SKA Project Series
LFAA tile processing module programming manual

G. Comoretto[1], C. Belli[1]

[1]INAF - Osservatorio Astrofisico di Arcetri

Arcetri Technical Report N° 2/2018
14-May-2018

Abstract

The Tile Processing Module is a board performing the data acquisition, channelisation and beamforming for 16 antennas in the SKA LoW Frequency Aperture Array instrument. 16 TPMs are connected using a 40 Gb Ethernet network, to compose a LFAA station with 256 antennas. The LFAA signal processing firmware is composed of modular elements, which are programmed using a standard memory mapped interface. Each module is managed by a software element (plugin), written in Python, which allows a higher level software to manage the board using a standardized set of functions. Here the interface for each module, and the associated plugin, is described for all modules used in the TPM signal processing chain.

1 Introduction

The Italian Tile Processing Module is a module that digitizes the signals form 16 LFAA antennas, combining them digitally into a *tile beam*. 16 iTPMs combine together their tile beams in one station beam, that is sent to the CSP for further processing. In this way 256 antennas and 16 TPMs operate as a single electronically steerable LFAA station.

The iTPM is programmed using a generic control protocol. Each board interfaces with the control system using a 1 GbE interface, with a simple UDP based protocol that allows to read and write a number of memory mapped registers and buffers. A software layer, written in Python, abstracts the control functions for each functional module in the board.

This report describes the programming interface for the portion of the iTPM that deals specifically with the signal processing functionality. For each module both the register level hardware interface and the Python module (plug-in) are described.

Section 2 describes the general structure of the iTPM firmware, of the signal pocessing chain and of the control software. Section 3 and subsequent ones describe the hardware and software interfaces for each module. Section 8 describes the front panel test point interface.

2 General overview of the iTPM firmware

Each iTPM contains two FPGAs, each one processing the signals from 8 antennas, according to the functional diagram in figure 1. iTPM firmware is identical in the two FPGAs. Behavior differences are selected using a dedicated FPGA input pin.

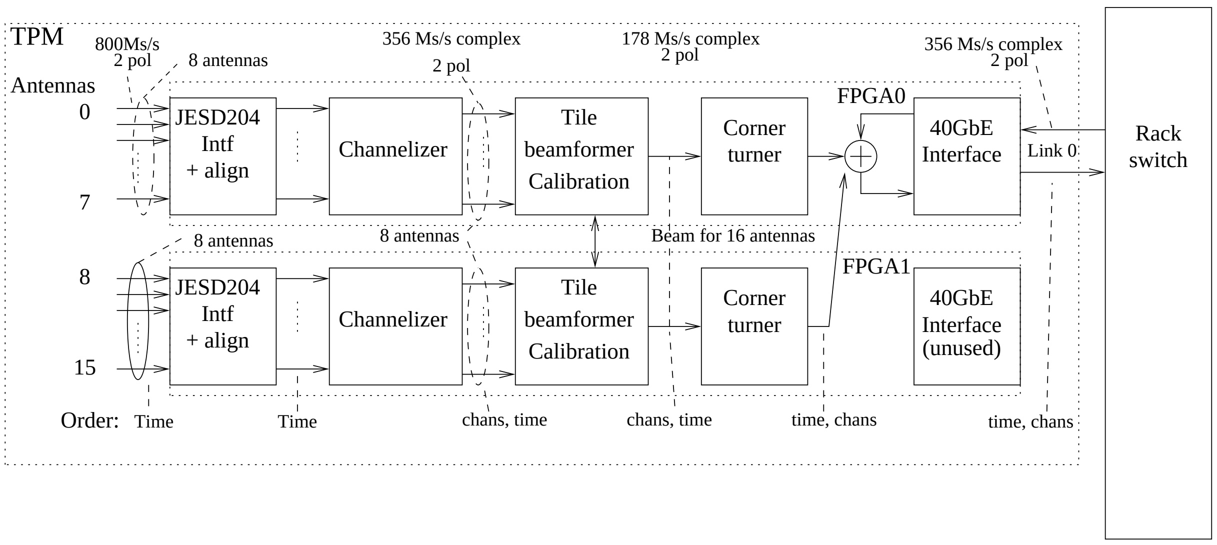

Figure 1: Signal processing in the iTPM

The firmware is composed of 3 concentric firmware layers:

- The board layer, containing board specific functionalities. Changing the board which implements the iTPM should cause changes only in this layer. In reality changes will have to occur also in the I/O ring layer, as this includes hardware specific device interfaces, e.g. with memory or Ethernet interfaces. Some DSP modules are specifically optimized for the Kintex family, but a generic, hardware agnostic version is also available and can be selected using VHDL generics.
- The input/output ring. This contains the physical interfaces to the ADC converter, 10/40 GB Ethernet, FPGA-to-FPGA interconnection, DDR memory and physical sensors. It also contains the AXI4 bridge ad distribution, and the streaming UDP interface towards the 1GB Ethernet port. It also contains clock and timing generation logic. (Figure 2)
- The DSP functionality. This layer is application specific, and contains the channelizer, beamformer, total power meter, and the diagnostic functionalities that allow monitoring and displaying of the signal at various processing stages. (Figure 3)

Each layer is composed of self-contained firmware blocks. Most of the block interface signals consists in a slave AXI4lite bus, for control, clock and reset signals, and data streams that use the AXI4 streaming interface. Specific structured types, common to the whole project, are used to define these busses. A top level, or wrapper module, is used to hold together the low level module (e.g. a proprietary core for a peripheral interface), the AXI4lite control interface and to translate signals from any internal bus format to the used AXI4 stream.

2.1 Control interface

The system is logically and functionally divided into modules. Registers for each module are described in a XML file, that is used in the firmware generation process to actually produce the VHDL code for the interface. These files are combined together to generate a symbol map for the whole board.

A copy of this file is also stored in a ROM included in the FPGA personality, that is used as a symbol table by the the control software to transparently access the registers using only symbolic names. Each register and bit field is identified by a hierarchic name, including the device (e.g. `fpga1`), the module name (e.g. `adc_power_meter`), the register name and optionally the bit field. For example the reset bit for the signal test generator in the first FPGA is symbolically addressed as `fpga1.test_generator.control.reset` This architecture is described in detail in the Pyfabil access layer manual [7].

The low level program software may access each register directly. A higher level of abstraction is implemented by a software module, written in Python, that is accessed using a limited set of functional methods. Modules are included in the main driver software as plug-ins, that are seen as object elements of the `tile` object. Whenever possible, parameters for the plug-in methods are given in physical units. Synchronization is performed on multiples of 256 ADC frames (276.48 μs).

2.2 Input/output ring

The input/output ring structure is described in the block diagram in fig. 2. Most interfaces use the vendor specific cores, enclosed in a top level wrapper that provides synchronization, an interface to the register architecture described in section 2.1, and maps the internal data stream to an AXI4 standard stream. These blocks are described in more detail in the firmware design document[2].

The analog-to-digital converters are interfaced via high speed links using the `jesd204_if_top` module, that includes the vendor specific JESD 204 core, plus the timing registers and state machine for the initial synchronization.

The `ddr_interface_top` module instantiates the vendor DDR interface, that provides DDR timing calibration, and maps its user interface to an `ddr_user` bus, presenting a standard, simple to use, memory interface.

The `f2f_x_top` interfaces use a high speed multiplexed bus, composed of two 18 LVDS lines, to implement a virtual parallel bus of 144+144 lines at 156.25 MHz. The `f2f_muxdemux` module converts this into two monodirectional busses of 105 lines running at 200 Mhz.

The `c2c_axi4lite` and `axi4s_c2c` modules respectively provide an AXI4linte master control interface and an UDP output stream, using the custom `c2c` bus. This in turn interfaces to the 1 GbE board control interface.

The `multi_udp_core` implements the UDP protocol layer for 4 Ethernet links. The `eth10_g_base` provides the Ethernet layer for the 4 10GbE links that compose the 40 GbE interface. In future releases, these will be replaced by a single instance of a 40 GbE Ethernet and UDP layers.

The `clock_manager` manages tha generation and synchronization of all clocks used in the design. The `pps_manager` receives the PPS signal, and synchronizes it to the internal main clock, resolving potential clock ambiguities.

2.3 The signal processing layer

The signal processing algorithm and structure is described in more detail in [6]. Its overall structure is described in figure 3.

Modules in the left column process the digitized samples into station beamformed samples, that are sent over the UDP 10/40 GHz link (`udp_tx_arr` stream in fig. 2 and fig. 3). Modules in the right column are used for monitor purposes. The programming interface is not shown. Each module is interfaced to the global Axi4lite control bus, and is addressed using a symbolic module name plus a register name. Module names are listed in table 1.

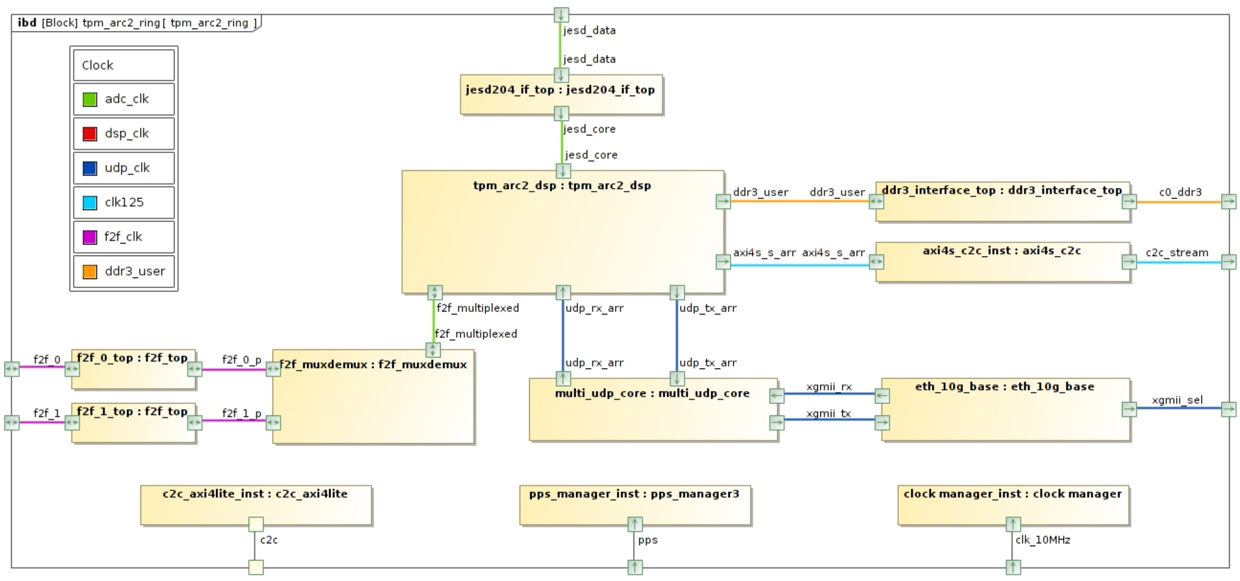
ibd [Block] tpm_arc2_ring [tpm_arc2_ring]
Clock
🟩 adc_clk
🟥 dsp_clk
🟦 udp_clk
🟦 clk125
🟪 f2f_clk
🟧 ddr3_user
jesd_data
jesd_data
jesd204_if_top : jesd204_if_top
jesd_core
jesd_core
tpm_arc2_dsp : tpm_arc2_dsp
ddr3_user
ddr3_user
ddr3_interface_top : ddr3_interface_top
c0_ddr3
axi4s_s_arr
axi4s_s_arr
axi4s_c2c_inst : axi4s_c2c
c2c_stream
f2f_multiplexed
f2f_multiplexed
udp_rx_arr
udp_tx_arr
udp_rx_arr
udp_tx_arr
f2f_0
f2f_0_top : f2f_top
f2f_0_p
f2f_muxdemux : f2f_muxdemux
f2f_1
f2f_1_top : f2f_top
f2f_1_p
multi_udp_core : multi_udp_core
xgmii_rx
xgmii_tx
eth_10g_base : eth_10g_base
xgmii_sel
c2c_axi4lite_inst : c2c_axi4lite
c2c
pps_manager_inst : pps_manager3
pps
clock manager_inst : clock manager
clk_10MHz

Figure 2: Internal block diagram of the input/output ring

Register ID	VHDL Module	Description
`test_generator`	`sample_test_generator`	Test signal generator and delay equalization
`channelizer`	`channelizer2_wrap`	Channelizer equalization
`beamf_fd`	`beamf_fd_wrap`	Tile (frequency domain) beamformer
`beamf_cal`	`beamf_fd_wrap`	Beamformer gain, phase and polarization calibration
`beamf_ring`	`beamf_ring_oa`	Station beamformer
`adc_power_meter`	`total_power_rfi_axi4`	Wideband total power meter and RFI detector
`lmc_gen`	`axi4s_lmc_gen`	Local monitor and control spigot generator
`side_channel_integrator`	`lmc_integrated_gen`	Coarse spectrometer and integrator

Table 1: Register ID for the software modules, and corresponding firmware modules in the DSP ring

ADC samples from the JESD core (`jesd_core` stream in fig. 2 and fig. 3) are optionally substituted with a test signal, and then aligned in the test signal generator. Each input signal is delayed by a fixed number of samples, to compensate for cable mismatch, but not for geometric delay due to beam steering. Aligned samples (`jesd` stream) are then channelized into 512 equispaced frequency channels. Channelization is a fixed operation that requires no parameters. At the channelizer output the signal is rescaled by a power of two, with a granularity of one spectral channel and one antenna. The rescaling is performed by discarding a number of bits in the channelizer result, is the same for all beams, and is used to bring the signal amplitude in a correct range for subsequent operations. Resulting samples are available either as 12+12 bits (`channelizer` stream) or $8 + 8$ bits complex samples (`channelizer_mon8` stream), for monitor purposes.

Tile beamforming requires several parameters, to specify the spectral region(s) being processed, the calibration coefficients for each antenna, polarization, frequency channel and beam, and the delay/delay rate for each antenna and beam. Delay is applied as a phase proportional to the frequency of the channel. Any nonlinear phase is included in the calibration coefficients. Tile beamformed samples are re-quantized to $12 + 12$ bits (`beamformer12` stream). These samples are expanded to 16 bits in the monitor section for practical reasons. Odd and even samples are exchanged across FPGAs using the `f2f` bus (modules `f2f_muxdemux` and `f2f_top` in figure 2).

Samples for each tile beamformer composing a station are added together in the station beamformer,

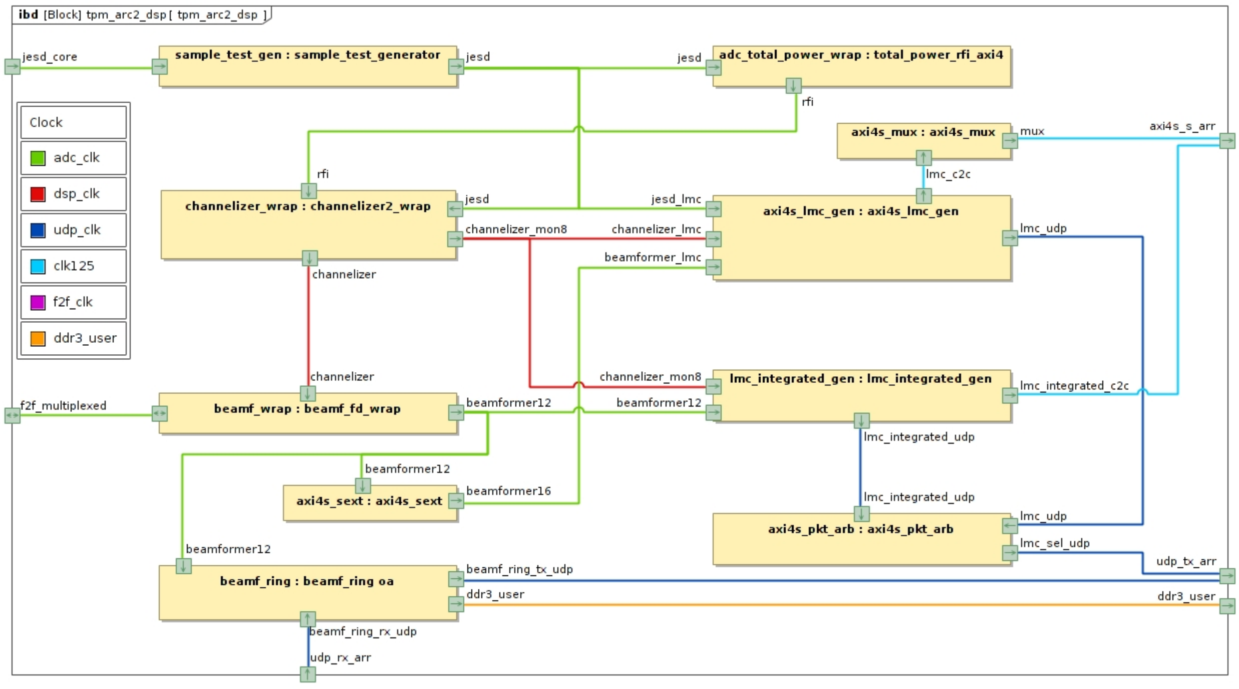

Figure 3: Internal block diagram of the DSP ring

and are then framed in CSP frames. Control points in this module are used to specify frame timing, format, and header content.

Raw samples are monitored in the total power module, that performs also a simple check for RFI presence. The integration time and threshold level for RFI detection are programmable.

A calibration spigot is formed by re-quantizing to 8 bits the samples for all antennas and polarizations and one specific spectral channel, at the channelizer output. These samples are organized in frames, with a SPEAD header (fig. 3 stream `lmc_udp`), and sent over the 40 GbE interface (stream `udp_tx_arr`). Parameters include the channel to be sent, and the start and stop time for the transmission. The destination IP address is specified in the 40 GbE interface. Spigots of the signal at various processing stages is also available for monitor and debug purposes.

Integrated spectrum of channelized and tile beamformed signals can be calculated. The integrator interface includes the signal to be monitored and the integration time. The integrated packets can be sent over the control 1GbE interface or the 40 GbE interface as UDP packets.

A summary of the signal processing sequence and required parameters (in red) is shown in figure 4. Monitor paths are shown in blue, and associated parameters in magenta.

2.4 Unimplemented functionalities

Some functionalities that will be implemented in the final LFAA tile processing are not yet implemented, and the corresponding programming interface has not been studied. This includes:

- Transient buffer. The structure and interfaces for the transient buffer subsystem is still in the definition phase and therefore was not included. Transient buffer data packets will be generated in a module included in the last tile of the station beamformer.
- Antenna sample buffer. An ECP has been raised for a buffer capable of storing a few ms of antenna raw data. If approved, this will be implemented as a separate module, interfaced to the realigned JESD samples stream and to the DDR memory.
- Threshold level adjustment for the RFI detector. The RFI detector is currently fixed at 1.185 (see section 7). The RFI threshold register is implemented in the hardware interface but not used.

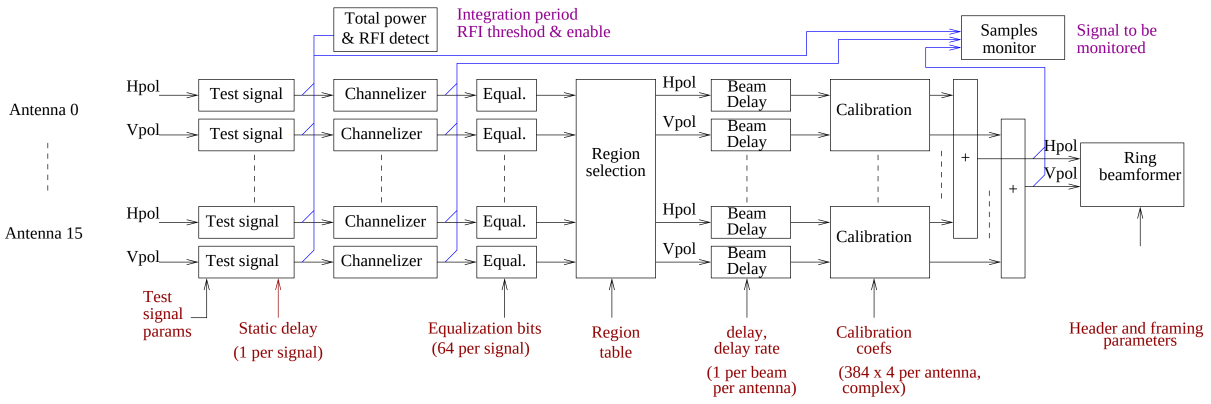

Figure 4: Signal processing parameters

- Using of a single 40 GbE interface in the station beamformer, for both FPGAs. Currently each FPGA uses its 40 GbE interface.

3 Test signal generator

The test signal generator allows the generation of simple test signals, for diagnostic purposes, and delays the resulting signal (physical or test) by an integer number of samples, to compensate for cable mismatch and/or to test the beamformer operations.

The test signal can be substituted to the input signals by selecting a bit mask (one bit per signal). It is composed of up to four components, of adjustable amplitude: two independent tones, a pseudo-random Gaussian noise, and a periodic pulse. Phase of the tones and of the Gaussian noise can be synchronized among FPGAs by specifying a trigger time. The pulses are always synchronized with respect to input frames. The first pulse in a frame will occur at the frame first sample.

3.1 Hardware interface

The module is accessed using the named prefix `fpga1.test_generator`. Registers are listed in table 2.

Frequency is specified in units of $f_s/2^{31}$, i.e. 0.3725 Hz per step at 800 MHz sampling rate. Phase is specified in 2^{16} steps per turn. Amplitude for each component is specified as an 8 bit unsigned, in the range 0–255. The maximum amplitude corresponds to 31.875 ADC units for the two tones, 26.03 ADC units RMS for the pseudo-random noise, and 127 units for the periodic pulses.

The periodic pulse frequency is set according to the following table:

Code	Period	Frequency
0	48	16.666667 MHz
1	72	11.111111 MHz
2	108	7.407407 MHz
3	144	5.555555 MHz
4	216	3.703704 MHz
5	288	2.777778 MHz
6	432	1.851852 MHz
7	864	0.925925 MHz

3.2 Software interface

The test generator is controlled by the `test_generator` plug-in, using the following methods:

```
test_generator.set_tone(dds, frequency, ampl=-1.0, phase=0.0)
test_generator.enable_prdg(ampl=-1.0)
test_generator.disable_prdg()
test_generator.set_pulse_frequency(freq_code, ampl=-1)
test_generator.set_delay(delay)
```

Register ID	Bit width	R/W	Description
`date_code`	32	R	Compile date (8 hex digits, YYYYMMDD)
`control`		RW	General control register, with subfields
`.reset`	1		Global reset
`.load_dds0`	1		Load DDS 0 on trigger event
`.load_dds1`	1		Load DDS 1 on trigger event
`.load_prdg`	1		Load pseudo-random generator seed on trigger event
`.trig_force`	1		Force a trigger event immediately
`.trig_req`	1		Request a trigger event at the specified time
`.pulse_period`	3		Pulse period code
`gain`	32	RW	8 bit gain (0-255) for each of 4 generators
`.dds0`	8		Gain for DDS 0 tone
`.dds1`	8		Gain for DDS 1 tone
`.prdg`	8		Gain for pseudo-random noise
`.pulse`	8		Gain for pulse train
`frequency_0`	32	RW	Frequency for DDS 0
`phase_0`	16	RW	Phase for DDS 0
`frequency_1`	32	RW	Frequency for DDS 0
`phase_1`	16	RW	Phase for DDS 0
`prdg_seed`	32	RW	Seed for pseudo-random noise generator
`channel_select`	16	RW	For each bit set, the corresponding input channel is substituted with the test generator output
`timestamp_req`	32	RW	Time at which the trigger is generated (frames x 256)
`delay_0`	8	RW	Delay in samples for input channel 0 (unsigned, 4 to 255)
`delay_15`	8	RW	Delay in samples for input channel 15

Table 2: Registers for the test signal generator and delay compensation

```
test_generator.channel_select(channel_select)
```

Methods accept frequency in Hz, amplitudes normalized in the range [0.0–1.0], phase in turns. If the amplitude is specified as -1.0 the previous (or default) value is retained. The timestamp, if specified, is expressed in units of 256 input frames. If not specified or set to -1, immediate load of the parameters is assumed (thus without any phase relationship among FPGAs).

Methods `set_tone`, `set_pulse_frequency`, `enable_prdg` set all the parameters for the two tones, the pulse generator and the pseudo-random white noise.

Each ADC channel has a settable delay, in steps of 1 sample and range of -123 to 127 samples. Delay is applied after the generator and is positive for increased delay. Method `set_delay` accepts an array of 16 signed 8 bit integers, one per input signal.

Method `channel_select` accepts a 16 bit mask. For each bit set to 1, the corresponding signal is substituted with the test signal. The test signal is the same for all inputs, but is delayed differently depending on the values specified with `set_delay`.

4 Channelizer

The channelizer operates without any programmable control. It is reset by the global reset signal, and begin producing channelized frames for each input frame, after a fixed filter preload time. Output samples are represented as $20 + 20$ bits complex samples, that must be re-quantized to 12 bits for further processing. As the observed signal has a spectrum that is very far from being uniform (white), the re-quantization must be performed on a per-channel basis. The equalization module allows to discard a variable number of bits, in the range from 1 to 8, and produces a spectrum that is roughly equalized, making the best use of the available 12 bits.

The 12+12 bit samples are further rounded to 8+8 bits for monitor purpose, using a fixed rounding block.

Equalization exponent (number of bits discarded) is specified in a large table, with one entry per frequency

Register ID	Bit width	R/W	Description
`date_code`	32	R	Compile date (8 hex digits, YYYYMMDD)
`block_sel`	5	RW	Memory block selection. LS bit selects which half of the frequency range, other bits select the input signal
`rescale_data`	256x32	W	Array of 256 words, 3 bits used in each word

Table 3: Registers for the equalization block in the channelizer

(512 total) and per signal (16). Table is divided into 32 banks, two per signal. First and second banks refer to channels 0–255 and 256-511 respectively.

4.1 Hardware interface

The module is accessed using the named prefix `channelizer`. Registers are listed in table 3.

Rounding coefficients (bits to be discarded minus 1) are specified in blocks of 256 words (32 bits each, only 3 used). First the `block_sel` register is set, then the 256 values are written to memory. For each signal two consecutive blocks are used, the first for channels 0–255 (0.0–199.4 MHz), and the the second for channels 256-511 (200.0-399.4 MHz). Operation is not synchronous, as it is not expected that these values will be varied during the observation. Only channels 64–447 will be actually used in the observations, so values specified outside this range are irrelevant.

4.2 Software interface

There is not a specific plug-in for this functionality. The software interface is currently implemented in the tile method `set_channeliser_truncation(trunc)`, that sets the truncation uniformly to the given number of bits for all signals and frequency channels. In the future it is expected that the truncation will be managed by the tile beamformer plug-in, as it is formally part of the calibration coefficients applied to the antenna signals.

5 Tile beamformer

The tile beamformer receives the samples from 8 antennas, 2 polarizations, selects a subset of the frequency channels, corrects them using calibration coefficients and beamforms them in one or more beams. Beamforming is performed in three steps:

- A subset of the channelized samples is selected, and a delay correction is applied in the frequency domain (phase shift) to align them to a common station delay center;
- Samples are calibrated for amplitude, phase and polarization errors;
- Samples for 16 antennas (tile) are summed together.

The first two functions correspond to two separate programming interfaces, for the tile beamformer and calibration modules. The tile adder requires no programming.

The tile beamformer is split among the two FPGAs in the TPM, that must be programmed in a consistent way.

5.1 Hardware interface

The tile beamformer module is accessed using the named prefix `beamf_fd`. Registers are listed in table 4. Processing is illustrated in fig. 5.

Channel selection is performed using a set of tables. The mechanism is optimized for hardware implementation, and is described in detail in [4]. A *region selection table* specifies a *region* for each group of 8 output channels. There are 16 regions in total, and up to 384 output channels (48 groups of 8 channels). Each region is associated to a set of contiguous physical spectral channels and to a pointing beam, using the *region offset table* and the *beam table*. The region offset is the difference between the physical channel number and the output channel number (see [4]), and must be even. Up to 8 separate pointing directions (beams) can be specified.

For each antenna `a` and each beam `b`, a physical delay and delay rate is specified in the `delay_<b>_<a>`. For example, `delay_3_2` refers to the setting for antenna 2 of beam 3. Each register is 32 bits long, with the

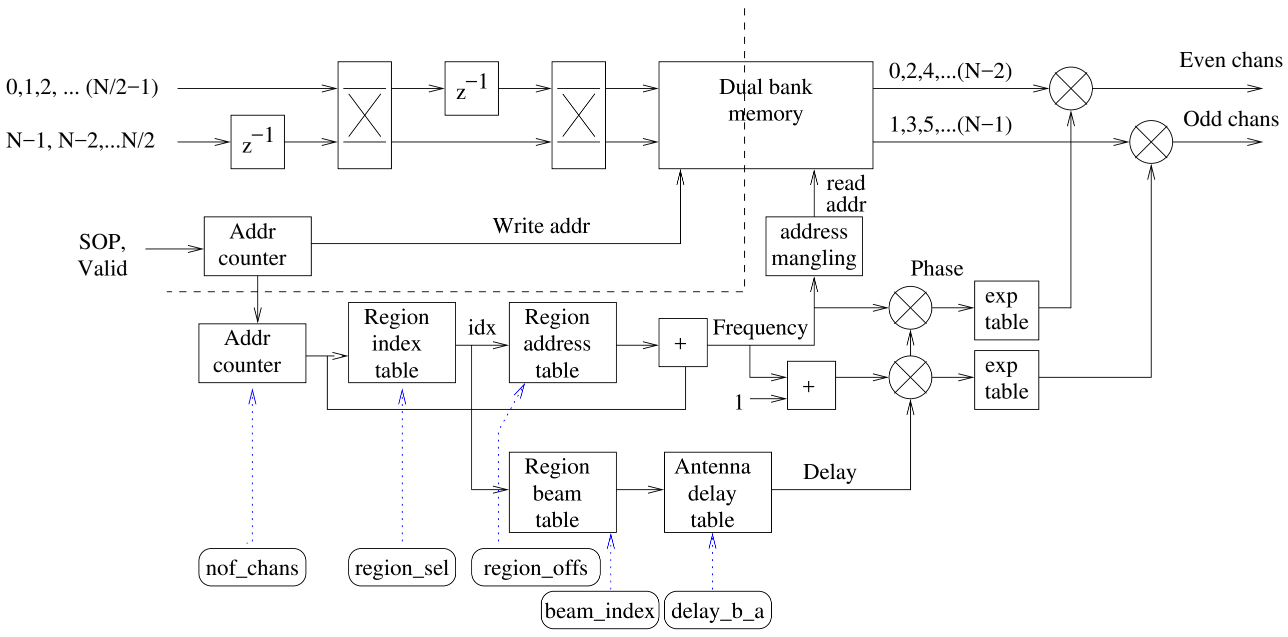

Figure 5: Tile beamformer sample processing. Programming interface registers and tables are shown in the bottom row

upper 20 specifying the delay in hardware units of $1280/2^{23}$ ns (152.6 fs, or 45.7 μm), and the lower 12 the delay rate in units of $(32/27)2^{-47}$ s/s (8.42 fs/s, or 2.54 μm/s).

Delay is specified by writing the delay registers, and actual values become effective when a trigger is generated. This is controlled by pulsing bits `control.load_delay` and `control.load_delay_immediate`. In the first case, the trigger will occur when the frame specified in register `load_time` is received. In the second case it will occur immediately (asynchronously). Delay is adjusted for delay rate every 1024 input frames, starting from the load time.

The calibration module has a separate interface, prefixed by the named prefix `calibration`. Registers are listed in table 5. As for the equalization module, the calibration table is divided into blocks, with 4 blocks per antenna (32 total), each one 512 (384 used) words long.

The total number of channels cannot be changed once the module is initialized, or the synchronization among the two FPGAs is lost. Therefore it is set to 384 by the software plug-in, and when less than 384 channels are used, the unused channels are discarded in the station module. The timing is not guaranteed if the total number of channels exceeds 384, even if in theory it should work up to about 420.

Calibration coefficients are stored as two 16 bit signed quantities, with the real part in the 16 LSB and the imaginary part in the 16 MSB. A unity gain corresponds to the value 1024, for a nominal quantization accuracy of 0.1% in the coefficients. Each table specifies the coefficients for the specified output channels (nominally 384) in output order. The inverse Stokes matrix for one antenna is stored into 4 consecutive tables, one per matrix term. These tables specify the matrix in the order:

0	X polarization direct element
1	X to Y polarization cross element
2	Y to X polarization cross element
3	Y polarization direct element

The calibration coefficients may include any rotation matrix (e.g. the parallactic angle), but do not include the geometric delay, that is specified in the beamformer module interface.

Coefficient memory is dual bank. One bank can be loaded while the other is being used for the actual calibration. Bit `control.cal_table_bank` is used to select between banks. Switching will occur synchronously at time specified by `load_time`, or immediately if bit `control.sw_table_bank` is pulsed.

Register ID	Bit width	R/W	Description
`date_code`	32	R	Compile date (8 hex digits, YYYYMMDD)
`control`		RW	General control register, with subfields
`.reset`	1		Global reset
`.load_delay`	1		Load delay table on trigger event
`.load_delay_immediate`	1		Load delay table, immediate
`load_time`	32	RW	Time at which the trigger is generated (frames x 256)
`current_frame`	32	R	Current frame number, divided by 256
`nof_chans`	6	RW	Total number of channels processed (divided by 8)
`tp_sel`	8	RW	Test point selection
`region_sel`	48×4	W	Region selection table
`beam_index`	16×3	W	Beam index table
`region_off`	16×9	W	Region offset table
`delay_0_0`	$20 + 12$	W	Delay and delay rate for antenna 0, beam 0
`delay_0_7`	$20 + 12$	W	Delay and delay rate for antenna 7, beam 0
`delay_7_7`	$20 + 12$	W	Delay and delay rate for antenna 7, beam 7

Table 4: Registers for the tile beamformer

Register ID	Bit width	R/W	Description
`date_code`	32	R	Compile date (8 hex digits, YYYYMMDD)
`control`		RW	General control register, with subfields
`.cal_table_bank`	1		Calibration bank used
`.sw_table_bank`	1		Force a bank switch immediately
`load_time`	32	RW	Time at which the trigger is generated (frames x 256)
`current_time`	32	R	Current frame number, divided by 256
`block_sel`	5	RW	Memory block selection
`cal_data`	32	W	Calibration coefficients

Table 5: Registers for the calibration module

5.2 Software interface

The Frequency Domain Beamformer plug-in, `beamf_fd`, controlls both the tile beamformer and the calibration modules.

The plug-in initializes the beamformer specifying a single region, in the range 50–350 MHz, with unity calibration matrix, and using a single beam.

The plug-in instantiates the methods:

- `set_regions(region_array)`
- `set_delay(delay_array, beam_index)`
- `load_delay(load_time=0)`
- `load_cal_curve(antenna, beam, cal_coefficients)`
- `load_antenna_tapering(tapering_coefs)`
- `load_beam_angle(angle_coefs)`
- `compute_calibration_coefs()`
- `switch_calibration_bank(load_time=0)`

Regions are set by method `set_regions()`. The region array is a 16×3 matrix, with each row specifying the starting channel, the size (number of channels) and the beam. The starting channel must be even (multiple of 2) and the size a multiple of 8 channels. The beam is an index in the range 0–7. Total number of channels must not exceed 384. If less than 384 channels are used, an extra region is created to reach 384 total channels. Hardware is programmed immediately as there is no need for synchronous operation and the region configuration is never changed during the observation.

Calibration quantities are specified using the following methods:

- `load_antenna_tapering()` specifies one beamforming taper value per antenna. Default is 1.0 for all antennas (uniform tapering). Tapering is the same for all beams. An antenna can be excluded from the beamforming by setting its tapering coefficient to 0.0.
- `load_beam_angle()` specifies a parallactic angle for each beam, in radians. The angle is the same for all antennas and defaults to zero (no parallactic adjustment). This angle is used to rotate the polarization matrix in following the source across its diurnal motion.
- `load_cal_curve()` specifies a calibration curve for one antenna and beam. The calibration curve is specified for 512 frequency points as a 512×4 complex matrix, with each row interpreted as a polarization matrix. Nominal gain is specified as a [1.0 0.0 0.0 1.0] vector.
- `compute_calibration_coefs()` combines together all the above informations, generates the actual calibration coefficients, and loads them in the inactive calibration coefficients memory bank.
- `switch_calibration_bank()` then switches the active and inactive bank, either synchronously, at the specified time (frame number/256) or immediately.
- `set_delay()` specifies the delay and delay rate for a given beam. The `delay` argument is a 8×2 array, with each row specifying the delay and delay rate (in seconds and seconds per second) for one antenna.
- `load_delay()` then loads the delay table, either synchronously, at the specified time (frame number/256) or immediately.

6 Station beamformer

The station beamformer receives the tile beamformed samples, reorganizes them into packets, manages the formation and accumulation of tile beams in one station beam, reformats the final packet in the format required by the CSP and generates the CSP SPEAD header. It is described in [5] and the low level programming interface is described in detail in [8]. The description here refers to the current module version, which uses one separate 40 GbE link per FPGA. The final module will use a single link, in FPGA0, but most of the hardware and firmware interfaces will remain the same.

6.1 Hardware interface

The module is accessed using the named prefix `beamf_ring`. Register usage is quite complex, so it has been splitted into three parts. General timing and control registers are listed in table 6, registers used only in the CSP formatter in table 7, and Monitor and debug registers in table 8.

Register ID	Bit width	R/W	Description
`date_code`	32	R	Compile date (8 hex digits, YYYYMMDD)
`control`		RW	General control register, with subfields
`.reset`	1		Global reset
`.first_tile`	1		Specifies the 1^{st} tile of a station chain
`.last_tile`	1		Specifies the last tile of a station chain
`.error_rst`	1		Resets the error counters
`frame_timing`			Frame temporization
`.int_block_len`	12	RW	Integration block length
`.int_block_ovl`	12	RW	Overlap between successive blocks
`start_frame`	32	RW	First frame in the integration (frames/256)
`last_frame`	32	RW	Last frame in the integration (frames/256)
`current_frame`	32	R	Current frame being processed (frames/256)
`frame_rate.first_tile`	16	RW	Frame rate for the first tile (TPM frames)
`frame_rate.last_tile`	16	RW	Frame rate for the last tile (CSP frames)
`ch_n`	9	RW	Number of processed channels

Table 6: General control and timing registers for the station beamformer

Most registers are initialized in hardware at power-up. The `reset` bit in the control register resets all internal state machines. It is safe to reset the module when it is not running, and it is recommended to do so before starting an integration.

Tiles are organized in a station chain, so the station beamformer must be set up consistently with the routing informations provided to the 40 GbE interfaces. A chain must include a *first tile* and a *last tile*, with each tile in the chain but the last sending 40 GbE packets to the next one. The last chain will send packets to the CSP. First and last tiles are selected by setting the associated bits in the control register. A chain may be composed of just one tile, in which case both bits will be set.

The first tile determines which channels are beamformed and in which order. The total number of channels is specified by the `ch_n` register, and must be even. Each FPGA will process half of these channels, with FPGA0 and FPGA1 processing even and odd channels respectively. Only the first `ch_n` channels from the tile beamformer are processed, and the others discarded.

The first tile assembles *TPM packets*. They are formed by 256 consecutive time samples for 4 frequency channels (4 consecutive odd or even channels in the two FPGAs), 2 polarizations, represented as $16 + 16$ bit complex integers. 8 consecutive TPM packets will end up into 4 *CSP packets*, containing each 2048 consecutive samples for one channel, re-quantized to $8 + 8$ bits. Register `frame_timing` specifies the higher order sequence, expressed in *CSP packets* (or groups of 8 *TPM packets*). The sequencer will continue to produce packets for the same set of channels for a total of `int_block_len` CSP packets, then will switch to the next group of 4 channels, until all channels will be processed. Then it will process another group of time samples, with optionally an overlap of `int_block_ovl` CSP packets.

This flexibility is not currently required. The default setting of `int_block_len=1` and `int_block_ovl=0` prduces a sequence of CSP packets with the same time interval of 2048 samples, for all channels, followed by the next time interval again for all channels.

Packet timing is specified by the `frame_rate` register, and is expressed in oversampled clock cycles. It must be low enough to transmit all required packets in a time interval shorter than the interval spanned by the transmitted samples. Using as a reference one CSP packet (2048 samples), this interval is 2.21184 ms, or 2^{19} clock cycles, in which at least `ch_n` TPM packets must be sent. For 384 channels, the `first_tile` register maximum value is 1365 (5.76 μs). Too a short value would cause packet overrun and Ethernet traffic clogging. Using four 10 GbE links (or one 40 GbE link) for the TPM chain, the packet transmission time is 3.32 μs. A safe value for the `first_tile` register is around 1228 (5.18 μs), or 90% of the maximum available time.

The CSP packets must be sent at slightly more than half this rate, since their number is half of the TPM packet number, and they must be sent before the next group of CSP packets is available. We have chosen a value for `last_tile` equal to 1.85 times that used for `first_tile`. This translates in a packet rate of one every 9.58 μs per FPGA, or 4.79 μs per station.

Packet generation corresponds to the actual LFAA observation, and must start and stop synchronously across the array. Start and stop times are specified in registers `start_frame` and `stop_frame` respectively. Packets are generated from start frame (included) and stop frame (excluded). Specifying a start frame of 1 and a stop frame of `0xffffffff` produces a continuous, never stopping flow of packets.

All timing registers are effective only in the first and last tiles in the chain. All other tiles will operate at the rate of the incoming packets, whatever this will be.

Last packet in the station chain will format the header for the CSP, and requires some extra informations to fill up the header. These are specified using the registers set in table 7.

Register ID	Bit width	R/W	Description
`ref_epoch_lo`	32	RW	Reference epoch (LSB) in Unix time (seconds)
`ref_epoch_hi`	8	RW	Upper 8 bits of reference epoch
`start_time`	32	RW	Start time for frame 0 wrt. ref. epoch (ns)
`frame_id`			
`.antenna_index`	8	RW	Number of antennas in station - 1
`.station_id`	16	RW	Station and substation code
`.sub_array_id`	8	RW	Subarray ID
`freq_beam_tab`	$64 \times X$	W	Table for start frequency and beam index
`csp_scaling`	3	RW	Output rescaling factor (bits discarded)

Table 7: Registers for the station beamformer

Absolute time is specified by a 40 bit reference epoch, in seconds (Unix time format) and a 32 bit offset, in nanoseconds. The sum of these two times correspond to the reference time for the first sample in packet 0. Packet number is multiplied by 1080 ns inside the SPEAD formatting module and added to the start time of each packet.

Frequency is computed from a table of frequency and beam indexes. Each packet is internally tagged with the sequence number of the sample in the tile beamformer output. These are grouped in groups of 8, corresponding to a contiguous block of frequency channels for the same beam. The corresponding frequency is stored in a table, specified in the `freq_beam_tab` for each group of 8 channels. Each entry in this table is 12 bits long, with the upper 3 bits specifying one of 8 beams and the lower 9 the frequency for the first channel in the group (must be even).

Entries in the `frame_id` register are copied verbatim in the SPEAD header. `csp_scaling` register is used to rescale the beamformed 16 bit sum to a 8 bit integer, depending on the number of antennas composing the station.

Some registers, listed in table 8 are used for diagnostics and status reporting. `tp_sel` is used in conjunction with the hardware test point pins to monitor specific internal signals, as described in section 8.

`core_timeout` is used to specify the maximum time before the DDR is assumed not to have responded to a read request, expressed in oversampled clock cycles. It is set in the firmware to a default value, and usually should not be modified.

The `error` register contains some error counts and flags. It is reset by bit `control.error_rst` and should always be zero. The frame error counter counts all frames affected by an error. Each flag is set if one error of the specific type is detected.

Register ID	Bit width	R/W	Description
`core_timeout`	16	RW	Timeout for DDR memory access
`error`		R	Error count register
`.frame_error`	16	R	Frame error count
`.ram_fifo_read_error`	1	R	
`.ram_fifo_write_error`	1	R	
`.spead_fifo_read_error`	1	R	
`.spead_fifo_write_error`	1	R	
`.corner_fifo_read_error`	1	R	
`.corner_fifo_write_error`	1	R	
`tp_sel`	8	RW	Test point select

Table 8: Registers for the station beamformer

6.2 Software interface

The station beamformer is controlled by the Station Beamformer `station_beamf` plug-in, using the following methods:

- `set_first_last_tile(isFirst, isLast)`
- `defineChannelTable(region_array)`
- `define_spead_header(stationId, subarrayId, numAntennas, refEpoch=-1, startTime=0)`
- `set_epoch(epoch)`
- `set_csp_rounding(rounding)`
- `start(time=0, duration=-1)`
- `abort()`
- `current_frame()`
- `is_running()`
- `report_errors()`
- `status_check()`

Timing and other *magic numbers* are set up at initialization.

Before integration start, method `set_first_last_tile()` must be specified for all tiles in the chain, and the 4 methods specifying the SPEAD header parameters and CSP rounding must be called for the last tile. Reference epoch can be set once using `set_epoch()`. In `define_spead_header()` the default value for the epoch leaves the one already programmed. The start time must be specified for the first frame after JESD interface initialization, i.e. for frame 0 in the numbering scheme returned by the `current_frame()` method.

`defineChannelTable()` accepts a 16×3 table, with each line containing (in order) the starting channel, the number of channels and the beam index. This is the same format required for the tile beamformer table.

Once the beamformer is set up, it is started using the `start()` method. Start and stop time are expressed in frames/256, i.e. the same units returned by `current_frame()`. By default integration is started 13.27 ms after the time returned by `current_frame()`, and lasts forever. In normal operation, it must be synchronized among all tiles and FPGAs composing the station, and a stop time must be specified. The stop time represents the first frame that is *not* processed.

If the beamformer is already running, the command is ignored. The `is_running()` method can be used to test this condition, or to verify whether the integration has successfully completed. A running integration can be stopped at any time by the `abort()` method. It immediately stops the generation of new packets by the first tile, but not the propagation of existing partial packets in the beamforming chain.

Status of the error registers can be queried using the `report_errors()` method. It returns two values, respectively for the frame error counter and for the error flag register. The `status_check()` method generates logging information if the frame error counter has a nonzero content.

7 Wide band total power detector

Each input data stream from the JESD interface has a dedicated total power detector. This is used both to actually measure the total power level on the 400 MHz input band, and to detect impulsive RFI.

The total power detector integrates the square of the raw ADC counts over a programmable number of frames. The minimum integration time is 256 input frames (276.48 μs), and the maximum is about 18.12 seconds, with a resolution step of 256 frames. Once started, the total power detector performs contiguous integrations until explicitly stopped. Start and stop is controlled by the bit `tp_run` in the control register.

The RFI detector operates by integrating the total power level over each frame (864 samples), and performing a running average of two frames (fast integrator). At the same time a single pole IIR filter is applied to the total power, producing an effective integration time of 2^{20} samples (1.31 ms, corresponding to a cutoff frequency of the low pass filter of 121 Hz). The result is normalized to 1024 samples, and is therefore $1024/864 = 1.185$ times higher than the value produced by the fast integrator. Without further rescaling, the 18.5% margin is sufficient for the statistical fluctuations in the signal amplitude not to cause spurious RFI flagging. As the input signal may vary for other reasons, the trigger threshold may be varied programmatically.

The RFI detector always counts the number of affected frames (channelized samples) over the same integration interval used by the total power integrator. Sample flagging is enabled independently for each signal, and is disabled at startup. When flagging is enabled, the flag signal is sent to the channelizer, delayed by the appropriate latency time, and applied to the affected channelized samples.

7.1 Hardware interface

When an integration is completed, the results are transferred to the `tp_counts` registers and the bit `ready` in the status register is set. The total power accumulator register is 48 bits long, but in the interface the result is represented by a 32 bit unsigned value. Even at the shortest integration time, the result may exceed 32 bits, and the least significant bits contain only noise. It is possible to select the bits transferred to the output register using the register field `discard_bits`$= d_b$. The actual number of least significant bits discarded is $b = 4d_b + 2$, while the most significant bits that are not copied to the result are tested for an overflow condition. If there is an overflow in one of the input channels, the corresponding bit in the `ovfl` field of the status register is set.

The nominal input RMS level is 19 ADC units. For an input level above 35-40 ADC units the ADC will produce a significant compression and nonlinear terms, and the analog chain will saturate completely around 60 ADC units. The minimum amplitude for meaningful operation is around 8 ADC units. The number of discarded bits is therefore set in order not to produce an overflow with an input level of 60 ADC unit. The minimum and maximum integration time for each value of d_b is reported in table 10. The maximum rounding error due to the finite number of bits in the result is reported, as an increase to the radiometric noise, and is referred to an input level of 8 ADC units. At the longest integration times this approaches 27%, but it must be noted that in this situation the $1/f$ noise is likely to be dominant with respect to the pure radiometric noise (-45 dB).

The RFI detector flags a channelized frame each time the power averaged over two frames, exceeds by a factor of 1.185 the power averaged over 2^{20} samples. The signal however could vary for intrinsic reasons,

Register ID	Bit width	R/W	Description
`date_code`	32	R	Compile date (8 hex digits, YYYYMMDD)
`control`		RW	General control register, with subfields
`.reset`	1	RW	Global reset
`.tp_run`	1	RW	Global run/stop
`.rfi_enable`	16	RW	RFI flagging enable
`.clear_ready`	1	RW	Clear ready flag
`.discard_bits`	2	RW	Scaling (number of bits to discard) in TP result
`status`		R	Status register
`.ready`	1	R	Total power readout available
`.ovfl`	16	R	Overflow flag (1 per input channel)
`integration_time`	16	RW	Total power integration time (frames/256)
`rfi_factor`	16	RW	Guard factor for RFI
`tp_counts`	16×32	R	Total power readouts
`rfi`	16×32	R	RFI affected frame counts

Table 9: Registers for the total power and RFI detector

d_b	Bits dropped	Min frames	Max frames	Max time	Noise incr.
0	2	1	21	6 ms	0.5%
1	6	22	345	95 ms	1.7%
2	10	346	552	1.53 s	6.8%
3	14	5524	65535	18.1 s	27%

Table 10: Integration time and excess quantization noise as a function of number of discarded bits

and therefore it is necessary to specify a different value. The `rfi_factor`= R_f multiplies the long term integration value, with a nominal value of 2048, before the comparison. The threshold for considering a particular frame affected by RFI is thus $1.185 R_f / 2048$. This feature is not yet implemented, i.e. the threshold is always set to 1.185, irrespective to the value specified in the `rfi_factor` register.

Each time the running mean for and input signal exceeds the threshold, a RFI event is generated for that antenna, and counted in the corresponding RFI event register. The register counts the events over the same integration period specified for the total power integration.

For each signal, if the corresponding bit in the `rfi_enable` field is set, the event causes the corresponding channelized sample to be flagged as invalid. Flagging is destructive (the original sample value is lost) and propagates to the corresponding beamformed sample. Even one flagged antenna in the station invalids the whole station. Therefore if some antennas are not used in the beamforming, the corresponding `rfi_enable` bit must be cleared.

7.2 Software interface

The total power plug-in has the following method. It is accessed with the symbolic name `adc_power_meter`. The most useful implemented methods are the following:

- `set_intTime(intTime)`
- `start_IntTime(intTime)`
- `start_TP(), stop_TP()`
- `enable_RFI(mask), disable_RFI()`
- `read_TpData()`
- `get_RmsAmplitude()`
- `read_RfiData()`

Other methods are used internally, but are accessible to the user.

- `initialise()`
- `wait_TpReady()`
- `wait_TpData()`

In all methods the integration time is specified in seconds, and all amplitudes are reported either in ADC units (amplitude) or units squared (power). Here below are presented more details about the methods listed:

- `set_intTime(intTime)` sets the integration time, without modifying the running status. If the integration is running the old integration time is not modified.
- `start_IntTime(intTime)` stops the current integration, if running, modifies the integration time and restarts the integration with the newly specified time.
- `start_TP()` and `stop_TP()` respectively starts and stops the total power integrator. Synchronous start is not supported yet.
- `enable_RFI(mask)` enables RFI flagging for the specified input channels. If RFI detects a signal power above the threshold level in one of the enabled input signals, the corresponding channelized samples are flagged. Masking is used not to generate RFI flagging for antennas that are not used in the beamforming. `disable_RFI()` is equivalent to using a mask of all zeros, effectively disabling the flagging operation.
- `wait_TpData()` waits for the total power data to become ready and returns them. Returns an empty list if the total power integrator is not running, and total power data from a previous integration is not available. Returns an empty list if the total power integrator is not running. `get_RmsAmplitude()` operates in the same way, but returns the RMS amplitude instead of the power.
- `read_RfiData()` Returns the number of RFI affected frames in the previous integration period.
- `initialise()` is automatically called at startup. The integration time is set to 10 ms, the total power integrator is started and RFI flagging is disabled.
- `wait_TpReady()` If the integrator is running, waits for data to become ready, returning `True`. Else returns `False`.
- `read_TpData()` reads the total power data if these are available, returning a list with 16 floating point elements, else returns an empty list.

8 Test points

Test point	Clock domain	Signal
0	ADC	PPS pulse
1	ADC	syncronization pulse
2	ADC	ADC reset (cross domain)
3	ADC	`jesd` stream `tvalid`
4	ADC	`jesd` stream `tlast`
5	DSP	Channelizer stream `tlast`
6	DSP	Tile beamformer `tvalid`
7	DSP	Tile beamformer `tlast`
8	UDP	DDR3 bus Request (to DDR)
9	UDP	DDR3 bus Read/NWrite (to DDR)
10	UDP	DDR3 bus Ready (from DDR)
11	UDP	DDR3 bus Data Ready (from DDR)
12	DSP	Station beamformer Test point 0
13	DSP	Station beamformer Test point 1
14	ADC	Tile beamformer Test point 0
15	ADC	Tile beamformer Test point 1

Table 11: Signal processing ring test points

Each FPGA has two dedicated lines that are connected to a total of 4 posts on each TPM front panel. These lines are used to directly monitor a number of internal signals, that are selected using dedicated registers. As these lines are used only for debug purposes, there is no plug-in associated with them.

Test points are selected at a global level using register `fpga<x>.regfile.tp_sel`. Test point pins 0 and 1 are selected by bits 3–0 and 7–4 respectively, according to table 11. There are 16 test points at the global level. Four of them connected to the test point outputs of the tile and station bamformer modules, that are in turn selected using a module specific, internal test point register. Each test point pin can be associated with any of the 16 internal test point signals. Test points are synchronized to the clock domain they belong to. There are mainly three clock domains: the ADC domain, at 200 MHz, the oversampled DSP domain, at 237.04 MHz, and the Ethernet and I/O ring domain, at 156.25 MHz.

Stream `tvalid` and `tlast` allows to follow the packets as they are being processed by the various modules. The `tvalid` for the input stream should become valid as soon as the JESD interface is initialized. The `tlast` pulses should be aligned among all TPMs in the system. The DDR bus bits can be used to verify DDR timing and bandwidth utilization.

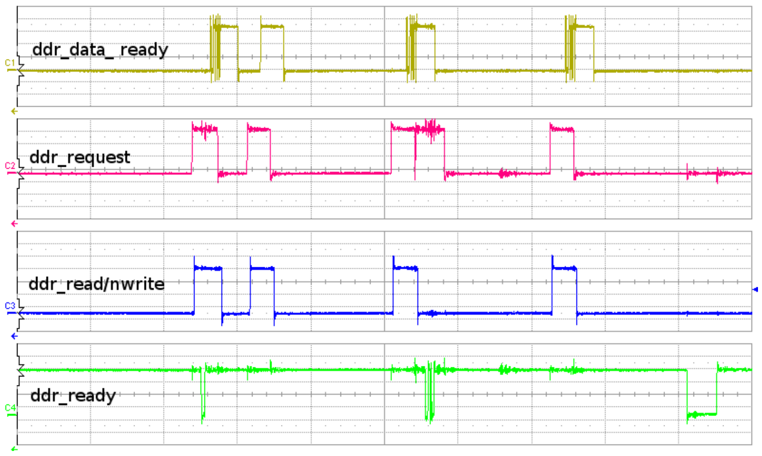

Figure 6: Example of test point use, showing DDR control lines

An example of the test point use is shown in figure 6. The test points shown are relative to the DDR bus. Signals `ddr_data_ready` and `ddr-request` are selected on FPGA1, and signals `ddr_read_nwrite` and `ddr_ready` on FPGA 2. Horizontal scale is 500 ns/division. As the two FPGAs are synchronized, these signals are very similar, with FPGA2 slightly delayed (about 20 ns) with respect to FPGA 1. The plot captures four memory read operations, with a memory write operation after the first read. An autorefresh cycle is also visible on the `ddr_ready` line.

The plot shows that each memory burst (256 DDR accesses) lasts 160–200 ns, that the memory access time is about 100–150 ns, and that the overall access efficiency is around 90% (900 ns for 1280 accesses at 1600 MHz).

8.1 Tile and station beamformers test points

Station and tile beamformers provide two test point signals each, selected with the same mechanism among 16 internal monitor points. Selection is performed using the same mechanism, specifying bits 3–0 and 7–4 respectively of the module `tp_sel` register.

Tile beamformer test points are listed in table 12.

The input bank selector switches as each time a new frame is received from the channelizer. Valid and start/end of frame bits can be used to follow the packets being processed. The F2F bits can be used to verify the correct working of the bus connecting the two FPGA, and its latency.

Station beamformer test points are listed in table 13.

DDR write and DDR read triggers are the pulses that start a DDR block write or read, respectively. Read and write requests are the actual requests to the DDR, valid for the whole burst. The `tvalid` and `tlast` signals monitor the response form the DDR to a read request.

The two FIFOs are used at the input of the station chain adder, respectively for the partial arriving downstream and for the tile samples stored locally. FIFO full bits should always stay false. Empty bits are false if some data is received in one of the two buffers.

Test point	Signal
0	Input bank select: square wave at half frame period
1	Negated reset
2	Memory strobe for block memory read
3	Ping pong status bit (output bank select)
4	Region selector output start-of-packet
5	Region selector output `tvalid`
6	Calibration block input `tlast`
7	Calibration block input `tvalid`
8	Calibration block output start-of-packet
9	Calibration block output `tlast`
10	Calibration block output `tvalid`
11	F2F bus input bit 48 (`tvalid`)
12	F2F bus input bit 49 (`tlast`)
13	F2F bus output bit 48
14	F2F bus output bit 49
15	Output bus valid signal

Table 12: Tile beamformer test points

The core state machine is normally in the *Idle* state (bits 13-12 = 00). On the arrival of a SPEAD packet it goes to state *Wait DDR*, then to state *Summing* when the corresponding packet is read from DDR. It cycles through these states until the whole packet has been processed, briefly passes through state *Done* and returns to *Idle*.

Test point	Submodule	Signal
0		DDR write trigger
1		DDR start read trigger
2		DDR read bus `tvalid`
3		DDR read bus `tlast`
4		Frame counter bit 0
5		Frame counter bit 1
6		Read request
7		Write request
8	Core	SPEAD input FIFO full
9	Core	SPEAD input FIFO empty
10	Core	DDR input FIFO full
11	Core	DDR input FIFO empty
13-12	Core	state machine: 1=Wait DDR, 2=Summing, 3=Done, 0=Others

Table 13: Station beamformer test points

List of acronyms

ADC: Analog to Digital Converter

AXI4: Advanced eXtensible Interface 4: standard digital bus protocol

ADU: Analog to Digital Unit: the amplitude of one ADC quantization step

CSP: Central Signal Processor

DDR: Double Data Rate: Implementations of DRAM using both clock edges for data transfer. DDR3 and DDR4 versions of the standard are used in the design

DDS: Direct Digital Synthesizer (oscillator)

DSP: Digital Signal Processing

ECP: Engineering Change Proposal

EMC: Electromagnetic Compatibility

EMI: Electromagnetic Interface

ENOB: Equivalent number of bits

F2F: FPGA to FPGA (bus, interconnection)

FFT: Fast Fourier Transformation

FIFO: First-in first-out buffer

FPGA: Field Programmable Gate Array

GbE: Giga Bit Ethernet

HDL: High Level Design Language

ICD: Interface Control Document

IICD: Internal Interface Control Document

I/O: Input/Output

IP: Internet Protocol

iTPM: Italian Tile Processing Module

JESD204: JEDEC Standard 204 for ADC serial interface

LFAA: Low Frequency Aperture Array Element or Consortium

LSB: Least significant bit

MATLAB: MATLAB simulation language and application

M&C: Monitor and Control

MSB: Most significant bit

PPS: Pulse per second

RFI: Radio Frequency Interference

RMS: Root mean square

ROM: Read Only Memory

SDP: Science Data Processing

SDRAM: Syncronous Dynamic Random Access Memory: Standard for bursting, fast memory. DDR3 and DDR4 implementations of SDRAM are used in the design

SKA: Square Kilometre Array

SKAO: SKA Organization (or office)

SPEAD: Standard Protocol for the Exchange of Astronomic Data

SW: Software

TBC: To be confirmed

TBD: To be decided

TPM: Tile Processing Module

UDP: User Datagram Protocol: standard Internet protocol

VHDL: VLSI High Level Design Language

WBS: Work Breakdown Structure

XML: eXtensible Markup Language

References

[1] UltraScale Architecture-Based FPGAs Memory Interface Solutions v6.0, *LogiCORE IP Product Guide*, Vivado Design Suite, **PG150** (2014).
`http://www.xilinx.com/support/documentation/ip_documentation/mig/v6_0/pg150-ultrascale-mis.pdf`

[2] R. Chiello: “iTPM Firmware Architecture”, LFAA document xx (2018)

[3] G. Comoretto: “LFAA timing and observation sequence”, in preparation (2015).

[4] G. Comoretto: “LFAA Tile Beamformer structure”, INAF - Osservatorio Astrofisico di Arcetri Internal Report no. 2/2015

[5] G.Comoretto: “LFAA Station Beamformer structure”, INAF - Osservatorio Astrofisico di Arcetri Internal Report no. 5/2016

[6] G. Comoretto, R. Chiello, M. Roberts, R. Halsall, K. Zarb Adami *et al.*: “The Signal Processing Firmware for the Low Frequency Aperture Array”, Journal of Astronomical Instrumentation, (2017),

[7] A. Magro, R. Chiello *et al.*: “A Software Infrastructure for Firmware-Software Interaction: The Case of TPMs”, ICSIGSYS 2017 conference

[8] E. Zaccaro: “CES Station Beamformer top level description”, Campera Electronic Systems internal report, 2018-03-10

[9] “SKA1 LFAA to CSP ICD”, SKA Office document no. SKA-TEL-SKO-0000142 (2015)

[10] J. Manely, M.Welz, A.Parson, S. Ratcliffe, R. van Rooyen: “SPEAD: Streaming Protocol for Exchanging of Astronomical Data”, SKA-SA internal memo, 2010/10/07

Contents

List of Tables

List of Figures