iTPM Firmware Architecture

DOCUMENT CHANGE RECORD

Author	Affiliation	Issue	Date	Paragraphs affected	Change information
R. Chiello	University of Oxford	1.1	18/04/2018	All	New document
R. Chiello		1.2	18/05/2020	All	Draft watermark removed

TABLE OF CONTENTS

1. SCOPE 5

2. ACRONYMS 6

3. REFERENCE DOCUMENTS 7

4. ITPM OVERVIEW 8

5. ITPM BLOCK DIAGRAM AND FUNCTIONS 10

5.1 Management and Control Plane 11
5.1.1 CPLD Firmware Architecture 12
5.1.1.1 CPLD UDP Core 13
5.1.1.2 CPLD Wishbone Internal Bus and Slaves 13
5.1.1.3 Memory Mapped Access 15
5.1.1.4 C2C Streaming interface 15
5.1.1.5 CPLD Clock and Reset 16
5.1.2 CPLD Bitfile Management 17
5.1.2.1 CPLD boot procedure 17
5.1.2.2 CPLD bitfile update procedure 17
5.1.3 CPLD Data Rates 20
5.1.4 CPLD FW Limitations 20

5.2 Acquisition and processing plane 21
5.2.1 FPGA Firmware Architecture 22
5.2.2 FPGA Clock and Reset Tree 23
5.2.2.1 Clock Tree 23
5.2.2.2 Reset Tree 25
5.2.3 FPGA I/O Ring 26
5.2.3.1 C2C AXI4Lite Bridge 26
5.2.3.2 AXI4Lite Interconnect 27
5.2.3.3 AXI4Lite Register File 27
5.2.3.4 AXI4Lite BlockRAM 27
5.2.3.5 JESD Interface 27
5.2.3.6 F2F - FPGA to FPGA Link 28
5.2.3.7 40G 30
5.2.3.8 DDR3 IF 30
5.2.3.9 AXI4S C2C Bridge 31
5.2.3.10 Clock Manager 31
5.2.3.11 PPS Manager 31
5.2.3.12 System Monitor 31
5.2.3.13 AXI4Lite DRP Bridge 32
5.2.4 DSP Section 32
5.2.4.1 AXI4Lite Interconnect 33
5.2.4.2 Sample Test Generator 33
5.2.4.3 Channelizer Wrapper 34
5.2.4.4 Tile Beamformer Wrapper 36
5.2.4.5 Station Beamformer Wrapper 37
5.2.4.6 Total Power RFI 37
5.2.4.7 AXI4S Packet Arbiter 38
5.2.4.8 AXI4S LMC Generator 38

5.2.4.9 AXI4S LMC Integrated Generator 40
5.2.5 LMC SPEAD Data Format 41
5.2.6 CSP SPEAD Data Format 42
5.2.7 iTPM Synchronization 48
5.2.7.1 Local synchronization 48
5.2.7.2 PPS Synchronization 48
5.2.7.3 Time-tagging 50
5.2.7.4 Command Synchronization 50
5.2.8 FPGA FW limitations 51

6. AXI4LITE MEMORY MAP 53

6.1 XML Memory Map Description 53

6.2 Automatic generation of VHDL code 54

6.3 Slave logic generation 54

6.4 Interconnection logic generation 55

6.5 Global XML memory map 56

7. FIRMWARE REPOSITORY AND BITFILE COMPILATION 57

7.1 Modules 57

7.2 Designs 58

7.3 FPGA Bitfile compilation 58

8. ITPM OPERATIONAL STATES 61

8.1 Initialization Procedure 61

9. FPGA RESOURCE UTILIZATION 65

10. FPGA POWER CONSUMPTION 65

1. SCOPE

Purpose of this document is to describe the architecture of the firmware implemented on the Italian Tile Processor Module (iTPM), providing information about the firmware components, main functions, interfaces and SW procedures needed to configure and initialize the board. The firmware repository structure is also presented along with description of the automated process to generate an FPGA bitfile. Lastly, implementation results including FPGA resource utilization and power consumption are reported.

The document is organized as follows: Section 4 provides an overview of the iTPM board from the hardware point of view, in Section 5 the iTPM block diagram is presented allocating the on-board devices into two distinct functional groups, namely the Monitor and Control section and the Acquisition and Processing section. The programmable components in each of the above groups, i.e. CPLD and FPGA, are then analyzed in more detail. Section 6 describes the procedure and tools to generate the FPGA memory map and related VHDL design files implementing the required register interfaces. Section 7 outlines the firmware repository structure and bitfile compile process. In section 8 the required procedure to configure and initialize the iTPM is described. Finally Sections 0 and 10 presents implementation results.

2. ACRONYMS

AD	Analog to Digital
ADC	Analog to Digital Converter
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
COTS	Commercial Off The Shelf
DDR	Double Data Rate
EEPROM	Electrically Erasable PROM
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
HW	Hardware
IF or I/F	Interface
IO or I/O	Input/Output
IP	Intellectual Property
ITPM	Italian Tile Processor Module
MAC	Media Access Control
Mbps	Megabits per second
MIPS	Millions of Instruction Per Second
PLL	Phase Locked Loop
PPS	Pulse per second
QSFP	Quad Small Form-factor Pluggable
RAM	Random Access Memory
SDRAM	Synchronous DRAM
SPI	Serial Peripheral Interface
SRAM	Static RAM
STD	Standard
SW	Software
SKA	Square Kilometre Array
TBC	To Be Confirmed
TBD	To Be Defined
UDP	User Datagram Protocol

3. REFERENCE DOCUMENTS

The following documents are referenced in this document. In the event of conflict between the contents of the referenced documents and this document, **this document** shall take precedence.

Ref ID	Issue
[RD1] www.analog.com/en/products/analog-to-digital-converters/high-speed-ad-10msps/ad9680.html	
[RD2] JEDEC STANDARD Serial Interface for Data Converters JESD204B	JULY 2011
[RD3] www.analog.com/en/products/clock-and-timing/clock-generation-distribution/clock-generation-devices/ad9528.html	
[RD4] www.xilinx.com/products/technology/ultrascale.html	
[RD5] www.latticesemi.com/Products/FPGAandCPLD/MachXO2	
[RD6] www.latticesemi.com/Products/PowerAndClockDevices/PlatformManager2	
[RD7] http://www.analog.com/en/products/clock-and-timing/clock-generation-distribution/ad9550.html	
[RD8] UDP/IPv4 FPGA command protocol - Harro Verkouter - verkouter@jive.nl	1.2
[RD9] SKA Italian Tile Processor Board User Manual – Sandro Pastore	1.1
[RD10] www.sanitaseg.com/project/gigabit-udpipmac/	
[RD11] MachXO2™ Family Data Sheet DS1035 Version 3.3, March 2017	
[RD12] WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores	B.3
[RD13] Tile Processing Module AAVS1 Firmware Power Report	1.0
[RD14] Xilinx PG150 - UltraScale Architecture-Based FPGAs Memory Interface Solutions	6.0
[RD15] ARM AMBA AXI Protocol v2.0 Specification	2.0
[RD16] AMBA4 AXI4-Stream Protocol v1.0	1.0
[RD17] Xilinx UG761- AXI Reference Guide	13.1
[RD18] Xilinx PG066 - JESD204	6.2
[RD19] Xilinx UG571 - UltraScale Architecture SelectIO Resources	1.6
[RD20] Arcetri Technical Report N. 4/2017 - SKA Project Series LFAA tile beamformer signal processing	
[RD21] LFAA memo 00xx - LFAA tile beamformer functional model	1.0
[RD22] Arcetri Technical Report N. 2/2015 - SKA Project Series - LFAA Tile Beamformer structure	
[RD23] Arcetri Technical Report N. 5/2016 - SKA Project Series - LFAA Station Beamformer structure	
[RD24] G. Comoretto et al. The Signal Processing Firmware for the Low Frequency Aperture Array. JAI 06	
[RD25] J. Manley et al SPEAD: streaming protocol for exchanging astronomical data, SKA document, 2010	
[RD26] Xilinx UG580 - UltraScale Architecture System Monitor	
[RD27] Science and Technology Facilities Council - UDP Core Architecture	2018.01

4. ITPM OVERVIEW

The Italian Tile Processor Module is 32 channels 1 GSPS 8 bits Analog to Digital conversion and processing board equipped with management functions to support the integration into a rack structure.

The following list sums the most relevant iTPM characteristics and features:

- Compact size, 6U format compatible
- 32 analog inputs capability with high quality and high isolation stackable RF 50 Ohm input connectors
- 16 dual channel 14 bit 1GSPS ADC AD9680 [RD1]
- High quality clock distribution system with JESD204B [RD2] Subclass 1 capability
- Low noise PLL AD9528 supporting JESD204B sysref signal generation [RD3] with ADCLK948 clock buffers
- 2 Xilinx Ultrascale XCU040 20 nm FPGA devices dedicated to data processing [RD4]
- 2 DDR3, 96 bit memory banks, 6+6 Gbit total size
- 2 40G Ethernet interface (QSFP), one for each FPGA
- High speed internal bus to connect the 2 FPGAs, 25 Gbit + 25 Gbit bidirectional
- Lattice Mach-XO2-7000 CPLD with instant ON flash memory dedicated to board management and control [RD5]
- Gigabit Ethernet connection for management
- Lattice L-ASC10 platform for power management and voltage and temperature monitoring [RD6]
- High efficient multistage power distribution system, with configurable start-up sequence.
- Support for 10 MHz reference clock and PPS input
- AD9550 Integer-N Clock Translator for Wireline Communications to generate 40G Ethernet Reference clocks [RD7]

The iTPM is provided by the manufacturer with a low-level management CPLD firmware to perform board configuration and monitoring through the Ethernet interface exploiting a UDP based protocol [RD8].

Further information about the iTPM can be found in [RD9].

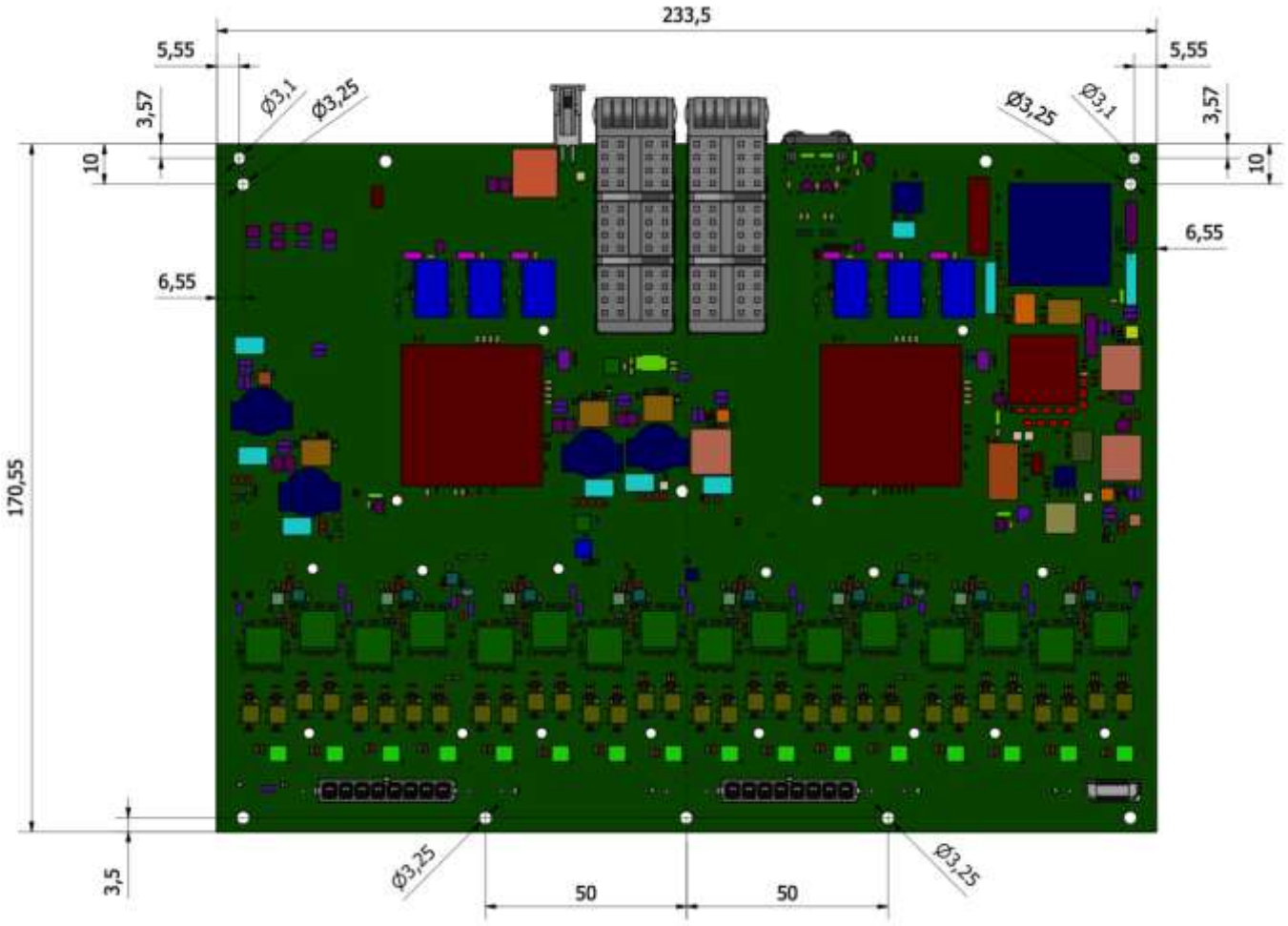


Figure 1. iTPM TOP mechanical layout

5. ITPM BLOCK DIAGRAM AND FUNCTIONS

In this chapter the main iTPM functions are described. First of all, the board block diagram is shown to help identifying which board physical component copes with each function. The iTPM is organized in two main sections: the management and control plane and the acquisition and processing plane.

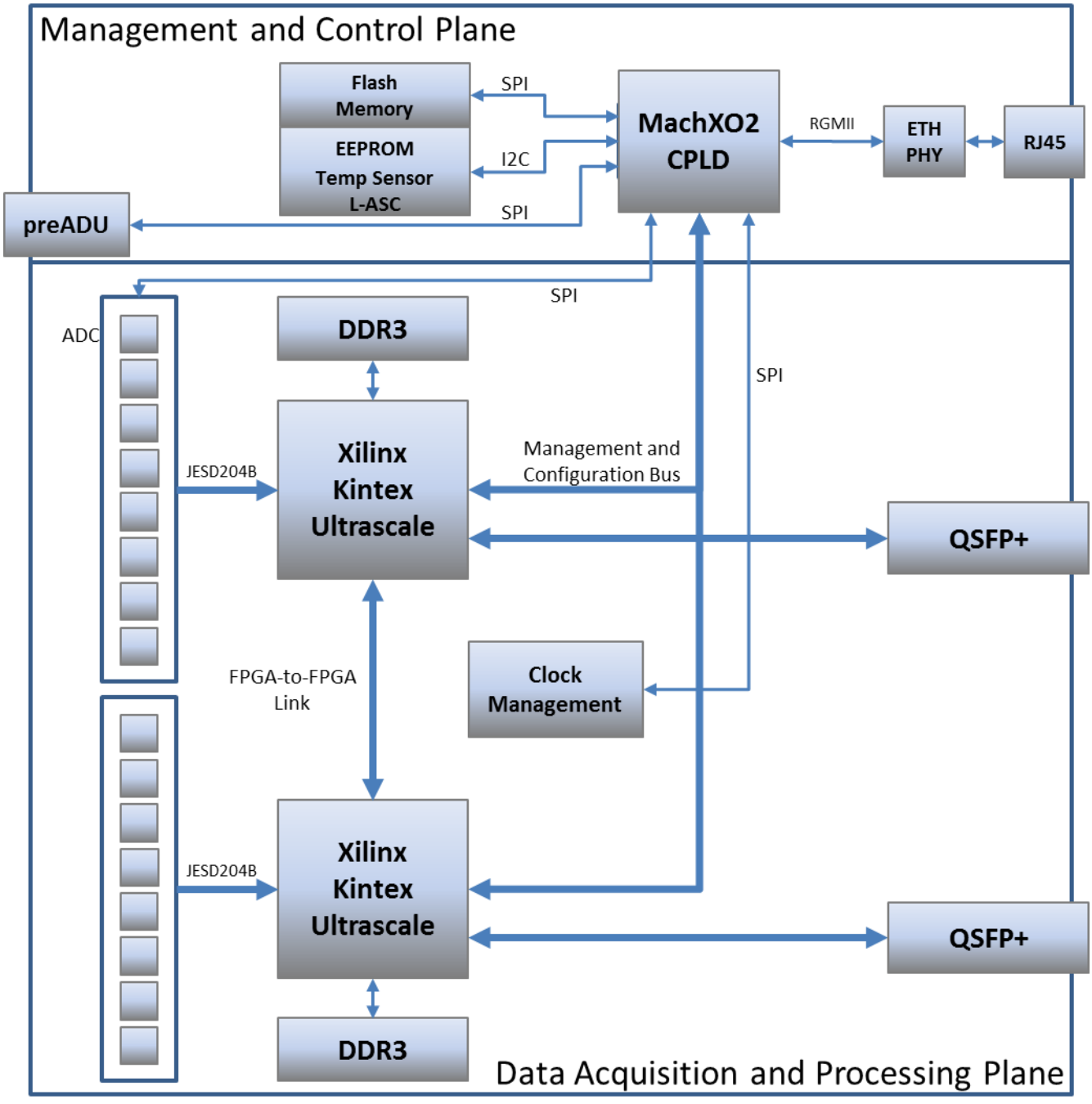


Figure 2. iTPM Block Diagram

5.1 Management and Control Plane

The management and control plane includes:

- FLASH based Lattice MachXO2 CPLD
- L-ASC System Monitor
- 1G Ethernet Connection
- EEPROM memory
- SPI FLASH memories
- Analog Front End (preADU) interfaces

The CPLD is the main component in the control plane. This device has an internal Flash memory and is auto-configuring at board power-up. It manages the access to the control registers of on-board devices through dedicated busses:

- RGMII to Ethernet PHY for the remote board control,
- FPGA management bus for FPGA register access and debug purposes
- SelectMap bus for fast configuration, reconfiguration and monitoring of the FPGAs
- C2C Stream bus for transferring data from the FPGAs to the CPLD
- I2C connected to the system monitor, temperature sensor and 2K bit EEPROM
- I2C connected to both QSFP+
- SPI connected to three identical high speed FLASH memories to store non-volatile configuration data
- SPI connected to the ADCs
- SPI connected to PLL

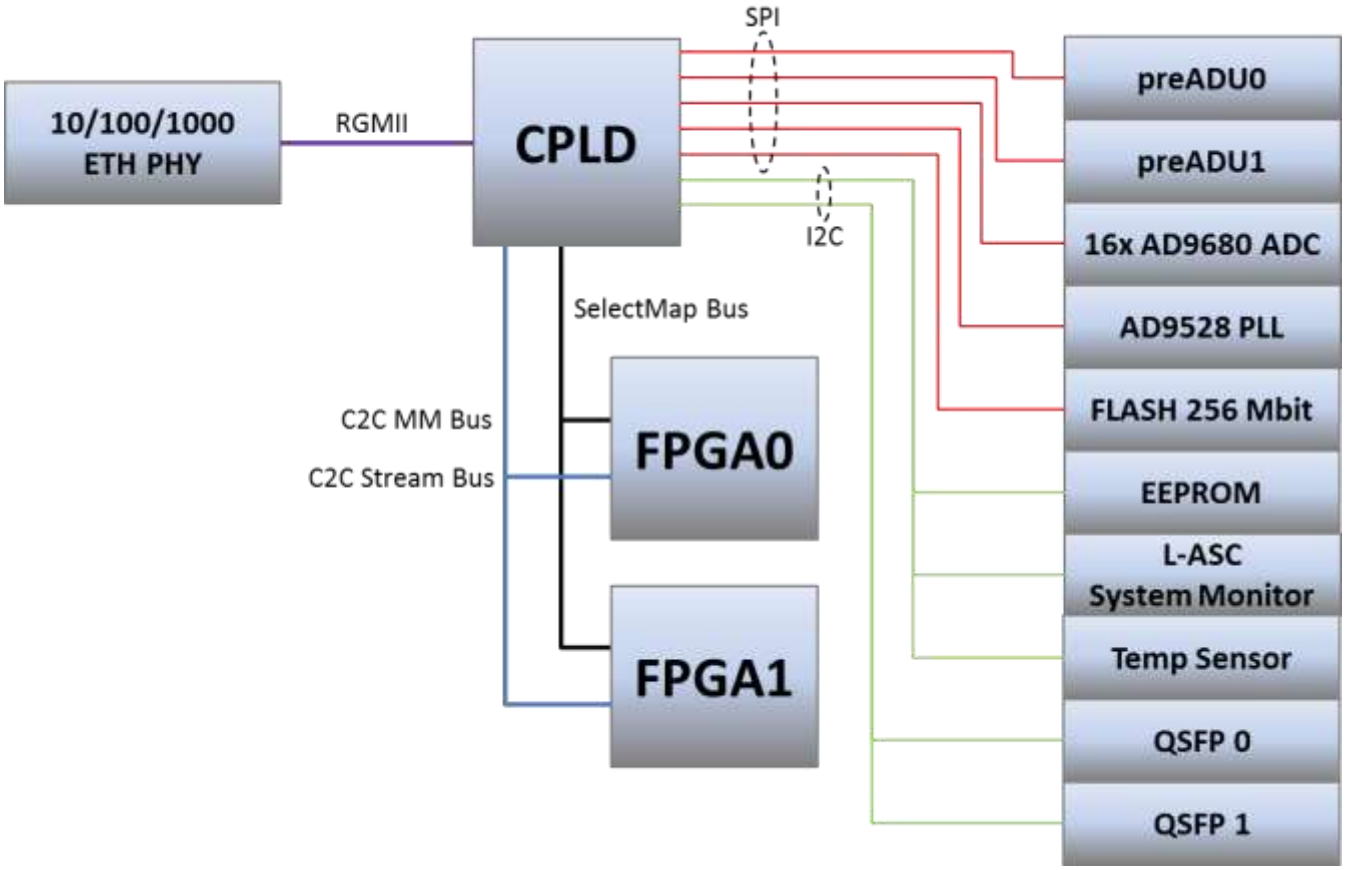


Figure 3. CPLD interfaces

5.1.1 CPLD Firmware Architecture

The following figure shows the CPLD firmware block diagram.

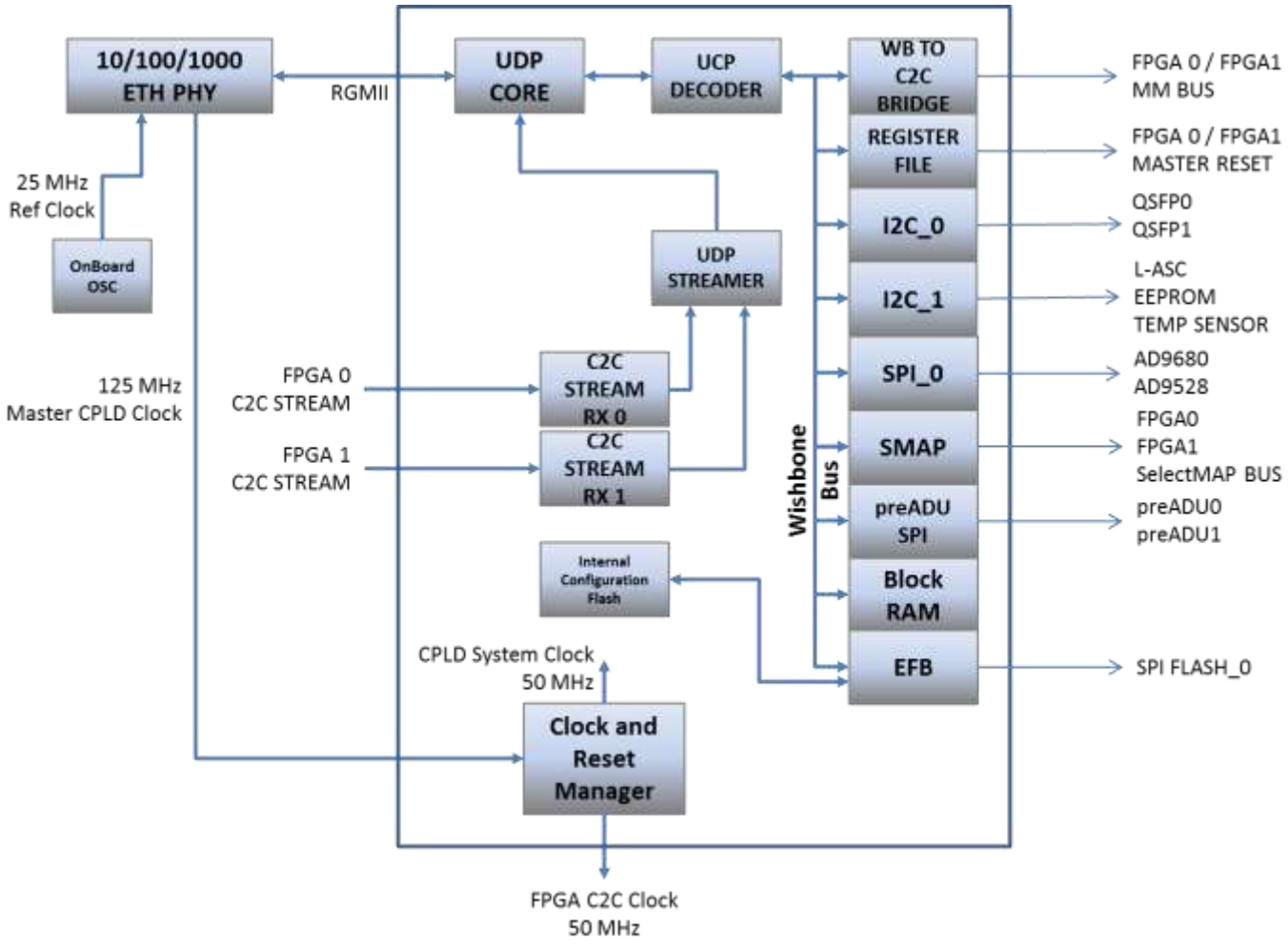


Figure 7. CPLD Firmware Block Diagram

The main CPLD responsibility is twofold:

- After board power-up the CPLD is responsible for enabling the on-board voltage regulators powering FPGAs, ADCs, PLL and Analog Front End. The power-up sequence can be customized to avoid excessive in-rush current.
- It acts as bridge between the network and the on-board devices such as FPGAs, ADCs, PLL, allowing the user to access them through the 1G Ethernet.

The user interface to the CPLD is based upon an UDP-IP core [RD10] which manages MAC, IP and UDP layers and allows the user to execute sequences of memory mapped bus accesses from a network device. The specific protocol implemented over UDP to convey the information about the memory mapped accesses is the Uniboard Control Protocol [RD8]. The CPLD is able to decode a UCP packet and translate the memory access into an internal bus access, thus allowing the user to access registers that have been memory mapped into the CPLD address space.

5.1.1.1 CPLD UDP Core

The CPLD is connected to a 10/100/1000 Ethernet PHY which enables the transmission and reception of Ethernet packets to/from an Ethernet network. The UDP core manages MAC, IP and UDP layers. Network configuration parameters, such as IP and MAC address, are stored in the on-board EEPROM and retrieved by I2C_1 interface after CPLD reset as described in the following section. The UDP core provides two distinct internal channels: the first channel is used to receive and transmit UCP packets allowing the user to access memory-mapped registers, while the second channel provides an interface to stream data from the FPGAs to the Ethernet network using the C2C Streaming bus.

5.1.1.2 CPLD Wishbone Internal Bus and Slaves

The CPLD implements a Wishbone [RD12] internal bus connecting multiple slaves using shared bus architecture with 32 bit address and data running at 50 MHz. The slaves are:

- Wishbone to C2C. It translates a Wishbone bus transaction to a C2C bus transaction as described in section 5.1.1.3.
- Register File. It contains several configuration and status registers such as reset registers and firmware version register.
- I2C_0. It consists of a register interface which commands an I2C master interface. SW instructs the interface to perform the required I2C access by writing the command registers and monitors the status registers to detect completion. This I2C interface is connected to both QSFP.
- I2C_1: Similarly to I2C_1, this interface this interface is connected to L-ASC, EEPROM and temperature sensor.

  The L-ASC platform manager provides voltage sensors that are accessed through I2C. The monitored voltages are listed in the following table:

Name	**Voltage (V)**	**Function**
5V0	5	Main switching regulator output
FPGA0_CORE	0.95	FPGA0 Core
FPGA1_CORE	0.95	FPGA1 Core
MGT_AV	1.8	FPGA Transceiver
MGT_AVTT	1.8	FPGA Transceiver Termination
VCC_AUX	1.8	FPGA VCC_AUX powered banks
AVDD1	1.25	ADC
AVDD2	2.5	ADC
AVDD3	3.3	ADC

Table 1. L-ASC monitored voltages.

The on-board EEPROM contains a pre-programmed and non-erasable unique MAC address. After power-up or reset, the interface reads autonomously the memory location specified in table Table 2 to retrieve the network configuration parameters that are used by to configure the UDP core. The EEPROM memory used to store network parameter is a Microchip 24AA02E which contains a unique MAC address written in the specified location in a write protected area and preloaded by the manufacturer. This guarantees that a unique MAC address is associated to the 1G interface of each TPM.

I2C Memory Offset	Network Configuration Parameter
0x0	IP Address byte 0 MSB
0x1	IP Address byte 1
0x2	IP Address byte 2
0x3	IP Address byte 3 LSB
0x4	Network mask byte 0 MSB
0x5	Network mask byte 1
0x6	Network mask byte 2
0x7	Network mask byte 3 LSB
0x8	Gateway IP Address byte 0 MSB
0x9	Gateway IP Address byte 1
0xA	Gateway IP Address byte 2
0xB	Gateway IP Address byte 3 LSB
0xFA	MAC address byte 0 MSB
0xFB	MAC address byte 1
0xFC	MAC address byte 2
0xFD	MAC address byte 3
0xFE	MAC address byte 4
0xFF	MAC address byte 5 LSB

Table 2. CPLD Network configuration to EEPROM addresses mapping

- SPI_0: It consists of a register interface which commands an SPI master interface. SW instructs the interface to perform the required SPI access by writing the command registers and monitors the status registers to detect completion. This SPI interface is connected to both ADCs and PLL.
- SMAP. The SelectMAP interface drives the SelectMAP and programming signals connected to the FPGA. It consists of an internal FIFO interface that is memory mapped on the Wishbone bus. Data written into the FIFO are transparently uploaded into the FPGAs configuration memory. It also supports a set of status and control register that permit to erase the FPGA configuration memory before uploading a new bitfile and monitor the FPGA done signal to detect successful FPGA configuration.
- preADU SPI. It accesses both preADU board connected to the ADU.
- BlockRAM. It consists of a 2Kbyte Block RAM connected to the Wishbone bus. This memory contains the CPLD memory map XML description, according to section 6.1, which can be read by SW.
- EFB. The EFB [RD11] includes a Serial Peripheral Interface (SPI), two I2Cs, a timer/counter peripheral and an access port to the internal Flash configuration memory. All of these hard IP peripherals are contained in the EFB block and are connected to the Wishbone bus in slave mode. Among the available peripherals, the CPLD firmware uses the following ones:
    - Access port to the Flash configuration memory which consists of a set of registers implementing a command interface that enable the access to the CPLD Flash memory. This feature permits to upload new CPLD bitfiles on the field using a fail-safe procedure as described in section 5.1.2.
    - SPI Master which enables access to the external SPI Flash memory. This peripheral must be active in order to enable the CPLD to load a bitfile from an external SPI Flash memory.

5.1.1.3 Memory Mapped Access

Memory-mapped registers can be accessed through network using UCP which supports 32 bits wide address and data transfers. The UCP decoder is able to receive a UCP packet from the UDP core, decode it and perform the corresponding bus access on the internal Wishbone bus [RD12]. The UCP decoder acts as Wishbone bus master accessing all the slave interfaces connected to the Wishbone bus. In particular, FPGA registers can be accessed via a Wishbone to C2C Bridge which translates the Wishbone bus transaction into a C2C MM (Memory Mapped) bus transaction. C2C MM bus is the custom shared local bus connecting both FPGAs to the CPLD. It is an address-data-command multiplexed and bidirectional 4 bits bus. A typical C2C MM write transaction is shown in the following figure.

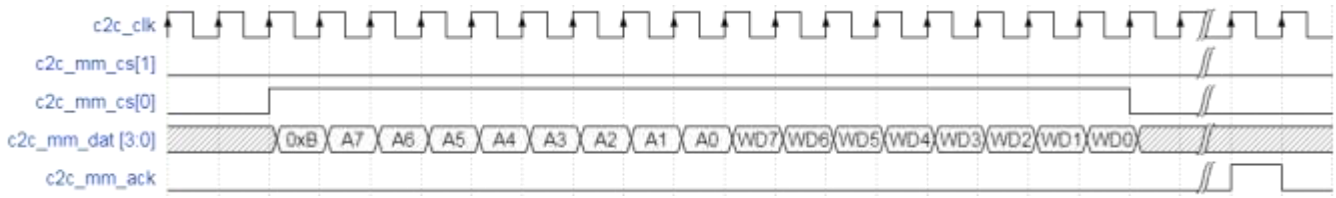


Figure 4. C2C Memory Mapped Write Transaction.

To start a write transaction the CPLD asserts one of the two c2c_mm_cs signals selecting the FPGA to address and transmits sequentially the command, address and write data of the transaction serializing them from MSB to LSB over the 4 bit c2c_mm_dat bus. Command word can be either 0xB for a write command or 0xA for a read command. After the last nibble of write data has been transmitted, the CPLD releases the data bus and it waits for the assertion of c2c_mm_ack signal which signifies the completion of the transaction in the addressed FPGA. The signal c2c_mm_ack is shared between the two FPGA and driven only by the addressed FPGA. Both FPGAs pull their c2c_mm_ack down when they are not addressed.

A typical C2C MM read transaction is shown in the following figure. In this case the CPLD transmits to the addressed FPGA the command and the transaction address. The FPGA acknowledges the CPLD asserting c2c_mm_ack for 8 clock cycles during which it drives the c2c_mm_bus with the requested 32 bits read data.

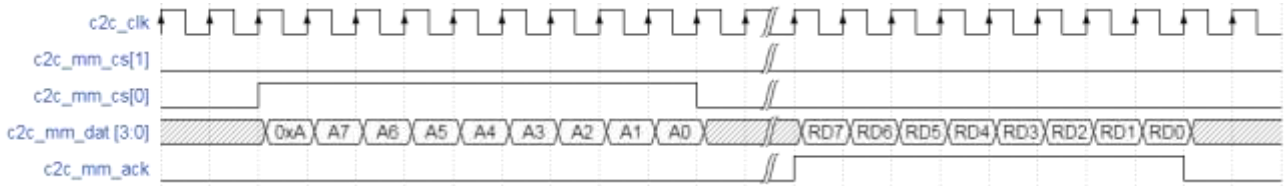


Figure 5. C2C Memory Mapped Read Transaction.

The FPGA is required to complete the bus transaction within 10 ms from the assertion of its c2c_mm_cs. In case of timeout, the CPLD replies to the UCP request originator with an UCP error packet.

5.1.1.4 C2C Streaming interface

Both FPGAs are connected to the CPLD by a C2C Stream bus; each packet transmitted from the FPGA to the CPLD over the C2C Stream bus is mapped into one or multiple UDP packets and transmitted over the network by the CPLD. A typical packet transmission over C2C streaming bus is shown in the following figure.

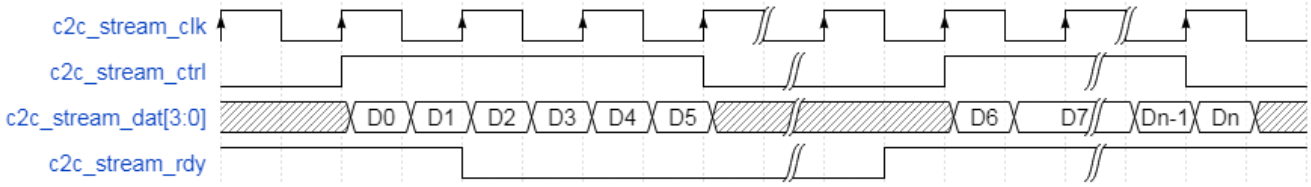


Figure 6. C2C Stream Packet Transmission.

The C2C Stream bus is a source-synchronous data bus consisting of 4 bits data and 1 bit control signals. Each FPGA is connected to the CPLD with a dedicated C2C Stream bus and dedicated source-synchronous clock generated by the FPGAs. Data are transferred over the bus at DDR; MSBs of each byte are transmitted at rising edge of c2c_stream_clk, while LSBs are transferred at falling edge. The c2c_stream_ctrl is encoded over two consecutive clock edges according to the following table.

Rising edge sampling	Falling edge sampling	Decoding
0	0	Idle
1	1	Byte Valid
1	0	Last Byte
0	1	MM Read Data

Table 3. C2C Stream control signal decoding.

The C2C Stream can transport MM read responses using the dedicated encoding. SW can select the MM read responses channel by writing the specific control register in the FPGAs.

The CPLD is able to shift the c2c_stream_clk phase internally for both C2C Stream bus. A procedure can be implemented by software to center the CPLD sampling clock in the C2C data valid window, this is described in more detail in section 8.1 .

Back-pressure can be applied by the CPLD to the FPGAs using the c2c_stream_rdy signal which is de-asserted when the FPGA internal FIFO can accept 15 bytes before getting full.

As the maximum length of Ethernet packets is limited, the CPLD will fragment the C2C packet into multiple UDP packets when the C2C packet length exceeds the maximum length programmed by software. The CPLD provides arbitration between the two FPGAs as only one UDP packet is transmitted at the same time.

5.1.1.5 CPLD Clock and Reset

The Ethernet PHY is connected to a 25 MHz reference oscillator and generates a 125 MHz clock which is fed to the CPLD which in turn uses an internal PLL to generate a 50 MHz clock used as internal system clock. This signal is also clocks the C2C MM bus and is forwarded to the FPGA as c2c_mm_clk. Main CPLD clocks and frequencies are listed in the following table.

After power-up the CPLD asserts its master internal reset. As soon as the internal PLL acquires lock to the 125 MHz master clock, the internal reset is de-asserted. The CPLD manages also the reset signal of FPGAs, ADCs and PLL. This reset signals are controlled by memory mapped registers directly accessible on the Wishbone bus.

5.1.2 CPLD Bitfile Management

The CPLD firmware is provided as closed source bitfile preload onto the iTPM by the manufacturer. The Lattice MachXO2 device is able to store one bitfile in its internal Flash memory and access an external SPI Flash memory where an alternative bitfile can be stored.

5.1.2.1 CPLD boot procedure

As shown in Figure 7, after board power-up the CPLD configuration state machine firstly searches for a valid bitfile in the internal Flash memory and then it switches to the external SPI Flash memory in case a valid bitfile is not found in the internal Flash.

5.1.2.2 CPLD bitfile update procedure

The dual-boot feature permits to implement a fail-safe procedure for updating the CPLD firmware on the field. The currently used bitfile is stored in the CPLD internal Flash memory and it is the first bitfile that the CPLD tries to load. The user can update the internal memory overwriting the current bitfile with the updated bitfile, this is possible as the internal Flash is accessed through the Embedded Functional Block (EFB) [RD11] which is in turn memory mapped into the CPLD address space and thus can be accessed using UCP commands. In case the updated bitfile has not been written correctly in the internal Flash memory, at the next board power-up or reboot the CPLD will not load the bitfile from its internal memory switching to the external SPI memory where the manufacturer stored a golden bitfile. The golden bitfile provides only the basic functionality which allows the user to access the CPLD internal Flash memory. In this way, with the golden bitfile running, it is possible to recover from a failed bitfile update procedure by overwriting the CPLD internal Flash memory. In case the CPLD is unable to load the Golden bitfile from the external Flash, e.g. as a consequence of an HW fault, the iTPM board becomes unreachable from the network and the CPLD must be checked using JTAG connectivity.

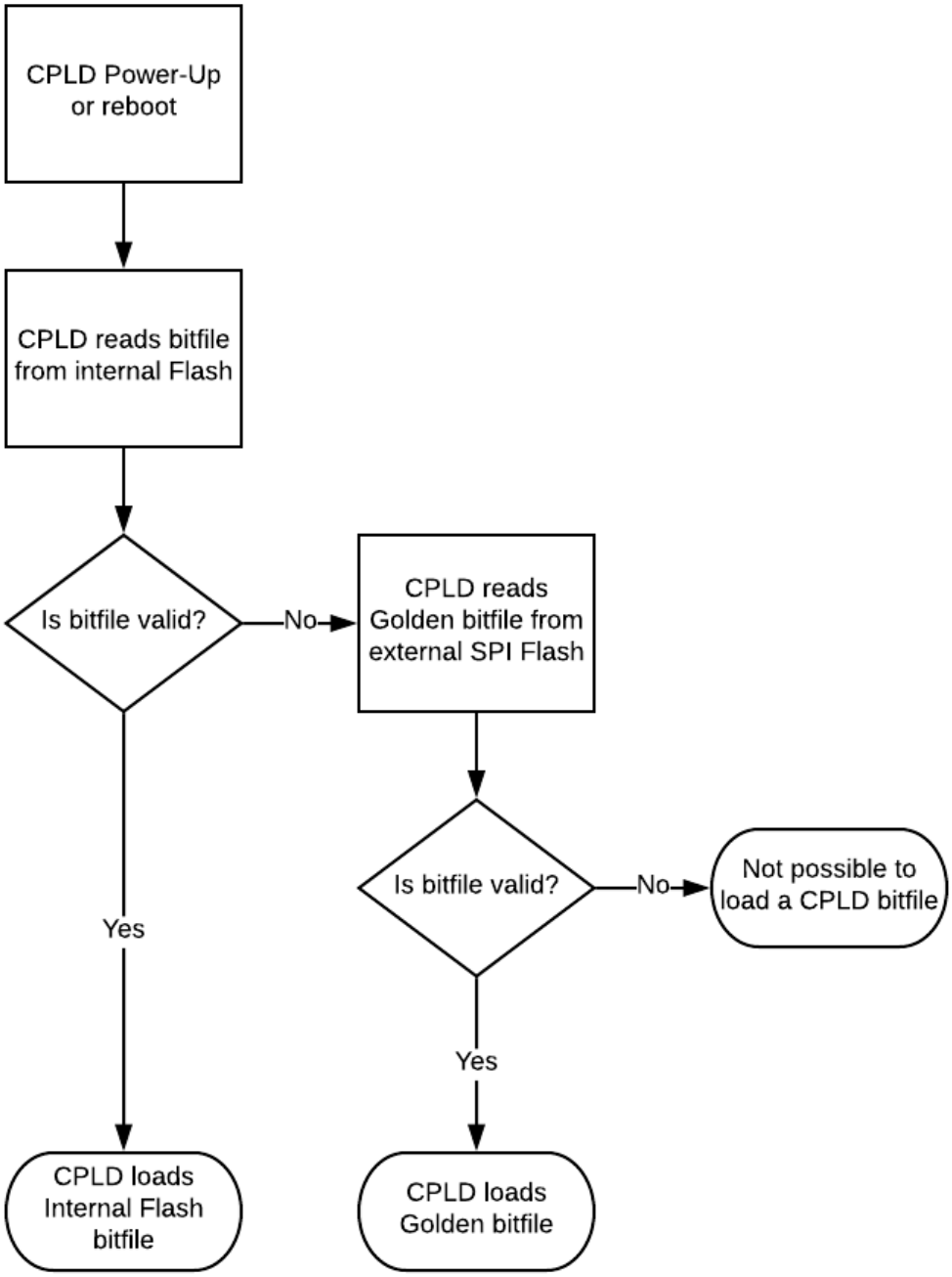


Figure 7. CPLD boot procedure.

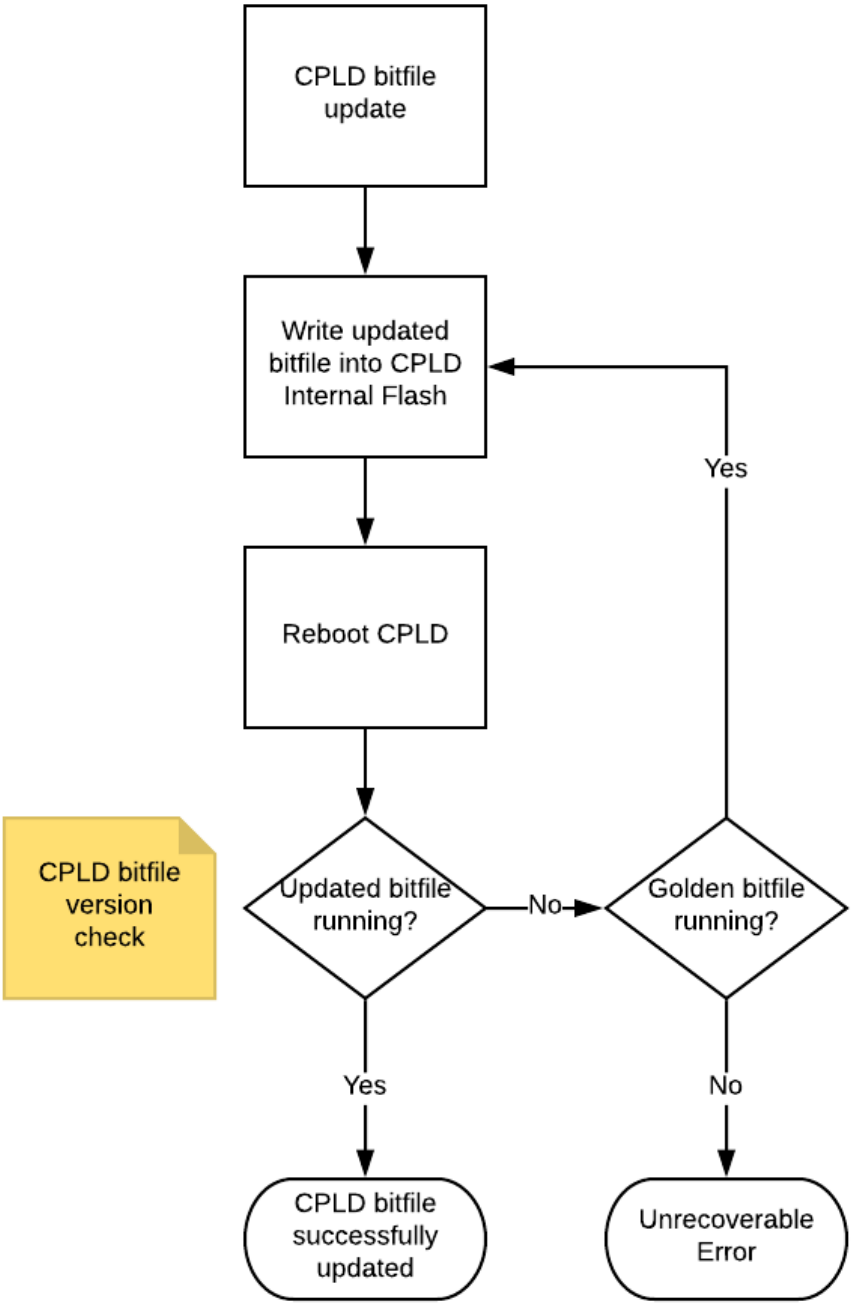


Figure 8. CPLD bitfile update procedure.

5.1.3 CPLD Data Rates

The following table shows the main CPLD data busses with related clock frequency and maximum and sustained data rate.

Bus	**Clock Fequency (MHz)**	**Data Width**	**Peak Data Rate (Mbyte/s)**	**Sustained Data Rate (Mbyte/s)**
C2C MM	50	4	25	Depends on SW.
Wishbone Bus	50	32	200	Depends on SW.
C2C Stream FPGA0	125	4 DDR	125	~50
C2C Stream FPGA1	125	4 DDR	125	~50
UDP Streamer to UDP	50	16	100	~100
RGMII RX	125	4	125	~125
RGMII RX	125	4	125	~125

Table 4. CPLD data rates.

The actual sustained C2C MM data rate depends on the transaction rate supported by SW considering UCP packet round trip delay. The maximum transaction rate supported by the C2C MM bus is ~0.8 MT/s.

The actual sustained internal Wishbone data rate depends on the transaction rate supported by SW considering UCP packet round trip delay. The maximum transaction rate supported by the wishbone bus is ~16 MT/s assuming minimal delay of slave response.

The UDP Streamer to UDP core data bus is limited in speed to 100 Mbyte/s. In case both FPGAs transmit data over the C2C Stream busses, the bandwidth is shared between the two FPGAs.

5.1.4 CPLD FW Limitations

The current FW provides access to only one of the three available SPI Flash memory chips. This limitation is due to a constraint in the connection of Chip Select signals to the CPLD when using the SPI peripheral embedded in the EFB block which is required to support the dual boot feature.

5.2 Acquisition and processing plane

The iTPM acquisition and processing plane includes ADCs, FPGAs, DDR3 memories, 40G connections and clock management devices. The iTPM FPGAs are two identical devices that provide suitable and efficient resources to implement the main board data acquisition and processing features. The two FPGAs have a compatible pinout and can use an identical bitfile. Each FPGA support an *fpga_id* pin that is detected as high logic value in the first FPGA and low in the second one allowing the same bitfile to behave differently in the two devices; for instance *fpga_id* is used to select the relevant *c2c_mm_cs* bit ignoring the transaction not addressing the selected device. Each FPGA provides the following interfaces:

- SelectMap interface for configuration, reconfiguration and monitoring
- Local bus interface to communicate with the CPLD. This consists of C2C MM and C2C Streaming busses
- 16 JESD204B data links to receive data from the ADCs
- JESD clock inputs, one for the transceiver and one for fabric internal logic
- Four Multi-Gigabit bidirectional data links connected to a QSFP connector to implement either 4x10G Ethernet interfaces or one 40Gbit Ethernet.
- 10G Ethernet clock PLL input
- Opto-isolated PPS input signal
- 96 bit DDR3L memory interface, x16 devices are soldered on board
- 100 MHz DDR clock
- FPGA to FPGA bus consisting of 2 groups of 18 LVDS data lines and dedicated clock and control lines supporting up to 51.2 Gbit monodirectional total throughput between the FPGAs.

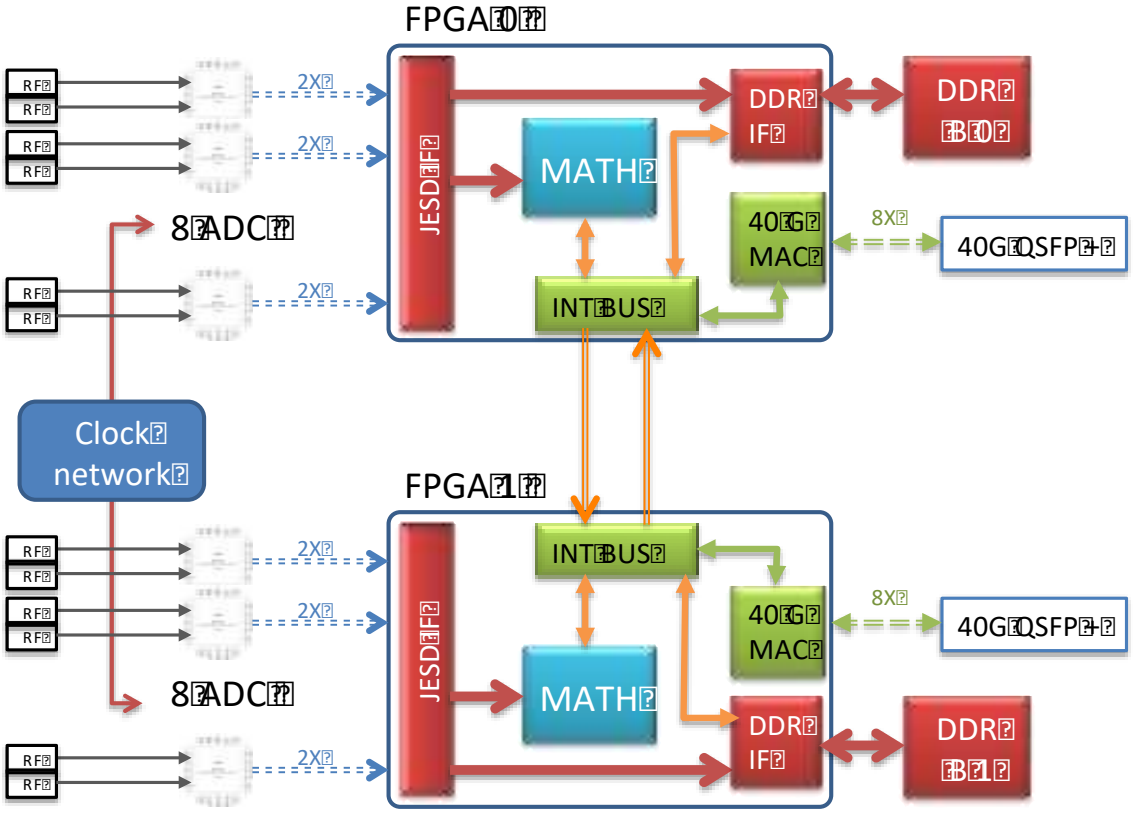


Figure 9. iTPM high speed data paths

5.2.1 FPGA Firmware Architecture

The FPGAs constitute the main processing elements on the iTPM. They provides high-speed interfaces to the ADCs, DDR3, 40G Ethernet and CPLD and implement the DSP logic to process acquired data acquired from the ADC. Additionally the FPGAs provide the functionality to extract data from the DSP chain and transmit it to the CPLD over the C2C streaming bus at suitable data rate such that the extracted data can be forwarded over the 1G for monitor, control and debug purposes.

The FPGA Firmware is mainly coded in VHDL and structured in such a way to simplify the transition to different boards, FPGA vendors and families. Technology dependent code has been isolated into specific portions and industry-standard interfaces implemented to enable the communication between main parts of the design; leveraging standard interfaces allows the designer to transparently modify the implementation of part of the design while maintaining compatibility with the rest of it.

The FPGA Firmware consists of a top level board wrapper, which is customized to support the specific board pinout; the firmware is then partitioned into two main sections. An I/O ring section includes all external interfaces and technology dependent modules such as PLLs, transceivers and physical layers. The I/O ring is directly connected to the DSP section using industry standard busses. The following block diagram shows the separation between the two sections. The following paragraphs describes FPGA clock and reset trees, I/O ring section and DSP section.

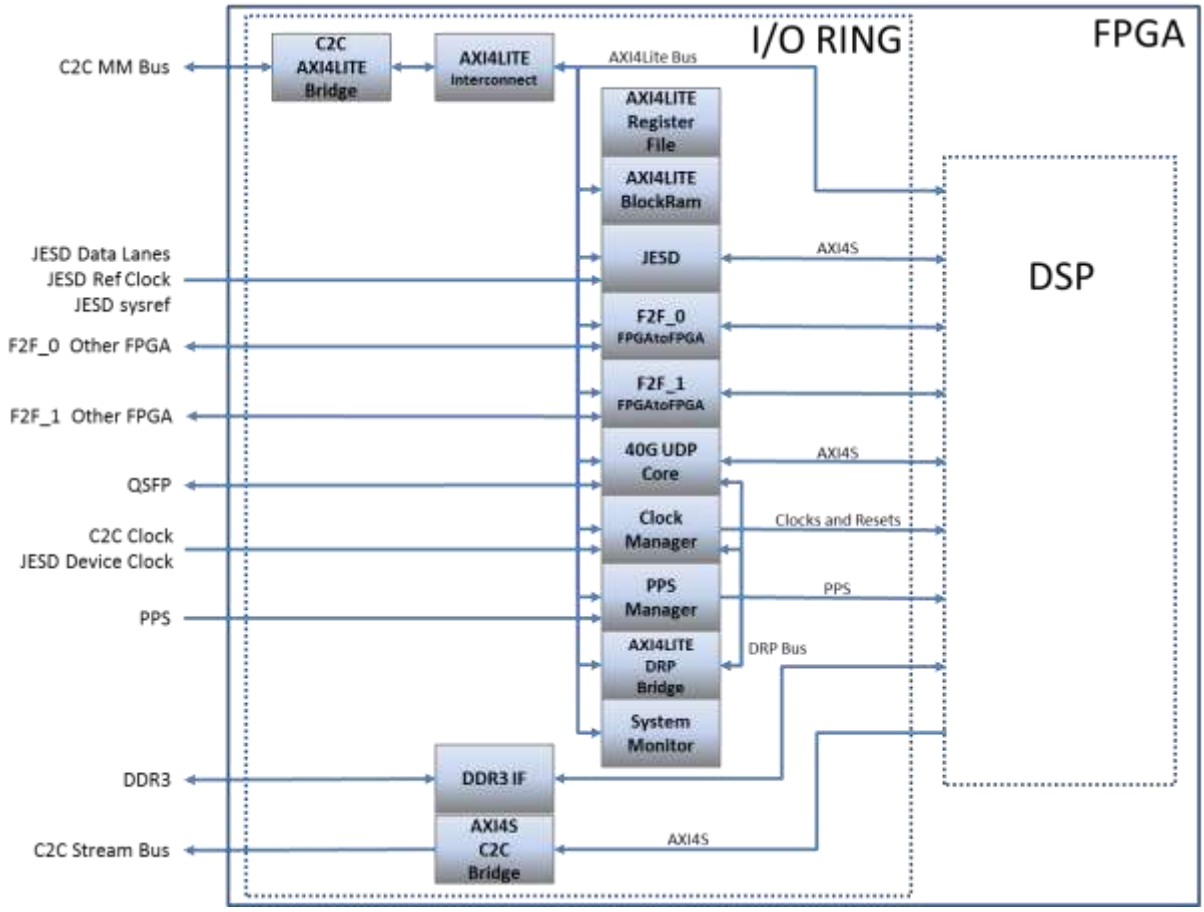


Figure 10. FPGA Top Level Firmware Block Diagram.

5.2.2 FPGA Clock and Reset Tree

5.2.2.1 Clock Tree

Figure 11 shows the external clocks sourced by the FPGAs and processed by MMCMs and PLLs to generate internal clocks needed by DSP section and other physical interfaces.

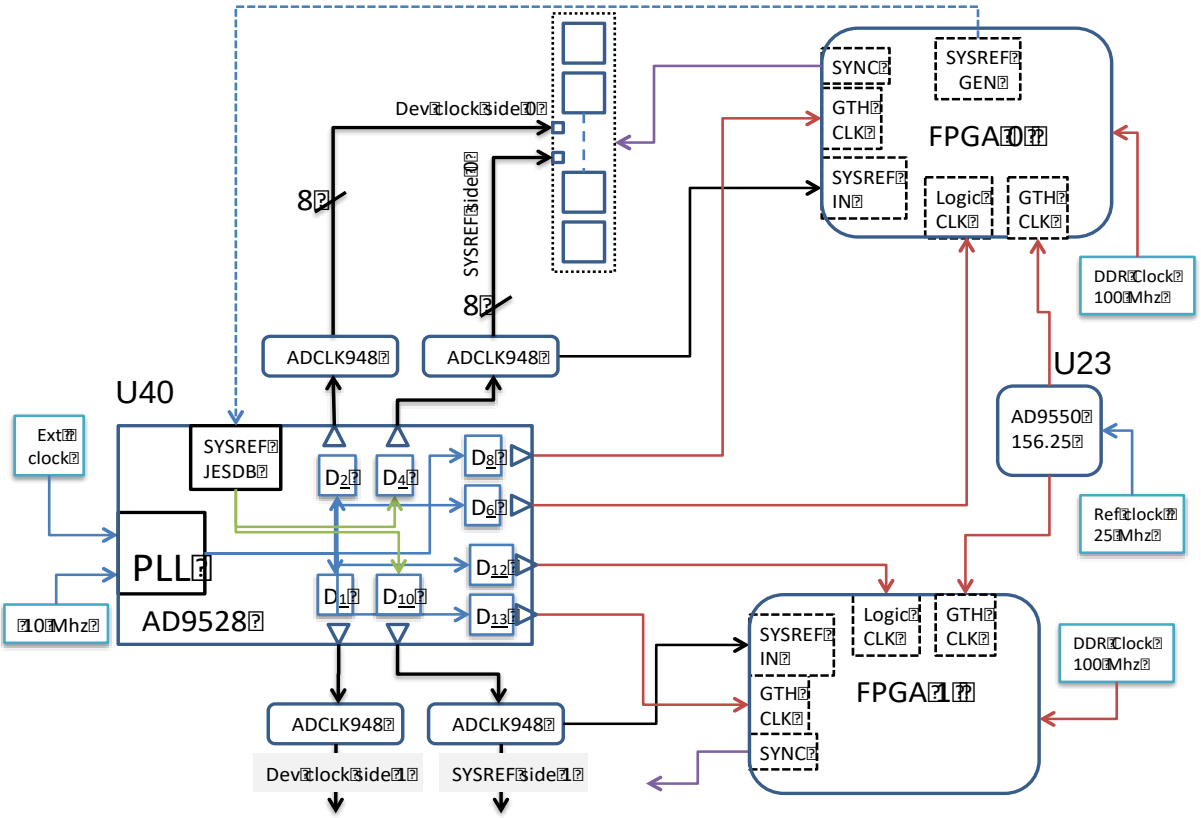


Figure 11. FPGA external clocks.

The external clocks are the following:

- C2C clock is generated by the CPLD; an MMCM is used to generate clk50, which is a phase aligned internal copy of C2C clock used to clock the internal AXI4Lite bus and the C2C MM interface, and clk125 used to clock the C2C Stream interface.
- JESD Device clock and JESD Reference clock are both generated by AD9528 PLL. The JESD Reference clock is fed directly to the QPLL which in turn generates the serial clock for the GTH transceivers. The JESD Device clock is fed to an MMCM generating the *hf_clk*, used in the PPS synchronization phase as described in section 5.2.7.2, and *adc_clk* which clocks the ADC data from the JESD interface into the DSP section. Another MMCM is used to generate *dsp_clk* by applying a frequency scaling factor of 32/27 to *adc_clk*; the generated clock is used as oversampling clock in the Channelizer. A double frequency dsp2x_clk is also generated; this clock is used in the optimized polyphase filter as shown in [RD13].
- 40G Reference clock is generated on-board by AD9550. This signal clocks both the 40G core and the F2F interfaces. In the 40G core it is used as reference clock for the GTH transceiver using a QPLL to generate the GTH serial clocks and the user clocks. The same 40G Reference clock is forwarded to the F2F cores using a BUFG_GT instance

allowing the F2F cores to use the high quality Reference clock and minimizing Jitter insertion. Both F2F cores use an internal PLL to generate the serial clocks required by the SERDES. A single F2F user clock is used for both interfaces to clock data in and out from the F2F cores.

- DDR clock is generated by an external oscillator. It feeds the Xilinx DDR3 IP core where it is conditioned by an MMCM which generates the ddr3_user_clk and the DDR3 PHY clock which is in turn forwarded to several TXPLL primitive to generate a 90 degrees phase shifted copy, as described in [RD14], which is used by the DDR3 physical interface to align the clock forwarded to the DDR3 chips to the center of the data window.

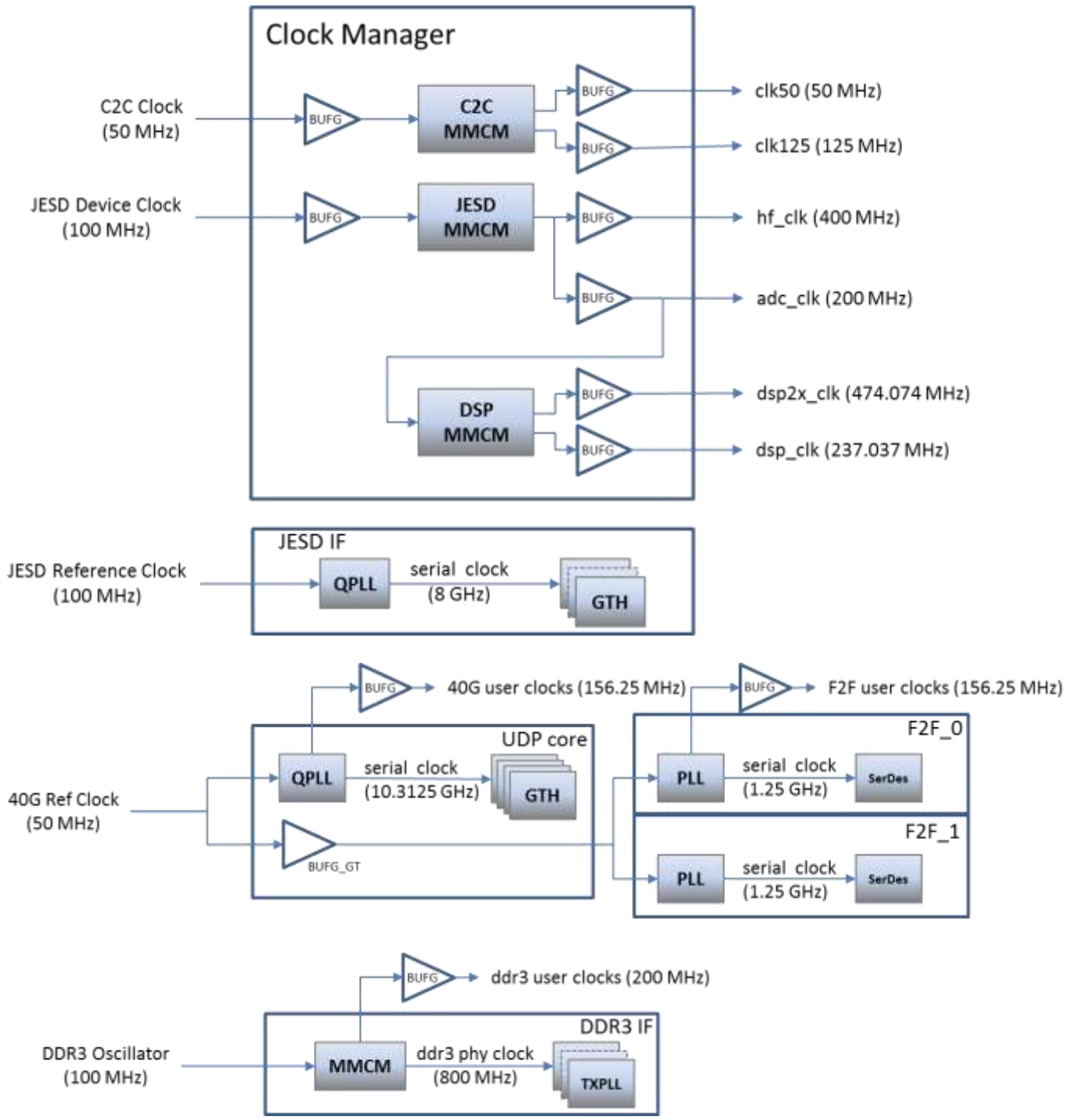


Figure 12. FPGA Clock Tree.

The following table reports the internal FPGA clocks, sources, frequencies and main usages.

Internal clock	Frequency (MHz)	Derived From	Usage
clk50	50	C2C MM clock	C2C MM Interface AXI4Lite internal bus
clk125	125	C2C MM clock	C2C Stream Interface
hf_clk	400	JESD Device Clock	PPS Manager high frequency counter
adc_clk	200	JESD Device Clock	JESD Interface, DSP section
dsp_clk	237.037	JESD Device Clock	DSP section
dsp2x_clk	474.074	JESD Device Clock	DSP section, optimized polyphase filter
f2f_user_clk	156.25	40G Reference Clock	F2F Interface
eth_phy_clk	156.25	40G Reference Clock	40G UDP, MAC and PHY layers
udp_clk	156.25	40G Reference Clock	40G UDP user interface
ddr_user_clk	200	DDR3 Oscillator	DDR3 user interface

Table 5. FPGA User Clocks.

5.2.2.2 Reset Tree

Each asynchronous reset signal is synchronized into the destination clock domain to assure that reset release is handled correctly. In the Clock Manager module the lock signal from the C2C MMCM is used to hold rst50, rst125, adc_rst and dsp_rst asserted until the C2C MMCM is locked to the C2C MM clock supplied by the CPLD. This ensures that after FPGA configuration the resets signal gets asserted at least once. The generated reset signals can be asserted also by writing in specific memory mapped register implemented in AXI4Lite Register File.

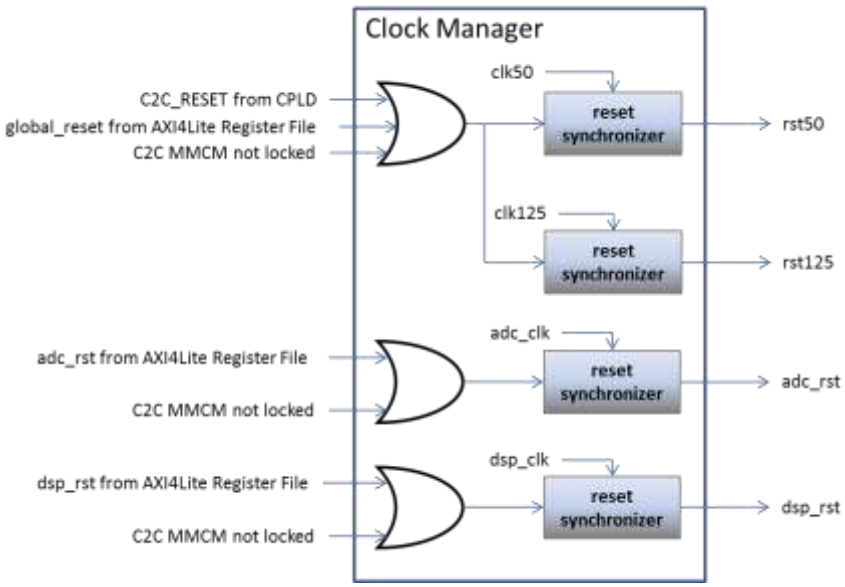


Figure 13. FPGA Reset Tree.

F2F, DDR3 and 40G reset are handled by specific finite state machines implemented in the vendor IP cores and can be triggered by writing in specific memory mapped registers implemented in AXI4Lite Register File.

5.2.3 FPGA I/O Ring

The FPGA I/O ring provides the interface to access on-board devices. It represents an abstraction layer where the technology dependent code has been encapsulated and standard interfaces have been made available to the DSP logic. The I/O ring section contains also reset, clock and PPS management logic. The following subsections describe each I/O ring submodule in more detail.

5.2.3.1 C2C AXI4Lite Bridge

The C2C AXI4Lite Bridge translates a C2C Bus transaction into an AXI4Lite Bus transaction acting as C2C Slave and AXI4Lite master. AXI4Lite bus is an industry standard local bus officially supported by Xilinx tools, further information can be found in these documents [RD15][RD16][RD17]. The implemented AXI4Lite bus consists of 32 bits wide address and data bus. The following figure shows a typical AXI4Lite write transaction.

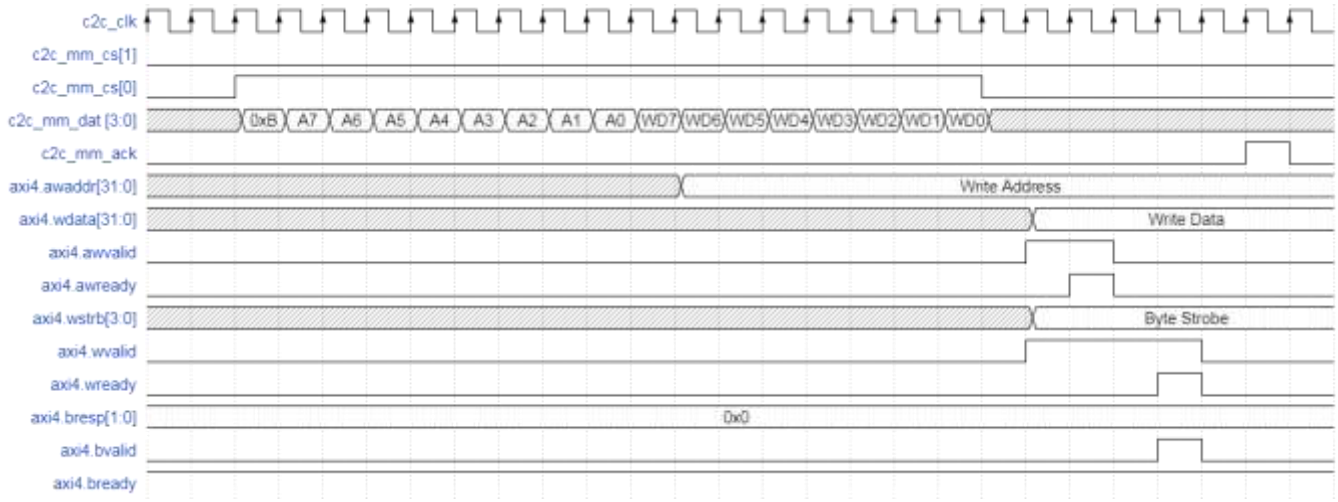


Figure 14. AXI4Lite Write Transaction.

The bus transaction is initiated by the C2C master in the CPLD and consists of several bus cycles during which address and data of the required memory access are multiplexed to fit onto the physical 8 bits C2C MM bus. The FPGA detects a bus transaction when the c2c_mm_cs signal is driven high by the CPLD; in that case, data on the C2C MM bus is sampled and de-serialized and the corresponding bus transaction on the internal AXI4Lite bus is started as soon as address and data are both available. When the AXI4Lite transaction is terminated, the C2C AXI4Lite Bridge asserts the *c2c_mm_ack* to acknowledge the CPLD.

Read transaction are handled similarly, no data is transferred from the CPLD towards the FPGA but on the contrary read data are transmitted back from the FPGA to the CPLD during the acknowledge phase, as shown in the following figure.

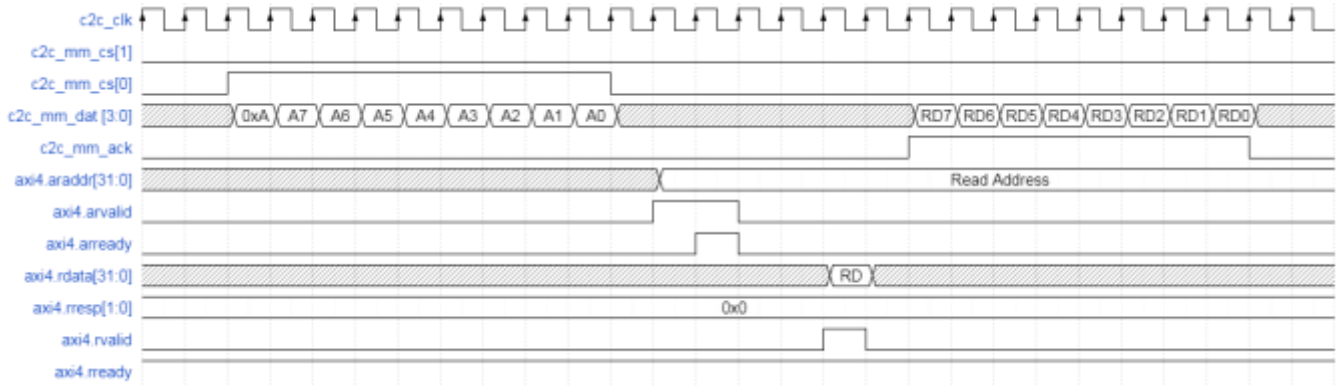


Figure 15. AXI4Lite Read Transaction.

5.2.3.2 AXI4Lite Interconnect

The AXI4Lite Interconnect implements a shared bus architecture that is used to enable a single bus muster, i.e. C2C AXI4Lite Bridge, to communicate with multiple bus slaves. The implemented shared bus architecture supports only one on-going bus transaction at any time, meaning that a bus transaction must be fully completed before the next bus transaction can be initiated.

5.2.3.3 AXI4Lite Register File

The AXI4Lite register file is a slave interface on the AXI4Lite bus. It consists of a set of registers used to control the FPGA and monitor the status of various signals. These registers include reset register, informational registers such as firmware version, implemented features and debug registers.

5.2.3.4 AXI4Lite BlockRAM

The AXI4Lite BlockRAM consists of a 64 Kbyte BlockRAM mapped onto the AXI4Lite bus. Primary purpose of the RAM is storing the global memory map file describing all the registers implemented FPGA. The memory map file is a compressed XML file following the conventions specified in section 6; it is generated and inserted into the FPGA bitstream using an automated process embedded into the tool chain, further details are provided in section 7.3.

5.2.3.5 JESD Interface

The JESD Interface is based upon the Xilinx JESD204B IP core [RD18] which manages the JESD protocol autonomously. The IP core contains several configuration registers that can be accessed through the AXI4Lite bus. Once configured and initialized the IP core streams ADC data towards the user logic using an AXI4S interface. It is not possible to apply back-pressure to the IP core as this would result in a data loss from the ADC.

As the Xilinx JESD204B core permits to specify a maximum of 12 JESD lanes per core, two identical IP cores managing 8 JESD lanes each are instantiated and wrapped in a top level JESD Interface module. The wrapper contains also the AXI4Lite arbitration logic to address the two cores.

JESD204B Subclass 1 is used as synchronization method to synchronize sampling and receiving devices, i.e. ADCs and FPGA respectively. The distinctive feature of Subclass 1 is deterministic latency which ensures that the delay from the point in time when the data is

sampled in the ADC to the moment when the same data is streamed out from the IP core towards the user logic is always the same after re-initialization of the JESD link. This feature is crucial in order to easily synchronize several iTPMs. In the current synchronization scheme the two Xilinx JESD204B IP cores in each FPGA share a single JESD204B sysref signal and a single JESD204B sync signal is generated by combining the sync output from each IP core; this configuration allows the two IP cores to be synchronized during the JESD204B link initialization. After JESD initialization, the IP cores are locked and stream ADC data, however a further step is necessary in order to enable the data streaming towards the DSP section. As serval JESD204B links are initialized across multiple iTPMs at different times, each JESD204B core would start streaming data asynchronously; this can be avoided if a synchronous signal across multiple iTPMs is used to enable the data transmission from the JESD core. In fact the JESD core wrapper starts streaming data towards the DSP section only after the assertion of its *start_pulse* input signal which is in turn generated by PPS Manager Module and derived from the external PPS signal as described in 5.2.7.2. Assuming that multiple iTPMs are fed with coherent and un-skewed copies of 10 MHz Reference clock and PPS signals, this two-step synchronization process ensures that all JESD Interfaces across all iTPMs start streaming data at the same ADC sampling clock edge. Lastly, as JESD204B Subclass 1 guarantees deterministic latency, the streamed ADC data are sample-synchronized across all cores.

As shown in Figure 16, once the data streaming is initiated by asserting AXI4S *tvalid* signal, data are framed into packets of 864 samples with 4 ADC samples per adc_clk clock cycle; AXI4S *tlast* signal is used to identify the last sample in a frame. Data streaming from JESD204B is never interrupted as AXI4S data rate matches ADC data rate exactly.

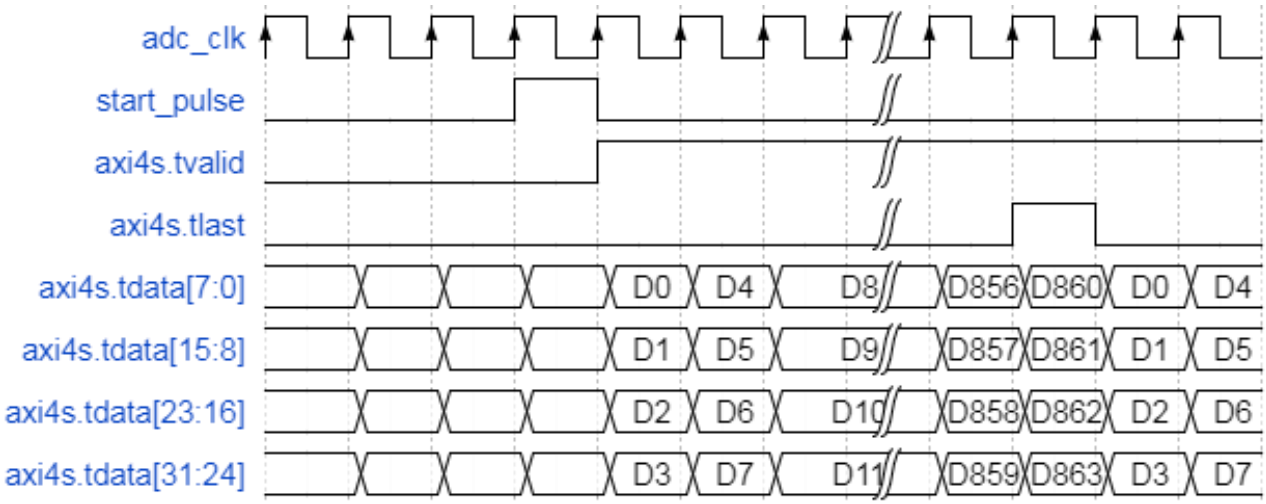


Figure 16. JESD204B AXI4S Streaming Interface.

5.2.3.6 F2F - FPGA to FPGA Link

The F2F module is based upon the Xilinx Select-IO IP core [RD19], which manages 18 LVDS lanes each running at 1.25 Gbit/s. Two Select-IO IP cores are used to manage the two sub-links that compose the F2F link. Each F2F sub-link can be configured either as input or output link at configuration time. When the F2F sub-link is configured as output link in one FPGA, the corresponding F2F sub-link in the other FPGA must be configured as input link to avoid conflicts on the F2F bus. In this way the F2F link can be configured as a bidirectional link supporting 22.5 Gbit/s in each directions or mono-directional link supporting 45 Gbit/s in only one direction. As the on-board F2F connections between the two FPGA are organized in a symmetric fashion, the same bitfile can be used for both FPGAs specifying the link direction in a configuration register.

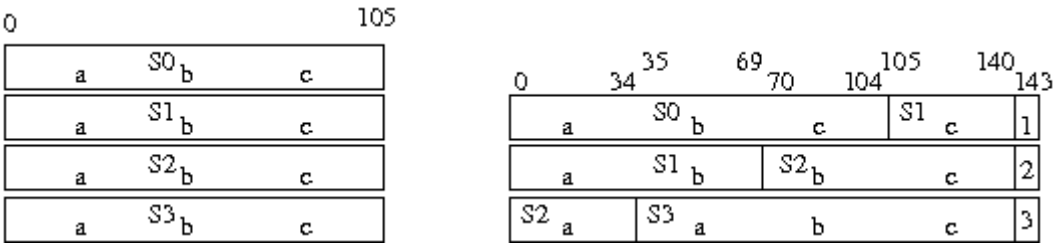
The F2F module consists of the following sub-module:

- Select-IO instance for sub-link 0.
- Select-IO instance for sub-link 1.
- AXI4Lite Register File, containing AXI4Lite memory mapped registers to control the core and monitor the link status.
- Calibration Module. This module implements a calibration routine that enables the receiving core to sample the data in the middle of the data valid window. The calibration routine consists of two phases: delay setting and word synchronization. After the initial configuration and during the delay setting phase, the transmitting core transmits a predefined pattern over the F2F link. On the receiving side, the calibration state machine performs a delay sweep using the Xilinx IDELAY module to identify the delay value that places the sampling time in the middle of the data valid window. Furthermore, when the optimal delay has been set, the calibration state machine search for the correct bit-slip to be applied in order to reconstruct properly the pattern. This procedure is repeated on each LVDS lane. When the delay setting phase is concluded, each LVDS lane is sampled at optimal time at the receiving side; however there could be a skew of several clock semi-periods among different lanes. This skew is compensated using a shallow FIFO in the receiving core and aligning each lane to the lane having the largest delay. This is accomplished in the synchronization phase, during which the transmitting FPGA transmits a specific synchronization word over each F2F link lane. Since the skew among lanes can be different, the word may appear at the receiving side of the link at different times; data are not released from the FIFO until all lanes receive the synchronization word, thus ensuring that all lanes are sample-aligned to the transmitting side.

The F2F user interface consists of a single data bus of 18*SERIAL_FACTOR bits, where SERIAL_FACTOR represents the ratio between the provided parallel clock and the internal clock generated by Select-IO core using a PLL. SERIAL_FACTOR is set to 8 in the current implementation resulting in 144 bits internal parallel bus.

Provided that the F2F link has been configured and calibrated, data presented on the user TX data bus in the transmitting FPGA will appear transparently on the user RX data bus in the receiving FPGA after the latency involved in the process.

On top of the F2F link a cross clock domain module allows the F2F bus to be seen in the 200 MHz adc_clk clock domain. F2F bus direction is selected by the FPGA position, bus 0 is used as output from FPGA1 and input from FPGA 0, and bus 1 is used in the opposite direction. 140 out of the 144 parallel lines from each de-multiplexed F2F bus, clocked at 156.25 MHz, are mapped to 105 lines clocked at 200 MHz, and the remaining 4 lines are used internally for handshaking. The protocol uses 4 clock cycles at 200 MHz for every 3 cycles at 156.25 MHz, with some idle cycles inserted when necessary. As the involved clocks are synchronous, once the protocol is started data presented on the transmitting side will appear on the receiving side as a continuous stream, with a fixed but slightly unpredictable delay.



The 105 lines are used to convey a few separate data streams, requiring some of these lines to be used for nested handshaking. 50 are used to exchange beamformed samples, and the remaining to access the DDR memory in FPGA1 from FPGA0.

5.2.3.7 40G

The 40G architecture is described in [RD27].

5.2.3.8 DDR3 IF

The DDR3 IF is a wrapper around the Xilinx Memory Interface IP core [RD14]. The user interface consists of separate address and data busses and few control signals permitting to have seamless access to the external DDR3 devices. Due to limitations on the maximum bit-width usable in the Xilinx MIG, only 4 DDR3 chips out of the six available are currently used. This translates to a 512 bits internal user data bus and 64 bits physical data bus, being the user interface clocked at 200 MHz and the external DDR3 interface clocked at 800 MHz and working in DDR mode.

Typical DDR3 read and write transactions are shown in the following waveform.

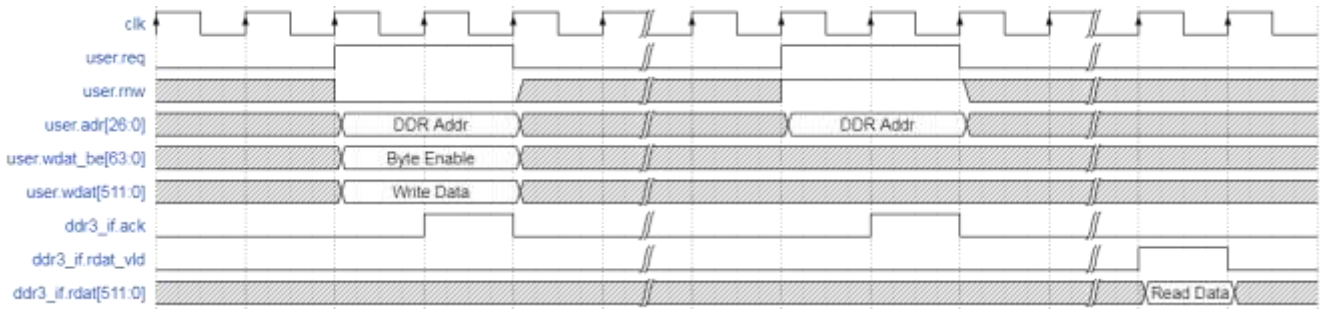


Figure 17. DDR3 Interface Read and Write transaction waveform.

In the first write transaction, user asserts *req* signal keeping *rnw* at low logic level and presenting transaction attributes defined by DDR Address, Write Byte Enables and Write Data signals. When the DDR3 core acknowledges the request asserting *ack* signal, the transaction attributes are internally registered and the Xilinx IP core starts processing the request.

The Xilinx IP core is able to internally pipeline multiple DDR commands; the delay from *req* assertion to *ack* assertion depends on the internal state of the Xilinx IP core.

Similarly, in the second transaction user asserts *req* along with *rnw* presenting the requested DDR address. Read Data follows after a delay determined by the controller read latency and its internal state. Read data is qualified by *rdat_vld* signal.

The Xilinx IP core is able to internally pipeline multiple DDR commands and supports multiple DDR3 bank management permitting to overlap command executions when they address different DDR3 banks; the delay from *req* assertion to *ack* and *rdat_vld* assertion depends on multiple factors, i.e. the internal state of the Xilinx IP core, the state of the DDR3 memory, the pattern of accessed DDR3 addresses, further information can be found in [RD14].

5.2.3.9 AXI4S C2C Bridge

The main purpose of this module is enabling the FPGAs to communicate to the CPLD using a channel having a throughput close to 1 Gbit/s in order to transmit data from the FPGA over the Gigabit link managed by the CPLD. This communication channel is mainly used for debug purposes.

This module receives an AXI4S stream and forwards it to the CPLD using the C2C streaming bus for transmission over the 1G network. It accepts a 32 bits wide AXI4S stream and serializes it over the 4 bit wide C2C Stream bus using DDR output registers. No internal buffering is provided; back-pressure signal *c2c_stream_rdy* from the CPLD is propagated backward on the AXI4S bus using *tready* signal, it is up to the AXI4S streaming module to stop the transmission when *tready* is not asserted.

5.2.3.10 Clock Manager

This module includes clock and reset generation logic. The FPGA clock and reset tree is described in 5.2.2.

5.2.3.11 PPS Manager

The PPS manager contains the logic used to sample the external PPS signal and generate an internal PPS signal that is synchronized across multiple TPMs to the single ADC sample. The synchronization scheme is described in a separate section of this document 5.2.7.2. The PPS manager provides an internally generated PPS signal when the external PPS signal is not available.

5.2.3.12 System Monitor

The System Monitor module is a Hard IP core providing ADCs for digitizing external voltages, on-chip supply voltage sensors and FPGA temperature sensor. Digitized values are written into specific registers that are memory-mapped on the AXI4Lite bus. The comprehensive list of monitored voltages is the following:

Name	**Voltage (V)**	**Description**
VM_AGP0	0,985 V	AD AGP group 0
VM_AGP1	0,985 V	AD AGP group 1
VM_AGP2	0,985 V	AD AGP group 2
VM_AGP3	0,985 V	AD AGP group 3
VM_CLK0B	0,881 V	Clock Buffer0 3,3V
VM_DDR0_VREF	0.675 V	DDR3 Vref
VM_DDR0_VTT	0.675 V	DDR3 Vtt
VM_FE0	0,934 V	preADU 3.5V
VM_MAN1V2	0,821 V	Management 1.2V
VM_MAN2V5	0,913 V	Management 2.5V
VM_MAN3V3	0,881 V	Management 3.3V
VM_MGT0_AUX	0,657 V	MGT0 AUX 1.8V
VM_PLL	0,881 V	PLL 3.3V

Note. Voltage are scaled with resistor partitors to adapt to voltage range handled by the FPGAs.

The System Monitor provides also access to the internal FPGA temperature diode.

Further information about Xilinx System Monitor can be found in [RD26].

5.2.3.13 AXI4Lite DRP Bridge

The AXI4Lite to DRP Bridge provides access to the Dynamic Reconfiguration Port implemented in several Xilinx primitives. The Xilinx primitive connected to the DRP Bridge are all GTH transceivers used in the JESD204B and 40G cores, related QPLLs and the MMCM used in the clock manager to generate *adc_clk*. For each of these devices the corresponding DRP register are mapped transparently to the AXI4Lite bus at specific offset. The DRP port allows the user to reconfigure dynamically and monitor the status of the related Xilinx primitive. In case of the MMCMs, this feature can be used to support several ADC sampling frequencies using the same bitfile. The GTH transceiver DRP port can be used to dynamically modify the operation parameters of the GTH such as pre-cursor, post-cursor, min-cursor, etc. Additionally it can be used to acquire eye diagram data to evaluate the link performance during normal operation.

5.2.4 DSP Section

Figure 18. DSP section block diagram shows the DSP section block diagram which consists of several processing and monitoring modules. Both the internal communication between the implemented modules and external communications to the IO ring section make use of AXI4Lite interfaces to access the configuration registers implemented in each module and AXI4S interface to implement data streaming interfaces. Auxiliary modules monitor and extract data at several points along the processing chain; extracted data are forwarded to the IO Ring and transmitted on either 1G or 40G Ethernet. Some of the modules are encapsulated in a wrapper that manages the interface with the standard AXI4 stream interconnections and permit to select among different implementation architectures, when multiple architectures have been developed. In the following subsections each module is detailed further.

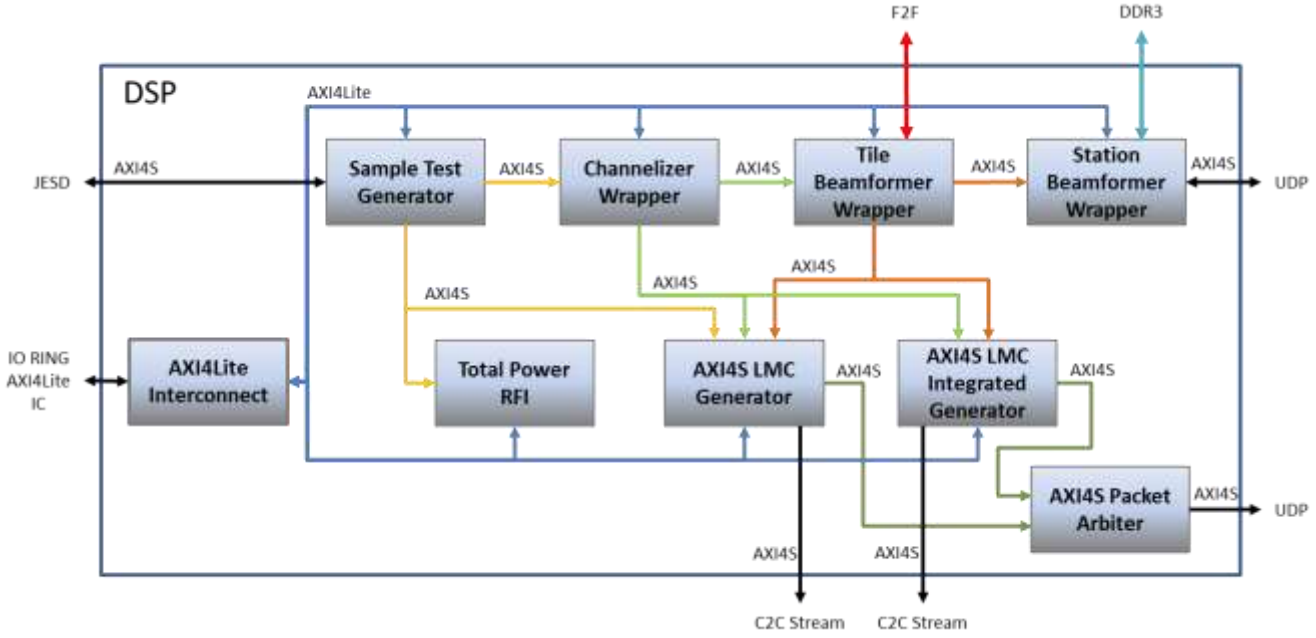


Figure 18. DSP section block diagram

5.2.4.1 AXI4Lite Interconnect

This module is connected to IO Ring AXI4Lite Interconnect and performs a similar function decoding AXI4Lite bus transactions and managing the connected AXI4Lite slaves. AXI4Lite register interfaces and interconnection logic are specified in a set of XML files which together define the global FPGA memory map. VHDL implementing AXI4Lite slave logic and interconnection is automatically generated using XML2VHDL Python script as described in section 6.2.

5.2.4.2 Sample Test Generator

The Sample Test Generator module is able to synthesize a test signal that is obtained by mixing the following components:

- 2 signal tones with independently specified phase and frequency
- Pseudorandom noise
- Periodic pulse

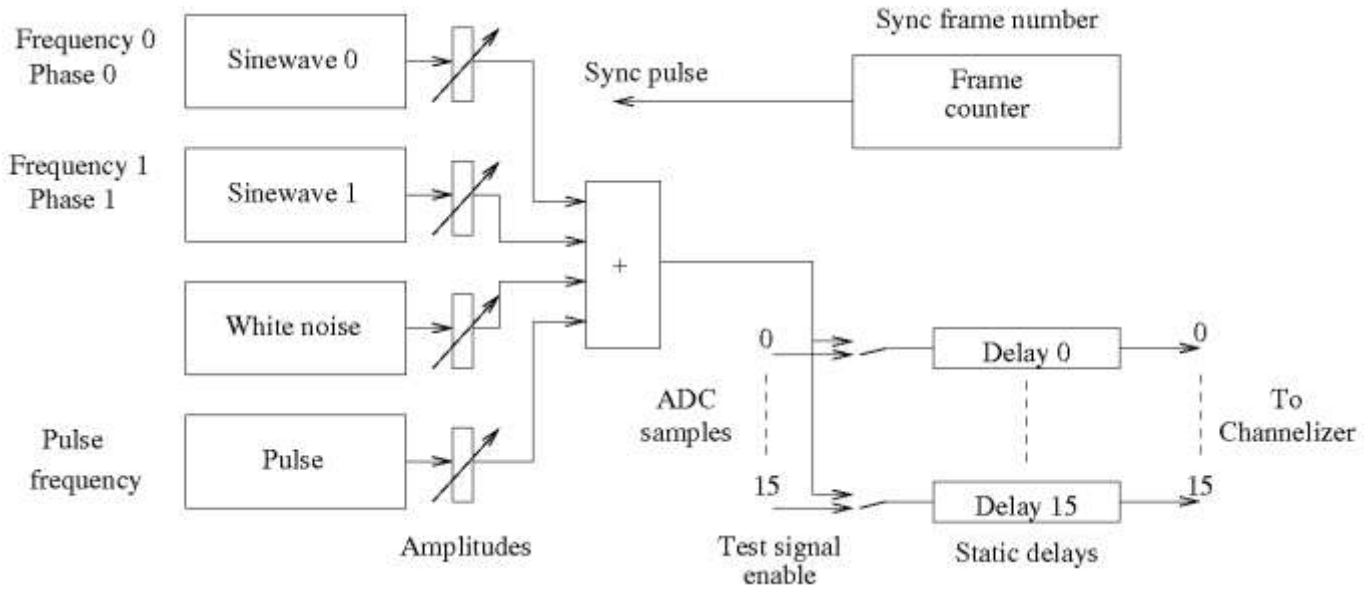


Figure 19. Structure of the sample test generator.

Tones are generated by a direct digital synthesizer, with a frequency resolution of 31 bits (0.37 Hz/step) and a phase resolution of 2048 steps/turn. The generated sinusoid is 8 bit wide and is rescaled to 32 ADC units. The pseudorandom noise is obtained as the average of 8 uniformly distributed 8 bit signed values. The resulting distribution is almost Gaussian, with zero mean and a standard deviation of 26.179 ADC units. The kurtosis is 2.85, slightly flatter than for a true Gaussian. The periodic pulse is a single nonzero sample of nominal value of 127 ADC units, which is repeated from 1 to 16 times in each 864 sample frame. All signals can be synchronized to a specific time, in order to have the same signal across multiple TPMs.
Amplitudes of each signal component can be independently specified, as an 8 bit signed quantity in the range 0 to 255/256. The maximum values for the tones and the pseudorandom noise RMS level are thus reduced to 31.875 and 26.03 ADC units, respectively.

The generated test signal can substitute the data from all or a subset of the acquired antenna. Additionally, this module is able to delay each antenna signal by an integer number of ADC samples on a per antenna basis within the delay range of 0 to 255 ADC samples, allowing for compensation of cable delays. Nominal delay is set to 128 samples, to allow both positive and negative delays.

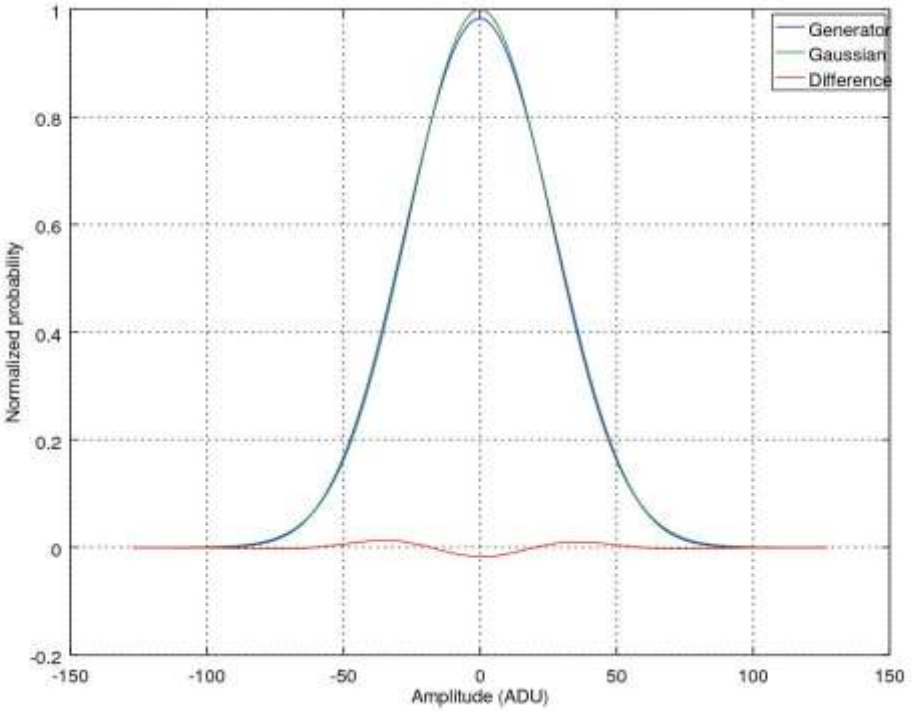


Figure 20. Distribution of the pseudorandom white noise sample amplitude.

5.2.4.3 Channelizer Wrapper

This module is a wrapper around the Channelizer core. Main functions implemented in the wrapper are:

- Channelization. Performs a channelization in frequency of the input samples for all 16 signals (8 antennas, 2 polarizations), using a polyphase oversampled filterbank. Produces a set of 512 complex frequency channels for every 864 input real samples. The implemented Channelizer is described in [RD20][RD21][RD24]. Channelizer is in turn composed of a polyphaser filter and a real valued input FFT engine.
- Post-FFT equalization by application of power-of-2 divider parameters which are defined for each frequency channel from each antenna. Parameters can be programmed via AXI4Lite registers. Each parameter specifies the number of LSB to round off the 20+20 bits Channelizer output sample to obtain a 12+12 bits data sample that is forwarded to the next stage of the DSP chain.
- Selection between behavioral and Xilinx Ultrascale optimized Channelizer implementations. Taking advantage of the specific features of Xilinx Ultrascale DSP primitives, the optimized code allows the Channelizer to efficiently use FPGA resources achieving higher clock frequencies, lower resources utilization and lower power consumption. Details about optimization are presented in [RD13]. The non optimized

version is implemented using parametrized, vendor independent VHDL code and can be easily tailored to different applications.

- Forwarding of channelized and equalized data to LMC modules including data width reduction and rounding to 8 bits.

The wrapper input AXI4 stream is composed of a total of 64 parallel 8 bit samples, organized in 8 antennas, 2 polarizations, and 4 parallel time multiplexed samples. Samples are numbered from oldest to newest, as in Figure 16. The total data width is 512 bits.

The polyphaser filter uses 18 bit coefficients and performs all the arithmetic in full precision. Coefficients are specified in a VHDL file, generated by a Matlab procedure from filter specifications. Result is rescaled to 12 bits by dividing the result by 2^{15}. As the maximum possible result Output format from the polyphaser filter is the same than for the input, but using 12 bits per sample. Timing is also different:

- Samples are clocked at a higher rate (32/27 times the input clock), in order to provide 1024 samples to the FFT engine for each 864 input ADC samples;
- Filtering is performed on an input segment that is partly overlapped, by 1024-864 = 160 samples;
- Output frame is rotated in order to always have a sample that is a multiple of 1024 samples as the first sample in a frame. This prevents frequency rotation of the PFB channels.

The FFT engine produces a frame of 513 channelized samples for each input frame. Channel 0 and 512 (real only) are packed in a single value, but in our application they are never used. Frame format is shown in Figure 21. Each polarization uses 48 bits of the data bus. Two contiguous signals represent the two polarizations for one antenna, and a total of 8 antennas are represented in 768 bits. The tvalid signal goes low after a reset, and returns high for the first valid channelized sample. This occurs after the whole filter buffer has been filled with valid data, i.e. after an initial delay of 16384 samples.

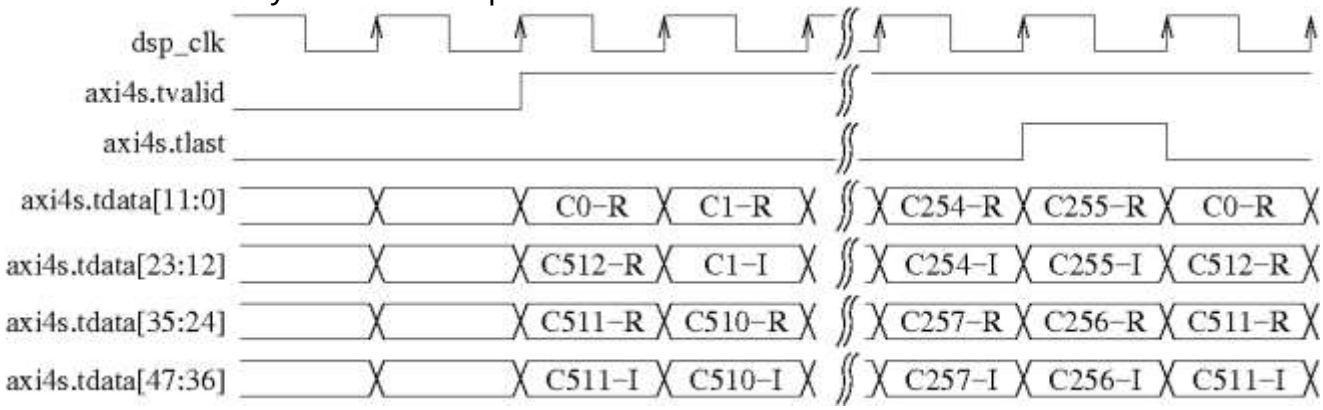


Figure 21. Polyphase filterbank output frame structure.

The FFT uses 18 bit arithmetic up to the final stage, which uses 20 bit accuracy. The channelizer never saturates up to input sample amplitudes of 70 ADU RMS, even for a tone entirely contained in a single channel. If a spectral channel overflows, the channelizer will flag the affected samples and all the DSP chain will propagate the flag. The expected RMS amplitude for normal operations is between 10 and 30 ADU, with a nominal value of 19 ADU. For amplitudes above 45 ADU RMS it is unlikely that the ADC will operate with sufficient linearity to provide useful data, and the above limit will not degrade the overall performance.

Resampling in the equalization stage checks for overflow. When the sample exceeds the representable value for 12 bits by less than a factor of 2 (i.e. it would be representable with 13 bits), the value is truncated to the maximum 12 bit value. If it exceeds that value by a larger value, it is flagged. In this way occasional large values (always present in a Gaussian noise) are

not flagged, but anomalous samples are. Samples flagged by the RFI wideband detector are also flagged.

5.2.4.4 Tile Beamformer Wrapper

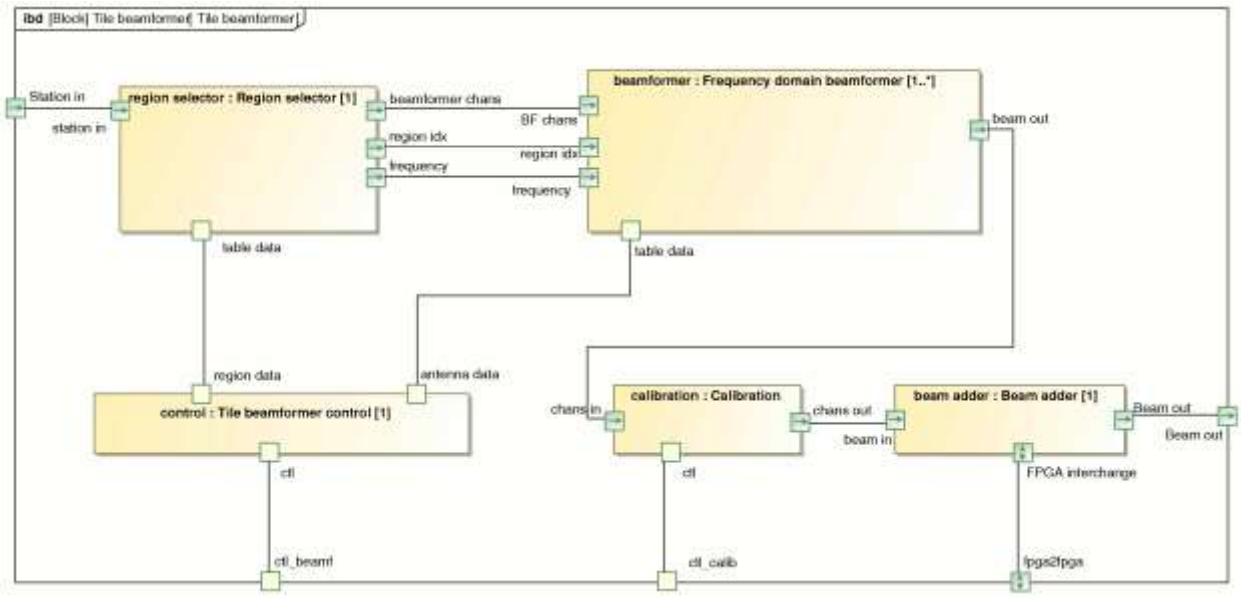


Figure 22. Tile beamformer structure.

The Tile Beamformer architecture is described in [RD22][RD24] and in Figure 22. It performs the following functions:

- Selection of up to 16 spectral regions. Regions may repeat and overlap. They represent different beams, subarrays, or simply disjoined parts of the spectrum. The total number of channels in all regions is limited to 384 (300 MHz).
- Delay correction is performed for each beam, by applying a phase correction proportional to the delay and channel frequency. Delay is computed internally to the module using a linear approximation (start delay and delay rate). Delay center is common for all antennas in a station.
- Calibration of each channel, including correction for polarization errors and rotation. Calibration is specified as a complex 4x4 matrix for each antenna and channel.
- Requantization to 12 bits of the result. This allows for bit growth using 16 bits samples in the station beamformer travelling sum.
- Tile beamforming. Aligned antenna signals are summed together. Odd and even channels are exchanged among FPGAs, with FPGA0 processing even channels and FPGA1 odd channels.

As at most 384 channels are selected, after the first block the signal is processed using the 200 MHz ADC clock. At this stage a frame is composed of 192 (odd or even) complex, 12+12 bit samples. Correspondence of logical channels to physical channels depend on programmed values in the selection block. Frame physical duration is 0.96 µs, with a frame period of 1,08 µs, and therefore the tvalid signal remains low for 0.12us (24 clock cycles) between frames. Frame structure is shown in Figure 23.

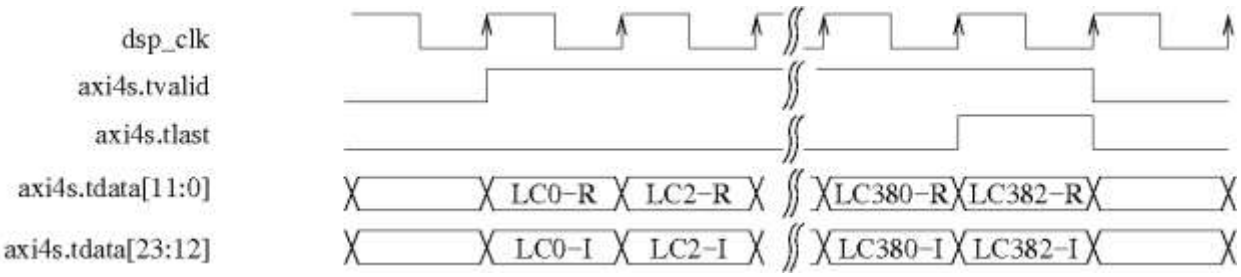


Figure 23. Tile beamformer output frame structure.

5.2.4.5 Station Beamformer Wrapper

The Station Beamformer architecture is described in [RD23] and [RD24]. It performs the following functions:

- 8 input frames of tile beamformed samples are stored in a FPGA internal memory buffer. These samples are then stored in DDR memory in time order, in blocks of 4 consecutive channels.
- In the first TPM serving a station, blocks of 256 consecutive samples, 2 polarizations, 4 frequency channels are framed in a UDP packet, together with a simplified SPEAD header, and sent to the 40G link via the UDP formatter
- Other TPMs wait for incoming packet, retrieve from DDR memory the corresponding packet, sum the two together and send the resulting packet to the 40G link
- Station Beamformer in FPGA1 reads the DDR memory in FPGA2 using the F2F interconnection.
- Last TPM in a station reformats 8 packets into 4 separate packets for the CSP (one per channel, 2048 samples each).
- Last TPM also reformats these packets for the transient buffer storage, on the MCCS servers. Transient buffer packetizer is not yet implemented.

Memory access occurs in bursts of 128 memory accesses with access interleaved over 8 DDR memory pages to optimize DDR bandwidth usage. Each memory word (48 bits used) stores one complex sample for both polarizations. An arbitrer is used to discipline simultaneous write and read accesses.
Sample size in the travelling sum is 16 bits, to account for bit growth in the sum of 16 tile beams. The final result is requantized to 8 bits for the CSP, and to 2 bits for the transient buffer. Transient buffer requantization is dynamic, with the 2 bit threshold computed for each 2048 sample packet and stored in the associated header.
AAVS1 version of the station beamformer operates identically on both FPGAs. Two 40 GbE interfaces are used for each TPM, one for the odd channels and one for the even channels. To reduce system costs, the SKA version will use only one GbE, in FPGA 0. The cornerturner operates identically in both FPGAs, while the adder and output buffer will be active only in FPGA0. When the beamformer in FPGA0 needs to retrieve a sample segment in FPGA1, it sends a request over the F2F interface, and FPGA1 sends the segment using the same interface.

5.2.4.6 Total Power RFI

The total power for each input signal is calculated in parallel on all antennas and polarizations. Total power level can be used to set the preADU attenuators to the optimal ADC input level, and in general to assist the calibration. The integrator keeps also two running means of the total

power level, over 2 input frames (2 microseconds) and over 2^{20} samples (1.25 ms). If the short term mean exceeds the long term mean by a programmable factor, the affected frames are considered contaminated by RFI, and optionally flagged.
Flagging is individually enabled on each input signal, in order not to include disabled antennas. The flag is applied on the channelized signals, at the channelizer output. The flag signal is delayed by the polyphase filter latency, and applied to all frequency channels in the affected frames. The beamformer flags the beam samples if one of the contributing signals is flagged.

5.2.4.7 AXI4S Packet Arbiter

The AXI4S Packet Arbiter selects one AXI4S stream among the connected input AXI4S streams. The selection happens on a per packet basis, meaning that an entire AXI4S is sourced from the selected AXI4S input and forwarded to the output before the next packet is processed. The arbiter pauses the non-selected stream by applying back-pressure using *tready* signal; a suitable buffer in the upstream module should be implemented to cope with the involved delays.

5.2.4.8 AXI4S LMC Generator

This modules is responsible for extracting data from the monitored AXI4S streams and packetizing them into SPEAD frames that are further forwarded towards the IO Ring in order to be transmitted over 1G or 40G network. The main performed tasks are:

- Reception of data streams from Sample Test Generator, Channelizer Wrap and Tile Beamformer Wrap.
- Selection and buffering of requested data according to software configuration.
- Encapsulation of buffered data into SPEAD packets.
- Transmission of SPEAD packets over AXI4S interfaces to either C2C Stream or 40G modules in the IO Ring.

The following figure shows the LMC Generator block diagram.

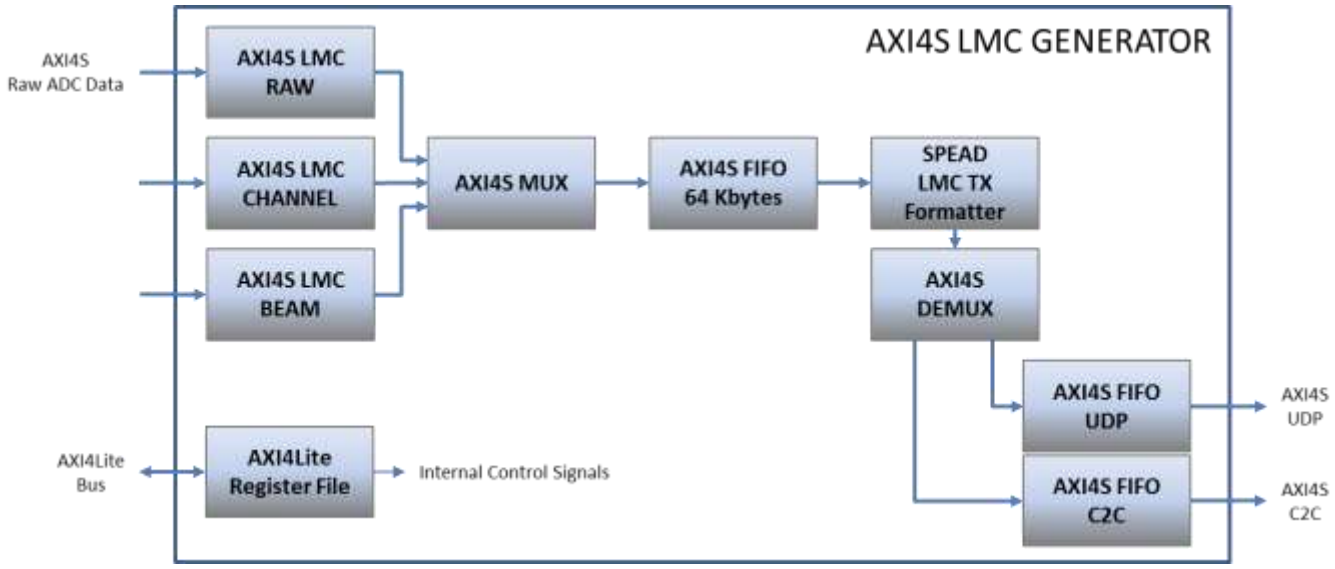


Figure 24. AXI4S LMC Generator block diagram.

The LMC Generator transmits monitored data when SW configures the AXI4Lite registers requesting the firmware to stream a specific data type. Available data types are:

- **Raw ADC Data** consisting of one snapshot of 32 Kbytes contiguous ADC data from each dual polarization antenna. Data snapshot from each antenna are extracted sequentially at different time. Data format is 8 bits real.
- **Synchronized Raw ADC Data** consisting of 64 Kbytes of synchronized and contiguous ADC data from each dual polarization antenna. A single data snapshot is extracted from all antenna at the same time. Data format is 8 bits real.
- **Channel Data** consisting of consecutive sample of channelized data from all dual polarization antenna. The number of consecutive samples is configured by SW. FW streams the configured number of samples of each frequency channel sequentially from frequency channel 0 to channel 511. Data format is 8+8 bits complex.
- **Continuous Channel Data** consisting of a continuous data stream of consecutive channelized samples of a single frequency channel programmed by SW from all dual polarization antenna. Data format is 8+8 bits complex.
- **Beamformed Data** consisting of 64 Kbytes of beamformed dual polarization data from the Tile Beamformer output. Data format is 16+16 bits complex resulted from the sign extension of 12+12 bits complex data from Tile Beamformer.

One specialized sub-module for each input data stream handles the related data types. The sub-modules are responsible for writing the incoming data into the 64 Kbytes AXI4S FIFO after being properly muxed. As the incoming data stream are generated in different clock domain, *adc_clk* and *dsp_clk,* the sub-modules are also responsible for clock domain crossing and data width conversion using shallow internal AXI4S FIFOs when necessary. After data has been written into the 64 Kbytes FIFO, the involved sub-module monitors the empty signal from the FIFO and waits until the FIFO is empty before writing the next data snapshot or clearing the request register written by SW. In case of Channel Data, the FIFO is continuously written as the write data rate never exceeds the read data rate, as shown in the following table.

Data Type	Source clock	Data Rate (Mbyte/s)	Note
Raw ADC Data	adc_clk	1600	Single dual polarization signal. 8 bits real data.
Synchronized Raw ADC Data	adc_clk	12800	All dual polarization signals. 8 bits real data.
Channel Data	dsp_clk	29.6	Dual polarization, single frequency channel. 8+8 bits complex data.
Continuous Channel Data	dsp_clk	29.6	Dual polarization, single frequency channel. 8+8 bits complex data.
Beamformed Data	adc_clk	1200	Maximum of 300 MHz bandwidth. Dual polarization. 16+16 bits complex data

Table 6. LMC FIFO Filling rate.

The FIFO is read by SPEAD LMC TX Formatter which is responsible for packetizing the data into several SPEAD packets generating the relevant SPEAD headers. SPEAD protocol is defined in [RD25] and widely adopted by the radio astronomy community.

SPEAD packet headers for each data type are generated according to Table 7. LMC Raw Data SPEAD Format., Table 8. LMC Channel Data SPEAD Format. and Table 9. LMC Beamformed Data SPEAD Format. Related SPEAD items are detailed in section 5.2.5.

5.2.4.9 AXI4S LMC Integrated Generator

Similarly to the AXI4S LMC Generator, this module receives several AXI4S data streams generated along the processing chain. Data is processed internally and integrated spectra are generated and then forwarded towards the IO Ring in order to be transmitted over 1G or 40G network. The related block diagram is shown in the following Figure.

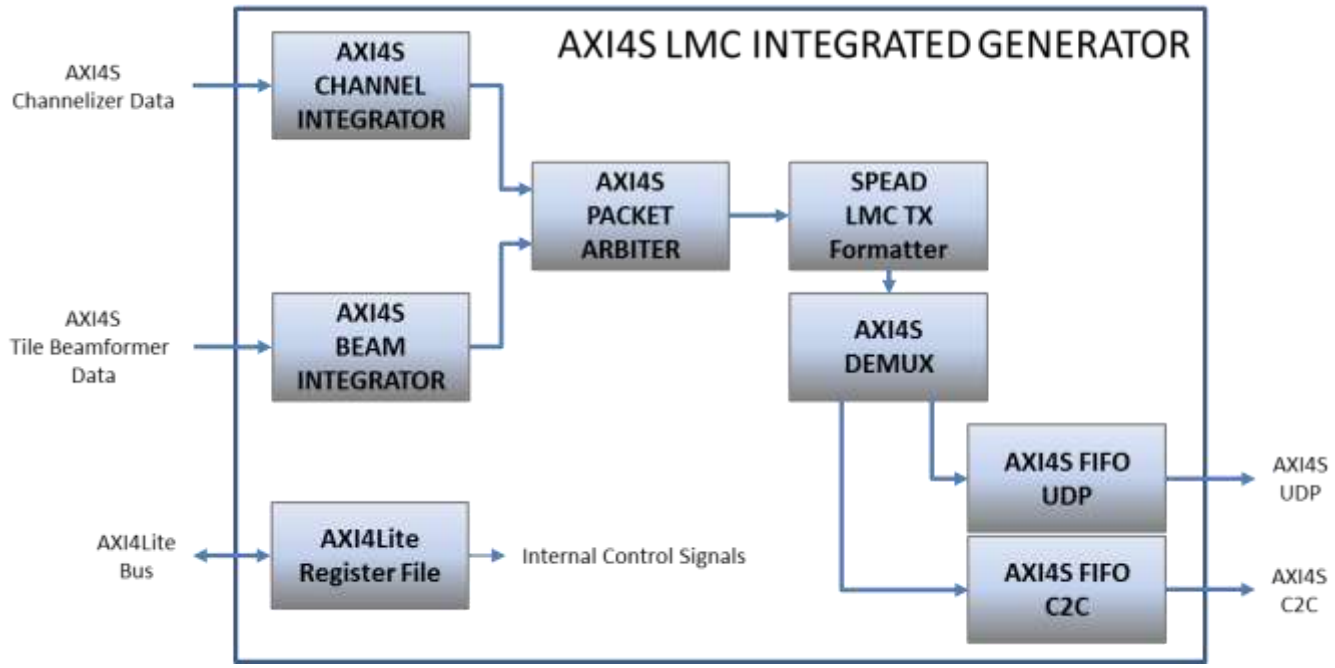


Figure 25. AXI4S LMC Integrated Generator block diagram.

The AXI4S Integrator module accepts an AXI4S stream from the Channelize or Tile Beamformer where each AXI4S packet corresponds to a channelized or beamformed frame respectively. Each sample in a frame is then interpreted as belonging to a specific frequency channel; channel power is calculated and accumulated over successive frames with the number of integrated frames programmable by SW. For each frequency channel the integrator calculates:

$$\sum_{k=0}^{N-1} a^2 + b^2$$

where N is the number of integrated frames, a and b are real and imaginary part of complex sample. Integrated spectra are transmitted after N frame have been processed; the internal accumulator buffer permits to read the last integrated spectrum while the new one is being written so that integrated spectra are processed seamlessly with no interruption.

The integrator processes one dual polarization antenna at a time. SW can specify the antenna to be processed or alternatively the integrator can cycles through the connected antenna generating one integrated spectrum per antenna at successive times.

The internal accumulator bit width can be specified by VHDL generic. Output values can be optionally rounded; SW can program the number of LSB to be rounded off before data transmission. In current implementation the internal accumulator is 48 bits wide, output data width is 16 bits for Channel Integrator and 32 bits for Beam Integrator.

Although the integrator generates an integrated frame having the same number of frequency channel as the input frame, SW can limit the number of frequency channels that are transmitted specifying a range of forwarded channels.

In contrast to the LMC generator, integrated data can be enabled during iTPM initialization and transmitted after the board has been synchronized. No successive intervention from the SW side is needed, however SW can stop the data transmission at any time by writing in memory mapped register. Available integrated data types are:

- **Integrated Channel Data** consisting of full integrated spectra (512 frequency channel, dual polarization) from the Channelizer. The integration time can be specified by SW as number of spectra to be accumulated. Data format is 16 bits real.
- **Integrated Beamformed Data** consisting of integrated dual polarization beamformed frames from the Tile Beamformer. Processed frequency channels depend on Tile Beamformer configuration. Data format is 32 bits real.

SPEAD packet headers for each data type are generated according to Table 7. LMC Raw Data SPEAD Format., Table 8. LMC Channel Data SPEAD Format. and Table 9. LMC Beamformed Data SPEAD Format. Related SPEAD items are detailed in section 5.2.5.

5.2.5 LMC SPEAD Data Format

As shown in section 5.2.4.8 and 5.2.4.9, specific portions of firmware extract data along the processing chain and encapsulate it into SPEAD packets that are transmitted over UDP. Three different SPEAD headers have been defined to support transmission of data fragmented into multiple SPEAD packets and support data reassembling at the receiving endpoint. Different data types imply distinct underlying assumptions that should be taken into account to correctly interpret and reassemble data.

Raw ADC Data, Synchronized Raw ADC Data and Beamformed Data represent a snapshot of consecutive data which are written into the internal FIFO at high speed and downloaded at a slower rate in order to be transmitted over the network. In this case SW can reassemble the 64 Kbytes buffer using the information provided in the SPEAD headers.
In case of Channel and Continuous Channel Data transmission, the firmware extracts a single frequency channel from the data stream and thus reduces the data rate to allow for continuous transmission over 1G Ethernet. The dimension of the transmitted data stream is either programmed by SW or indefinite. In the first case SW specifies the number of consecutive channelized samples belonging to a frequency channel to be transmitted before switching to the next channel; in the second case data is continuously transmitted unless stopped by SW.

Data is fragmented into multiple SPEAD packets depending on the channel used for transmission. In case of transmission over 1G Ethernet, data is forwarded to the C2C interface and subsequently sent to the CPLD which encapsulates it over UDP and transmits over the 1G network. As UDP packets managed by the CPLD are limited to fit into the standard Ethernet frame having length of 1514 bytes, the SPEAD formatter fragments the data into 1 Kbytes SPEAD packets. When data is transmitted over 40G Ethernet Jumbo frames are used and data fragmentation happens at 8192 bytes limit.

In regards to integrated data transmission, the total size of transmitted data is 16 Kbyte for Integrated Channel Data and a maximum of 1536 bytes for Integrated Beam Data with fragmentation occurring as explained above.

It should be noted that the above sizes are calculated per FPGA. As each FPGA transmits data autonomously, SW should reassemble data according to the FPGA identifier included in the SPEAD header. Raw and Channel data are related to distinct antenna in each FPGA, while Beamformed data are logically split between two FPGAs, with one FPGA processing and transmitting even frequency channels and the other FPGA handling odd frequency channels.
SPEAD packet headers for each data type are generated according to Table 7. LMC Raw Data SPEAD Format., Table 8. LMC Channel Data SPEAD Format. and Table 9. LMC Beamformed Data SPEAD Format. Related SPEAD items are detailed in Table 11. Predefined SPEAD items. and Table 12. Custom SPEAD items.

5.2.6 CSP SPEAD Data Format

The data format of SPEAD packets transmitted to CSP is depicted in Table 10. CSP packets contain 2048 samples, each sample consisting of dual polarization complex 8 bit data corresponding to 32 bits.

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Magic # [0x53]								Version [0x04]								Item Pointer Width [0x40]								Item ID Width [0x18]								Reserved [0x0000]																				Number of Items [0x08]															
1	heap_counter [0x0001]															FFU								FFU								packet_counter																											
1	pkt_len [0x0004]															FFU								packet_payload_length																																					
1	sync_time [0x1027]															FFU								unix_epoch_time (s)																																					
1	timestamp [0x1600]															FFU								timestamp (ns)																																					
1	lmc_capture_mode[0x2004]															FFU								lmc_capture_mode																																					
1	lmc_rawdata_info [0x2000]															FFU								FFU																																start_antenna_id								nof_included_antenna							
1	lmc_tpm_info [0x2001]															FFU								tpm_id								station_id																											
0	sample_offset [0x3300]															payload_offset = 0x0																		

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Magic # [0x53]								Version [0x04]								Item Pointer Width [0x40]								Item ID Width [0x18]								Reserved [0x0000]																				Number of Items [0x08]															
1	heap_counter [0x0001]															FFU								FFU								packet_counter																											
1	pkt_len [0x0004]															FFU								packet_payload_length																																					
1	sync_time [0x1027]															FFU								unix_epoch_time (s)																																					
1	timestamp [0x1600]															FFU								timestamp (ns)																																					
1	lmc_capture_mode[0x2004]															FFU								lmc_capture_mode																																					
1	csp_channel_Info [0x3000]															FFU								FFU								beam_id																				frequency_id															
1	lmc_antenna_info [0x2003]															FFU								tpm_id								station_id																				nof_contributing_antenna															
0	sample_offset [0x3300]															sample_offset																		

0x0001	heap_counter	logic_channel_id	For LMC data, this filed is always 0. For CSP data, it stores the logical channel identifier.
		packet_counter	For LMC data, after data has been requested by SW, the first SPEAD packet has packet counter cleared to 0; the packet counter is incremented by 1 at each transmitted SPEAD packet. In Continuous Channel Data mode packet counter wraps to 0. For CSP data, packet_counter is continuous and starts form 0 each time after the system is synchronized.
0x0004	pkt_len	Length in bytes of included SPEAD payload. Value reflects the register value programmed by SW.	
0x1011	center_freq	Nominal center of the channel in Hz.	
0x1027	sync_time	Time at which the system was last synchronized (armed and triggered by a 1PPS) in seconds since the Unix Epoch.	
0x1600	timestamp	Timestamp of start of this observation. It counts the number of data frames since last sync_time (1.08 uS).	

Table 11. Predefined SPEAD items.

0x2000	lmc_rawdata_info	start_antenna_id	Identifier of first antenna included in transmitted data. As each iTPM handles 16 dual polarization antenna, this parameter has range [0, 7] for the first FPGA and [8, 15] for the second FPGA.
		nof_included_antenna	Total number of antennas included in current data. This parameter has value 1 for Raw ADC Data mode or 8 for Synchronized Raw ADC Data mode.
0x2001	lmc_tpm_info	tpm_id	Identifier of TPM within the station. This parameter is configured by SW in memory mapped register.
		station_id	Station identifier. This parameter is configured by SW in memory mapped register.
		fpga_id	FPGA identifier within TPM. This parameter reflects the logic value of *fpga_id* pin.
0x2002	lmc_channel_info	start_channel_id	Identifier of first frequency channel included in current data packet. In integrated mode it should take into

			account that a single spectrum may be fragmented in multiple packets.
		nof_included_channel	Total number of frequency channels included in current data packet. This parameter is always set to 1 in non-integrated mode.
		start_antenna_id	Identifier of first antenna included in current data transmission. As each iTPM handles 16 dual polarization antenna, this parameter is either 0 or 8 in Channel data modes. It has range [0, 7] for the first FPGA and [8, 15] for the second FPGA in integrated data modes.
		nof_included_antenna	Total number of antennas included in current data packet. This parameter is always set to 8 in Channel data modes. It is set to 1 in integrated data modes.
0x2003	lmc_antenna_info	tpm_id	Identifier of TPM within the station. This parameter is configured by SW in memory mapped register.
		station_id	Station identifier. This parameter is configured by SW in memory mapped register.
		nof_contributing_antenna	It indicates how many antenna contributed to the beamformed sample. This parameter is configured by SW in memory mapped register. It takes into account the possibility to mask some antenna data. Masked antanna don't contribute to the calculate beamformed samples. This parameter has range [0, 16].
0x2004	lmc_capture_mode	mode	0 = raw_adc_data 1 = synchronized_raw_adc_data 4 = channel_data 5 = continuous_channel_data 6 = integrated_channel_data 8 = tile_beam_data 9 = integrated_tile_beam
0x2005	lmc_beam_info	fpga_id	FPGA identifier within TPM. This parameter reflects the logic value of *fpga_id* pin.
0x3000	csp_channel_info	beam_id	Beam identifier. Each logical channel carries information for a

			physical coarse channel and for a beam. Information for this correspondence is static, and could be broadcasted to all receiving nodes in a look-up table, but can be carried along with the data for convenience.
		frequency_id	Frequency identifier. Each logical channel carries information for a physical coarse channel and for a beam. Information for this correspondence is static, and could be broadcasted to all receiving nodes in a look-up table, but can be carried along with the data for convenience.
0x3001	csp_antenna_info	subarray_id	The sub-array ID identifies the sub-array to which the logical station belongs, whereby the number of sub-arrays can be any integer number between 1 and 16 inclusive.
		station_id	Logical station identifier.
		nof_contributing_antenna	The number of antennas indicates how many antennas contributed to the sample.

Table 12. Custom SPEAD items.

5.2.7 iTPM Synchronization

The iTPM synchronization provides synchronous ADC data sampling across multiple iTPMs driven by the same 10 MHz Reference clock and PPS signal. Assuming that no skew exists between each Reference clock and PPS copy forwarded to each iTPMs, this would result in un-skewed data, within one ADC sampling period, forwarded from the JESD Interface to the DSP section across multiple boards. The iTPM synchronization requires a two level synchronization procedure: a local synchronization of clocks and JESD links and multi-board synchronization using PPS signal to guarantee that data sampled by the ADC are coherently time-tagged across multiple iTPMs.

5.2.7.1 Local synchronization

AD9528 contains two PLLs; the first PLL (PLL1) controls the on-board 100 MHz VCXO and generates a 100 MHz clock which is locked to the External 10 MHz Reference clock. The 100 MHz generated clock is:

- used by the second PLL (PLL2) to synthesize the ADC sampling clock (800 MHz),
- forwarded to the FPGA as Reference and Device clocks.

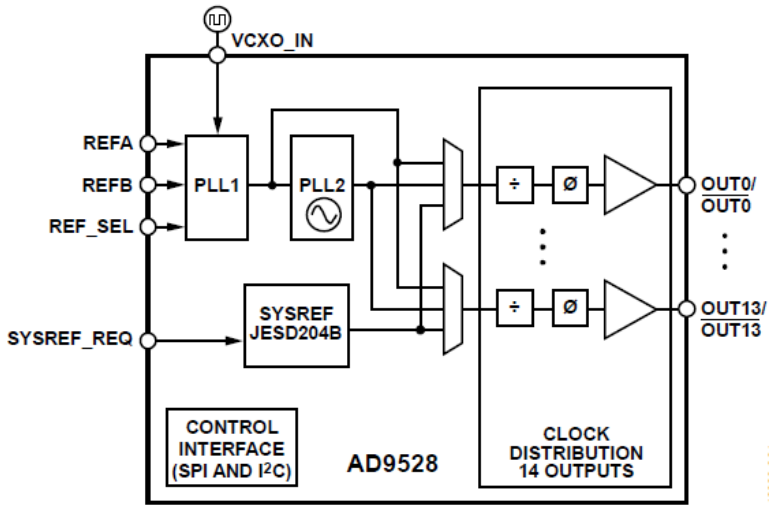


Figure 26. AD9528 Functional Block Diagram.

The FPGA generates the required *adc_clk* and *dsp_clk* clocks internally as described in section 5.2.2.1. This scheme ensures to obtain higher frequencies phased clocks as MMCMs in the FPGAs and PLL2 in AD9528 align the phase of generated clocks to their input clocks; moreover it guarantees that adc_clk signals in different iTPMs are phase-aligned to the 10 MHz Reference clock.

5.2.7.2 PPS Synchronization

As shown in section 5.2.3.5, JESD204B links are initialized and synchronized in a predictable way using the JESD204B deterministic latency feature; however when the JESD initialization happens at different times, ADC data cannot be time-tagged univocally across multiple iTPMs. This is achieved using the system wide PPS signal to generate an internal PPS signal that is synchronous to adc_clk; being *adc_clk* locked to the 10 MHz Reference Clock with the same phase relationship among different iTPMs, this signal can be used to synchronize several iTPMs and ultimately represents the trigger that enable the ADC data streaming from the JESD Interface to the DSP section.

The above synchronization scheme requires that the external PPS is sampled in the *adc_clk* clock domain in a stable manner, i.e. all iTPMs sample the external PPS at the same clock edge of *adc_clk*. Using the *adc_clk* to sample the PPS directly would cause meta-stability issue when the PPS signal edges don't meet setup and hold requirements of the sampling register. In order to avoid meta-stability issues a configurable PPS sampler has been developed.

The PPS sampler generates an internal signal, *pps_sample*, synchronous to JESD device clock, and uses it to enable the external PPS sampling. *pps_sample* can be configured as continuous or periodic, thus defining the PPS sampling windows which can range from 10 ns to 150 ns.

In case of a 50 ns PPS sampling window, the procedure to initialize the PPS sampler requires the following steps:

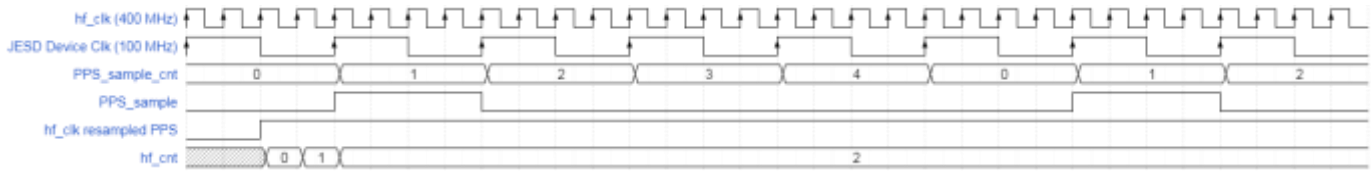


Figure 27. PPS not centered in sampling window.

- In reference to Figure 27. PPS not centered in sampling window. a free-run counter *PPS_sample_cnt* generates a 20 MHz periodic pulse signal *pps_sample* in the JESD Device clock domain.
- External PPS is sampled and synchronized into the high frequency *hf_clk* clock domain; as shown in Figure 27. PPS not centered in sampling window.using the counter *hf_cnt* running on *hf_clk* the number of clock cycles from PPS rising edge detected in the *hf_clk* domain to a rising edge of *pps_sample* is determined. As the external PPS is sampled by a DDR register, the current granularity of *hf_cnt* is 1.25 ns. The value represents the relative delay between a PPS rising edge and a *PPS_sample* rising edge.
- By checking *hf_cnt* value after a PPS rising edge, the phase of *pps_sample* is shifted in order to center the PPS rising edge within two *pps_sample* pulses as shown in Figure 28. PPS centered in sampling window.
- The PPS signal is then sampled in the JESD Device clock domain only when *pps_sample* is asserted; an internal PPS pulse is generated in the JESD Device clock domain when a rising edge is detected and it is propagated synchronously into adc_clk clock domain.

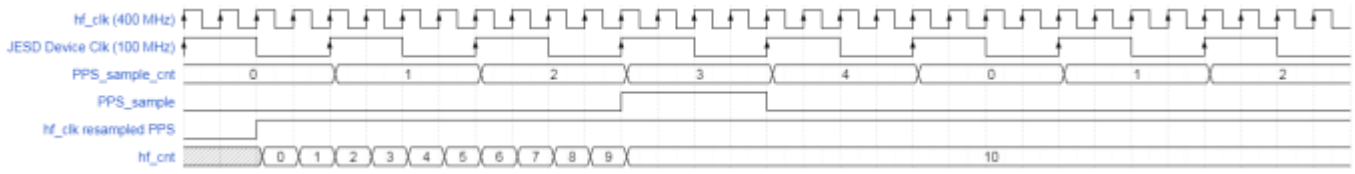


Figure 28. PPS centered in sampling window.

The procedure requires that variability between two successive PPS pulses referred to JESD Device clock is smaller than +/- 3.75 ns; if this constraint is met, the procedure can be repeated simultaneously on multiple boards resulting in a synchronized PPS pulse in the *adc_clk* domain

across multiple iTPMs. Anomalies in the result of the synchronization procedure can be detected by checking that the final value of *hf_cnt* is included in a specific range.

5.2.7.3 Time-tagging

After local and PPS synchronization, each iTPM generates a synchronous internal PPS signal in *adc_clk* clock domain which is in turn synchronous to the 10 MHz Reference clock. The internal PPS signal can be used to increment a time counter *curr_time* in the PPS Manager module which is initialized by software to contain UTC time. As the internal PPS signal is synchronous across multiple iTPMs, the *curr_time* counter is incremented synchronously at each PPS rising edge as well. Software can then initiate data streaming towards the DSP section by writing into the register *sync_time* an UTC time set in the future. When the *curr_time* assumes the same value as *sync_time*, the *start_pulse* signal is generated; this will cause the JESD Interface to start streaming data as 864 framed samples. By counting the number of streamed frames and knowing *sync_time*, each frame, and ultimately each sample in a frame, can be traced to an absolute time t_{Frame} using the equation:

$$t_{Frame} = sync_time + 864 * time_stamp * \delta t_r$$

where δt_r is the ADC sampling period and $time_stamp$ represents the number of counted frames. Channelized samples correspond one-to-one to ADC frames. A fixed offset is introduced by the firmware, and depends on the specific details of the channelizer implementation. A delay of 128 clock cycles are introduced by the cable delay compensation, to accommodate both positive and negative delays. The polyphaser filterbank introduces a fixed delay due to the necessity of preloading the filter taps before producing useful samples. As a result of both effects, channelized sample time is $\delta t_0 = 1008\,ns$ later than t_{Frame} as computed above.
Internally the time stamp is computed using 40 bit values. The programming interface uses or specifies only the 32 most significant bits of the time stamp counter, and samples are transmitted in multiples of 256 samples. Time corresponding to the first sample in a block is thus equal to:

$$t_{Sample} = sync_time + \delta t_0 + 256 * 864 * \delta t_r * frame_block$$

The synchronization time for any operation, except the initial ADC synchronization, can be specified with a granularity of $256 * 864 * \delta t_r$ = 276.48 µs. It is important to note that frames do not divide exactly one second, and usually channelized samples will not occur at integer seconds.

5.2.7.4 Command Synchronization

As the iTPMs are controlled using UCP over 1G network, delays occurring both in the controlling machine and the network cause the commands to be received and executed with different latencies. Some commands may be required to be logically executed as station wide commands and synchronized across multiple iTPMs; for instance, updated calibration coefficients may be applied at a specific time within a station. For those commands that are required to be synchronized a specific firmware implementation and software interaction is needed.

As described in the previous sections, each iTPMs manages a *time_stamp* counter that is incremented at each data frame boundary in a synchronous way across multiple boards. The execution of synchronized commands is delayed by firmware to occur at data frame boundaries.

A specific firmware module *timestamp_trigger* implements a local frame counter and a set of AXI4Lite registers. Software can arm *timestamp_trigger* by programming its internal AXI4Lite registers to a specific frame value, namely *timestamp_req*, causing *timestamp_trigger* to pulse its *trigger* output when the internal frame counter equals the programmed value. As shown previously the frame counter is related to the absolute time.

The generic procedure for synchronized command execution is shown in the following Figure. If synchronization is successful, a trigger pulse is eventually generated by *timestamp_trigger*; the firmware implementation can use this pulse to synchronize the internal execution. The delay applied by software should be chosen to allow the controlling machine to execute N UCP commands, where N is the number of iTPMs to be synchronized and the UCP command is a single register write command used to arm *timestamp_trigger.* Software is always able to detect whether excessive delays prevented it to arm the controlled iTPMs by comparing the requested trigger time to the time read from one iTPM after arming them.

5.2.8 FPGA FW limitations

The current FW uses a 4x10G UDP core consisting of 4 independent 10G lane where the first two lanes are used by the Station Beamformer and the third lane is used for LMC data. Additionally each FPGA uses its QSFP interface meaning that each iTPM needs two 40G connections to the switch. Both these limitations are planned to be removed by using a 40G core and transferring beam and LMC data from one FPGA to the other one via F2F link before in such a way that a single FPGA transmits the data processes by both FPGA over the network.

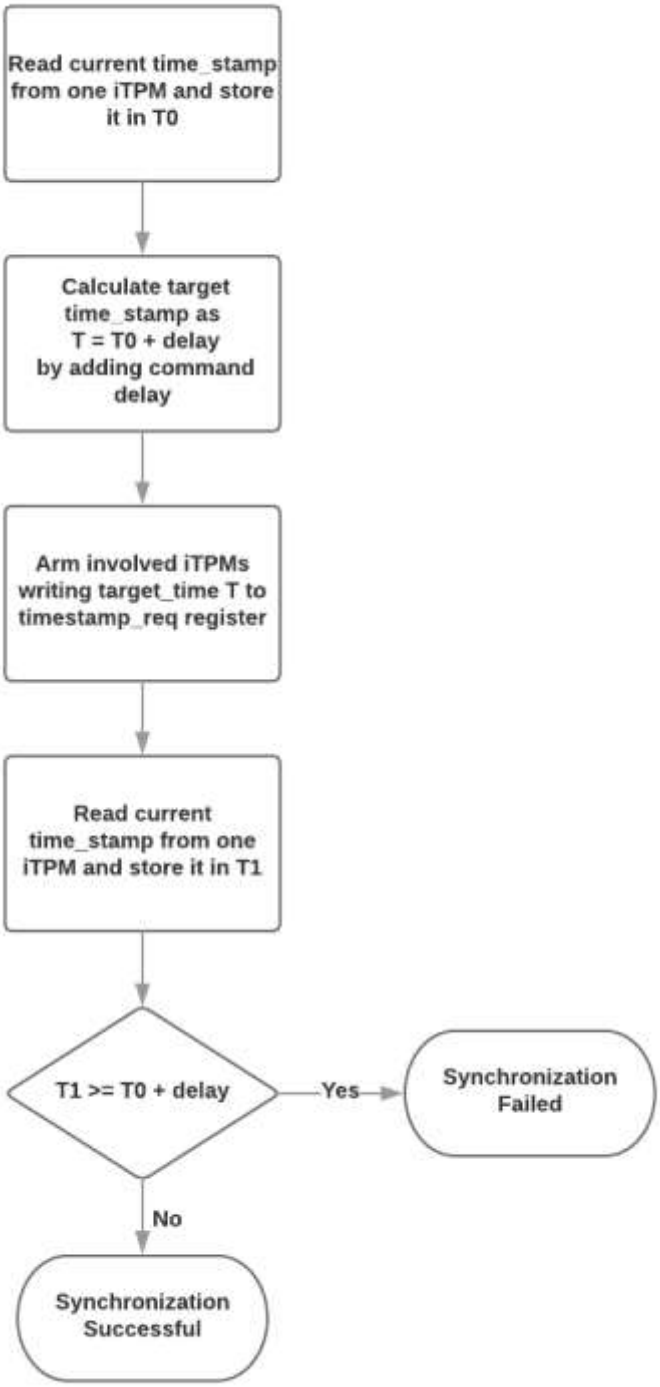


Figure 29. Command synchronization

6. AXI4LITE MEMORY MAP

6.1 XML Memory Map Description

Registers in a memory map can be logically split into components or blocks (such as Channelizer, Tile Beamformer, Station Beamformer and so on), and in turn each register can be composed of multiple bit-fields, where disjoint sets of bits have different interpretations in the firmware. It is natural to handle this type of structures in a hierarchical way using tree structures. Starting from a root node, representing a TPM memory map, the first level child nodes represent components, the second level nodes represent registers and third level nodes represent bit-fields. In the TPM these trees take the form of an XML file. Depending on its role, each XML node can have a number of attributes defined:

- **id**, which is the register, memory or bit-field name/mnemonic;
- **address**, specifying the registers physical address;
- **permission**, specifying the access permission for the register or memory block (read, write or read-write);
- **size**, the size in words of the register or memory block, the default being 1 (32-bits);
- **mask**, indicating which bits of the register compose a bit-field;
- **description**, which is a textual description for the register.

The above attributes form the essential attribute set that constitutes the information enabling the software to perform basic accesses to memory mapped registers. When a name to physical address translation routine is implemented in the software, register can be transparently displaced within the memory map.

In order to support the generation of VHDL code, the structure above is further expanded both hierarchically and in the number of defined attributes. The hierarchical expansion consists of the possibility of a) grouping together sets of registers by defining deeper structures of nodes and b) embedding a component into the current node by linking an existent XML file, thus enabling the reuse of predefined register interfaces.

Within this hierarchical representation, the actual physical address of a node is calculated by traversing the XML tree from the considered node to the root node and accumulating the value of the address attribute.

The following firmware specific attributes are also added to the previous ones:

- **hw_permission**, which controls the access type from the firmware user logic for which three modes are available: continuous write, write with enable and no write (the user logic can always read the register);
- **hw_prio**, which controls the priority between user logic and bus when a simultaneous write (at the same clock edge) occurs;
- **hw_rst**, which specifies the value that the register should assume after reset;
- **hw_dp_ram**, which, in case of a node having size greater than 1, controls the instantiation of a dual port block RAM having one port connected to the bus logic and the other available to the user logic (the block RAM can be further customized by specific attributes controlling data width, initialization via binary or hexadecimal file and read latency on both ports);
- **array**, which replicates the underlying node structure by the specified number of times (the offset between each array element is specified by the related attribute array offset);
- **link**, which links the specified XML register specification file to the current node.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
    <node id="regfile">
        <node id="rev" address="0x0" mask="0xFFFFFFFF"
            permission="r" hw_rst="g_rev"
            description="Revision register"/>
        <node id="fpga_id" address="0x4" mask="0x00000001"
            permission="r" hw_rst="no" hw_permission="w"
            description="FPGA identifier"/>
        <node id="reset " address="0x10">
            <node id="adc" mask="0x00000001" permission="rw" hw_rst="0x1"
                description="ADC clock domain reset"/>
            <node id="dsp" mask="0x00000002" permission="rw" hw_rst="0x1"
                description="DSP clock domain reset"/>
    </node>
</node>
```

Listing 1. XML memory map description example.

6.2 Automatic generation of VHDL code

The Python script xml2vhdl.py has been developed to automatically generate the VHDL code implementing a register interface described by an XML file following the conventions described above. This generates VHDL code that implements slave interfaces on a standard bus (currently supported buses are AXI4Lite and Wishbone) and the logic to interconnect multiple slaves to a single master using a shared bus architecture. This solution is adequate for the TPM where only one entity accesses the bus as master and a high throughput is not needed. However, this can be extended to include more complex and efficient bus architectures.

6.3 Slave logic generation

Listing 1. XML memory map description example. is an example of an XML describing a simple slave where three registers are specified, the last one being composed of two single bit-fields. In this case the generated VHDL code consists essentially of a package file and a top level entity with its architecture. The generated package contains the VHDL definition of the register set VHDL record and the functions and procedures to execute bus address decoding, register reset, read and write, according to the specified attributes.

Defining the register set as a VHDL record has two advantages. Firstly, when the user logic interface is defined in terms of the same record, it becomes independent from the specific register set, meaning that it doesn't change when registers are modified, added or removed from the XML file, allowing the designer to instantiate the slave top level entity once without requiring successive modifications. Secondly, the script is able to replicate the hierarchy expressed by the node tree in the XML file by creating structures of records where each record is defined in terms of lower level records. In this way the designer has the possibility to partition the registers within a slave in order to obtain a more organized and structured design.

Listing 2 shows the generated VHDL records for the considered XML example. In the top level record t_axi4lite_regfile, the member reset is defined in terms of the lower level record t_axi4lite_regfile reset.

In reference to Listing 3, the VHDL entity ports include the bus interface ports and the user logic ports, which is implemented as a VHDL record defined in the generated package. The output record of type t_axi4lite_regfile contains the instantaneous value of each defined register. When the attribute hw_permission is set to 'w', the user logic writes into the corresponding registers

assigning a value to the corresponding member of the input record. Since the hw_rst attribute of rev register is not a hexadecimal value, the script creates the VHDL generic g_rev which is used as reset value.

```
type t_axi4lite_regfile_reset is record
    adc: std_logic;
    dsp: std_logic;
end record;

type t_axi4lite_regfile is record
    rev     : std_logic_vector(31 downto 0);
    fpga_id : std_logic;
    reset   : t_axi4lite_regfile_reset;
end record;
```

Listing 2. Generated VHDL register records.

```
entity axi4lite_regfile is
generic (
    g_rev   : std_logic_vector(31 downto 0)
);
port(
    -- Bus interface
    axi4lite_aclk           : in std_logic;
    axi4lite_aresetn        : in std_logic;
    axi4lite_mosi           : in t_axi4lite_mosi;
    axi4lite_miso           : out t_axi4lite_miso;
    -- User logic interface
    axi4lite_regfile_in     : in t_axi4lite_regfile;
    axi4lite_regfile_out    : out t_axi4lite_regfile
);
end entity;
```

Listing 3. Generated VHDL top level entity.

6.4 Interconnection logic generation

One or more slaves can be interconnected together to form a bus system, this is done by defining an XML root node having the attribute type=”ic” and linking the previously defined XML files of the slaves as child nodes. In this context the attribute address of each child node represents the base address of the corresponding slave. The generated interconnection logic is responsible for performing the base address decoding and activating the slave selected by the current bus address.

The system composed by the interconnection logic and the controlled slaves can in turn be linked to a source node in an interconnection definition, enabling the creation of hierarchical bus systems. When linking a slave (or a bus system), two actions are performed: a) the address attribute of each register in the slave receives an offset which is defined by the address attribute of the source node, b) the id attribute of the source node is concatenated to the id attribute of each slave node. While doing these operations the script performs address and name checking to avoid overlap of memory areas and duplication of register names.

6.5 Global XML memory map

Starting from the slave nodes to the system root node, at each step of the code generation process the script generates an output XML file. This XML is a flattened version of the input XML where the attributes related to the VHDL generation have been stripped out. By performing this action recursively for each node, at the end of the process the script generates the global XML memory map which has the following characteristics: the hierarchy consists only of components, registers and bit-fields; for each node the attribute address represents the actual physical address; for each node the id attribute is unique.

The global XML contains the entire memory map in a single file that can be conveniently used as interface to the SW. In the TPM firmware generation toolchain targeting the Xilinx FPGAs, the global memory map is automatically generated, compressed and then embedded into the compiled bitfile by initializing a block RAM memory as described in section 7.3. In this way the compressed XML memory map file for the running firmware results in turn memory mapped at a specific address.

7. FIRMWARE REPOSITORY AND BITFILE COMPILATION

Firmware repository is currently located at https://tirgo.arcetri.astro.it/svn_radio/Channelizer/ using SVN version control system. Access to repository can be requested to [TODO email].

The repository uses a folder structure to aid the management of several designs and library components. A design is considered as a unit where several library components are used to perform higher level tasks. The designs folder contains source file that are needed to successfully generate a bitfile for one or multiple FPGA target boards; these include design specific VHDL files, constraints files and XML memory map files.

Focusing on the firmware related folders within the repository, the current structure is as follows:

- Channelizer (root)
  - Trunk
    - Firmware
      - designs. This folder contains several design subfolders; each subfolder contains source files for a specific design.
      - modules. This folder contains several subfolders; each subfolder contains source files related to a specific library component.
      - tools. This folder contains scripts and utilities to manage the repository and drive FPGA vendor tools in order to compile a design.

7.1 Modules

Each module folder consists of a specific structure of subfolder:

- Module
  - doc. This folder contains documentation specifically related to the module.
  - src
    - vhdl. This folder contains VHDL source files.
      - sim. This folder contains simulation files, including test benches and models, used to verify the module.
    - xml. When an AXI4Lite register interface is used in the module VHDL files, this folder contains the XML memory map files to generate the related VHDL code
    - xml2vhdl_generated. This folder contains VHDL and XML code automatically generated by xml2vhdl Python script.
  - hdllib.cfg. This file specify the module configuration including links to the source files that have to be included to successfully implement this module.

Modules can be self-consistent or dependent on other modules; they cannot have dependencies on design specific source files.

7.2 Designs

Each design folder consists of a specific structure of subfolder.

- Desing
  - build. This folder contains Vendor tool specific project files which are automatically generated by make scripts.
  - doc. This folder contains documentation specifically related to the design.
  - src
    - constr. This folder contains constraint files organized in different subfolders where each subfolder contains constraint files for a specific target board.
    - ip. This folder contains Vendor specific IP cores, for instance DDR3, SelectIO, and JESD204B. IP cores are contained in different target board sub-folder.
    - netlist. This folder contains netlists that are required to compile a design. A netlist is a previously synthesized module that is included into the compilation of the design.
    - revision. This folder includes a single file revision.txt containing the design revision numbers as Major Version, Minor Version and Build Number.
    - vhdl. This folder contains design specific VHDL source files.
      - sim. This folder contains simulation files, including test benches and models, used to verify the module.
    - xml. This contains XML memory maps files.
    - xml2vhdl_generated. This folder contains VHDL and XML code automatically generated by xml2vhdl Python script.

7.3 FPGA Bitfile compilation

FPGA bitfile compilation is managed by Python 2.7 based make scripts stored in tools/make repository folder. Supported operative systems are Linux Ubuntu 14.04 and Microsoft Windows 7. The required Vivado version is 2016.2.

After SVN Checkout it is necessary to initialize the repository by executing the following command from tools/config folder:

python setup.py

The script generates a config.txt file in tools/config folder. In the generated config.txt in the [MAIN] section it is then necessary to indicate the path to the Vivado executable in VIVADO_EXE field, for example:

VIVADO_EXE = opt/Xilinx/Vivado/2016.2/bin/vivado

Once the repository is initialized, the make script support a completely automated compile flow where Vivado XPR project files are generated from scratch so that no manual management of project files is needed.

The following command should be executed from /tools/make subfolder in order to generate AAVS1 bitfile:

python make.py -d tpm_test -b itpm_v1_1

where the switch -d indicates the design to be compiled and the switch –b indicates the target board, in this case iTPM board version 1.1.

The compilation process consists of several steps as shown is the following Figure 30. As first step all the XML memory map files are processed and VHDL code implementing the corresponding AXI4Lite slaves and interconnects is generated. The make script search for XML files in the module directory and re-generates VHDL files overwriting the previously generated VHDL files. The top-level XML memory map is generated, compressed into a .zip files, converted to block RAM initialization files and stored in a specific location.
After XML processing, the make script generates the Vivado XPR project file. The XPR file contains reference to all source files necessary to successfully compile the bitfile and several other project settings. If an hdllig.cfg file exists in a module folder, referenced source files are included into the XPR project. If hdllib.cfg doesn't exist or in case of design VHDL files, all VHDL files found in the relevant subfolders are included. IP cores, netlists and constraint files found in the pertinent design folder are included into the XPR project provided that they are relevant to the selected target board.

The global XML memory map file, previously converted to a block RAM initialization file, is directly referenced by the generated VHDL code describing the AXI4Lite BlockRAM module as initialization file for the implemented block RAM. In this way the XML memory map is automatically embedded, and in turn memory mapped, into the compiled bitfile.

Once the XPR has been generated, the make script provides to increment by one unit the build number stored in revision.txt file and then drives Vivado to perform the required steps to generated the FPGA bifile. A TCL script, run by Vivado when a compilation is started, reads the revision numbers from revision.txt and initializes the value of the relevant VHDL generic so that the revision numbers are memory mapped into the firmware revision register and can be read by SW.

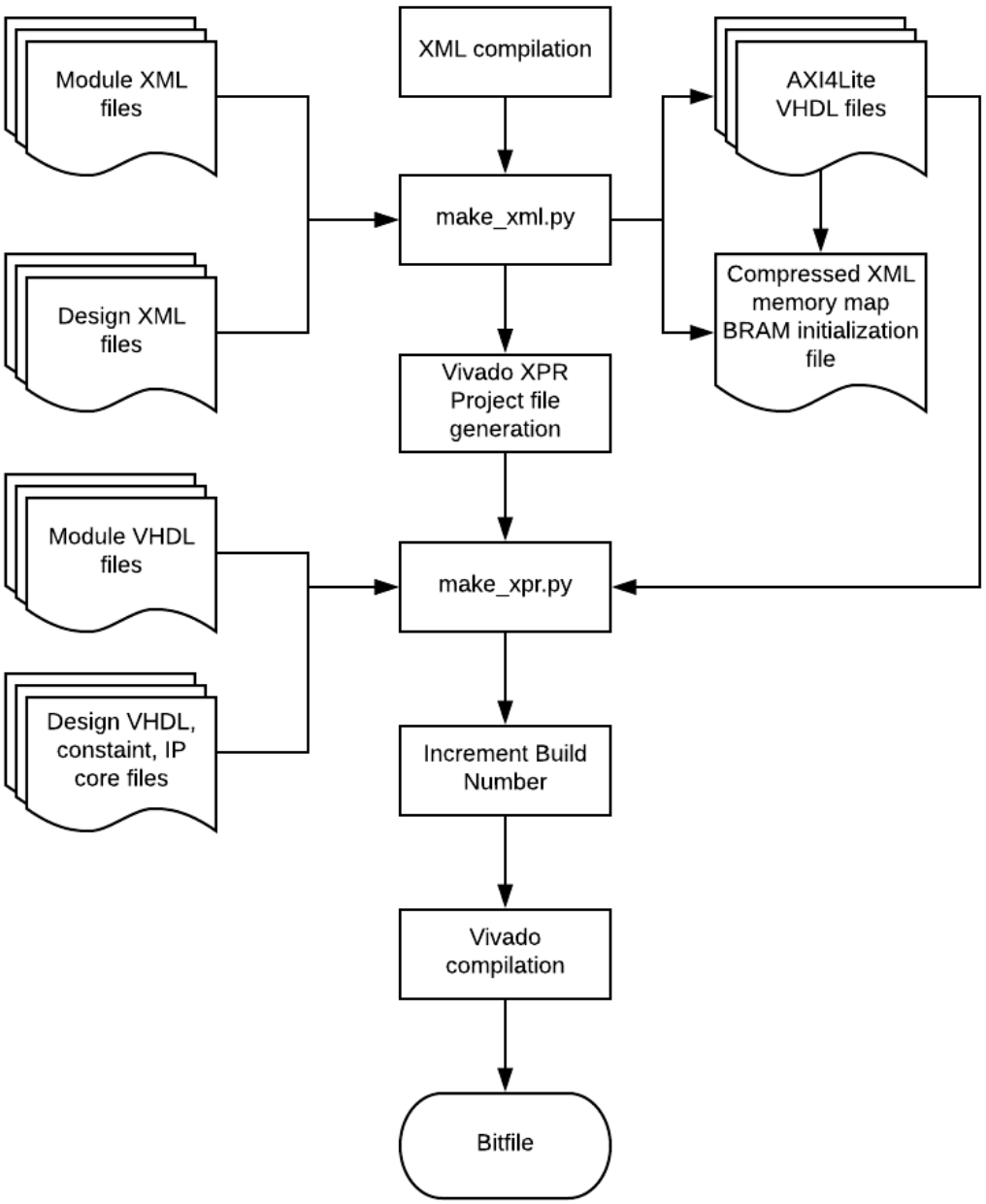


Figure 30. Bitfile compilation.

8. ITPM OPERATIONAL STATES

The iTPM operational states can be described by the following diagram.

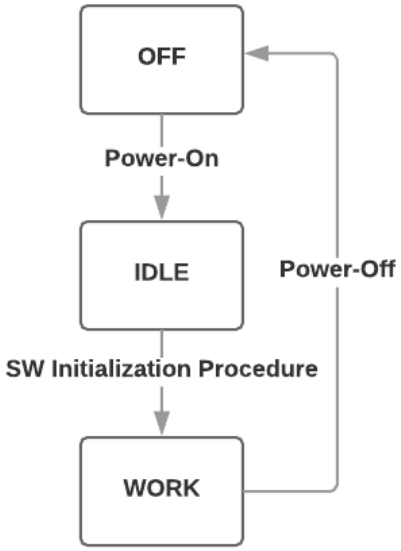


Figure 31. iTPM Operational States.

From the OFF state after power is applied to the iTPM, the CPLD boots loading the work firmware from its internal Flash memory or the golden firmware from the SPI Flash according to section 5.1.2.1; the CPLD provides to enable the on-board DC-DC converters and initializes the 1G network. In this state the iTPM is idle, with no firmware running on the FPGAs, no clock being generated by AD9528 and ADCs in reset state.

To bring the board into the WORK state an initialization procedure consisting of a specific sequence of commands issued by SW needs to be executed.

8.1 Initialization Procedure

The initialization procedure consists of several steps that are required to be executed with a specific order after the iTPM has been powered to ensure that relevant clocks are available and external devices are initialized when required.

CPLD XML Read

SW reads the CPLD XML containing the CPLD memory map. The memory map represents the symbol table by means of which SW can associate a register name to a physical address.

FPGA Programming

SW transfers the FPGAs bitfile over the 1G network addressing the SMAP slave in the CPLD. Bitfile is the written by the CPLD using the SelectMap interface. FPGA bitfile is fragmented

	into multiple UCP packets.
FPGA XML Read	SW reads the FPGA XML to acquire the FPGA memory map.
AD9528 PLL Configuration	SW addresses the SPI slave in the CPLD to access AD9528 configuration registers. AD9528 is configured to generate the required clocks and 0.3125 MHZ periodic SYSREF signals at corresponding outputs. SW can check the AD9528 SPI status registers status to detect if the PLL receives the external 10 MHz reference clock and the verify the lock status of the two internal PLLs. In case the 10 MHz reference clock is not available, the external VCO will be free-running with all derived clocks still generated.
AD9680 ADC Configuration	SW addresses the SPI slave in the CPLD to access AD9680 configuration registers. Each ADC is configured separately. SW sets the transferred data width to 8 bits and configures the JESD transmission interface with suitable parameters. SW can check the AD9860 SPI status register to detect if the ADC receives the sampling clock generated by AD9528.
C2C Stream Interface Calibration	For both AXI4S Stream interfaces connected to the FPGAs, SW configures the AXI4S C2C Bridge module in the related FPGA to generate a test pattern and verifies the sampled data in the CPLD. SW configures the CPLD internal PLL to apply a 2π/32 (250 ps) phase shift to the sampling clock and verifies the sampled data again. The process is repeated 32 times and a delay value is chosen in order to place the CPLD sampling clock edge in the middle of the valid window.
F2F Initialization and Calibration	SW programs the F2F control registers in both FPGA to select desired direction of each link. SW commands the F2F interfaces to execute calibration and checks the link status to verify if both F2F interfaces are calibrated and operational. Calibration process is described in section 5.2.3.6.
JESD204B Initialization and Synchronization	JESD cores in each FPGA are initialized and configured. SW starts the JESD204B synchronization procedure which is performed autonomously by FW according to [RD18].
DDR3 Initialization	SW resets the DDR3 Interface. Upon exiting from

	reset FW execute DDR3 initialization and calibration according to [RD14]. SW checks relevant status register to detect errors.
40G Initialization	40G link initialization and synchronization is performed autonomously by FW. SW checks relevant status register to detect errors. Several seconds are necessary to establish a valid link with the connected switch/NIC; this delay should be taken into account by SW to avoid packet transmission before the link has been established. SW can check the receiver status by reading relevant status register.
FPGA Time Synchronization	SW writes *curr_time* register into both FPGAs. This register is incremented when the FPGA detects a PPS rising edge. SW should check whether the two write commands to the FPGAs have been executed across a PPS rising edge which results in non-coherent *curr_time* in the two FPGAs. To verify correct synchronization the procedure described in Figure 32 is used.
PPS Synchronization	PPS synchronization procedure is executed according to section 5.2.7.2.
Start Acquisition	SW writes *sync_time* register into both FPGAs. After a PPS rising edge that causes *curr_time* to have the same value as *sync_time,* data streaming from the ADCs is started as described in section 5.2.7.3.

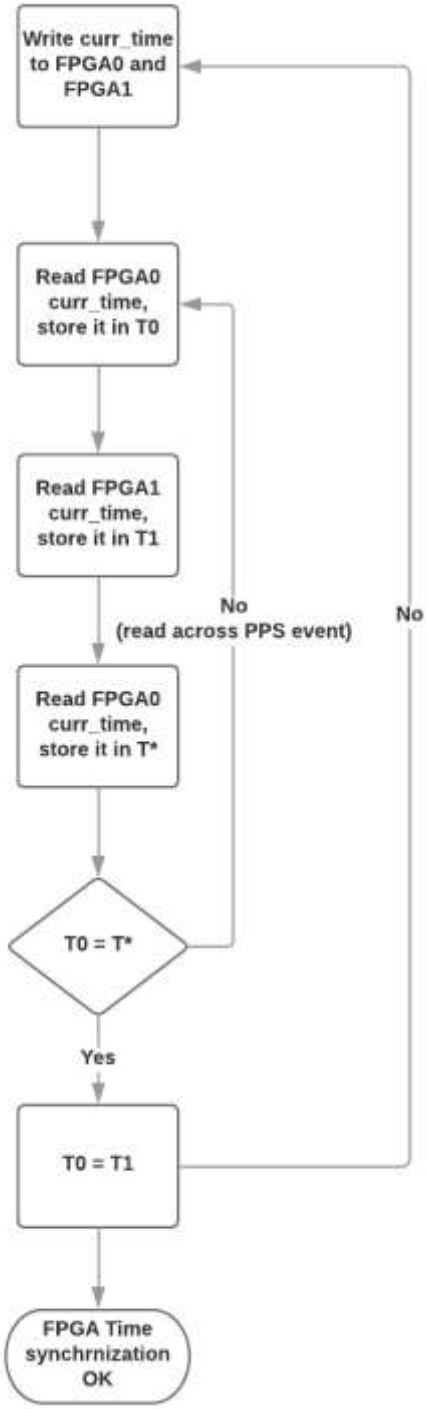


Figure 32. FPGA time synchronization procedure.

9. FPGA RESOURCE UTILIZATION

The following table shows utilization of each FPGA resource.

Resource	Utilization	Available	Utilization %
LUT	146047	242400	60.250412
LUTRAM	21923	112800	19.435284
FF	220327	484800	45.446987
BRAM	445	600	74.166664
DSP	1416	1920	73.75
IO	269	520	51.73077
GT	20	20	100.0
BUFG	32	480	6.666667
MMCM	5	10	50.0
PLL	5	20	25.0

10. FPGA POWER CONSUMPTION

The FPGA power consumption is evaluated by measuring the whole iTPM power consumption under load and after erasing the FPGA birstream memory. The difference between the two figures gives an estimation of the power consumed by two FPGAs. More specifically, a precise protocol has been defined in order to compare power consumed by different firmware. The protocol consists of the following steps:

- The voltage and current provided to the TPM are monitored
- No signal is connected to the TPM input channels
- The internal test generator is programmed to generate pseudo random noise at full amplitude.
- Each antenna signal is delayed to result staggered in time by 4 clock cycles with respect to the previous signal. Signal are delayed to prevent the DSP block in the optimized polyphase to process data-tap coefficient multiplication in two subsequent clock cycles as two antenna signal are combined in a single stream at double clock frequency.
- The delayed noise signal is selected as input to the Channelizer for all antenna inputs. Pseudo random noise is chosen for similarity to the astronomical signals and because it drives the toggle rate in the FPGA to large values resulting in higher power consumption in the DSP section.
- Power under load is measured after at least 15 minutes to allow for temperature stabilization.

In addition, after power under load is measured, the FPGAs programming memory is erased by a SW command. After 15 minutes the absorbed power is measured again.

The measured power under load consumed by the iTPM in the above condition is 140.3±0.6 W; this figure doesn't take into account the preADUs power consumption. After erasing the FPGAs the absorbed power drops to 77.6±0.6 W resulting in an estimated total FPGA power consumption of 31.4±0.6 W per FPGA. Note that this figure takes into account the loss in the FPGAs voltage regulators.

END OF DOCUMENT