrea/feng/MAD3$ ./mad_start.py

====================================

Connecting to ROACH board named "feng"...  ok
   Deprogramming FPGAs
   Programming feng with bitstream mad_full_corr_2014_May_13_2258.bof
   Calibrating ADC on feng
    SUCCESS: adc sync test returns 1 (1 = ADC syncs present & aligned)
    SUCCESS: adc sync test returns 1 (1 = ADC syncs present & aligned)
    SUCCESS: adc sync test returns 1 (1 = ADC syncs present & aligned)
    SUCCESS: adc sync test returns 1 (1 = ADC syncs present & aligned)
    SUCCESS: adc sync test returns 1 (1 = ADC syncs present & aligned)
```

The ADC is locked to the two signals provided by the Maser Atomic Clock, a ultrastable 10 MHz sinewave and a PPS. The host computer is synchronized via a local NTP server providing the station time. The station time is a high precision clock locked again to the Maser 10 MHz sinewave and PPS, and verified periodically comparing a time provided by a GPS receiver. During the initialization phase, after an ADC alignment test, the host computer waits for a “new second” (having decimals parts less than 0.1s). This means a PPS signal has been recently received from the firmware and then send a "sync arm" (labeled “arm rst” on the next Simulink picture) signal to the firmware which arrives for sure within two PPS signals, the next PPS signal arriving to the firmware will generate the master reset and will be the "t_zero" time of the firmware, that time will be stored to local registers and attached to each observations file header. Each data packet has a 8 byte counter header which allows to know the acquisition time of each sample.



Frequency channels

The MAD3 science case is a regular array of 9 dual polarization antenna for a total of 18 input signals. The polyphase filters and the FFT logic blocks are optimized to manage parallel streams of 2^n inputs, that's way we will develop the system to manage 32 input. As you can see in the previous picture only the even output (dout0, dout2, dout4…) have been used where each lines carries 4 interleaved time domain streams.

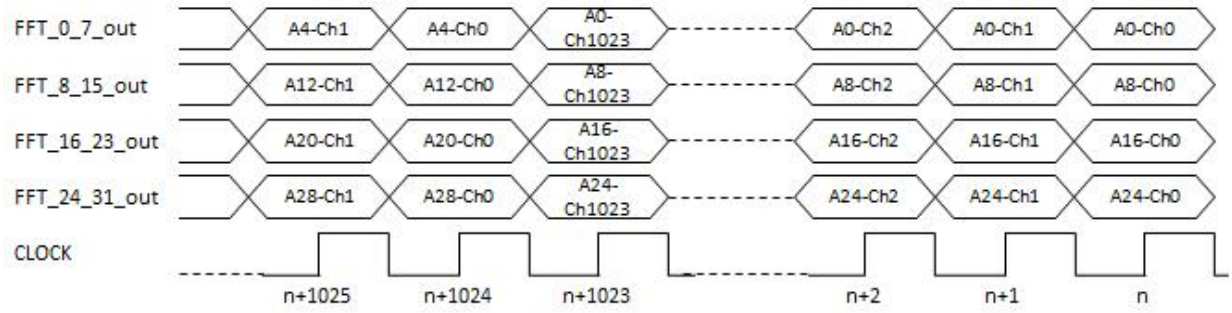


The CASPER green block of "pfb_fir" combined the "fft_biplex_real" implement a poliphase filter which consist of: multiply the analog signal to a discrete SYNC wave that correspond to a convolution of a RECT in frequency domain to filter each sub-band. The number of sub-bands and the number of the coefficients describing the SYNC is customizable with the block parameters, the higher those numbers the more FPGA resources is required. The Simulink block used for the PFB is a customized PFB version to use the ADC64in, called "pfb_fir_mux" which takes care about the 4 clock cycles to have the next value of the same stream, and, a reorder block is needed before the FFT block which expect to receive continuous stream of the same time domain data stream to compute the discrete Fourier transform. The output of the FFT are the N frequency channels of each input and one line carries 8 streams in the following order: 0-4-1-5-2-6-3-7.

The width of the data is set to 36 bit for the complex number (36 bit is the unit of the Xilinx Virtex5 registers) and the format is Fix_18_17 for both the Real and the Imaginary values. The number of sub-

bands in output from the FFT are 1024 (it is necessary to set both the size of the PFB and the size of FFT to 2048 points).

The order of the data leaving the FFT green blocks is as follow:

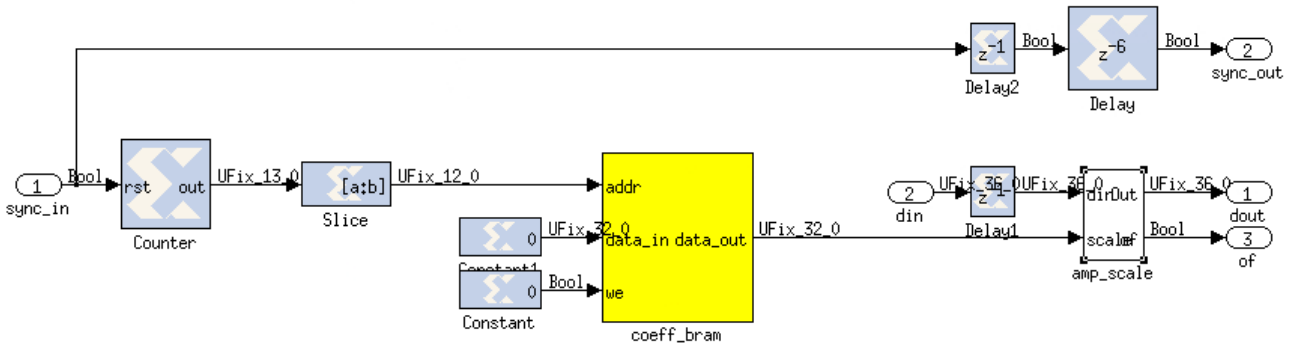


The next table shows an extract of the frequencies of each sub-bands, the left and right limits and the center frequency of the first and the second harmonic. As you can see, the second harmonic has a reversed order. The central sub-band of the 1024 FFT bins is the 512th which contains the frequencies around the 10 MHz and the 30 MHz.

Channel	1-Left	1-Center	1-Right	2-Left	2-Center	2-Right
508	9912109,4	9921875,0	9931640,6	30068359,4	30078125,0	30087890,6
509	9931640,6	9941406,3	9951171,9	30048828,1	30058593,8	30068359,4
510	9951171,9	9960937,5	9970703,1	30029296,9	30039062,5	30048828,1
511	9970703,1	9980468,8	9990234,4	30009765,6	30019531,3	30029296,9
512	9990234,4	10000000,0	10009765,6	29990234,4	30000000,0	30009765,6
513	10009765,6	10019531,3	10029296,9	29970703,1	29980468,8	29990234,4
514	10029296,9	10039062,5	10048828,1	29951171,9	29960937,5	29970703,1
515	10048828,1	10058593,8	10068359,4	29931640,6	29941406,3	29951171,9
516	10068359,4	10078125,0	10087890,6	29912109,4	29921875,0	29931640,6

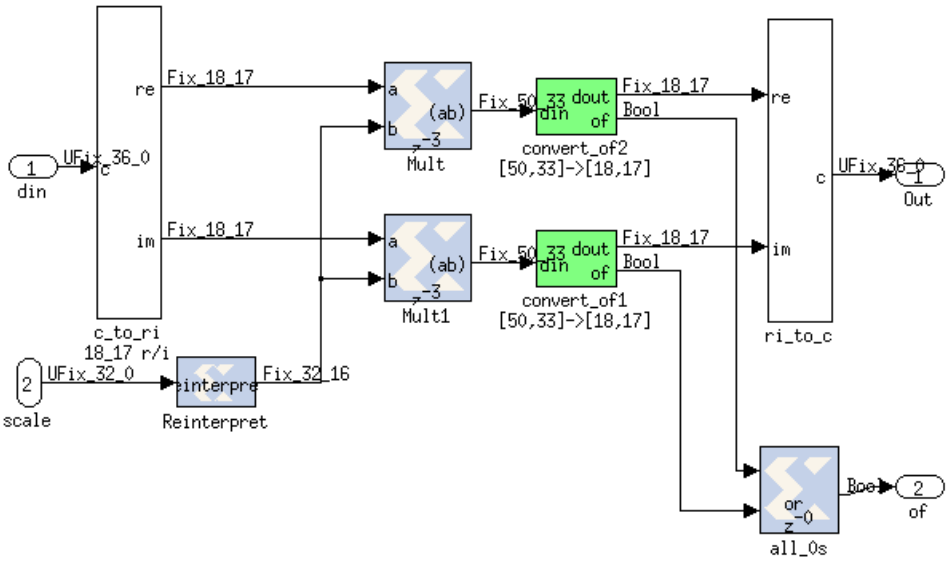
Amplitude Equalization

The data streams reach 2 stages of multipliers that allow to equalize in amplitude and calibrate in phase the antennas. It is possible to equalize and calibrate each frequency channel independently thanks to the presence two BRAM (Block RAM of length 1024 each) which allow to load coefficients by the host computer. Since the output of each multipliers has the bit width increased, a data cast (quantization) is needed.



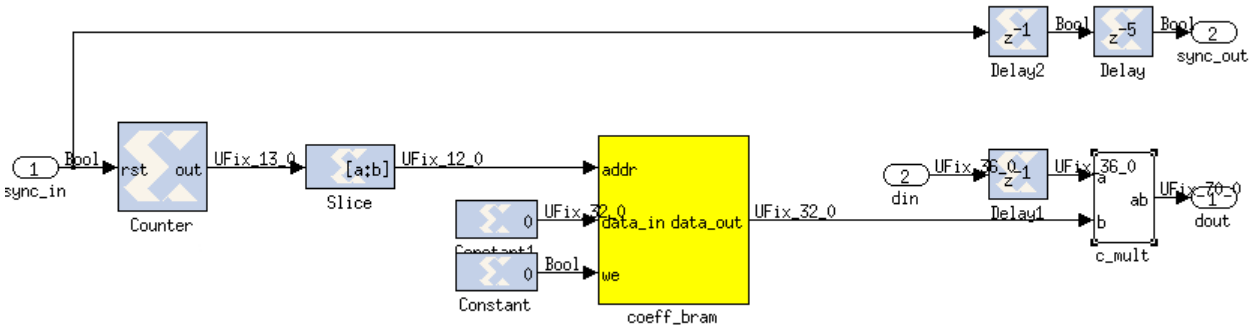
The previous picture show in details the amplitude equalization stage: going into the Simulink subsystem you will find the BRAM yellow block, and another subsystem “amp_scale” making the multiplication.

There are two multipliers that computes the multiplication between the real and imaginary part of the complex number (fixed point 18.17 each) of the frequency channel and a coefficient in the fixed point 32.16 format. The result is converted to a Fixed 18.17 again.

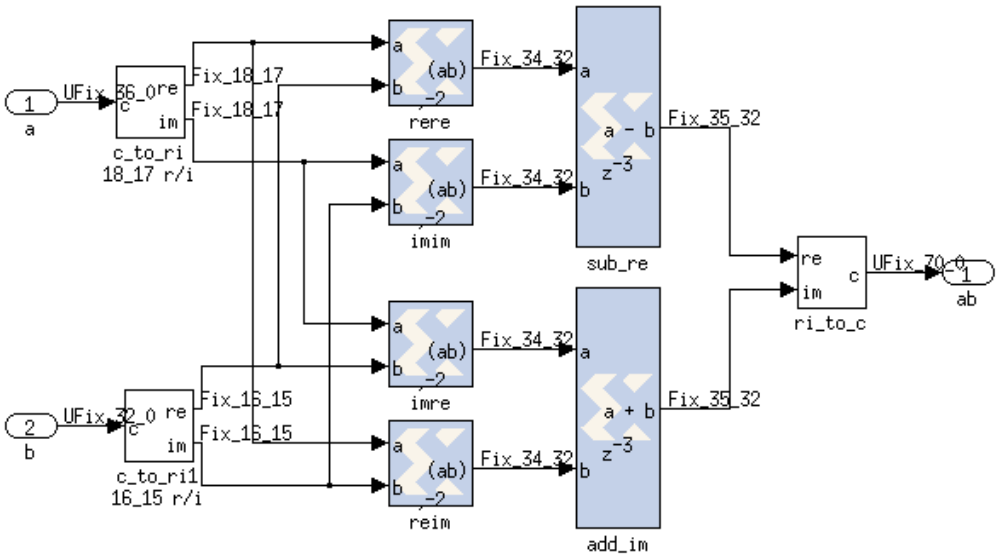


Instrumental Phase Calibration

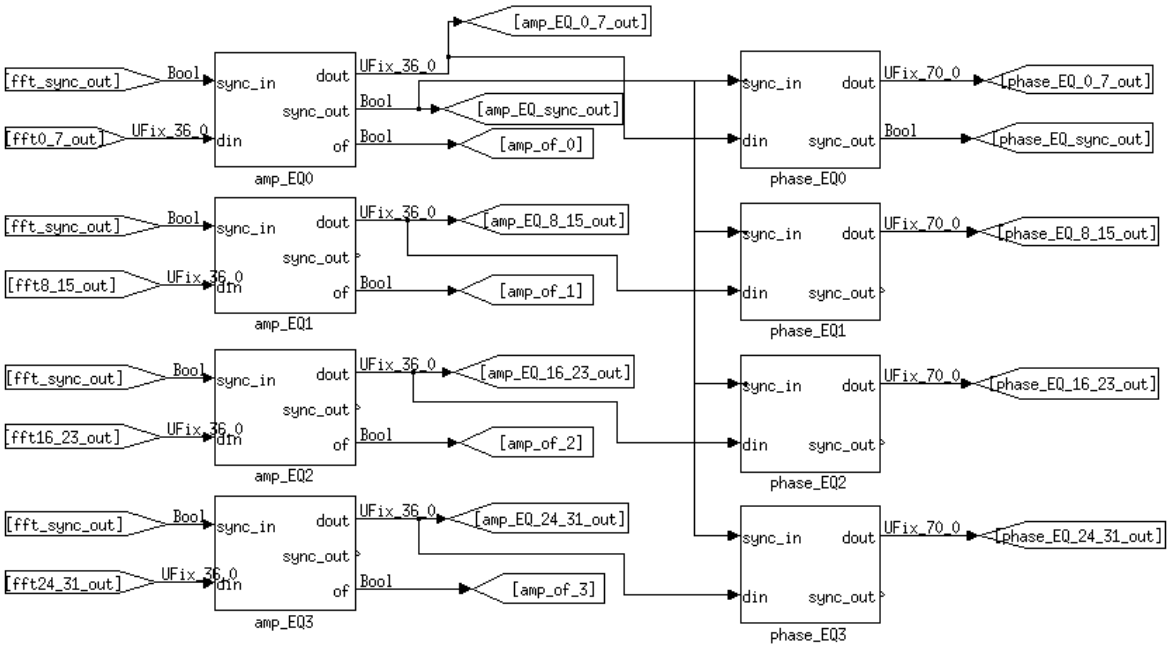
The phase calibration stage is similar to the previous stage without the “real mult” subsystem which is a complex multiplication between the frequency channel and the phase correction loaded into the BRAM.



The phase correction factor is a complex number having the real and imaginary part written has a fixed point 16.15 each. The data output is a complex number where real and imaginary part have the format 35.32 each. This 70 bit number will be the final number which will be used for the auto correlation and cross correlation products in the new firmware, while the old branch of the firmware will keep a quantized data of 8.7 bit for real and imaginary part each.

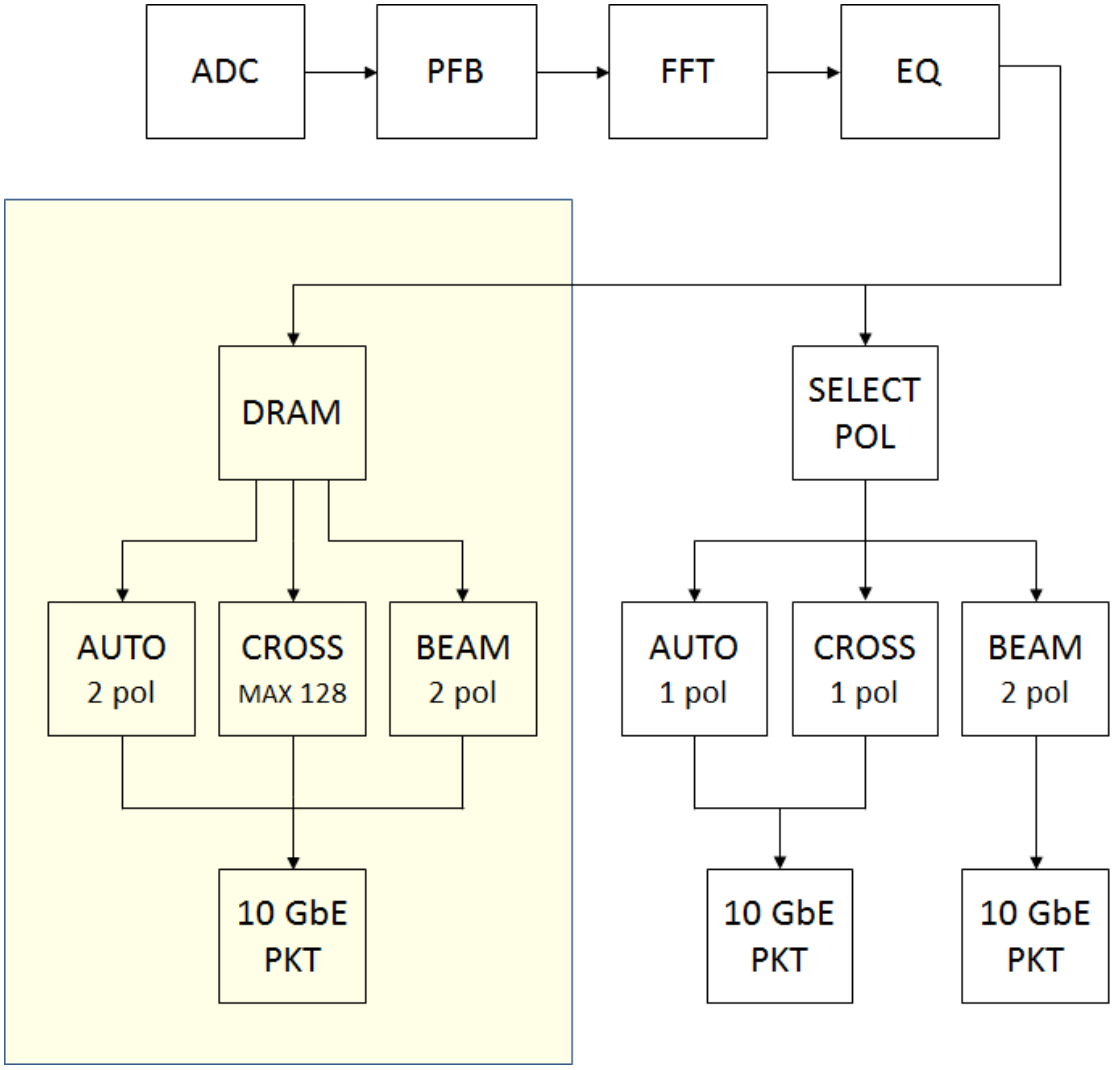


Each stage of multiplier is grouped in subsystems and shown as in the follow picture in the top level Simulink model file.

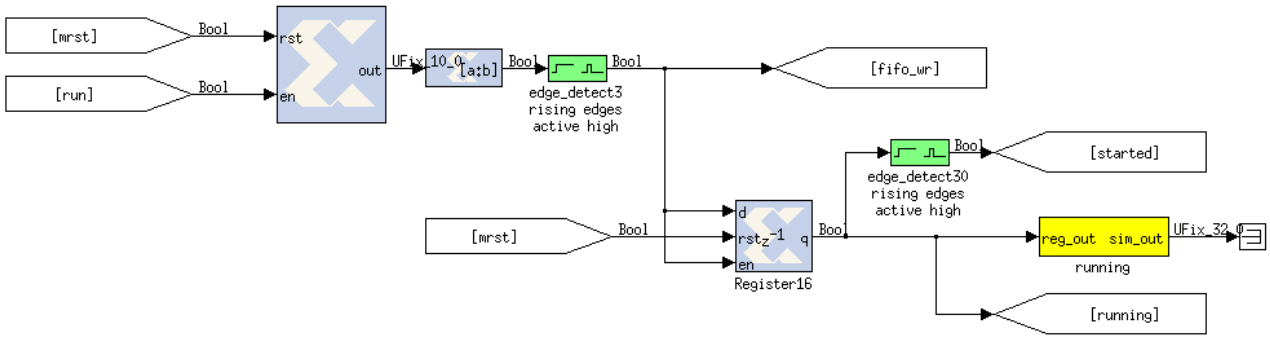


The 64 bit system

The following scheme describes the “State Machine”: 32 streams of data are channelized and equalized. Then the frequency channel bin containing the sine wave transmitted by the drone is cached by a combination of FIFOs and a Dual Port RAMs.

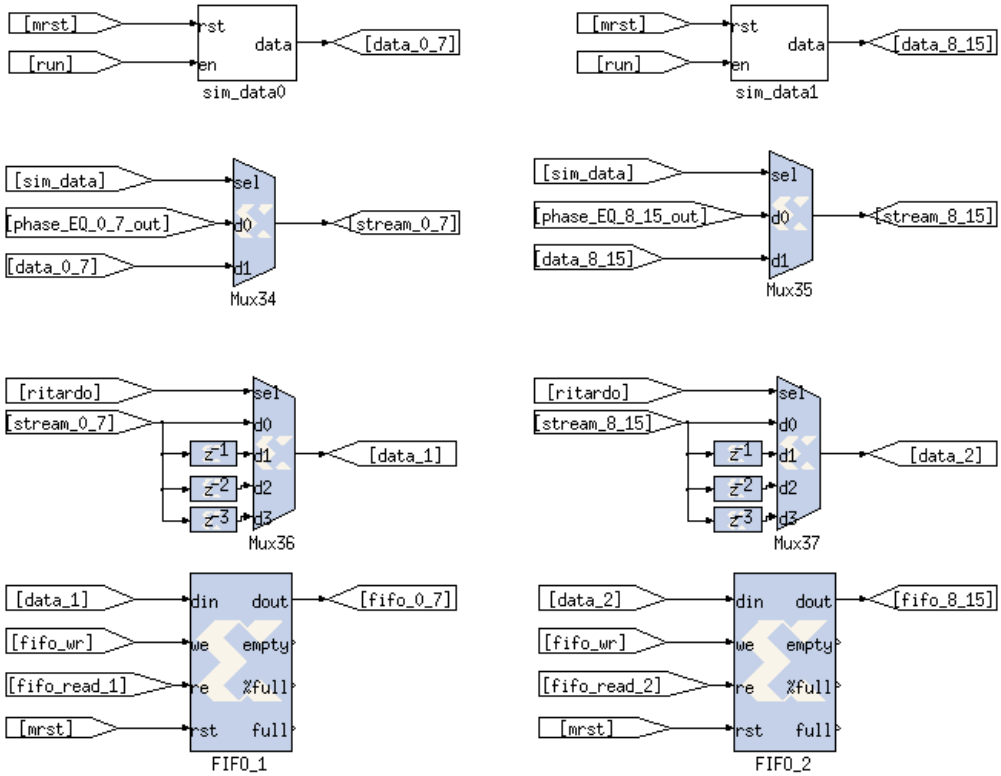


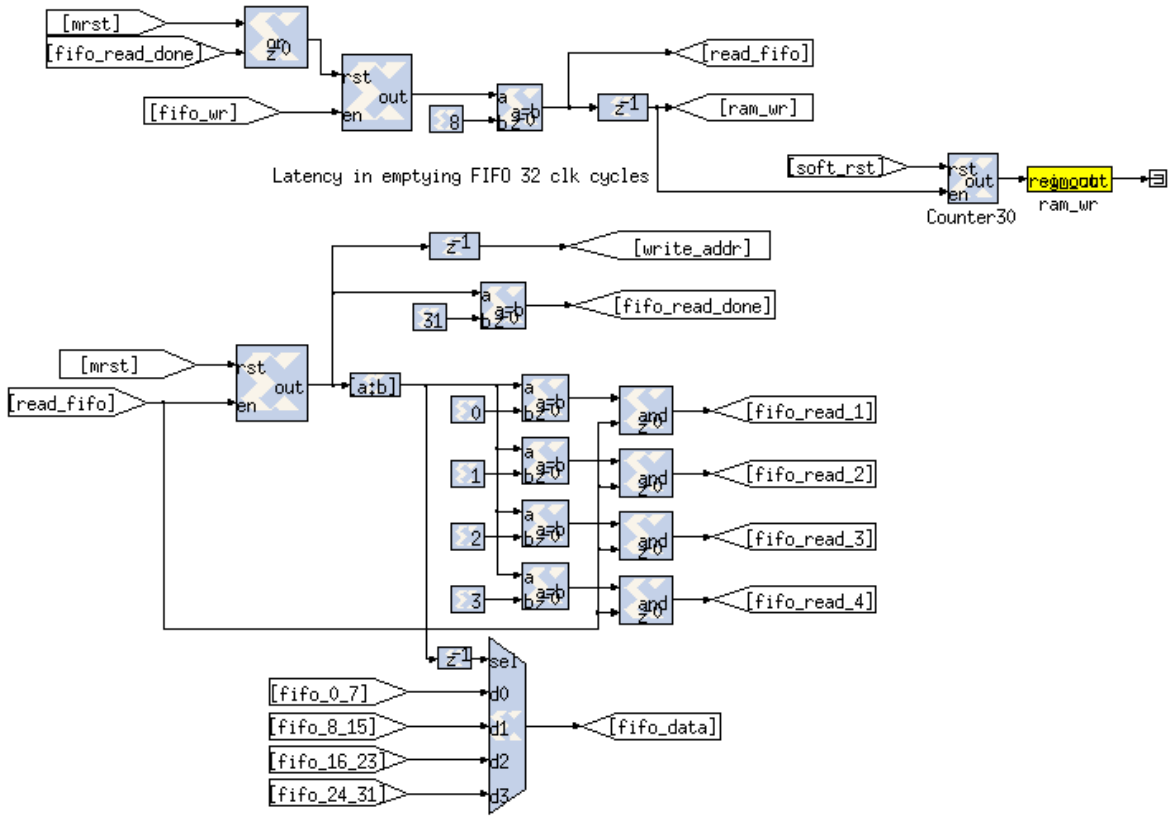
The first raw of the block diagram is the common part between the two independent systems producing more or less the same output with different resolution (bit quantization). There are 4 parallel streams of antenna spectra (organized by antennas and not by frequency channels) where few logic blocks catch the 512th channel where we expect to find the sine wave transmitted by the drone at 408 MHz.



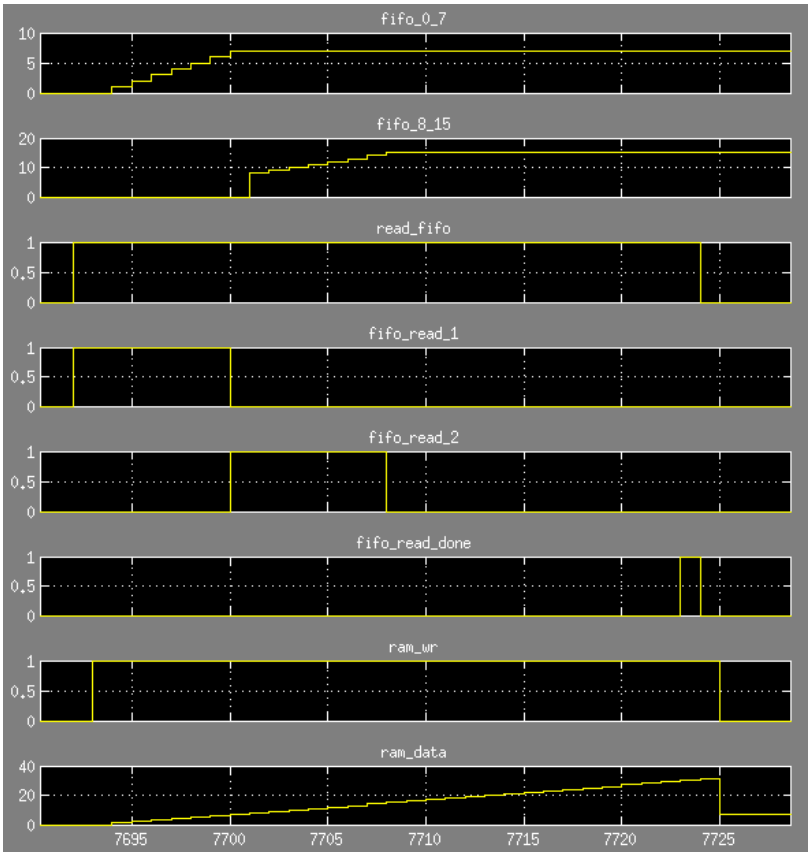
The edge detect green block (set to rising edge, active high) over the 10th bit of a 13 bit counter (that counts from 0 to 8191) generates the FIFO write enable signal. It will produce a one clock high level signal every 1024 clock cycle (each spectra is composed of 1024 frequency bin), starting from the first 512th channels. The master reset of that counter is aligned with the sync signal escaping from the last equalization block to ensure that the first sample is the frequency channel #0 and aligned with the timestamp of the first spectra saved in the header BRAM.

The 32 frequency channels reach the 4 FIFOs (4 streams of 8 spectra each) in a window of 8192 clock cycles then few logic blocks allow to empty one FIFO at a time shortly afterwards in order to save that data into two twins Dual Port RAMs.

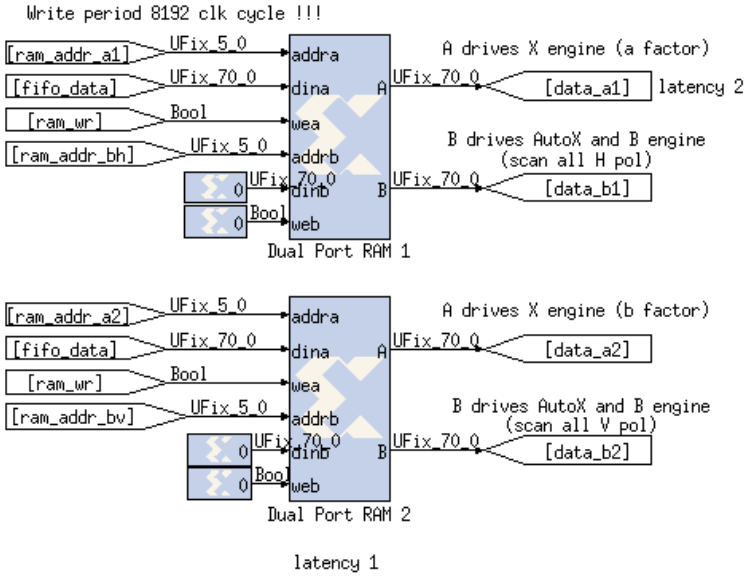




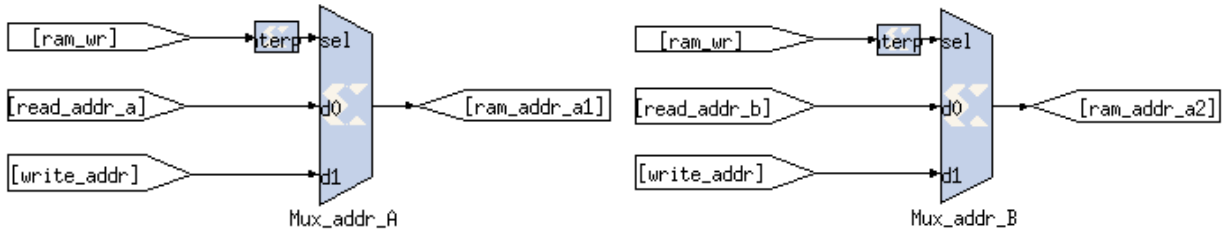
The next picture shows the result using the simulation logic blocks instantiating a constant number for each spectra in the right order. Considering the latency clock cycles when reading data from a FIFO, the 32 frequency channels stored in the DRAM is a ramp between values 0 and 31.



Each dual port RAM contains the same values at the same address. Each address represents the index of the data stream that is the antenna number. Due to the structure of the parallel FFT a remap of that addresses is needed. The choice to have 2 twins Dual Port RAM is to extract 2 factors at the same clock cycle for the Cross-Correlation engine, while, the choice of a Dual Port (whit respect to a simple RAM) allows to run other engine at the same time, the Beam-Engine and the Auto-Correlation engine (that work using the same input).

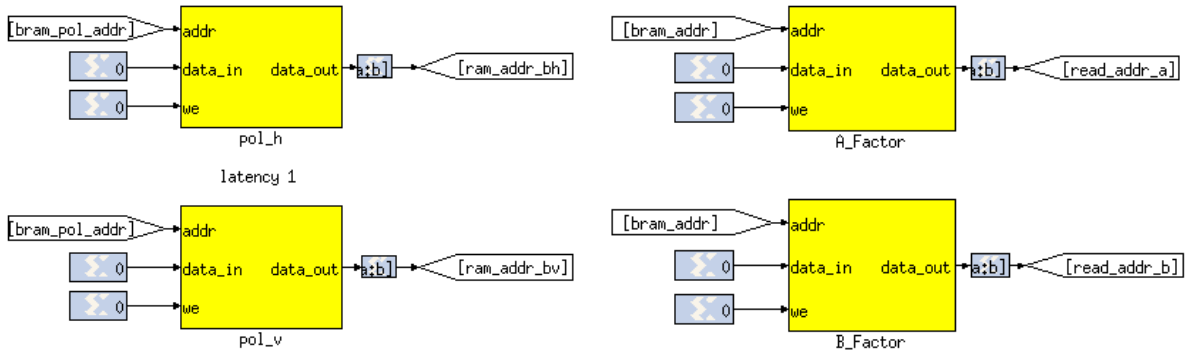


Two multiplexer change the source of the address line of DRAM port A for write operations and read operations.



This system has been designed to work with dual pol antenna and produces in output the sequence of all the Auto-Correlation data (both pol-H and pol-V), the user defined Cross-Correlations and the total Beam of each pol. The Correlations set to be computed is loaded by the user into FPGA BRAMs called “A_Factor” and “B_Factor” and it is the complex multiplication between A and the conjugate of B.

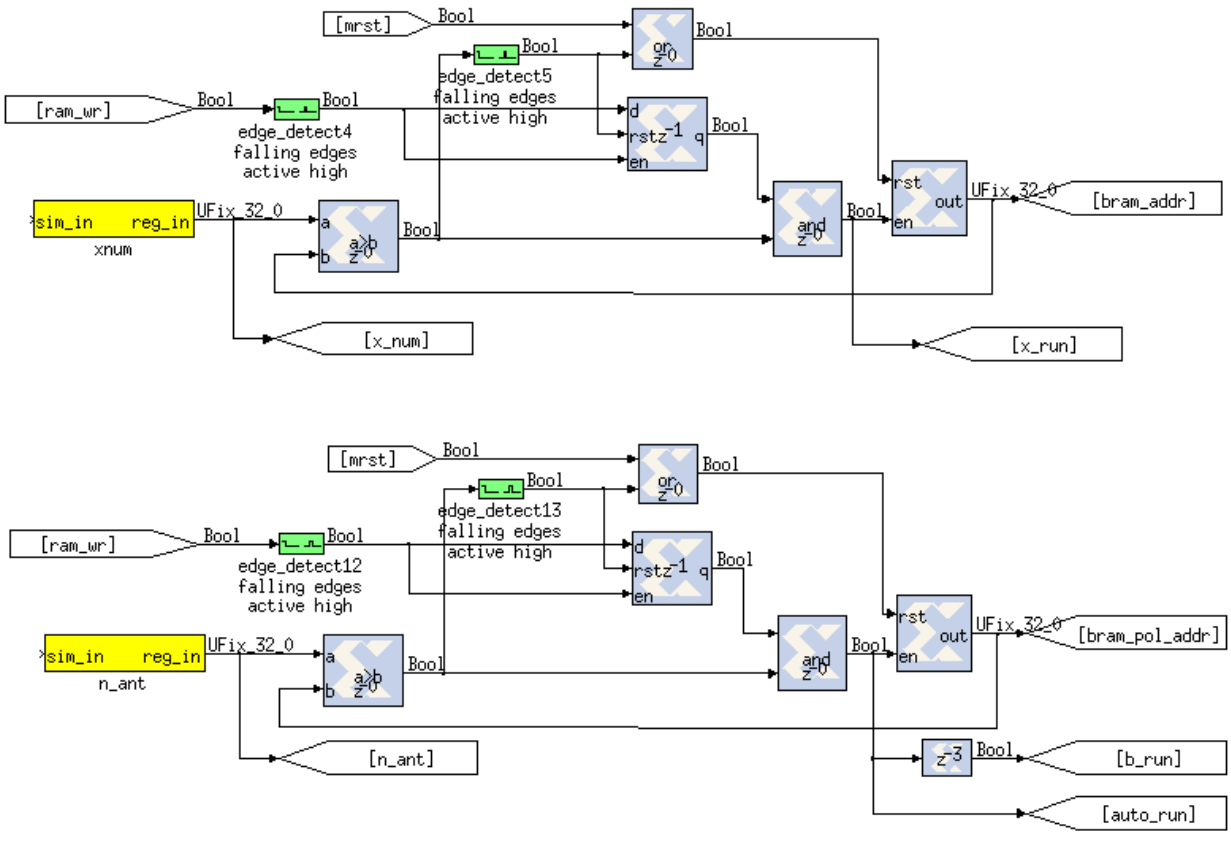
That BRAMs as well as the two BRAMS “pol_h” and “pol_v” contain the values of the addresses of the interested antennas that must match the ADC inputs where the MAD antennas have been connected. The firmware can only read values from the BRAM (see the write enable signal of the BRAM always set to zero), while the user can write to them via Ethernet. The width of the BRAM values is 32 bit, due to the limit of 32 antennas a slice of the last 5 bits provides the range values of 0-31. The size of the BRAMs is the minimum instantiable that is 1024 nevertheless further logics set the real maximum limit to 128, that means, there is not allowed ask to compute more than 128 Cross-Correlations or Auto-Correlation.



This limit has been fixed in order to do not exceed the 10Gbit packets volume limit for the 64 bit size of the output data, and, to fit the data rate allowed by the host computer that receives these packets that has been properly configured and equipped with a set of disk in RAID which guarantees write speed of about 350MB/s.

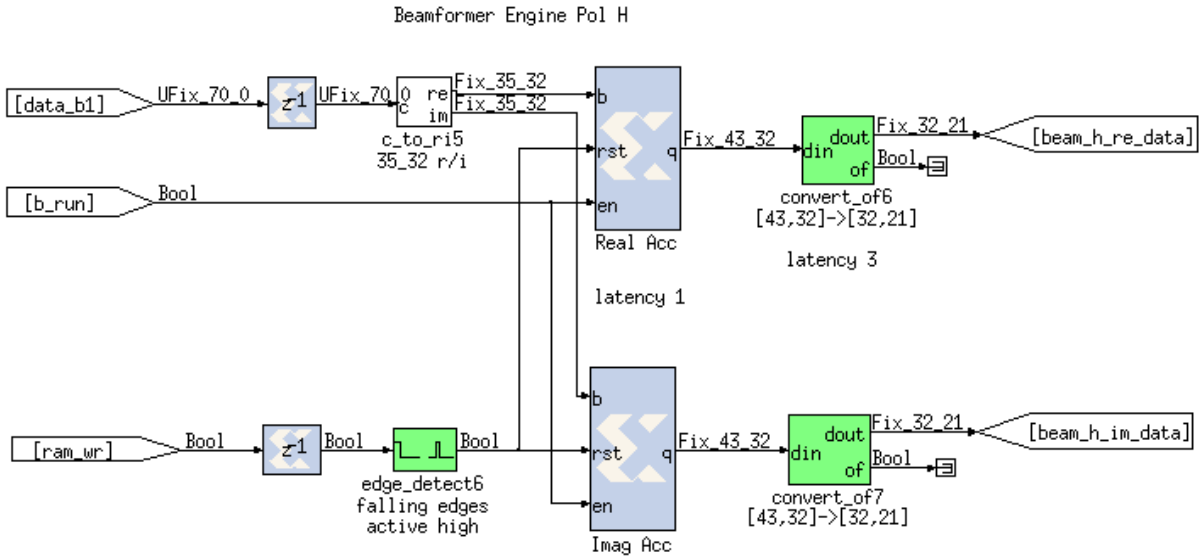
There are two software register into the FPGA firmware that store the number of the antennas used (that means the number of ADC input signals used) and the number of the correlations that have to be computed. These two numbers tell the firmware how many iterations must be done to extract valid data from BRAMs.

When the DRAM write enable goes down the computation engines starts. The DRAM output port A gives values for the X-Engine (correlation products) while the output port B values for the Beam-Former and Auto-Correlation products. The address lines A and B of each dual port must have the correct address of each factor. An additional logic (shown in the next pictures) must generate the address line for the BRAM to output valid addresses for the dual port RAMs.



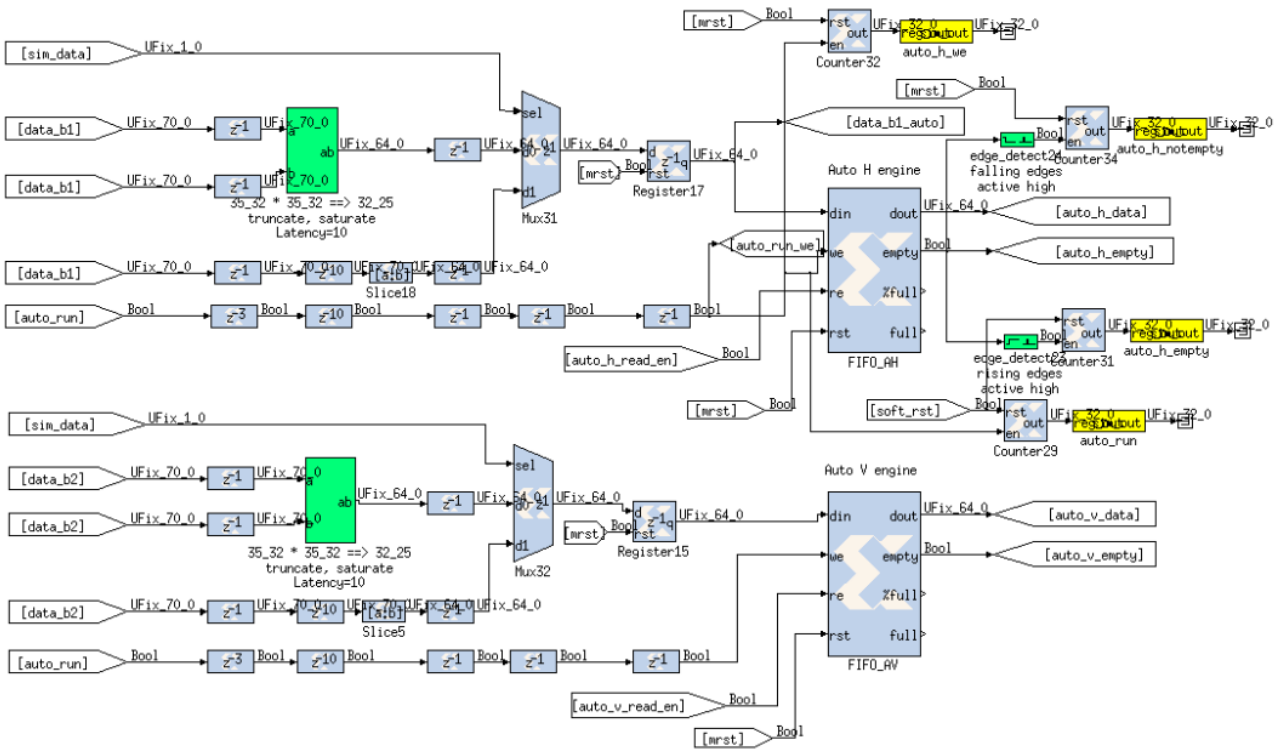
As seen in the above pictures, “bram_pol_addr” and “bram_addr” values are the addresses for the BRAM, while BRAM output signals called “ram_adrxxx” are the addresses for the dual port RAMs.

Each engine receives the respective input data and takes different latency for the computations depending on the complexity. The easiest engine is the Beam-Former that makes a complex sum of each antenna frequency. This is done using a simple Xilinx Accumulator block which needs to be reset to zero value at least a clock cycle before the first valid data is present at the input port “b”. The sum must be enabled rising a “en” signal in line with the input data and the result of the sum is available after a clock cycle.

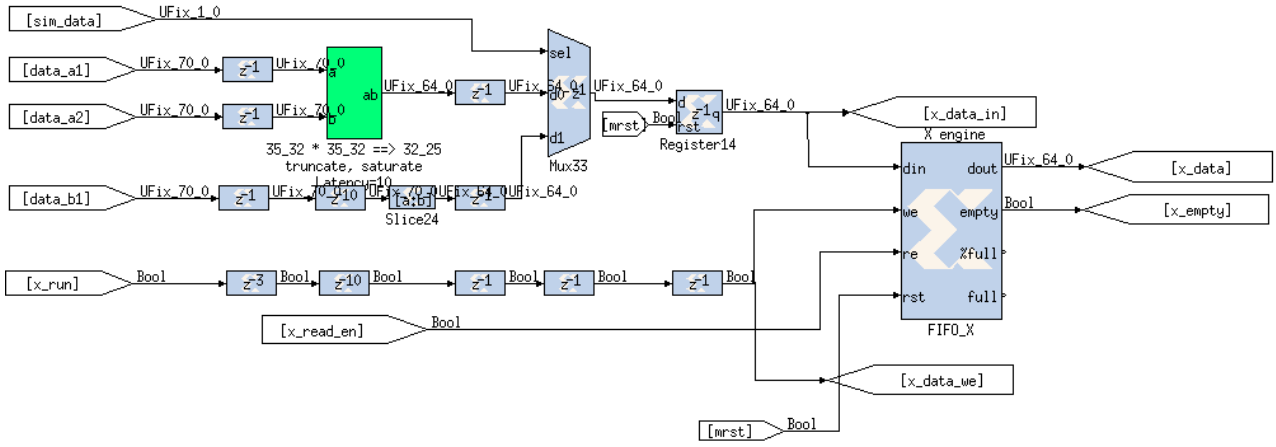


The output of the accumulator which grows up a bit for each sum (worst case) must be quantized due to the limited bandwidth. We have decided to have an output beam formed made by 64 bit for the complex number (32 for the Real part, 32 for the Imaginary part).

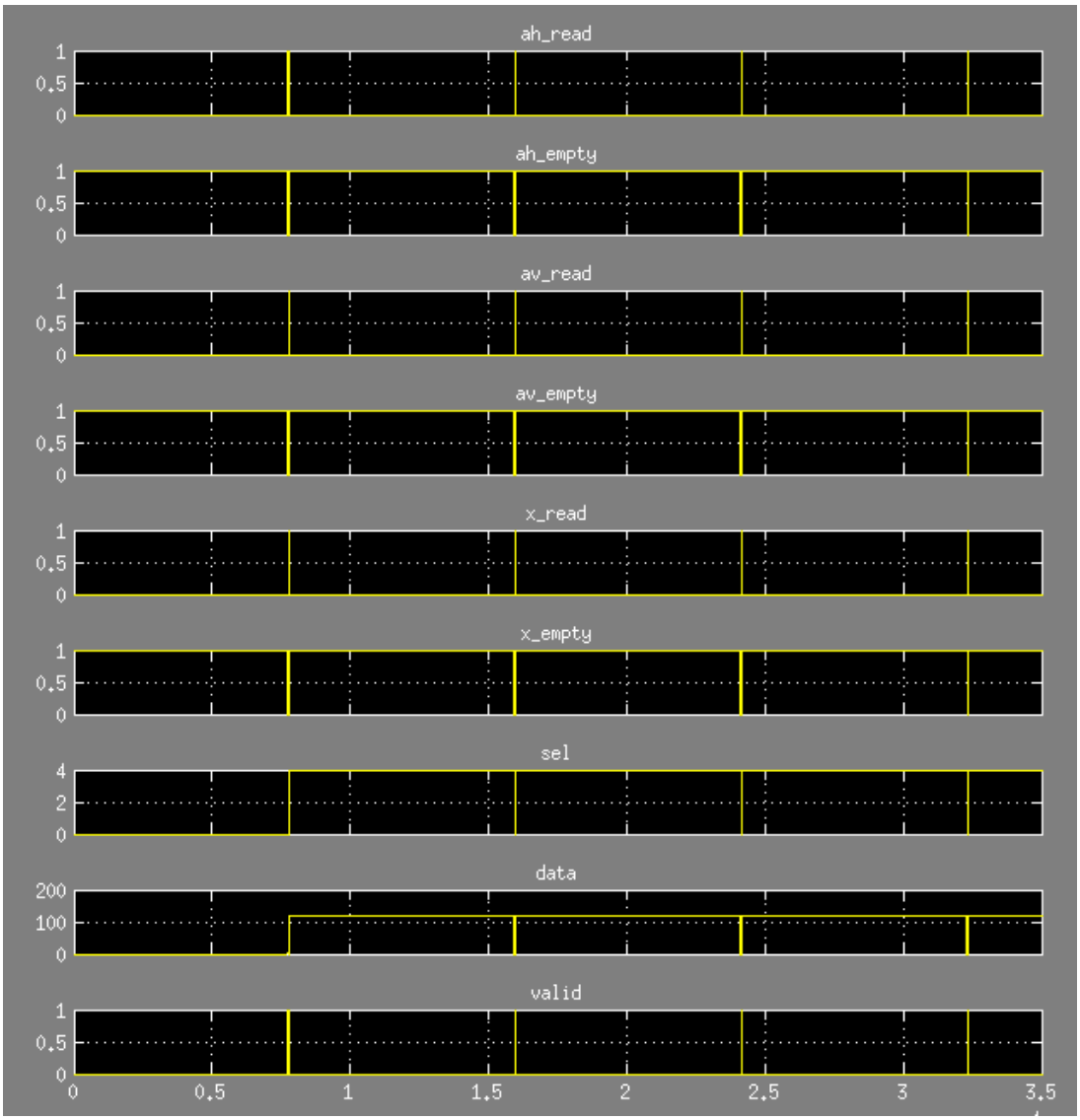
The same antenna frequencies signals go to the Auto-Correlation product. It is a complex multiplication between a value and complex conjugate of the same value done by the green block on the left of the next picture where the upper engine computes the polarization H and the lower engine the polarization V.



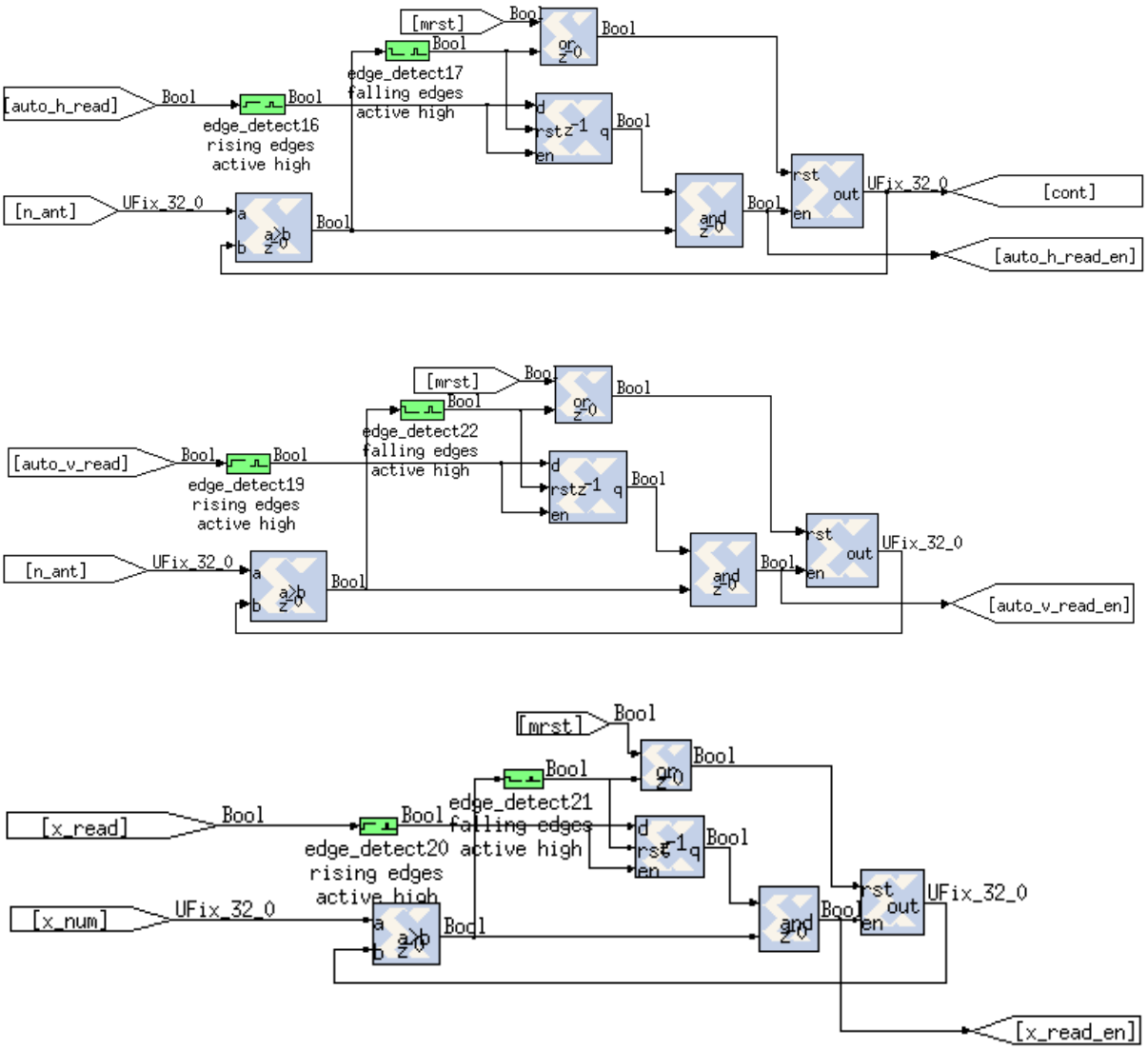
Each single result is stored to a FIFO ready to be sent to the 10 Gbit packetizer. A “FIFO Empty” signal will tell if the FIFO is empty or there are still complex values to read. The X-Engine that computes the cross-correlation products (interference fringes) is equal to the previous engine unless the product factors are not the same values but the frequencies of different antenna of the array. In any case, the output of the green block is internally quantized to match the 64 bit constraints.



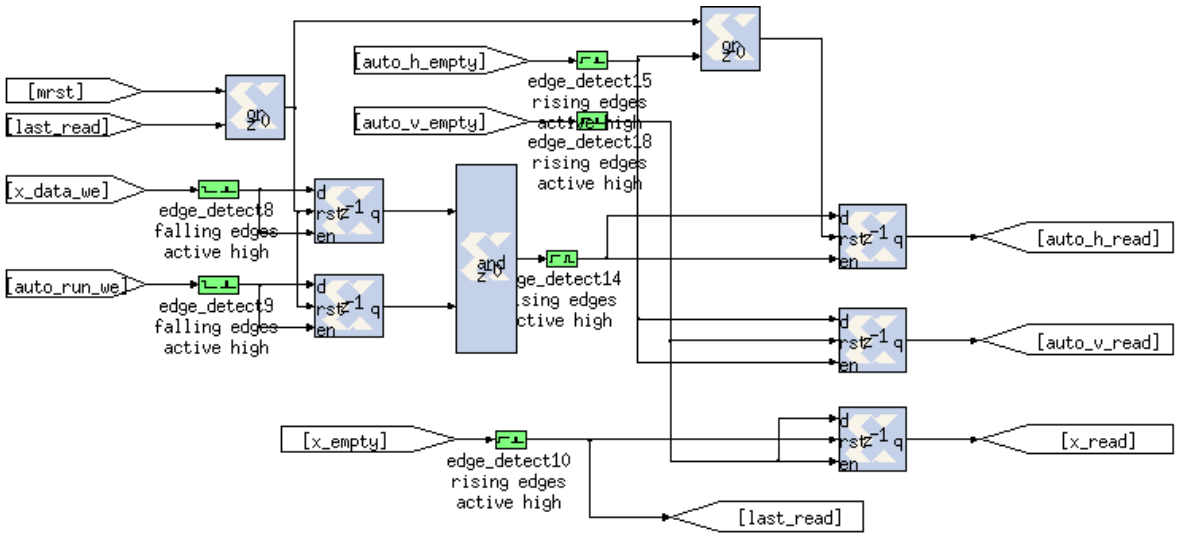
These engines run for a short period because it works with input buffered data. The next set of products (or accumulations in case of the Beam-Former) will be available in about 8000 clock cycles. That sounds like there is enough time to make thousands computations, but the data rate and the data volume of the output data saturate the output bandwidth (especially using 64 bits). We have chosen to limit to a maximum of 128 Auto-Correlations and 128 Cross-Correlations products. For the MAD test case those numbers guarantees to acquire the 9 dual polarization antennas with the full correlation matrix (9 Auto-H, 9 Auto-V, 36 Cross-Correlations and the 2 Beams of H and V). The next picture is very interesting shows the burst of data every 8192 clock cycles.



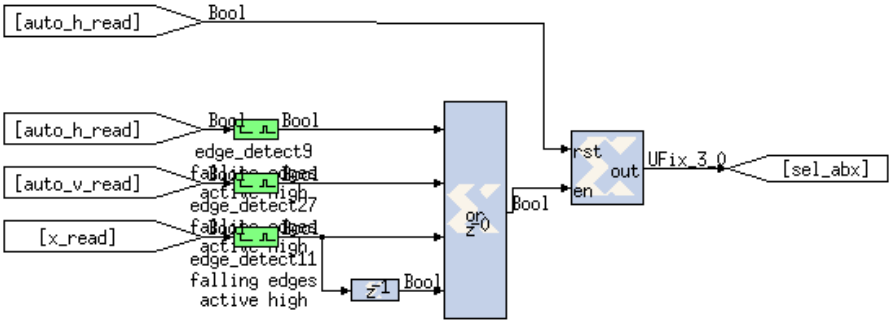
All engines have finished to compute their products and now the FIFOs are full of data and must be emptied. A few logic blocks produce, in sequence, the read enable signals to read the FIFO data in the order: AUTO-H, AUTO-V, CROSS. The next pictures shows how it is done.

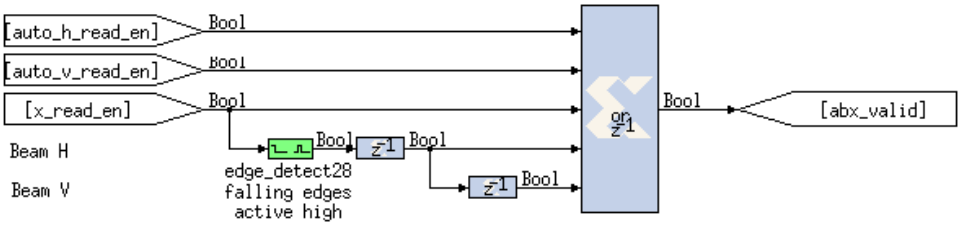


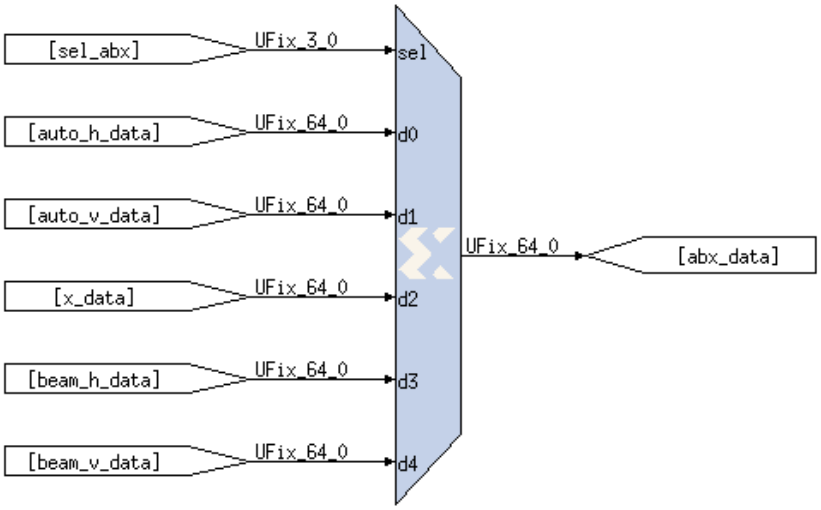
A high level control logic must exist to implement the machine state that serializes the output data. The next picture shows the logic that allow the FIFO-H read enable to start at first, then the FIFO-H, and last the FIFO-X. At the end it is needed to present to the data bus of the packetizer also the output of the beam-former accumulators.



A big multiplexer is demanded to switch between data flow and the selection of the mux input is shown on the next pictures.



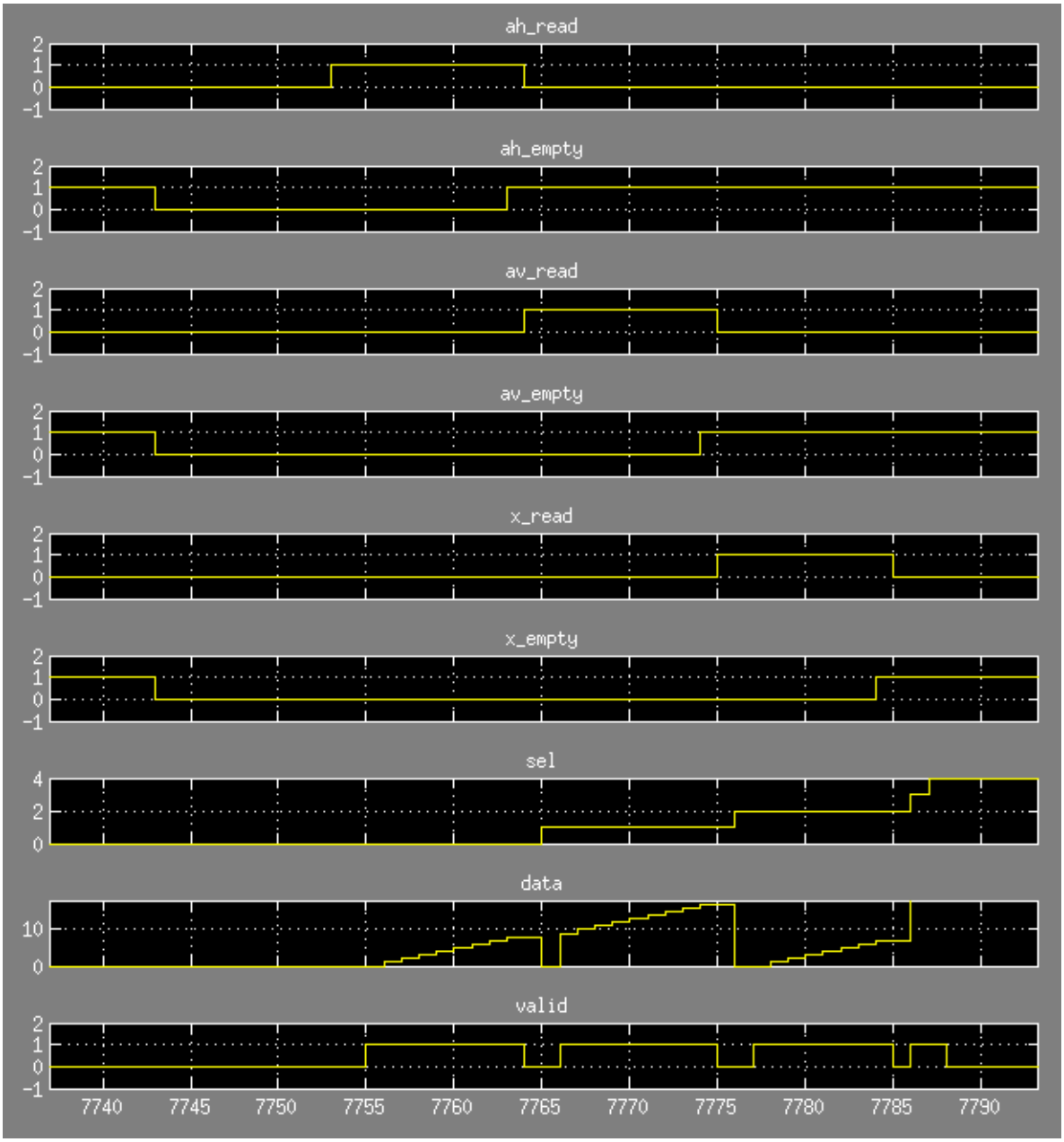




Using the simulated data it is shown on the next picture the behavior of this part. The simulation stimulus used are: number of antennas 9, number of cross correlations 8 (minimal correlation matrix), and the address of the ADC input used are from zero to 17.

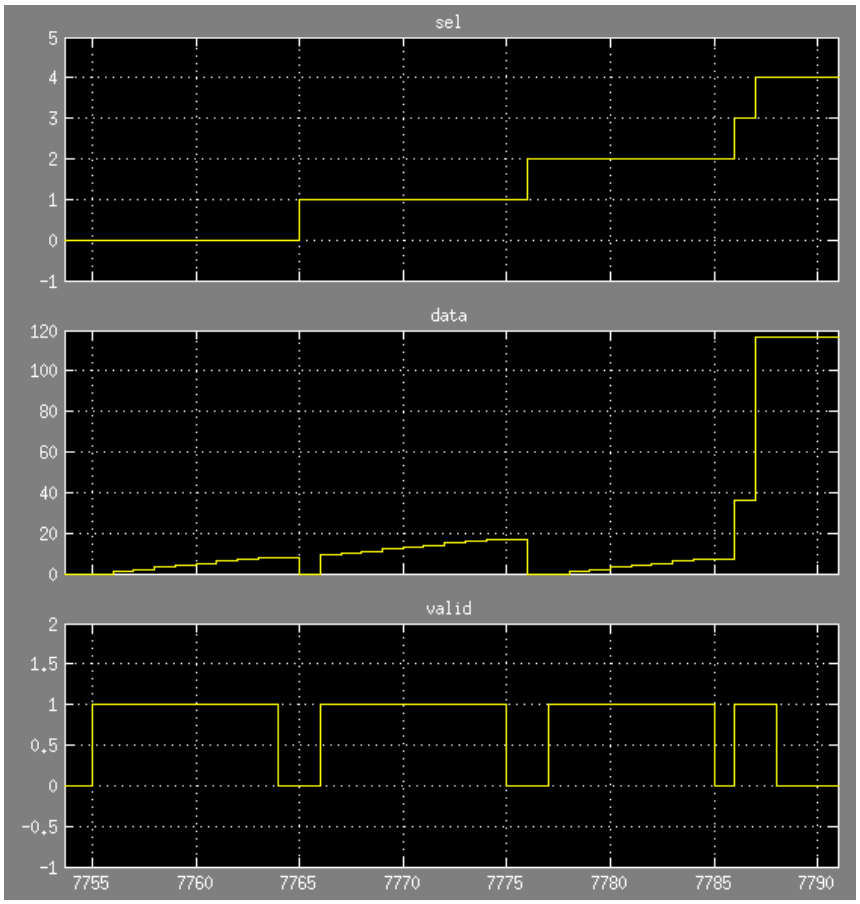
Following the path with the “sim_data” Boolean value set to 1, the AUTO-H values must be a series from 0 to 8, the AUTO-V a series from 9 to 17, and the Cross-Correlation a series from 0 to 7. Looking at the bottom signal called “valid” you will see the “data” over the output serial bus, and the upper signals suggest which data stream is. The last spot of data is the BEAM-H and the BEAM-V that need always only 2

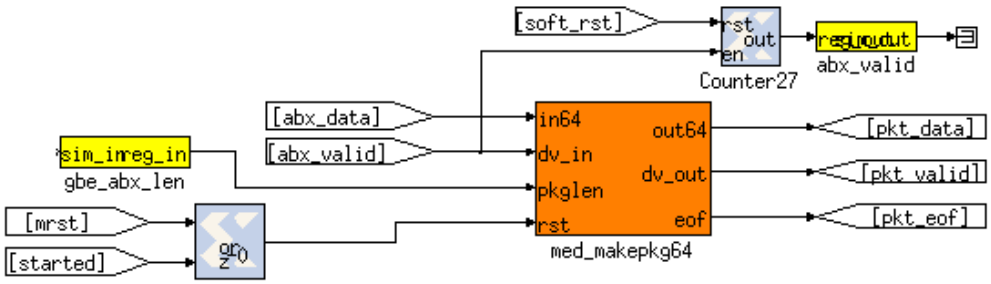
clock cycles while the AUTO and CROSS windows can have different length depending on how many antennas is connected and how many cross-correlations have been programmed.

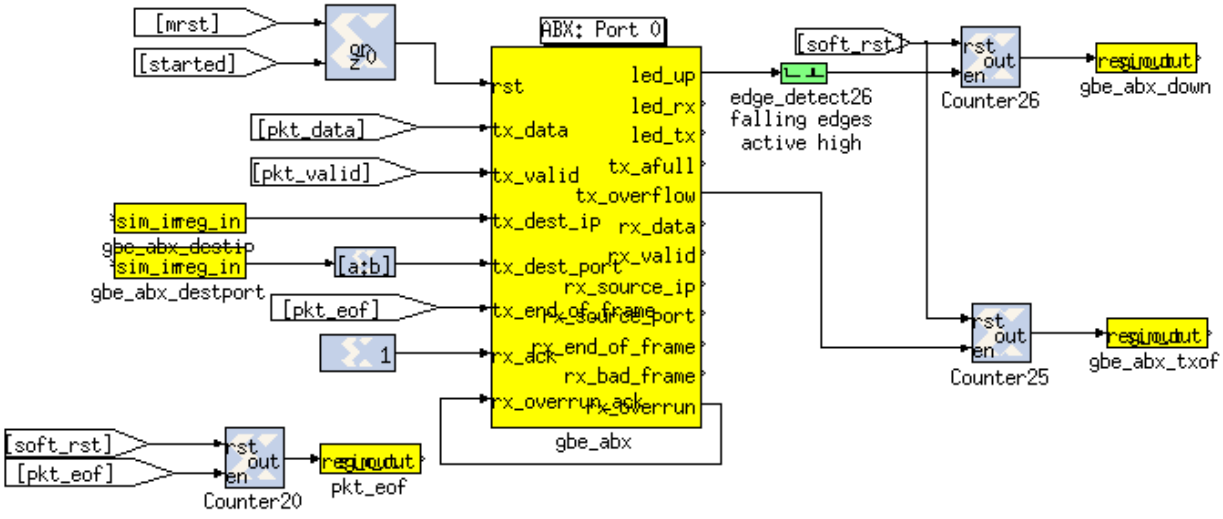


The next picture has been produced zooming in Y the previous picture in order to demonstrate that the Beams have the right accumulated values that is 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 = 36 for the polarization H and 9 + 10 + 11 + 12 + 13 + 14 + 15 + 16 + 17 = 117 for the polarization V.

The “med_makepkg64” orange block is the block the prepare the 10 Gbit packet. It expects to have a data valid signal in line with the 64 input data and the size of the packet. Since this block prepend a counter in each packet (useful to compute in post-elaboration the timestamp of each data) it is important to have the size parameter multiple of AUTO-X + AUTO-V + CROSS + BEAM-H + BEAM-V, otherwise input data validated on the same clock cycle where the counter is written will be lost. That counter is a 64 bit value that is written the clock cycle after the “pkt_eof” signal for the 10 Gbit has been raised. The size of the packet can be loaded and modified by the user writing the software register called “gbe_abx_len”.


sel
5
4
3
2
1
0
-1
data
120
100
80
60
40
20
0
valid
2
1.5
1
0.5
0
-0.5
-1
7755
7760
7765
7770
7775
7780
7785
7790


[soft_rst]
rst
out
en
Counter27
abx_valid
[abx_data]
[abx_valid]
in64
dv_in
pkglen
rst
out64
dv_out
eof
[pkt_data]
[pkt_valid]
[pkt_eof]
sim_imeg_in
gbe_abx_len
[mrst]
[started]
med_makepkg64


[mrst]
[started]
ABX: Port 0
rst
led_up
led_rx
led_tx
tx_afull
tx_overflow
rx_data
rx_valid
rx_source_ip
rx_source_port
rx_end_of_frame
rx_bad_frame
rx_overrun
[pkt_data]
tx_data
[pkt_valid]
tx_valid
sim_imeg_in
tx_dest_ip
gbe_abx_destip
sim_imeg_in
[a:b]
tx_dest_port
gbe_abx_destport
[pkt_eof]
tx_end_of_frame
1
rx_ack
rx_overrun_ack
gbe_abx
[soft_rst]
edge_detect26
falling edges
active high
Counter26
gbe_abx_down
Counter25
gbe_abx_txof
[soft_rst]
[pkt_eof]
Counter20
pkt_eof

Control Software

All the control software have been developed in Python 2.7 scripts.

System Start Up

To start up the system you need to run a Python script that reads the configurations in a file.

```
oper@fahal:~/MAD4/bee2$ python mad4_start.py --help
Usage: mad_start.py <ROACH_HOSTNAME_or_IP> [options]

Options:
  -h, --help            show this help message and exit
  -p, --skip_prog       Skip FPGA programming (assumes already programmed).
                        Default: program the FPGAs
  -c CONFIG_FILE, --config=CONFIG_FILE
                        Select the Configuration file
  -e, --skip_eq         Skip Default Equalization.  Default: Equalize Amp 3.5
                        and Phase 0 to 18 ant
```

Where the configuration file has these informations:

```
oper@fahal:~/MAD4/bee2$ more configura.conf
[OBSERVATION]
observ_site   = Medicina
antenna_type  = Vivaldi_2.0
ants          = 9
pols          = 2
freq_channel  = 512

[CONFIG_FILE]
pol_h    = mad_pol_h.conf
pol_v    = mad_pol_v.conf
xcorr    = mad_corr.conf
amp_eq   = mad_amp_eq.conf
phase_eq = mad_phase_eq.conf
adc_curve = adc_curve.txt
header = header.conf
rx_network = rx_network.txt
rx_map     = rx_mad_best.txt

[FENG_CONF]
roach_name = feng
katcp_port = 7147
bitstream  = mad_full_corr_2014_May_13_2258.bof
adc_debug  = False
adc_name   = x64_adc
pkt_len    = 0
head_len   = 2048
clock_rate    = 160000000
sample_rate   = 40000000
pfb_size           = 11
pfb_in_signals     = 0
pfb_window         = Hamming
pfb_in_bitwidth    = 12
```

```
pfb_out_bitwidth   = 18
pfb_coeff_bitwidth = 18
pfb_quant_behavior = Round_EvenValues
fft_size           = 11
fft_shift          = 1877
fft_in_bitwidth    = 18
fft_out_bitwidth   = 18
fft_quant_behavior = Round_EvenValues
fft_of_behavior    = Wrap
gbe-0           = gbe_abx
gbe-0_dest_ip   = 3232238347
gbe-0_dest_port = 7200
gbe-0_pkt_len   = 648
gbe-1           = not_used
gbe-1_dest_ip   = 0
gbe-1_dest_port = 7201
gbe-1_pkt_len   = 648
gbe-2           = gbe_beam
gbe-2_dest_ip   = 3232238348
gbe-2_dest_port = 7202
gbe-2_pkt_len   = 648
gbe-3           = gbe_corr
gbe-3_dest_ip   = 3232238348
gbe-3_dest_port = 7203
gbe-3_pkt_len   = 648
```

Example of use:

```
oper@fahal:~/MAD4/bee2$ python mad4_start.py

==================================

Connecting to ROACH board named "feng"...  ok
   Deprogramming FPGAs
   Programming feng with bitstream mad_full_corr_2014_May_13_2258.bof
Writing base_conf...
Setting N ant to  9
Setting Frequency channel to  512

 HPOL  ANT-F  VPOL   ANT-F
------------------------------
 H001    8    V001     7
 H002    15   V002     3
 H003    11   V003     0
 H004    12   V004     16
 H005    13   V005     4
 H006    24   V006     5
 H007    14   V007     2
 H008    9    V008     1
 H009    10   V009     6


 Correlation List (A * Bconj)
-----------------------------------
 V003 <---> V008    H001 <---> H008
 V003 <---> V007    H001 <---> H009
 V007 <---> V002    H009 <---> H003
 V007 <---> V005    H009 <---> H004
```

```
 V007 <---> V006    H009 <---> H005
 V007 <---> V009    H009 <---> H007
 V007 <---> V001    H009 <---> H002
 V003 <---> V004    H001 <---> H006

Starting interface gbe_abx
Set UDP packets destination IP:Port to 192.168.11.11:7200
Set UDP packets size to 648 (64 bit + 1 counter)
UDP packets started!

Starting interface gbe_beam
Set UDP packets destination IP:Port to 192.168.11.12:7202
Set UDP packets size to 648 (64 bit + 1 counter)
UDP packets started!

Starting interface gbe_corr
Set UDP packets destination IP:Port to 192.168.11.12:7203
Set UDP packets size to 648 (64 bit + 1 counter)
UDP packets started!

EQ AMP FLAG:  True , it means the amplitude equalization block is active!

   Calibrating ADC on feng
    SUCCESS: adc sync test returns 1 (1 = ADC syncs present & aligned)
    SUCCESS: adc sync test returns 1 (1 = ADC syncs present & aligned)
    SUCCESS: adc sync test returns 1 (1 = ADC syncs present & aligned)
    SUCCESS: adc sync test returns 1 (1 = ADC syncs present & aligned)
    SUCCESS: adc sync test returns 1 (1 = ADC syncs present & aligned)
 Arming F Engine and setting FFT Shift...  Armed.
Expect trigger at 11:59:01 local (10:59:01 UTC). Updating header BRAM with
t_zero=1512212341
Read from header t_zero=1512212341
Updating header BRAM with fft_shift=1877
Read from header fft_shift=1877

Loading Amp calibration file: eq/default_amplitude_correction.txt
done

Loading phase calibration file: eq/0_phase_correction.txt
done
Antenna  0, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna  1, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna  2, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna  3, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna  4, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna  5, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna  6, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna  7, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna  8, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna  9, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna 10, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna 11, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna 12, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna 13, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna 14, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna 15, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna 16, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna 17, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 18, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 19, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
```

```
Antenna 20, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 21, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 22, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 23, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 24, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna 25, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 26, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 27, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 28, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 29, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 30, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 31, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Modifying calibration coefficients for antenna 0
Modifying calibration coefficients for antenna 1
Modifying calibration coefficients for antenna 2
Modifying calibration coefficients for antenna 3
Modifying calibration coefficients for antenna 4
Modifying calibration coefficients for antenna 5
Modifying calibration coefficients for antenna 6
Modifying calibration coefficients for antenna 7
Modifying calibration coefficients for antenna 8
Modifying calibration coefficients for antenna 9
Modifying calibration coefficients for antenna 10
Modifying calibration coefficients for antenna 11
Modifying calibration coefficients for antenna 12
Modifying calibration coefficients for antenna 13
Modifying calibration coefficients for antenna 14
Modifying calibration coefficients for antenna 15
Modifying calibration coefficients for antenna 16
Modifying calibration coefficients for antenna 17
Modifying calibration coefficients for antenna 18
Modifying calibration coefficients for antenna 19
Modifying calibration coefficients for antenna 20
Modifying calibration coefficients for antenna 21
Modifying calibration coefficients for antenna 22
Modifying calibration coefficients for antenna 23
Modifying calibration coefficients for antenna 24
Modifying calibration coefficients for antenna 25
Modifying calibration coefficients for antenna 26
Modifying calibration coefficients for antenna 27
Modifying calibration coefficients for antenna 28
Modifying calibration coefficients for antenna 29
Modifying calibration coefficients for antenna 30
Modifying calibration coefficients for antenna 31

Connecting to ROACH board named "feng"...  ok

 Writing phase coefficients... done
 Updating phase coefficients on header bram... done

 Writing amp coefficients... done
 Updating amp coefficients on header bram... done
```

Check Antenna Power

There are two Python Scripts to read the Power at ADCs level.

Read Antenna Power

The first script is just to read the dBm Level and Effective Number of Bits used.

```
oper@fahal:~/MAD4$ python mad_adc_pwr.py --help
Usage: mad_adc_pwr.py [options]

Options:
  -h, --help            show this help message and exit
  -c CONFIG_FILE, --config_file=CONFIG_FILE
                        Configuration File [Default: "./configura.conf"]
  -r ROACH_NAME, --roach_name=ROACH_NAME
                        Configuration File [Default: "./configura.conf"]

oper@fahal:~/MAD4$ python mad_adc_pwr.py
Connecting to ROACH board named "feng"...  ok

 HPOL   BITS     dBm    VPOL    BITS     dBm
-----------------------------------------------

 H001   8.03    -8.11   V001    8.14    -7.42
 H002   8.02    -8.18   V002    8.06    -7.97
 H003   7.98    -8.39   V003    8.02    -8.16
 H004   8.05    -8.03   V004    8.03    -8.14
 H005   7.94    -8.61   V005    8.04    -8.05
 H006   7.90    -8.85   V006    7.99    -8.31
 H007   7.91    -8.76   V007    8.01    -8.20
 H008   7.92    -8.72   V008    8.00    -8.25
 H009   8.05    -8.01   V009    8.02    -8.16

 2017-12-02 10:44:55.795170 UTC
```

Interactive Antenna Equalization

The second script can be used also to equalize the level of the signals by changing the digital step attenuator value of the MAD receivers:

```
oper@fahal:~/MAD4/bee2$ python mad_equalize_rx_ch.py

Connecting to ROACH board named "feng"...  ok
Connetion to  192.168.69.1 :5002... ok!
Connetion to  192.168.69.2 :5002... ok!
Connetion to  192.168.69.3 :5002... ok!
Connetion to  192.168.69.4 :5002... ok!


 HPOL   BITS     dBm    RxdB    diff    VPOL    BITS     dBm    RxdB    diff
-------------------------------------------------------------------------------

 H001   7.68    -10.15  10.0    10.2    V001    7.68    -10.10  10.0    10.1
 H002   7.68    -10.15  10.0    10.2    V002    7.70    -9.99   10.5    10.0
 H003   7.65    -10.32  8.0     10.3    V003    7.69    -10.07  9.0     10.1
 H004   7.74    -9.79   10.0    9.8     V004    7.70    -9.98   9.5     10.0
 H005   7.70    -10.03  10.5    10.0    V005    7.69    -10.07  9.0     10.1
 H006   7.71    -9.94   10.0    9.9     V006    7.63    -10.42  10.0    10.4
 H007   7.71    -9.94   9.0     9.9     V007    7.66    -10.22  8.5     10.2
 H008   7.68    -10.12  10.5    10.1    V008    7.70    -9.99   10.0    10.0
 H009   7.71    -9.96   9.5     10.0    V009    7.67    -10.19  9.5     10.2

 2017-12-02 11:29:28.490257 UTC

 Press [h] key to Equalize Pol H
 Press [v] key to Equalize Pol V

 Press [q] key to Exit
```

Load Coefficients

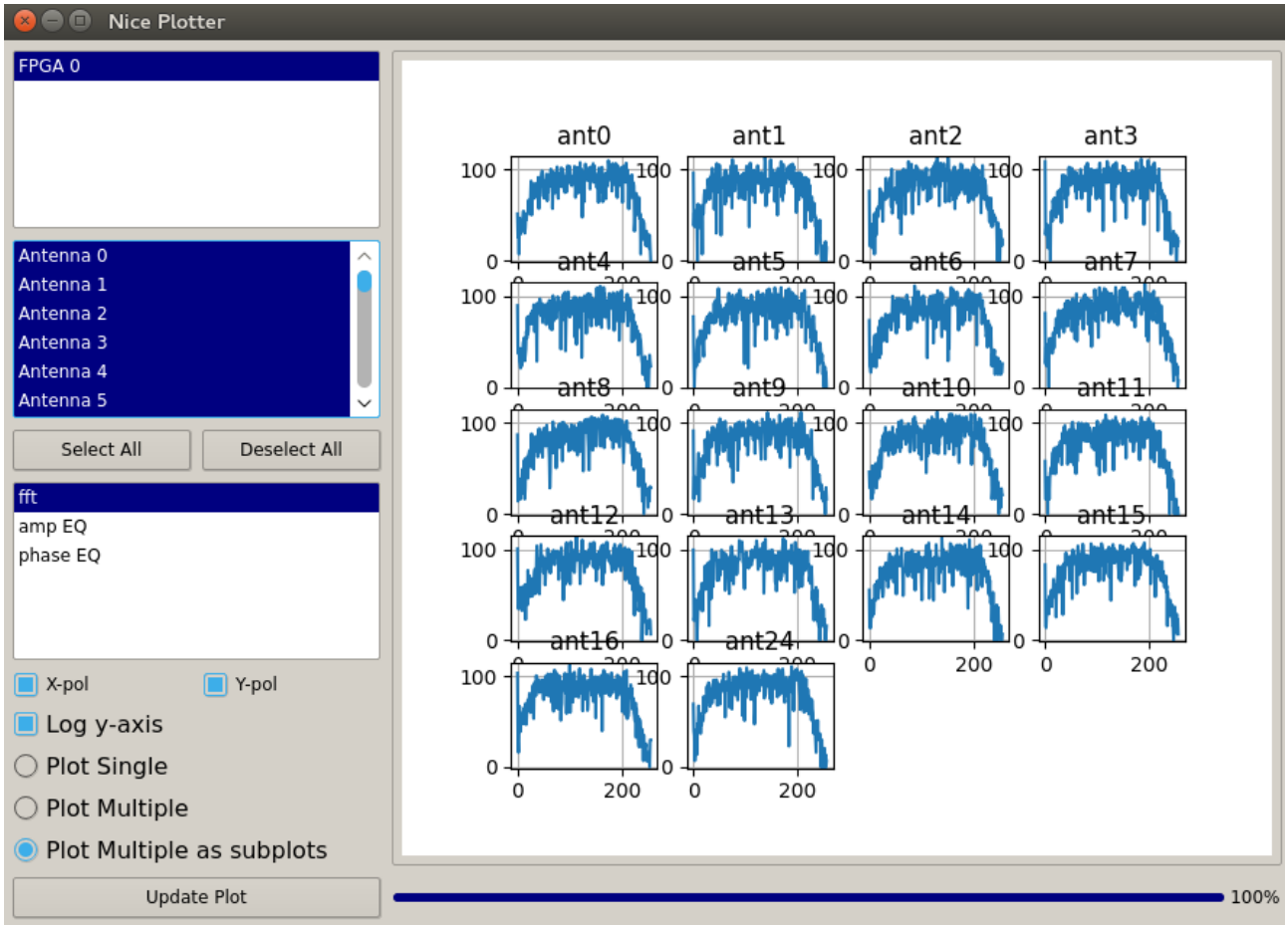
There is a Python script that allows to load Phase and Amplitude coefficients to the backend.

```
oper@fahal:~/MAD4/bee2$ ./mad_eq.py -A eq/default_amplitude_correction.txt -P
eq/0_phase_correction.txt

Loading Amp calibration file: eq/default_amplitude_correction.txt
done

Loading phase calibration file: eq/0_phase_correction.txt
done
Antenna  0, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna  1, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna  2, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna  3, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna  4, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna  5, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna  6, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna  7, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna  8, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna  9, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna 10, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna 11, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna 12, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna 13, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna 14, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna 15, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna 16, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna 17, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 18, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 19, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 20, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 21, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 22, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 23, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 24, (channel 0): AMPLITUDE: 3.50      PHASE (degs): 0.0
Antenna 25, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 26, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 27, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 28, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 29, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 30, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Antenna 31, (channel 0): AMPLITUDE: 0.00      PHASE (degs): 0.0
Modifying calibration coefficients for antenna 0
Modifying calibration coefficients for antenna 1
Modifying calibration coefficients for antenna 2
Modifying calibration coefficients for antenna 3
Modifying calibration coefficients for antenna 4
Modifying calibration coefficients for antenna 5
Modifying calibration coefficients for antenna 6
Modifying calibration coefficients for antenna 7
Modifying calibration coefficients for antenna 8
Modifying calibration coefficients for antenna 9
Modifying calibration coefficients for antenna 10
Modifying calibration coefficients for antenna 11
Modifying calibration coefficients for antenna 12
Modifying calibration coefficients for antenna 13
Modifying calibration coefficients for antenna 14
Modifying calibration coefficients for antenna 15
```

```
Modifying calibration coefficients for antenna 16
Modifying calibration coefficients for antenna 17
Modifying calibration coefficients for antenna 18
Modifying calibration coefficients for antenna 19
Modifying calibration coefficients for antenna 20
Modifying calibration coefficients for antenna 21
Modifying calibration coefficients for antenna 22
Modifying calibration coefficients for antenna 23
Modifying calibration coefficients for antenna 24
Modifying calibration coefficients for antenna 25
Modifying calibration coefficients for antenna 26
Modifying calibration coefficients for antenna 27
Modifying calibration coefficients for antenna 28
Modifying calibration coefficients for antenna 29
Modifying calibration coefficients for antenna 30
Modifying calibration coefficients for antenna 31

Connecting to ROACH board named "feng"...  ok

 Writing phase coefficients... done
 Updating phase coefficients on header bram... done

 Writing amp coefficients... done
 Updating amp coefficients on header bram... done
```

Plot Instantaneous Spectra

It is possible to plot instantaneous spectra by using the script *./snap_mad.py*:



The Spectra plot title will be referring to the Feng antenna name.

Save Raw Data

There is a script developed in C to save UDP raw data packets *./c_save_udp*. The usage is shown in the following example. The function argument is the name of the output file and will be stored in the *data* directory. The beginning of the saved file will contain the Header Bram information such the list of the correlations, the T_Zero time to compute the timestamp for each samples, and all the field described in the Firmware Chapter. The recording will terminate injecting a Keyboard interrupt Key (CTRL+C).

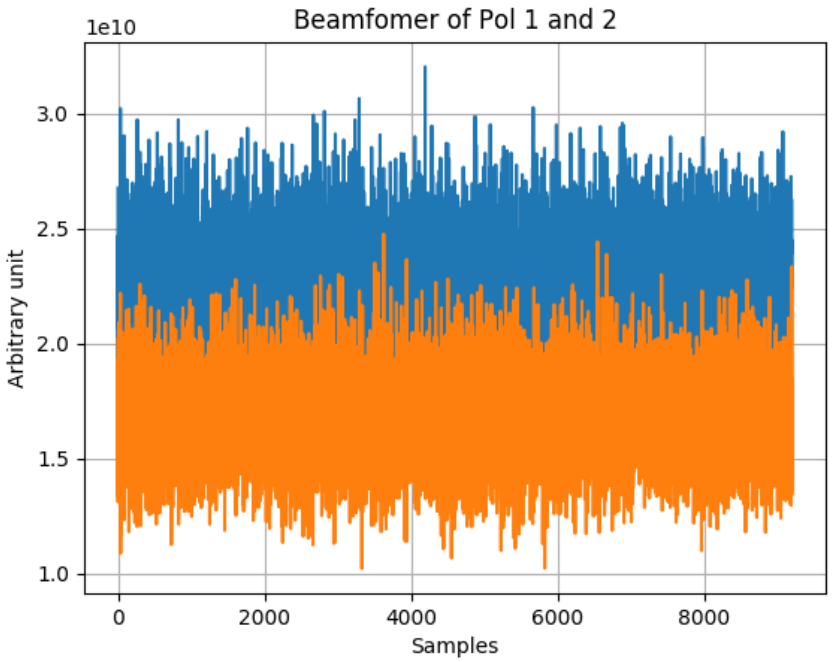
```
oper@fahal:~/MAD4$ ./c_save_udp abx
Server : Socket() successful
Server : bind() successful
./mad_header_fullcorr.py -o data/2017-12-02_152409_NEW_abx.dat
Connecting to ROACH board named "feng"...  Writing file header... done!
Receiving packets from 192.168.11.188:7200
First Counter: 120232

^C
oper@fahal:~/MAD4$ ls -al data/2017-12-02_15
2017-12-02_151225_NEW_test.dat  2017-12-02_152409_NEW_abx.dat
oper@fahal:~/MAD4$ ls -al data/2017-12-02_152409_NEW_abx.dat
-rw-rw-r-- 1 oper oper 49540232 Dec  2 15:24 data/2017-12-02_152409_NEW_abx.dat
```

Plot Raw Data

There are two way to plot the saved data. Using the script *plot_mad.py* you need to specify as function argument what to plot:

```
oper@fahal:~/MAD4$ python plot_mad.py --help
Usage: plot_mad.py [options]

Options:
  -h, --help            show this help message and exit
  -p, --plus_marker     Plot with '+' (plus markers)
  -f FNAME, --fname=FNAME
                        The name of the file to be plotted
  -s INT_SIZE, --int_size=INT_SIZE
                        Integration Size in milliseconds
  -i INDICE, --index=INDICE
                        Select which 16 bits plot (Re-Pol1=0, Im-Pol1=1,Re-
                        Pol2=2, Im-Pol2=3), Default=0
  -b, --beam            Show the beam with power
  -a, --plot_auto       Plot all auto-correlations
  -c, --plot_cross      Plot all correlations
  -x, --plot_complex    Plot real and imaginary components
```

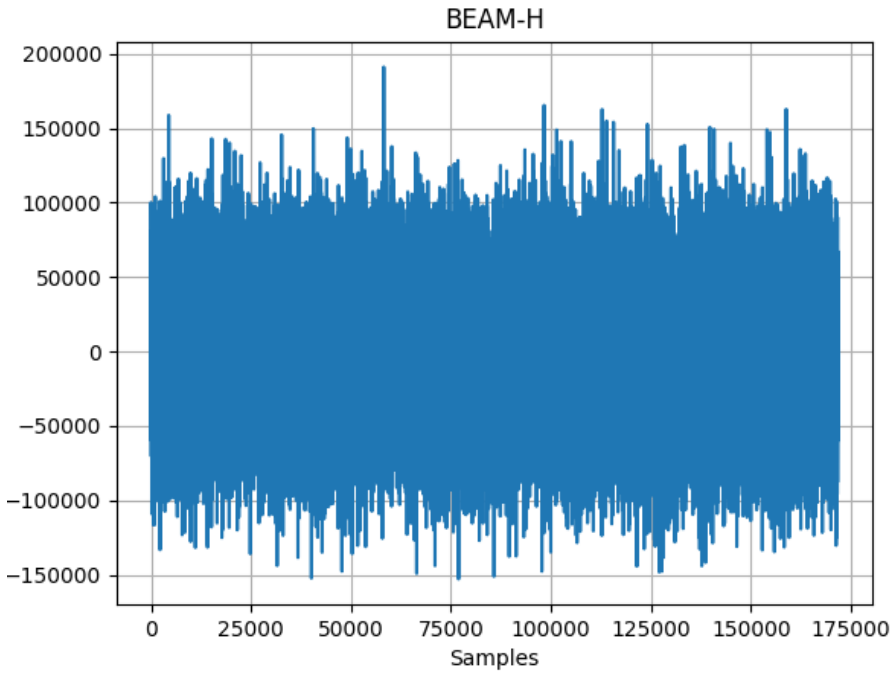
```
oper@fahal:~/MAD4$ python plot_mad.py -b  -f data/2017-12-02_152409_NEW_abx.dat
Reading file:  data/2017-12-02_152409_NEW_abx.dat

Grabbing Header
T zero is  2017-12-02 14:22:19
Header size is  2048
Packet size is  672
100%

Plotting a beam
Process terminated in 0:00:02.342316
```



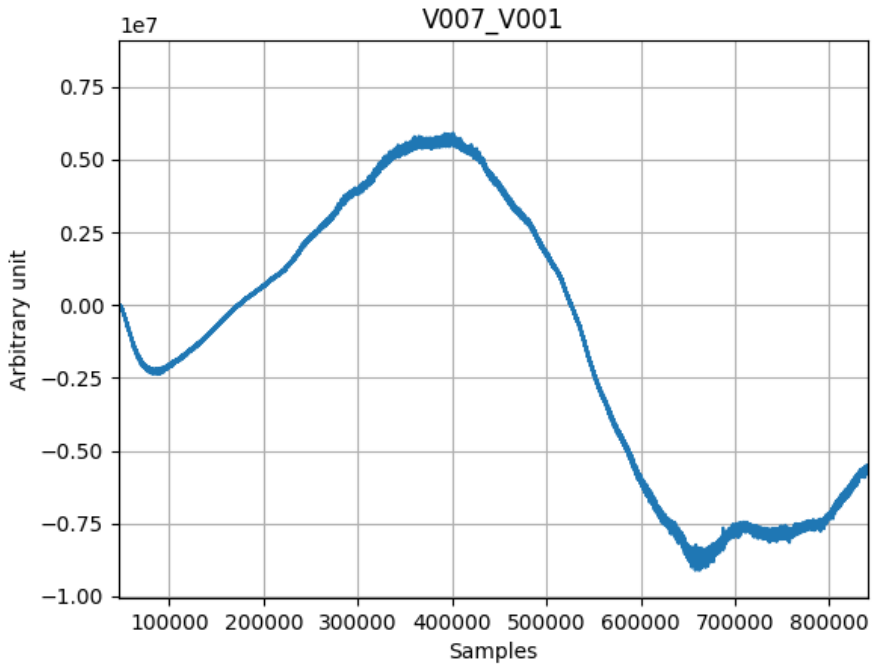
As alternative, you can use an interactive script that shows the list of what is contained in the file:

```
oper@fahal:~/MAD4$ python plot_mad_new.py -f data/2017-12-02_152409_NEW_abx.dat
Reading file:  data/2017-12-02_152409_NEW_abx.dat

Grabbing Header
T zero is  2017-12-02 14:22:19
Header size is  2048
Packet size is  672 (5384 Bytes)
Number of dual pol antenna 9
Number of baselines 16
Retrieving Data Map...
Total elements  36

 0: H001_H001       1: H002_H002       2: H003_H003       3: H004_H004
 4: H005_H005       5: H006_H006       6: H007_H007       7: H008_H008
 8: H009_H009       9: V001_V001      10: V002_V002      11: V003_V003
12: V004_V004      13: V005_V005      14: V006_V006      15: V007_V007
16: V008_V008      17: V009_V009      18: V003_V008      19: H001_H008
20: V003_V007      21: H001_H009      22: V007_V002      23: H009_H003
24: V007_V005      25: H009_H004      26: V007_V006      27: H009_H005
28: V007_V009      29: H009_H007      30: V007_V001      31: H009_H002
32: V003_V004      33: H001_H006      34: BEAM-H         35: BEAM-V
```

```
Select a stream to plot: 34
Plotting BEAM-H (34)...
100%Process terminated in 0:00:02.297335
```



The following picture shows the onboard computed visibility between the antenna #7 and antenna #1 in a UAV flight along the V pol:



UDP packets data format

The Data Set

As explained above the content of UDP packets depends on the correlation products programmed and the number of antenna used. In general, the sequence AUTO-X, AUTO-V, CORR, BEAM-H, BEAM-V is always valid. The following table is an example for an output of a backend configured with 9 dual polarization antennas and 16 Correlation products for both polarizations.

Offset	Size (Byte)	Name	Description
0	8	PKt_Counter	The counter of the packet
8	4	IM_Auto-H001	Imaginary part of the autocorrelation of antenna H001
12	4	RE_Auto-H001	Real part of the autocorrelation of antenna H001
16	4	IM_Auto-H002	Imaginary part of the autocorrelation of antenna H002
20	4	RE_Auto-H002	Real part of the autocorrelation of antenna H002
24	4	IM_Auto-H003	Imaginary part of the autocorrelation of antenna H003
28	4	RE_Auto-H003	Real part of the autocorrelation of antenna H003
32	4	IM_Auto-H004	Imaginary part of the autocorrelation of antenna H004
36	4	RE_Auto-H004	Real part of the autocorrelation of antenna H004
40	4	IM_Auto-H005	Imaginary part of the autocorrelation of antenna H005
44	4	RE_Auto-H005	Real part of the autocorrelation of antenna H005
48	4	IM_Auto-H006	Imaginary part of the autocorrelation of antenna H006
52	4	RE_Auto-H006	Real part of the autocorrelation of antenna H006
56	4	IM_Auto-H007	Imaginary part of the autocorrelation of antenna H007
60	4	RE_Auto-H007	Real part of the autocorrelation of antenna H007
64	4	IM_Auto-H008	Imaginary part of the autocorrelation of antenna H008
68	4	RE_Auto-H008	Real part of the autocorrelation of antenna H008
72	4	IM_Auto-H009	Imaginary part of the autocorrelation of antenna H009
76	4	RE_Auto-H009	Real part of the autocorrelation of antenna H009
80	4	IM_Auto-V001	Imaginary part of the autocorrelation of antenna V001
84	4	RE_Auto-V001	Real part of the autocorrelation of antenna V001
88	4	IM_Auto-V002	Imaginary part of the autocorrelation of antenna V002
92	4	RE_Auto-V002	Real part of the autocorrelation of antenna V002
96	4	IM_Auto-V003	Imaginary part of the autocorrelation of antenna V003
100	4	RE_Auto-V003	Real part of the autocorrelation of antenna V003
104	4	IM_Auto-V004	Imaginary part of the autocorrelation of antenna V004
108	4	RE_Auto-V004	Real part of the autocorrelation of antenna V004
112	4	IM_Auto-V005	Imaginary part of the autocorrelation of antenna V005
116	4	RE_Auto-V005	Real part of the autocorrelation of antenna V005
120	4	IM_Auto-V006	Imaginary part of the autocorrelation of antenna V006
124	4	RE_Auto-V006	Real part of the autocorrelation of antenna V006
128	4	IM_Auto-V007	Imaginary part of the autocorrelation of antenna V007
132	4	RE_Auto-V007	Real part of the autocorrelation of antenna V007

136	4	IM_Auto-V008	Imaginary part of the autocorrelation of antenna V008
140	4	RE_Auto-V008	Real part of the autocorrelation of antenna V008
144	4	IM_Auto-V009	Imaginary part of the autocorrelation of antenna V009
148	4	RE_Auto-V009	Real part of the autocorrelation of antenna V009
152	4	IM_Cross_V003_V008	Imaginary part of the cross correlation V003_V008
156	4	Re_Cross_V003_V008	Real part of the cross correlation V003_V008
160	4	IM_Cross_H001_H008	Imaginary part of the cross correlation H001_H008
164	4	Re_Cross_H001_H008	Real part of the cross correlation H001_H008
168	4	IM_Cross_V003_V007	Imaginary part of the cross correlation V003_V007
172	4	Re_Cross_V003_V007	Real part of the cross correlation V003_V007
176	4	IM_Cross_H001_H009	Imaginary part of the cross correlation H001_H009
180	4	Re_Cross_H001_H009	Real part of the cross correlation H001_H009
184	4	IM_Cross_V007_V002	Imaginary part of the cross correlation V007_V002
188	4	Re_Cross_V007_V002	Real part of the cross correlation V007_V002
192	4	IM_Cross_H009_H003	Imaginary part of the cross correlation H009_H003
196	4	Re_Cross_H009_H003	Real part of the cross correlation H009_H003
200	4	IM_Cross_V007_V005	Imaginary part of the cross correlation V007_V005
204	4	Re_Cross_V007_V005	Real part of the cross correlation V007_V005
208	4	IM_Cross_H009_H004	Imaginary part of the cross correlation H009_H004
212	4	Re_Cross_H009_H004	Real part of the cross correlation H009_H004
216	4	IM_Cross_V007_V006	Imaginary part of the cross correlation V007_V006
220	4	Re_Cross_V007_V006	Real part of the cross correlation V007_V006
224	4	IM_Cross_H009_H005	Imaginary part of the cross correlation H009_H005
228	4	Re_Cross_H009_H005	Real part of the cross correlation H009_H005
232	4	IM_Cross_V007_V009	Imaginary part of the cross correlation V007_V009
236	4	Re_Cross_V007_V009	Real part of the cross correlation V007_V009
240	4	IM_Cross_H009_H007	Imaginary part of the cross correlation H009_H007
244	4	Re_Cross_H009_H007	Real part of the cross correlation H009_H007
248	4	IM_Cross_V007_V001	Imaginary part of the cross correlation V007_V001
252	4	Re_Cross_V007_V001	Real part of the cross correlation V007_V001
256	4	IM_Cross_H009_H002	Imaginary part of the cross correlation H009_H002
260	4	Re_Cross_H009_H002	Real part of the cross correlation H009_H002
264	4	IM_Cross_V003_V004	Imaginary part of the cross correlation V003_V004
268	4	Re_Cross_V003_V004	Real part of the cross correlation V003_V004
272	4	IM_Cross_H001_H006	Imaginary part of the cross correlation H001_H006
276	4	Re_Cross_H001_H006	Real part of the cross correlation H001_H006
280	4	IM_Beam-H	Imaginary part of the BEAM-H
284	4	RE_Beam-H	Real part of the BEAM-H
288	4	IM_Beam-V	Imaginary part of the BEAM-V
292	4	RE_Beam-V	Real part of the BEAM-V

The yellow records on the above table (of length 36 = 9 AutoH +9 AutoV + 16 Correlations + 2 Beam)is a repetitive part on the UDP packet data field. The packet size should be a multiple of this number, the explanation has been given in the previous chapter talking of the packetizer block that sends a new packet counter one clock after the 10 Gbit end of frame signal.

The packet size used for the MAD3 experiment was 648 (words of 8 Bytes, 5184 Bytes) that means there are 648/36=18 blocks of data in each packets. The packet size parameter must not take into account of the packet counter. Since the period of each spectra is 1/(40MHz/2048 (fft real sample))= 51.2 microseconds, the period of one packet is 51.2 *18 =0.9216 milliseconds. Combining this information with the t_zero field of the output file header and the counter of each packet you have the timestamp of each data. The counter of the packet is also useful to know if there are some missing packets.

Offset	Size (word of 64 bit)	Name
0	1	PKt_Counter
1	36	Data set #1
37	36	Data set #2
73	36	Data set #3
109	36	Data set #4
145	36	Data set #5
181	36	Data set #6
217	36	Data set #7
253	36	Data set #8
289	36	Data set #9
325	36	Data set #10
361	36	Data set #11
397	36	Data set #12
433	36	Data set #13
469	36	Data set #14
505	36	Data set #15
541	36	Data set #16
577	36	Data set #17
613	36	Data set #18

The software that will receive data from the UDP socket will have to read 648+1 words of 8 Bytes.

Mapping Antenna/Rx/ADC

The map of the configuration is written in several file.

Firmware Antenna Map

These files create an association between MAD antenna nicknames (Pol+Ant number) and the Firmware antenna Numbering (aka Feng numbers).

```
oper@fahal:~/MAD4$ more mad_pol_h.conf
H001=8
H002=15
H003=13
H004=10
H005=14
H006=24
H007=11
H008=12
H009=9

oper@fahal:~/MAD4$ more mad_pol_v.conf
V001=7
V002=5
V003=0
V004=16
V005=2
V006=6
V007=1
V008=4
V009=3
```

BEST Receivers Map

This file create an association between the name of a BEST Receivers (first column) and the Slave ID for the Carrier RS485 communication (last column).

```
oper@fahal:~/MAD4$ more rx_map_best.txt
1N-1-1 0 0 11
1N-1-2 0 0 12
1N-1-3 0 0 13
1N-1-4 0 0 14
1N-2-1 0 0 21
1N-2-2 0 0 22
1N-2-3 0 0 23
1N-2-4 0 0 24
1N-3-1 1 1 31
1N-3-2 1 1 32
1N-3-3 1 1 33
1N-3-4 1 1 34
1N-4-1 1 1 41
1N-4-2 1 1 42
1N-4-3 1 1 43
1N-4-4 1 1 44
1N-5-1 2 2 51
1N-5-2 2 2 52
1N-5-3 2 2 53
```

```
1N-5-4 2 2 54
1N-6-1 2 2 61
1N-6-2 2 2 62
1N-6-3 2 2 63
1N-6-4 2 2 64
1N-7-1 3 3 71
1N-7-2 3 3 72
1N-7-3 3 3 73
1N-7-4 3 3 74
1N-8-1 3 3 81
1N-8-2 3 3 82
1N-8-3 3 3 83
1N-8-4 3 3 84
```

MAD Receivers Map

Following the previous configuration file here the link between MAD antennas and RX Carrier Slave ID.

oper@fahal:~/MAD4/bee2$ more rx_config.txt

```
H001 8  0 44
V001 7  0 51
H002 15 0 84
V002 5  0 52
H003 13 0 81
V003 0  0 41
H004 10 0 82
V004 16 0 11
H005 14 1 71
V005 2  1 64
H006 24 1 83
V006 6  1 73
H007 11 1 24
V007 1  1 23
H008 12 1 43
V008 4  1 61
H009 9  2 54
V009 3  2 62
```

Network Receivers Map

Here the IPs list of the Receiver carrier boxes.

```
oper@fahal:~/MAD4/bee2$ more rx_network.txt
192.168.69.1
192.168.69.2
192.168.69.3
192.168.69.4
```

Acronyms

ADC	Analog to Digital Converters
BEST	Basic Element for SKA Training
FFT	Fast Fourier Transform
MAD	Medicina Array Demonstrator
PFB	PolyPhase Filter Bank
RX	Receiver
SKA	Square Kilometer Array
UDP	User Datagram Protocol