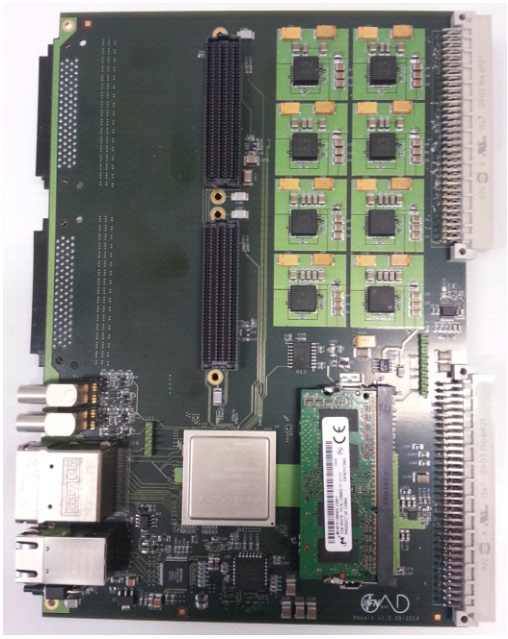

USER'S MANUAL

ALICE Pixel IT Barrel

MOSAIC Test System

INFN sez Bari, CAD service

July, 2015

Revision Sheet

Release No.	Date	Revision Description
Rev. 0	04/17/2015	User's Manual Template, datasheet, documentation int/ext, checklist
Rev. 1	05/03/2015	draft and conversion to word format
Rev. 2	05/25/2015	first draft of all sections
Rev. 3	06/15/2015	changed disposition for addressing methods (classes IPbus WBB)
Rev. 4	06/25/2015	start conversion to doxygen
Rev. 5	07/06/2015	second draft all section but doxymentation
Rev. 6	07/20/2015	changed disposition for addressing methods (classes IPbus WBB)
Rev. 7	07/23/2015	revised receiver buffer
Rev. 8	07/29/2015	current release

USER'S MANUAL

TABLE OF CONTENTS

TABLE OF TABLES

TABLE OF FIGURES

1 GENERAL INFORMATION

1.1 System Overview

MOSAIC is the acronym of "**MO**dular **S**ystem for **A**cquisition **I**nterface and **C**ontrol".

A multi device testing platform implemented by a single Field Programmable Gate Array chip.
The firmware infrastructure of its main component, a Field Programmable Gate Array FPGA provides the possibility to easy add user modules to the system.

Mainly conceived for the testing of particle physics detectors and their related electronics the MOSAIC is able to read data through high speed serial links (ten receivers on a single board at rate up to 6.6 Gbps) or using slower channels such as general purpose Low Voltage Differential Signals LVDS lines (up to 126 on a single board).

The board comes equipped with four programmable LEMO Input/Output complying with Nuclear Instrument Module standard and two FMC-LPC mezzanine slots in order to provide connectivity/housing to many various Device Under Test interfaces (DUT in the following) and in various test setup.

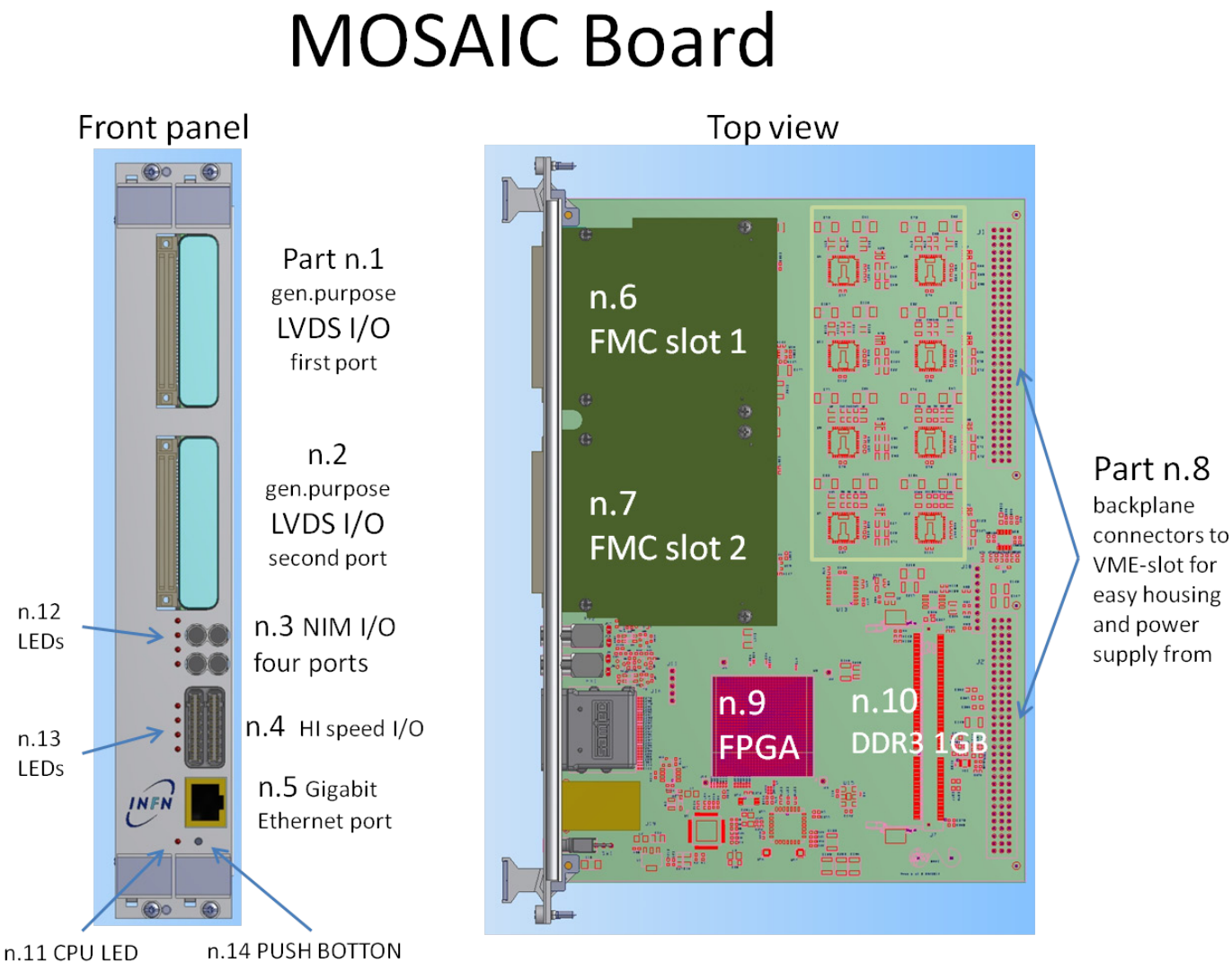

Figure 1: Main parts of MOSAIC board are enumerated in a mechanical drawing. LVDS I/O ports n.1 and 2; four NIM I/O n.3; HI Speed I/O ports n.4; Ethernet plug n.5; Two FMC-LPC mezzanine slots n.6 and 7; Field Programmable Gate Array FPGA n.9. 1GB-DDR3 n.10 and status LEDs n.11-13.

The system provide internally FPGA, a generator of data and a pulser to test connectivity respectively to external PC and DUT. More details are in following sections 2.5-23 and 2.6-25.

The board also host DDR3 1GB memory for the temporary store of data.

An Ethernet connection is provided for data transmission, control of operations and additional functions such as firmware upgrade, IP address configuration and board diagnostics at startup. The Ethernet interface is compliant with 803.x 10/100/1000 **only full duplex** more details in following section 1.1.5.

The FPGA architecture also provides an IP Bus transactor therefore IP Bus protocol is used for configuration and monitoring.

The same Field Programmable Gate Array houses an 8-bits microprocessor, supervision of transmission of data to an external PC and ancillary functions.

The board is a 2-unit wide 6U (6 x 12 inches) Versa Modular Eurocard VME standard for easy housing and power supply in a standard VME bus crate which would be the easiest way to supply power to the board, however this is not strictly necessary. Indeed the board only require 2 different supply namely at minus 12V for NIM I/O and +5V for the effective supply of the circuit (through several stabilized power supplies which you see located on the top right of the board). Moreover a power lines of +12V is fed to FMC slots, in case it could be required by added hardware.

The system is conceived as client/server interface for users. Software routines to access features of the system are under development.

No radiation hardness. The board does not include many of the provisions necessary when operating in irradiated area.

As an example of possible application, the proposed circuit will be used to perform the functionality test of the future Inner Tracker pixel barrel for the upgrade of the ALICE experiment.

Here following the description details about main parts as depicted in the mechanical drawing of the MOSAIC board Figure 1.

1.1.1 First I/O LVDS ports

About this first Input Output I/O Low Voltage Differential Signal LVDS ports we can highlight:

a. Both LVDS port are made by a Robinson Nugent P 50E-068-P1-SR1-TG multi-pin connector, whose pin-out is shown in the following Figure 2.

b. All LVDS signals are connected to FPGA pins (§bank number 13, 15 and 16) according to the classification proposed hereby which assign to every pin the suffix EXT followed by channel number, connector row and signal polarity.

c. In the released MOSAIC device all signals are compliant with LVDS.

1.1.2 Second I/O LVDS ports

Second I/O LVDS port, it apply same specifications as the previous first LVDS port.

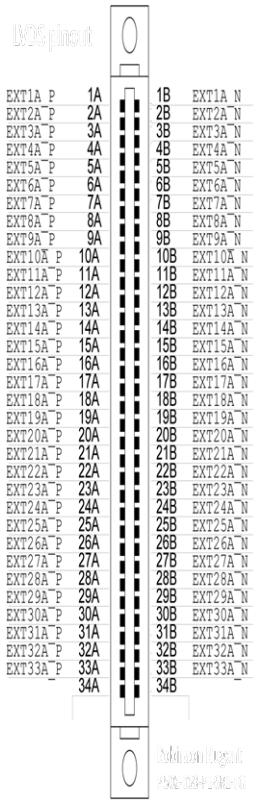

Figure 2: LVDS I/O port pinout. Robinson Nugent P 50E-068-P1-SR1-TG multipin connector.

1.1.3 LEMO ports

a. There are four LEMO connectors of four separate channels provided for boards cooperation, for instance with signal such as Common clock, Trigger, Busy.

b. The four channels are compliant with fast logic NIM standard also known as NIM logic.

c. With reference to the picture in *Figure 3*, each connector can be configured to work as input or output by soldering jumpers located at the back of and close by the same connectors namely J8, J9, J17 and J18 respectively for the NIM signals numbered from 1 to 4. Each jumper merely consist of three pads; a soldering contact of the upper pad (I) with the middle pad (II) configure the channel as output and vice versa a soldering contact between the middle and the lower pad (II in contact with III) set the related channel as input. It is worth to notice that the adjectives upper and lower are here used with respect to orientation of Figure 1.

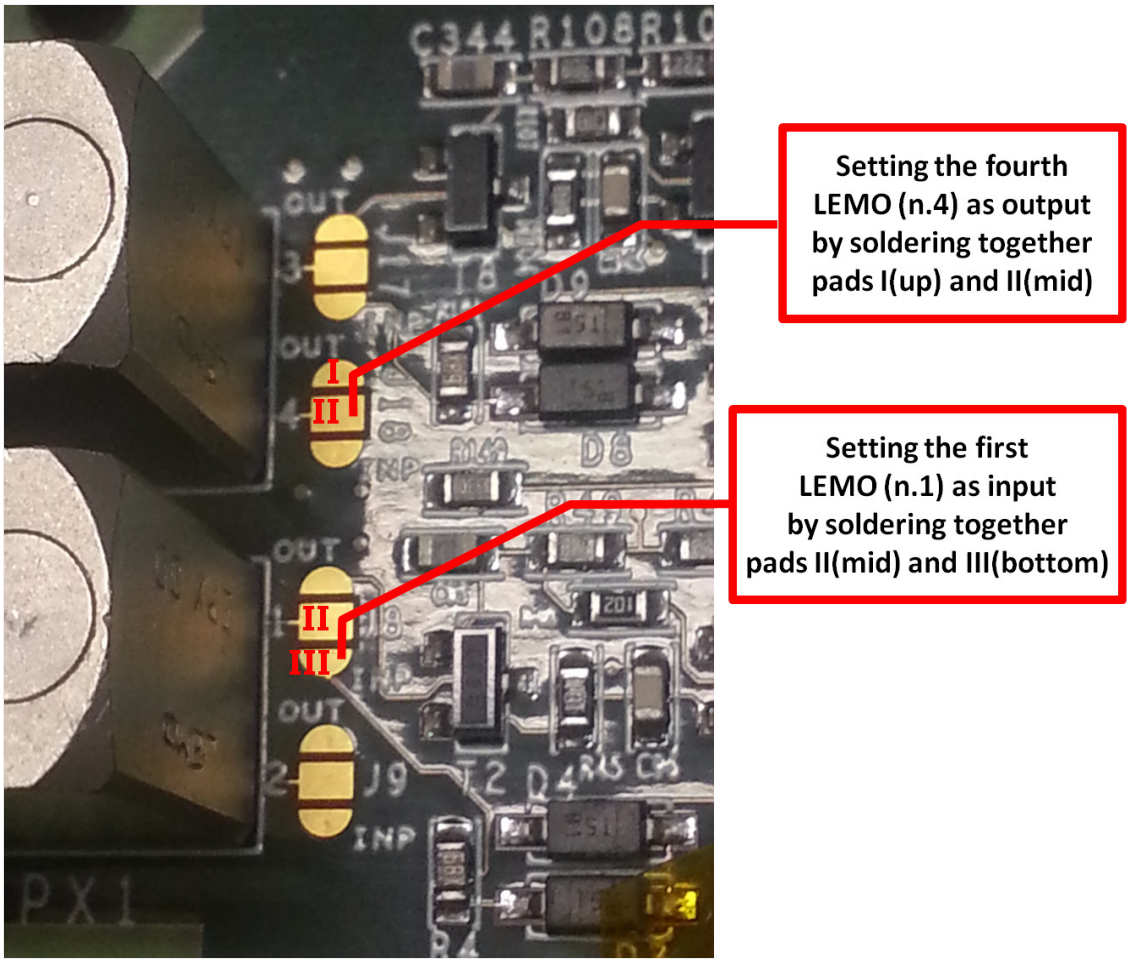

Figure 3: the soldering jumper pads for settings of LEMO ports as input or output.

1.1.4 High Speed Input/Output ports

a. On the front panel there is a connector devoted to receive high speed signals and more in general to be connected with high speed devices. These channels are connected with ten high speed serial links and other ports of FPGA, following the pin-out shown in Table 1.

b. Please NOTICE that only pins from first two rows (A and B) are connected resulting that the bottom plug is totally unconnected. Therefore lower plug cannot be used and it has to be left empty.

c. Moreover all pins named GND are directly connected to the same ground on the PCB.

d. The two channels named HSA and HSB at pins 31-35 of the row B are connected to M-LVDS drivers (multipoint low voltage differential signalling) compliant with standard TIA/EIA-899. As you expect for differential signals each channel take up two pins, denoted by (+) for positive and (-) for negative, beside ground pins.

e. There are last two differential channels for clock distribution to the DUT. Those are located at pins 31-35 of the row A and referred to as CLK40 in reference to an expected Front-End clock frequency of 40MHz.

f. In the matter of serial links velocity we can point out that respective clocks signal are supplied by a Phase Locked Loop which is programmed through the FPGA. Therefore serial link are versatile and can be easily adapted to work at different frequencies in various applications.

g. The port consists of a connecter HDI6-035-01-RA with cage HDC-035-01. According to the producer (Samtec) HDI6 denotes the 0,635 mm Eye Speed® HD connector at High Speed with High Density Receptacle, while 035 indicates the number of positions (per row).

The pinset is shown in the following Figure 4.

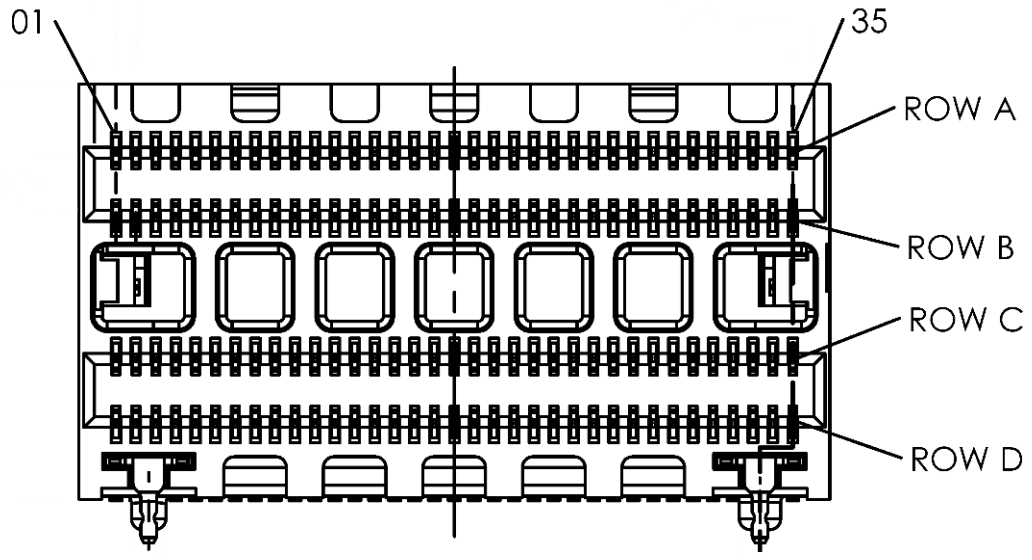

Figure 4: pinset of the High Speed connector located on the front panel (HDI6-035 Eye Speed® by Samtec). Please notice that lower plug cannot be used because C and D rows are unconnected.

HDI6 connector pin number	Signal	HDI6 connector pin number	Signal
A1	HSRX08N	B1	HSTX08N
A2	HSRX08P	B2	HSTX08P
A3	GND	B3	GND
A4	HSRX09N	B4	HSTX09N
A5	HSRX09P	B5	HSTX09P
A6	GND	B6	GND
A7	HSRX00N	B7	HSTX00N
A8	HSRX00P	B8	HSTX00P
A9	GND	B9	GND
A10	HSRX01N	B10	HSTX01N
A11	HSRX01P	B11	HSTX01P
A12	GND	B12	GND
A13	HSRX02N	B13	HSTX02N
A14	HSRX02P	B14	HSTX02P
A15	GND	B15	GND
A16	HSRX03N	B16	HSTX03N
A17	HSRX03P	B17	HSTX03P
A18	GND	B18	GND
A19	HSRX04N	B19	HSTX04N
A20	HSRX04P	B20	HSTX04P
A21	GND	B21	GND
A22	HSRX05N	B22	HSTX05N
A23	HSRX05P	B23	HSTX05P
A24	GND	B24	GND
A25	HSRX06N	B25	HSTX06N
A26	HSRX06P	B26	HSTX06P
A27	GND	B27	GND
A28	HSRX07N	B28	HSTX07N
A29	HSRX07P	B29	HSTX07P
A30	GND	B30	GND
A31	CLK40B_P	B31	HSB+ (MLVDS)
A32	CLK40B_N	B32	HSB- (MLVDS)
A33	GND	B33	GND
A34	CLK40A_P	B34	HSA+ (MLVDS)
A35	CLK40A_N	B35	HSA- (MLVDS)

Table 1: Pinout connection of HighSpeed connector (HDI6). Notice that GND pins are already connected through PCB. The two MLVDS signals to the DUT are referred as HSA and HSB.

1.1.5 Ethernet port

An Ethernet Gigabit connection with hardware control of IP fragmentation and TCP checksum provides communication with external PC (exploiting Direct Access Memory). The system sustains a transfer rate of about 120 MB/s. The interface implemented in MOSAIC is compliant with IEEE 802.3x 10/100/1000 flow control with pause packet, UDP/IP for control and monitoring and TCP/IP protocol for data transfer. The Ethernet connection has to be used **only in full duplex mode**. Indeed half duplex mode is not supported by MOSAIC board. A switch can be used to connect more boards if use together is required. Clearly, high transfer rate capability or full speed connection are suggested for best performance. For further detail about protocol please refer to RFC indicated in following paragraph Project References at page 1-13.

1.1.6 FMC first slot

Field Programmable Mezzanine Card (FMC) first slot.

Vita-57	Samtec	Molex
CC-LPC-10L	ASP-127796-01	45971-4307
MC-LPC-10L	ASP-127797-01	45970-4307
legenda		
CC	Socket of Carrier Side	
MC	Terminal of Module Side, Mezzanine Card (MC)	
LPC	Low Pin Count	
10	10 mm height	
L	with Lead (no lead-free)	

Table 2: Standard and vendor codes for FMC plugs and socket installed on the MOSAIC board.

About the two FMC slots (this and the next one) we can highlight that:

a. The connector used is an ASP-127796-01 by Samtec which is compliant with VITA 57 std.

b. The VITA 57 SEAM/SEAF Series system provides 160 I/Os (this apply to Low Pin Count configuration LPC in the following) in a selectively loaded 40 x 10 configuration, in 10mm stack heights. A recap and comparison of FMC relevant codes for MOSAIC board is in Table 2.

c. According to VITA 57.1 FMC LPC connection table the pin out is depicted in Table 3.
Please note that only four columns (denoted by letters C,D,G and H) are used whereas other pins are Not Connected and indicated as NC in Table 3; that is known as Low Pin Count.

d. Each FMC slot is connected also to one high speed transceiver among the other available inside FPGA (see DP0_C2M and M2C signals at pins 2,3, 6 and 7 of row C) and its related clock (GBTCLK0_M2C signals at pins 4 and 5 of row D).

1.1.7 FMC second slot

Second FMC slot applies specifications same as first FMC slot.

	K	J	H	G	F	E	D	C	B	A
1	NC	NC	VREF_A_M2C	GND	NC	NC	PG_C2M	GND	NC	NC
2	NC	NC	PRSNT_M2C_L	CLK1_M2C_P	NC	NC	GND	DP0_C2M_P	NC	NC
3	NC	NC	GND	CLK1_M2C_N	NC	NC	GND	DP0_C2M_N	NC	NC
4	NC	NC	CLK0_M2C_P	GND	NC	NC	GBTCLK0_M2C_P	GND	NC	NC
5	NC	NC	CLK0_M2C_N	GND	NC	NC	GBTCLK0_M2C_N	GND	NC	NC
6	NC	NC	GND	LA00_P_CC	NC	NC	GND	DP0_M2C_P	NC	NC
7	NC	NC	LA02_P	LA00_N_CC	NC	NC	GND	DP0_M2C_N	NC	NC
8	NC	NC	LA02_N	GND	NC	NC	LA01_P_CC	GND	NC	NC
9	NC	NC	GND	LA03_P	NC	NC	LA01_N_CC	GND	NC	NC
10	NC	NC	LA04_P	LA03_N	NC	NC	GND	LA06_P	NC	NC
11	NC	NC	LA04_N	GND	NC	NC	LA05_P	LA06_N	NC	NC
12	NC	NC	GND	LA08_P	NC	NC	LA05_N	GND	NC	NC
13	NC	NC	LA07_P	LA08_N	NC	NC	GND	GND	NC	NC
14	NC	NC	LA07_N	GND	NC	NC	LA09_P	LA10_P	NC	NC
15	NC	NC	GND	LA12_P	NC	NC	LA09_N	LA10_N	NC	NC
16	NC	NC	LA11_P	LA12_N	NC	NC	GND	GND	NC	NC
17	NC	NC	LA11_N	GND	NC	NC	LA13_P	GND	NC	NC
18	NC	NC	GND	LA16_P	NC	NC	LA13_N	LA14_P	NC	NC
19	NC	NC	LA15_P	LA16_N	NC	NC	GND	LA14_N	NC	NC
20	NC	NC	LA15_N	GND	NC	NC	LA17_P_CC	GND	NC	NC
21	NC	NC	GND	LA20_P	NC	NC	LA17_N_CC	GND	NC	NC
22	NC	NC	LA19_P	LA20_N	NC	NC	GND	LA18_P_CC	NC	NC
23	NC	NC	LA19_N	GND	NC	NC	LA23_P	LA18_N_CC	NC	NC
24	NC	NC	GND	LA22_P	NC	NC	LA23_N	GND	NC	NC
25	NC	NC	LA21_P	LA22_N	NC	NC	GND	GND	NC	NC
26	NC	NC	LA21_N	GND	NC	NC	LA26_P	LA27_P	NC	NC
27	NC	NC	GND	LA25_P	NC	NC	LA26_N	LA27_N	NC	NC
28	NC	NC	LA24_P	LA25_N	NC	NC	GND	GND	NC	NC
29	NC	NC	LA24_N	GND	NC	NC	TCK	GND	NC	NC
30	NC	NC	GND	LA29_P	NC	NC	TDI	SCL	NC	NC
31	NC	NC	LA28_P	LA29_N	NC	NC	TDO	SDA	NC	NC
32	NC	NC	LA28_N	GND	NC	NC	3P3VAUX	GND	NC	NC
33	NC	NC	GND	LA31_P	NC	NC	TMS	GND	NC	NC
34	NC	NC	LA30_P	LA31_N	NC	NC	TRST_L	GA0	NC	NC
35	NC	NC	LA30_N	GND	NC	NC	GA1	12P0V	NC	NC
36	NC	NC	GND	LA33_P	NC	NC	3P3V	GND	NC	NC
37	NC	NC	LA32_P	LA33_N	NC	NC	GND	12P0V	NC	NC
38	NC	NC	LA32_N	GND	NC	NC	3P3V	GND	NC	NC
39	NC	NC	GND	VADJ	NC	NC	GND	3P3V	NC	NC
40	NC	NC	VADJ	GND	NC	NC	3P3V	GND	NC	NC

Table 3: FMC LPC connector pinset (NC = Not Connected).

1.1.8 Plugs to Versa Modular Eurocard 6U compliant chassis.

Aside providing easy housing one can notice that only few pins are used, therefore connected, among those available in VME bus as shown in [Table 4]. Specifically the MOSAIC board is provided to be connected to VME backplane merely in order to supply power at +5 Volts to the board circuits (and its main component the FPGA chip). Besides, other two power lines from VME back planes are connected, those are the pin 31 A and C respectively to the NIM (converter) circuitry and to the FMC plugs.

Pin No.	J1/P1 (top)			J2/P2 (bottom)		
	Row A	Row B	Row C	Row A	Row B	Row C
01	D00	BBSY*	D08	User defined	+5 V	User defined
02	D01	BCLR	D09	User defined	GND	User defined
03	D02	ACFAIL*	D10	User defined	Reserved	User defined
04	D03	BG0IN*	D11	User defined	A24	User defined
05	D04	BG0OUT*	D12	User defined	A25	User defined
06	D05	BG1IN*	D13	User defined	A26	User defined
07	D06	BG1OUT*	D14	User defined	A27	User defined
08	D07	BG2IN*	D15	User defined	A28	User defined
09	GND	BG2OUT*	GND	User defined	A29	User defined
10	SYSCLK	BG1IN*	SYSFAIL*	User defined	A30	User defined
11	GND	BG3OUT*	BERR*	User defined	A31	User defined
12	DS1*	BR0*	SYSRESET*	User defined	GND	User defined
13	DS0*	BR1*	LWORD*	User defined	+5 V	User defined
14	WRITE*	BR2*	AM5	User defined	D16	User defined
15	GND	BR3*	A23	User defined	D17	User defined
16	DTACK*	AM0	A22	User defined	D18	User defined
17	GND	AM1	A21	User defined	D19	User defined
18	AS*	AM2	A20	User defined	D20	User defined
19	GND	AM3	A19	User defined	D21	User defined
20	IACK*	GND	A18	User defined	D22	User defined
21	IACKIN*	SERCLK	A17	User defined	D23	User defined
22	IAOUT*	SERDAT	A16	User defined	GND	User defined
23	AM4	GND	A15	User defined	D24	User defined
24	A07	IRQ7*	A14	User defined	D25	User defined
25	A06	IRQ6*	A13	User defined	D26	User defined
26	A05	IRQ5*	A12	User defined	D27	User defined
27	A04	IRQ4*	A11	User defined	D28	User defined
28	A03	IRQ3*	A10	User defined	D29	User defined
29	A02	IRQ2*	A09	User defined	D30	User defined
30	A01	IRQ1*	A08	User defined	D31	User defined
31	-12 V	+5V STDBY	+ 12 V	User defined	GND	User defined
32	+5 V	+ 5 V	+ 5 V	User defined	+ 5 V	User defined

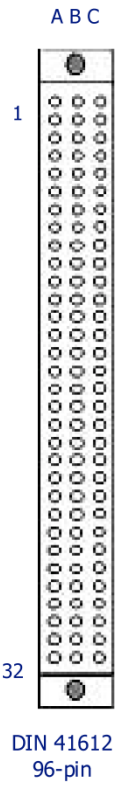

Table 4: VME Backplane connectors and pin layout

1.1.9 FPGA

The main component of MOSAIC board is a Field Programmable Gate Array FPGA

a. Artix-7 FPGA XC7A200T Package FFG1156

b. More details about firmware architecture will be discussed in further section .

1.1.10 DDR3 memory

Double Data Rate Random Access Memory DDR3L SDRAM SODIMM

a. The MOSAIC board comes furnished with a 1.35V RAM module which is used as cache for the temporary store of data before to be transferred to an external PC.

b. The part number of the module is MT4KTF12864HZ-1G6K1 by Micron.

c. The module is 204-pin, small-outline dual in-line memory module (SODIMM).

d. Module have memory storage of 1GB (128 Meg x 64), supply voltage VDD = 1.35V (1.283–1.45V) bandwidth of 12.8 GB/s Memory Clock 1.25ns and data rate up to 1600 MT/s.

e. At the moment the only RAM module supported by firmware is the above mentioned MT4KTF12864HZ-1G6K1 by Micron.

The front panel depicted on the left of Figure 1, hosts also the LEDs and a Push Button described in the following.

1.1.11 CPU LED

Function: this is a single led controlled by the CPU (internal of FPGA) which allows users to know the board state by its different blinking modes as in Table 5.

Located: front panel, down to the left.

Blinking timing (s)		Meaning mode
ON	OFF	
1.0	1.0	Normal operation mode
0.3	0.1	Golden image loaded (please refer to Section 3.2)
0.1	0.1	IP configuration request
0.1	1.0	Error detected during startup

Table 5: CPU led blinking modes and meanings.

1.1.12 LEDs first block

Those four Leds together with the four of the following second block are sequentially numbered from zero to seven starting **from the bottom toward the top** as in Figure 5.

The eight Leds have different utilization during normal operations and during startup.

In the latter case leds are used to inform about errors detected during startup, displaying a binary code of the error according to the numeric code summarized in Table 5. E.g. In case of SODIMM initialization timeout the second and fourth led (also denoted as Led[1] and Led[3]) will light up to indicate the error code number ten, namely 1010 in binary which is 0x0A in hexadecimal.

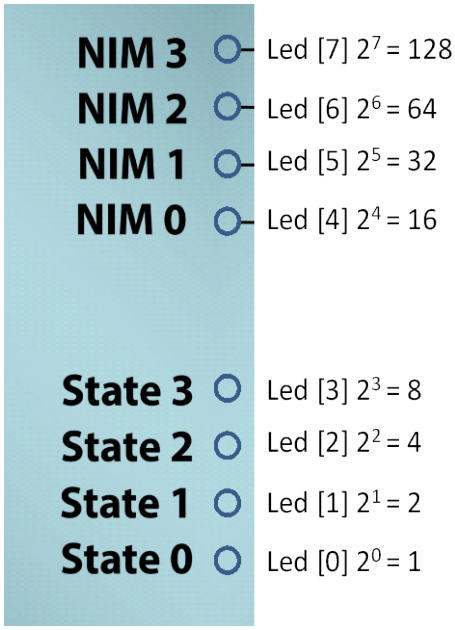

Figure 5: layout of leds on front panel and their binary assignment.

CODE [hex]	Error description
01	Ethernet PHY initialization timeout
02	One wire bus initialization error
03	No IP address stored in EEPROM
04	EEPROM write error
08	SODIMM module not detected
09	Wrong SODIMM module. Memory model is not valid for the firmware
0A	SODIMM initialization timeout

Table 6: numeric code for errors at startup as displayed by LED blocks part n.11 and n.12.

All errors but the "No IP address stored in EEPROM" are blocking, namely the board hangs flashing the led and does not responds to Ethernet requests. In case of "No IP address stored in EEPROM" it will perform only the IP address configuration as described in the relate section.

What following, it refers to the specific firmware implemented for the test of pAlpide: during normal operations every led has its specific meaning as summarized in following Table 7.

Tag	Led/bit Num.	meaning
NIM 3	7	Unused
NIM 2	6	Front End Clock
NIM 1	5	Front End Pulse
NIM 0	4	Front End Trigger
State 3	3	Unused
State 2	2	BUSY It also lights if RUN is off
State 1	1	Running
State 0	0	Front End PLL status = LOCKED

Table 7: function of front panel leds during working operation.

Lastly, please note that the lowest Led (Led or bit [0]) indicates that the Phase Locked Loop, which generates the different read out clock, is locked. Since it is possible also to select an external clock it means that everything is working correctly.

1.1.13 LEDs second block

See above description of part.11.

1.1.14 PUSH BUTTON

The push button is used during IP address request procedure, details in the following Section 3.1.

Located: front panel, down to the right.

1.2 Project References

List of the references that were used in preparation of this document.

1. For more specifications concerning IPbus packet and transaction header, and the format of possible IPbus request/response please refer to "*The IPbus Protocol An IP-based control protocol for ATCA/μ TCA*" **Version 1.4 - draft 1** 18th Oct, 2011 available at link

   https://svnweb.cern.ch/trac/cactus/export/156/trunk/doc/ipbus_protocol.pdf

2. MOSAIC system utilizes communication protocol complying with Ethernet 802.3x pause packet only full duplex (**no half duplex**) 10/100/1000.

3. For further reference please refer to RFC http://www.rfc-editor.org/rfc.html

4. RFC 768 User Datagram Protocol August 28, 1980 UDP

5. RFC 791 Internet Protocol September 1981 IPv4

6. RFC 792 INTERNET control message Protocol September 1981 ICMP

7. RFC 793 TRANSMISSION CONTROL PROTOCOL September 1981 TCP

8. RFC 826 An Ethernet Address Resolution Protocol November 1982 ARP

9. Wishbone Bus Specification rev.B4 System-on-Chip Interconnection Architecture for Portable IP Cores by OpenCores Organization (2010) Richard Herveille, rherveille@opencores.org www.opencores.org

10. UM10204 I2C-bus specification and user manual Rev. 6 — 4 April 2014 by NXP

11. Doygen version 1.8.9.1 doxygen reference manual available at http://www.stack.nl/~dimitri/doxygen/ Copyright © 1997-2015 by Dimitri van Heesch free software under GNU General Public License

12. pALPIDEfs datasheet ver.1.0b by ALICE ITS ALPIDE development team Last ed. 19/03/2013

13. TDR Alice IT Barrel

1.3 Authorized Use Permission

Usage of the hardware and software described in the present documentation is limited to its owner via the terms of its development. MOSAIC test system is wholly owned by INFN, and may not be used or referenced without their express consent.

In no event shall INFN or any of the contributors be liable for any direct, indirect, incidental, consequential, exemplary, or special damages (including, but not limited to procurement of substitute goods or services; loss of use, data, or profits; or business interruption) resulting in any way from the use of this specification. By adopting this specification, the user assumes all responsibility for its use.

This is a preliminary document, and is subject to change without notice.

Verilog® is a registered trademark of Cadence Design Systems, Inc.

1.4 Information

For additional information, CAD Service Team can be contacted through Team Leader/ INFN sez Bari Giuseppe De Robertis (Giuseppe.DeRobertis@ba.infn.it).

2.0 SYSTEM SUMMARY

2 SYSTEM SUMMARY

This section provides a general overview of the system and outlines its operation modes.

2.1 System Architecture: Firmware

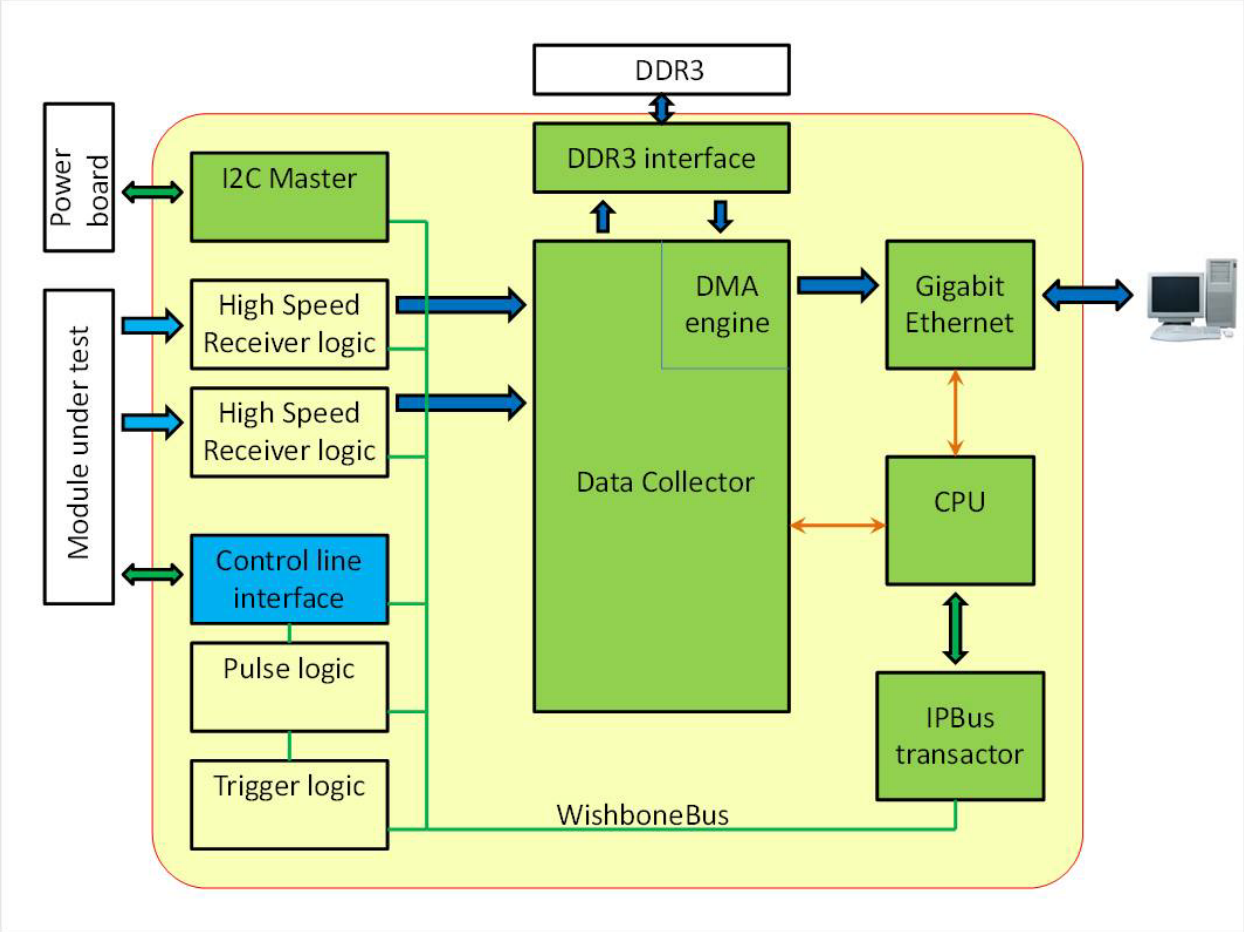

Figure 6: Firmware architecture of MOSAIC system.

The Figure 6 above, depict the firmware architecture of MOSAIC system. Starting from the top right we have an Ethernet interface (compliant with Ethernet 802.3x 10/100/1000 full duplex only) for the communication to an external computer. The latter can be a PC or a mainframe in charge for the control of operations, collection and further elaboration of data. The Ethernet interface can access the RAM memory contents thanks by direct access memory engine (DMA) and communicate with an internal CPU. The CPU, implemented in hardware internally to the FPGA, is mainly an 8-bit microcontroller with many functions such as CPU, RAM, ROM, I/O, interrupt logic, timer, etc..

Moreover we have a data collector unit and an IP-Bus transactor those two in connection with the above mentioned CPU. Thanks to the IP Bus transactor unit, the configuration and monitoring of operations can be done externally using the IP Bus protocol, for more details please refer to [1] of Project Ref. page 1-13.

It is also provided an I2C interface for the connection with an expected power section of DUT. Besides we have the high speed receivers in charge of acquiring data from the DUT and the logic block of control signals including a pulse generator (pulser) and a logic block to handle the trigger.

As portrayed in Figure 6 communications between those modules and IPBus transactor are conducted through a Wishbone Bus ®OpenCores which specifications can be found at item [9] of list on page 1-13. In the following the addressing through Wishbone bus will be referred to by symbolic reference.

About high speed receiver it is worth to notice that rear to the dedicated hardware there is logic blocks to handle the coming data flux to the data collector and then to the storage in DDR3 memory. During the design, this part, in charge of overseeing the data-flux to the collector, was referred to as receiver-buffer, that is how it will be indicated also in the following.

2.2 Receiver Buffer

Since an high speed receiver get data this module (one for each high speed receiver) packs data in fixed length records of 512 bits, that is to say 64 Bytes. The records are written in a cache, that is a FIFO memory of 64KB (organized in 1024x64B namely with 1024 locations of 64B width). At the same time the receiver buffer counts how many bytes of received data it has written in the cache. Moreover, agreed that data comes from events, there will be a trailer marker to spotlight the conclusion of each single event (understandably called End of Event). This buffer also registers the number of received End of Event.

Communication between the Receiver Buffer and the DUT takes place by four signals, a typical waveform is depicted in Figure 7:

- Front End Clock, denoted by CLKWR (elsewhere by FE_CLK);
- Input Data Bus, denoted by DIN;
- Write Enable of Input Data, denoted by WR;
- End Of Event input denoted by EOE.

In Figure 7, a snapshot of waveforms for the above mentioned signals shows a bunch of eight data, considered in three different cases indicated by the letters A, B and C respectively:

A. In this first case, the Receiver Buffer realizes that two events are contiguous by the halfway End Of Event signal and it interprets the coincident data (the fourth, simultaneous with EOE) as the last data in the first event. Therefore both events contain four data each.

B. In this second case, both EOE and the WR signal are de-asserted at the same time. The Receiver Buffer store the last data in coincidence with the EOE and it correctly stops to acquire other data, whether or not signals on DIN bus changes. Indeed Write Enable (WR) is de-asserted.
The single event contains eight data.

C. The last case does not really differs from previous case in behaviour of the Receiver Buffer, again eight data are interpreted as a single event and treated accordingly.

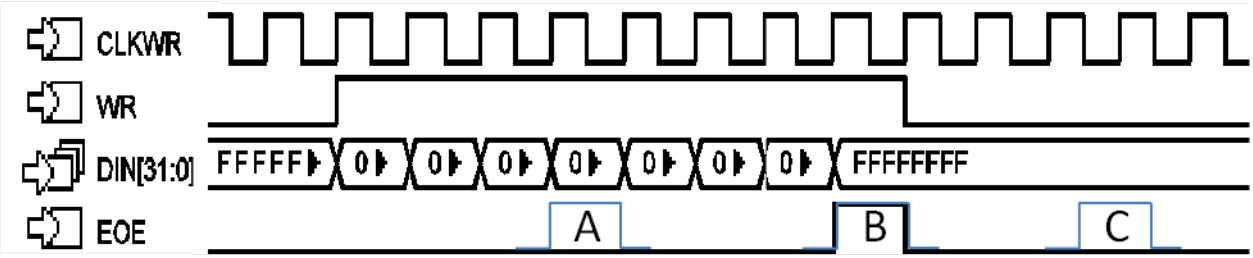

Figure 7: waveforms signals between receiver buffer and DUT

Under certain conditions the receiver asserts a request to be read by the Data Collector (reading request). For the sake of clarity, Data Collector is asked to read data and it will perform the reading as soon as it can, which is to say, instantly whether Data Collector is not still busy with a previous reading or afterward it becomes not busy. In any case, the receiver buffer tries caching data in the FIFO as it gets data, except that overflow[1] occur.

[1] Which is to say an overrunning of memory's boundary with overwriting of adjacent memory locations.

2.2.1 Reading request and latency of packets in receiver buffer

First and foremost, conditions under which the receiver buffer asserted reading request, depends on user settings of latency. Basically a reading request is asserted on four conditions:

- if half of the memory storage of FIFO is filled up;
- if, despite a previous request at half memory full, an overflow occurs;
- if the Run is closed;
- if timeout is elapsed, under latency settings of a fixed value for timeout;
- At every End of Event, if this is possible and under "zero latency" settings.

As presumable, the first three cases are independent of latency mode. In fact under these circumstances a reading request is always asserted, disregarding current latency mode. Instead, about the two last conditions, please notice that the receiver buffer works differently according with data packet latency settings. Leaving aside for the moment how to set latency mode (Wishbone bus, IP bus and dedicated software routines are described in following Sections), users can choose among the following three:

- zero latency mode;
- timeout mode;
- infinite latency mode.

As first possibility, latency mode equal zero means that the receiver buffer sends a reading request to the data collector as soon as it receives the trailing signal End of Event. Please note: this does not entail that, in the buffer to be read, is contained neither an entire event nor an event alone.

In second instance, a timeout value can be set by the user, so as to, the currently stored data should be sent to the data collector at the moment of timeout elapsing.

In the last mode (infinite latency), receiver buffer will ask to read data content, principally when half of its storage memory capability is filled up.

It is worth to notice that, in any case and regardless the latency mode setting, a request for reading is sent afterward the storage memory become half full, without concern of UN-completed events. This has been provided as protection against dangerous overflowing of receiver buffer and consequent loss of data.

Clearly, the last two latency modes can be employed by the user in order to pack together several events, taking into account that the receiver buffer disregards the fact that the packet would contain also portions of events. Actually we can always expect that there will be events not completed, in the first part as well as in the tail of the data packet.

As an effect of reading request the Data Collector will load the content of FIFO cache memory, organize and store the packet in the DDR3 waiting for the transmission. Anyway the Data Collector cannot generally perform the loading as soon as the receiver ask for that, for instance because it is already busy reading from another receiver. And at the same time the receiver buffer keeps to cache data in the FIFO whether or not its reading request is being satisfied.

Clearly an overflow occurs if there is a record to be written in FIFO while this is full. That can happens under conditions of high data rate (with respect to output throughput) and it will cause a loss of data and a corrupted packaging. Indeed that circumstance cause the receiver to stop collecting data and to generate a request to the Data Collector for being read as soon as it can (namely as soon as the latter is not busy with previous reading). This and other occurrences have to be indicated by flags. Moreover the packaging has to report the total number of data and the number of completed events in its content.

2.2.2 The header of packets from receiver buffer

Therefore at the beginning of the packet there is an header of 64B. The header is structured as follows: the first (most significant) 13 words to indicate channel, namely which receiver data come from, and other three words (3x32 = 96 bits) where to store information pertinent to that specific channel.
Such header arrangement is depicted in [Table 8]; here following some explanation about its meaning.

As we said the first field report which channel, namely which receiver, sent the data stored in the packet. Channel info employs 13 words equal to 60 Bytes equal to the 416 most significant bits of the header.

Beside there is another field of three words which is denoted BLOCK INFO. As one can presume, there are stored, packet concerning main information, that is: how many data and event tail the packet contains and the reason why reading request was asserted.

In detail within BLOCK INFO, the first word (4 bytes) contains the number only of event tail, contained in the same packet. The first field reports the amount of End of Event occurred during the collection of the packet, which can differ from the number of events. Indeed, admitted value can be zero in the meaning that the current packet contains only a fraction of one entire event. Moreover up to two fragments can be within the packet wrapped (leading and trailing) complete events.

The second word is the Flags field as depicted in Table 9. Indeed it mainly consists of four bit registers which are high in case the transmitted packet has been subjected to one of four conditions:

1. the LSB, namely bit[0], is high to indicate that the packet sent finished with an End Of Event;
2. the next more significant bit[1] indicates that an overflow occurs upon FIFO cache memory writing, as above explained;
3. the third register, if high, means that current packet has been sent afterwards an elapsed timeout;
4. last fourth bit is set high when the same referring packet is sent in concurrency with a stop of the running, namely cause running signal has been de-asserted.

2.2.3 Size of data packet for transmission: the role of Data Collector

When the packet is stored in FIFO, Receiver Buffer assert a reading request and the Data Collector comes in charge of loading the data from FIFO. As we noticed the width of FIFO is fixed (64B), disregarding the amount of bytes contained in a single packet and stated in the third field of BLOCK INFO. That can imply that Data Collector will fill the tail of the packets with sequence of zero, up to have a packet structured in multiple of 64B and long enough to be sent complying with protocol requirement.

2.2.4 Remarks about Flags

Concerning flags of BLOCK INFO it is worth to notice that in some case the packet will be sent with any flag high, namely in the case of a reading request due to half memory filled up. Under second condition, in case of Overflow, the appropriate flag is set high (@bit 33). At last in a third case, as consequence of closing a run, the End of Run flag is set high (@bit 35). As example, considering a packet with any flag high (namely all BLOCK INFO flags field set to zero), user can realize that reading request has been asserted as consequence of FIFO half full and the tail of the packet contain an incomplete event.

Finally please observe that above described circumstances, as reported by flags, can be concurrent hence more than one flag can be high in the same packet meaning that all reported circumstances had occurred. For instance we can have a packet ending with a complete event (first flag @bit 32) which has been sent cause of a timeout and/or an overflow both occurred during package of data, therefore we will have also pertinent flags set high (third flag @bit 34 and/or second flag @bit 33). The Flag content will be 0111 in that first case of packet finishing with complete event and sent cause of timeout and overflow or, as another instance, flags would be 0101 in case again of a packet again finishing with complete event but sent cause timeout and not overflow.

Address in Byte				
63 16	15 12	11 8	7 4	3 0
RESERVED 12 words = 48 Bytes	**CHANNEL INFO** (1 word = 4 Bytes)	**BLOCK INFO** (3 words = 12 Bytes)		
		End of Event counter (4 Bytes)	**Flags** (4 Bytes)	**Size in Byte** (4 Bytes)

Table 8: Header protocol for packaged data from Receiver Buffer.

Address in bit (within the Flags field)				
31 4	3	2	1	0
FLAGS (4 Bytes = 32 Bits)				
Reserved - zero filled (28 bits)	**End of Run** (1 bit)	**Timeout** (1 bit)	**Overflow** (1 bit)	**Closed event** (1 bit)

Table 9: format of FLAGS field in the header.

2.3 Addressing Wishbone Bus

WISHBONE is a flexible System-on-Chip (SoC) design methodology. WISHBONE establishes common interface standards for data exchange between modules within an integrated circuit chip. Its purpose is to foster design reuse, portability and reliability of SoC designs. WISHBONE is a public domain standard.

Please notice that in MOSAIC the standard is restricted at fixed 32-bit data port with BIT granularity (width of 32 bits with NOT-allowed partial writing or reading). In other words MOSAIC uses a 32 bit specification of WBB bus, namely with 32 bit width for both data and addressing. Also due to this specification, some of the signals in the WISHBONE standard are optional in the standard and here not necessary therefore not present on this specific interface.

Beside what concerning Pulser, Data Generator and Trigger Control described in Sections 2.5, 2.6 and 2.7, the present section report the offset addresses of other useful blocks interconnected within MOSAIC by Wishbone bus, in detail the Controller of I2C, the Run Controller and the pAlpide Control Interface.

The address space for the Wishbone bus is structured in two fields in such a way that address is done by the sum of Base Address to address the specific block and Relative Address (offset) to address single registers within that selected sub-system. A list of Base addresses is reported in the following Table 20, while for every module is provided a dedicated table with offset reference.

Please note that numeric values for addressing can differ in final release (not the symbolic reference to) as a matter of fact the running arrangement is established in the package file **`mapping.pkg`** for the base addresses and in dedicated package files for the relative part (offset). Therefore the absolute addresses (which is to say the numeric values) have to be modified from there.

Finally, for specification and more details about writing reading and addressing Wishbone Bus please refer to the Wishbone Bus protocol documentation at reference item [9] at page 1-13.

I^2C Controller: registers mapping		
Address name	**register function**	**Note**
ADD_SHIFT_OUT	A 32-bit R&W register for data output and to control operations	see fields
Fields within SHIFT_OUT		
STOP	*1-bit to command a stop to I2C slave*	*bit[31]*
START	*1-bit to command a start to I2C slave*	*bit[30]*
MASTER_ACK	*1-bit for the acknowledge from Master*	*bit[29]*
IGNORE_ACK	*1-bit to ignore Master acknowledge*	*bit[28]*
dummy	*20 bits to fit SHIFT_OUT to bus width*	*to fit 32 bits*
DATA_OUT	*A 8 bit R&W register for data output*	*bit[7:0]*
ADD_SHIFT_IN	An 8-bit read only register for reading data back	
ADD_PAR_IN	An 32-bit read only register for reading parallel inputs	
N.B.: absolute addressing can differ in final release. The current arrangement is stated by the proper package file (i2c_master.pkg) and it has to be be modified from there.		

Table 10: mapping of I2C controller

Run Controller registers mapping		
Address name	**register function**	**Note**
ADD_RUN_CTR	A 2-bit R&W register to control starting of Run	two fields single bit
Single bit fields within Run Control register, namely RUN_CTR bits		
RUN	*1-bit register to start running*	*bit[0] zero by default*
PAUSE	*1-bit register to pause running*	*bit[1] zero by default*
ADD_ERROR_STATE	A 32-bit R&W register for storing the error state	
ADD_ALMOST_FULL_THRESHOLD	A 32-bit R&W register for current value of Threshold of almost full flag for the ddr3 memory buffer	
ADD_LATENCY	A 32 bit R&W register for control Data Latency	bits UNUSED see fields
fields within Latency register		
MODE	*2-bit field to latency mode setting*	*bit[31:30] zero by default*
TIMEOUT	*24-bit register for the TIMEOUT value*	*ONLY bit[23:0] UNUSED [29:24]*
enumerator for setting Latency Mode in previous register, namely in MODE reg.		
latModeEOE	*Send reading request on End Of Event; namely zero latency mode*	*default mode*
latModeTimeout	*Send reading request on timeout elapsing; namely timeout mode*	
latModeMemory	*Send reading request on half memory filled; namely infinite latency mode*	
ADD_TEMPERATURE	NOT IMPLEMENTED; future development FGPA internal temperature	DO NOT USE not working
ADD_RESERVED_0	NOT IMPLEMENTED reserved 0 (first)	DO NOT USE not working
ADD_RESERVED_1	NOT IMPLEMENTED reserved 1 (second)	DO NOT USE not working
ADD_RESERVED_2	NOT IMPLEMENTED reserved 2 (third)	DO NOT USE not working
ADD_CFG	A 32 bit R&W register for the configuration setting	
fields within Configuration register, namely in CFG reg, for clock setting		
CFG_EXTCLOCK_SEL_BIT	reg. to Enable external clock	bit[0]
CFG_CLOCK_20MHZ_BIT	to switch clock from 40 to 20 MHz	bit[1]
N.B.: absolute addressing can differ in final release. The current arrangement is stated by the proper package file (run_control.pkg) and can be modified from there.		

Table 11: mapping of Run Controller

pAlpide Control Interface: registers mapping		
Address name	**register function**	**Note**
ADD_WRITE_CTRL	A 32-bit R&W register to control writing operations	see fields below
Fields within WRITE_CTRL		
opcode	*8-bit reg for the current value of Operation Code*	*bit[31:24] see list below*
chipID	*8-bit reg for the current value of Chip ID*	*bit[23:16]*
regAddr	*16-bit reg for the current value of address where to write*	*bit[15:0]*
List of Operation Codes, enumerator for opcode field of Write Control Register above		
OPCODE_STROBE_2	*Trigger*	
OPCODE_GRST	*command a Chip Global Reset*	
OPCODE_RORST	*command a Readout Reset*	
OPCODE_PRST	*command a Pixel matrix reset*	
OPCODE_STROBE_6	*Trigger*	
OPCODE_BCRST	*command Bunch Counter Reset*	
OPCODE_ECRST	*command Event Counter Reset*	
OPCODE_PULSE	*Calibration Pulse trigger*	
OPCODE_STROBE_10	*Trigger*	
OPCODE_RSVD1	*Reserved for future development*	
OPCODE_RSVD2	*Reserved for future development*	
OPCODE_WROP	*command a Start Write Unicast or Multicast Operation*	
OPCODE_STROBE_14	*Trigger*	
OPCODE_RDOP	*command a Start Unicast Read*	
ADD_WRITE_DATA	A 16-bit reg for data to write	
ADD_READ_DATA	A 16-bit register for read data	
Fields within READ_DATA		
syncOK	*1-bit to acknowledge that input data are correctly synchronized with Front End clock*	*bit[27]*
ChipID_OK	*1-bit to acknowledge that Chip ID is received in good order*	*bit[26]*
DataL_OK	*1-bit to acknowledge that Read Data Low Byte is received in good order*	*bit[25]*
DataH_OK	*1-bit to acknowledge that Read Data High Byte is received in good order*	*bit[24]*
rdChipId	*An 8 bit reg for read value of chip ID*	*bit[23:16]*
rdData	*A 16 bit reg for read value of Data*	*bit[15:0]*
ADD_DATA_PHASE	A 3-bit read only register for the input data phase respect to FE_CLK	
N.B.: absolute addressing and enumerated values can differ in various releases. The running arrangement must be modified always by editing the proper package file, in this case control interface.pkg		

Table 12: mapping of pAlpide Control Interface

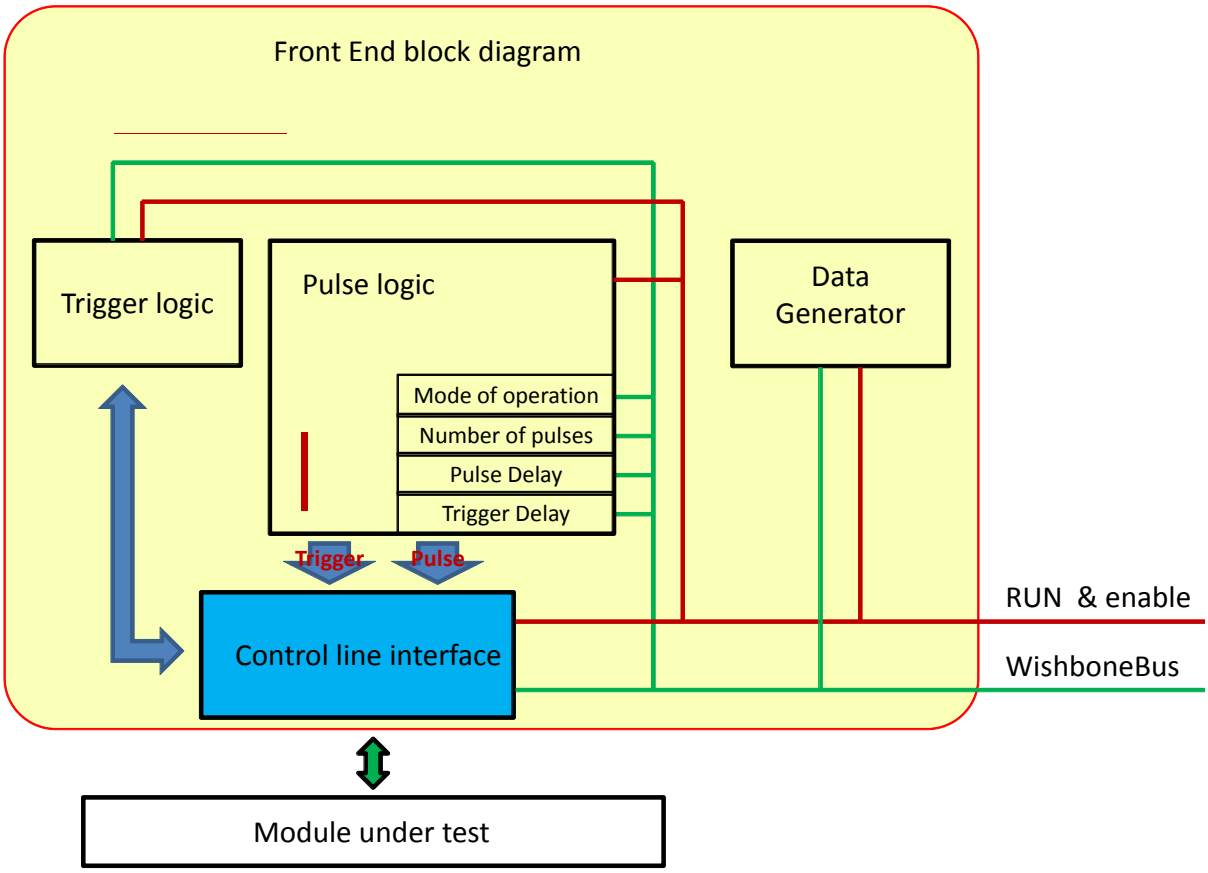

Figure 8: block diagram of Front-End.

The diagram in Figure 8 above, illustrates the organization of the front end logic. Among other blocks, it is worth to notice that the system also implements (inside the FPGA) a generator of data as sequential numbers and a pulser, provided for testing connectivity respectively to external PC and DUT. Previously we gave some details about how to address single blocks in MOSAIC, through Wishbone bus. The performances relating to those data generator and pulser are subject of the following sections.

2.5 Pulser

As said before, the MOSAIC system provide generation of pulses and trigger. Three (3) different mode of operation are provided:

- only Pulse;
- only Trigger;
- Pulse and Trigger (default mode).

The desired values for number of pulses, trigger delay and pulse delay are written through Wishbone bus at different addresses enumerated by symbolic names in Table 13. Please note that absolute addressing can differ in final release, as a matter of fact the running arrangement is established in the proper package file named **`pulser.pkg`** and can be modified from there.

Every subsequent overwriting/update of those values stops the execution of that batch of N pulse and starts another one from the beginning, with current (which is to say at that moment) values of delays and number of pulses. This restart occurs at the timing set for trigger generation, namely after internal trigger is generated (if enabled) or at the moment when it should be (if not enabled).

2.5.1 How to set different operation modes for pulsing

There are different modes of operations controlled by enable registers. Those are single bit fields within the register for operation mode. For the sake of completeness we report here that this register is named `opMode_wbb`, but this is redundant due to the fact that the same register can be addressed, through Wishbone bus, by the symbolic reference `ADD_MODE`. Moreover, as we will see later in the following Section 4.1, software routines are provided in order to spare users from addressing Wishbone bus.

In detail enabling registers are:

- `enable_pls` for enabling pulse, which is to say generate a pulse at due timing;
- `enable_trg` for enabling trigger, namely generate internally a trigger at due timing.

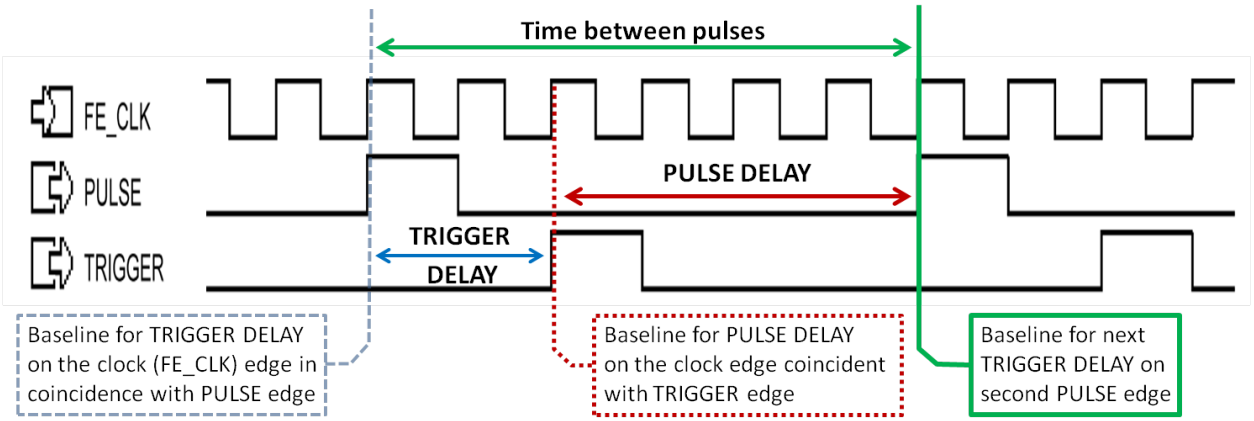

Figure 9: a snapshot of waveform illustrates timing of pulses and delay in the pulser.

2.5.2 Trigger and Pulse Delay

The following Figure 9 shows the timing of both internally generated trigger at set timing after pulse and pulse at set timing after generated trigger.

Value for the trigger delay has to be written in unit of Front-End clock period (FE_CLK in the following), addressing Wishbone bus at `ADD_TRIGGER_DELAY`. Default value is zero.

Value for the pulse delay has to be written in unit of Front-End clock period (FE_CLK in the following), addressing Wishbone bus at `ADD_PULSE_DELAY`. Default value is zero.

In conclusion the **time interval between subsequent pulses is the sum of** the two set delay values, namely **pulse delay plus trigger delay**.

2.5.3 Number of pulses and general cautions

The desired value for the number of pulses in a bunch has to be written addressing Wishbone bus at `ADD_PROG_NUMPULSES`. Since default value is zero, the generation of pulse does not start at all, as well as in the case that set value is zero.

On the contrary, the combination of RUN signal and ON flag asserted, commands to sample the number of pulses and to store the current value in a register clocked at (synchronized with) FE_CLK.

At last there is a 32 bit read only register for the current number of pulses sent to the DUT, therefore that amount can by monitored addressing Wishbone bus at `ADD_STATUS_BAR` reference.

Please notice that it is mandatory to supply the set values written in the input registers before the strobe signal on the Wishbone bus. At last, users can overlook this caution using the IP bus transactor or the software routines we will described later, in the Section 4.1.

Pulser registers mapping		
Address name	**register function**	**Note**
ADD_MODE	A 2-bit R&W reg. for setting of operation mode (opMode_wbb)	Pulse and Trigger enabled by default
ADD_TRIGGER_DELAY	A 16 bit R&W reg. where to store program value for the delay between pulse and following trigger	zero by default
ADD_PULSE_DELAY	A 32 bit R&W reg. where to store program value for the delay between trigger and following pulse	zero by default
ADD_PROG_NUMPULSES	A 32 bit R&W reg. where to store program value for the number of pulses	zero by default generation does NOT start if this is zero
ADD_STATUS_BAR	A 32 bit Read only reg. to monitor the current number of pulses already sent to the DUT	
fields in operation mode register, namely opMode_wbb		
enable_pls	*field in MODE register for Enabling Pulse output*	*bit[0] high by default*
enable_trg	*field in MODE register for Enabling Trigger output*	*bit[1] high by default*
N.B.: absolute addressing can differ in final release. The current arrangement is stated by the proper package file (pulser.pkg) and can be modified from there.		

Table 13: Mapping of Pulser registers

2.6 Data generator

As said, the MOSAIC system provides a generator of data sample which will be composed by sequential numbers and that can be useful to test communication with external and during the developing of control software. The reference clock for the generation of data, it is that of Wishbone bus.

The sample generated is user-adjustable in terms of:

- Event size, namely the number of data inside a single event/bunch/sample;
- Event delay between subsequent bunches.

The desired values for number of data and delay are written through Wishbone Bus at different addresses. Thereafter, user can start generation by asserting the specific signal (switching ON).

De-asserting the same signal will provoke a stop in the data generation. Please notice that Stop does not imply abort. To the contrary, the module will finish generation of the current packet and it will wait the set delay before to check the enable status in order to decide whether to generate a next event or to stop generation permanently.

Besides, every subsequent overwriting/update of those values does not stop the generation of that sample and the start of another occurs after the current has been completed according with updated values of event size and delays between packets. Furthermore set value for event delay is up-loaded at the end of event generation, therefore please **avoid to update event delay during the generation** until the end of the current event.

Therefore it is not guarantee that data from subsequent bunches were sequentially ordered as supposed. Moreover in case of an overflowing of the receiver buffer, the internal storing can be truncated disregarding generation of data. Hence there can be following data that are not sequential to the previous data, even within the same bunch and therefore in a single transferred packet too.

2.6.1 How to set Data Generator

Symbolic names for addressing the above mentioned registers (for Data Generator main parameters) are summarized and listed in Table 14. Please note that absolute addressing can differ in final release, as a matter of fact the running arrangement is established in the proper package file, in the present case it is named **`generator.pkg`**, and it has to be modified from there.

Data Generator registers mapping		
Address name	**register function**	**Note**
ADD_MODEON	modeOnReg: A 2-bit R&W register to control starting of generation	zero by default MODE reg. is unused
ADD_EVSIZE	EV_SIZE: A 22 bit R&W register for program value of event size	number of data in each single event zero by default
ADD_EVDLY	EV_DELAY: A 32 bit R&W register for program value of event delay	delay between two subsequent events zero by default
bits within operation Mode and ON register, namely modeOnReg		
ON	*1-bit register to enable generation*	*bit[0];* *zero by default*
MODE	*1-bit register for future development*	*UNUSED bit[1];* *zero by default*
N.B.: absolute addressing can differ in final release. **The current arrangement is stated by its package file (generator.pkg) and can be modified from there.**		

Table 14: Mapping of Data Generator registers

In conclusion, there are three different registers that can be set through the Wishbone bus:

1. MODE (un-used) and ON register, addressed by the symbolic reference `ADD_MODEON`;
2. Event size, addressed by `ADD_EVSIZE`. Default value is zero;
3. Event delay, addressed by `ADD_EVDLY`. Default value is zero.

Again as we have seen in the case of the Pulser block, users can access registers also by IP bus transactor or overlook those particulars using software routines we will described later at following Section 4.1.

2.7 Trigger Control Unit

The present module is in charge for:

a. enabling an external trigger;

b. read number of received triggers on a 32 bits counter;

c. read the time interval passed by the first trigger on a 64 bits timer.

At the moment (July 2015) it is still under development a class which implements functions for the previous purposes. Hence, the module has to be accessed by Wishbone bus or through IP bus transactor, as every other parts can always be dealt with. In this framework, the pertinent registers in the Trigger Control Unit are illustrated in the following Table 15. Among others, it has been defined a symbolic reference (`EN_EXT_TRG`) to the configuration register in charge for the enabling of external trigger; that can be useful later during programming. Please notice that counters are reset by asserting of Run signal.

TRIGGER CONTROLLER UNIT: TriggerControl		
Address name	**register function**	**Note**
ADD_CFG	A 16-bit configuration register	see field below
EN_EXT_TRG	*1-bit register field of ADD_CFG for enabling external trigger*	
ADD_TRIGGER_CTR	A 16-bit read only register for the current amount of triggers	Reset by RUN signal
ADD_TIME_L	A 16-bit read only register for Least Significant Bits of a TIMER which counts the time elapsed starting from the first trigger	bits 31:0 of TIMER Reset by RUN signal
ADD_TIME_H	A 16-bit read only register for Most Significant Bits of a TIMER which counts the time elapsed starting from the first trigger	bits 63:32 of TIMER Reset by RUN signal

Table 15: Mapping of registers in Trigger Controller Unit.

2.7.1 Enable external trigger

External signals are synchronized to the Front End Clock Domain through a two stage DFFs pipeline and shaped at one FE clock duration. At this stage, if external trigger is enabled (directly addressing the `ADD_CFG` or by the `EN_EXT_TRG` reference) the above-said shaped signal is send to `TRIGGER` output.

Besides, if requested by the corresponding enable bit, the shaped signal from external trigger is added with the internal trigger signal generated by the Pulser as previously seen (in OR). Otherwise the first mentioned goes alone.

Notice: correlation timing is not granted for the external trigger, in the mean that the exact delay above-mentioned is set only for the internal trigger. A priori the external one is therefore uncorrelated with respect to pulse, unless you make that from outside and under the only condition of cycle duration and synchronicity with FE clk.

2.8 Function and Data Flows

While command to different blocks are sent through and by the Wishbone Bus, user does not really need to access to that. Instead, the MOSAIC system provide an IPbus transactor therefore control of operations can be done externally by means of IPbus control packets going from a software (UDP) client to the hardware target. The complete software client is still under development while at the moment the CAD service INFN Bari has already developed software routines for the direct generation of above described IPbus control packets. Those routines are described in a separate documentation file (produced by Doxygen as referred in following Section 4.1).

2.9 IP bus control packet

As we said MOSAIC system provide an IP Bus transactor for the communication from/to external through an Ethernet connection, namely a simple, IP-based protocol for controlling internal hardware.

In effect MOSAIC is furnished of an internal bus with 32-bit data transfer and 32-bit word addressing , i.e. allowing up to 2^{34} bytes to be addressed. By the way please notice that it is word addressable not bit addressable. The choice of 32-bit widths is fixed in this protocol, though the target host is free to ignore address or data lines if desired.

An IPbus packet consists of a set of transactions, that is to say requests and responses, between the user and one or more target devices (for controlling devices on a virtual bus). Here immediately below the very basic terminology to understand the following:

1. IPbus transaction, an individual IPbus request or response, e.g. a block read request.
2. IPbus packet, one or more individual IPbus transactions that are concatenated together to form the payload of the transport protocol.
3. IPbus client, the software client that generates IPbus transaction requests to control an IPbus host device.
4. IPbus host, the (hardware) device that responds to – and is controlled by – IPbus transaction requests from an IPbus client.
5. Transport protocol, the protocol responsible for transporting the IPbus packet to/from the client/host, e.g. the User Datagram Protocol (UDP).

The IPbus specific information is contained within the payload of the UDP packet, which in turn is wrapped within an IP packet and an Ethernet packet. In an IPbus packet can be queued several IPbus transaction requests, each consisting of an IPbus transaction header and its body. The packet returned by the hardware target would follow the same format, except that each IPbus transaction request would be exchanged for the relevant response format.

Since an IPbus packet consists of a set of transactions, in order to improve transport efficiency, IPbus transactions can be concatenated together as necessary. The recommended transport is UDP. Therefore, each transaction or set of transactions must fit into a single UDP packet.

Note that the maximum size of a standard Ethernet packet is 1500 bytes; with an IP header of 20 bytes and a UDP header of 8 bytes, this gives the maximum IPbus packet size of 368 x 32-bit words, or 1472 bytes. In fact the IPbus protocol does not support fragmentation, hence the maximum block size is the same as the Ethernet.

The specifications to be employed for MOSAIC system, concerning IPbus packet header, IPbus transaction header, and the format of possible IPbus request and response, are detailed in the following sections 2.9.1, 2.9.2 and 2.9.3. For more general details about IPbus packet header, IPbus transaction header, and the format of possible IPbus request/response please refer to the IPbus protocol specification document at reference item [1] at page 1-13.

2.9.1 IPbus Header

Each IPbus v1.4 transaction must start with a 32-bit header of the following format. There is no overall header, however each individual IPbus transaction within the packet has its own header. The number of transactions in a given UDP packet must be deduced from the length of the packet and its content. Each IPbus transaction carries a 32-bit header which content describes the particular transaction request/response according to a standard format described in the following Table 16.

IPbus v1.4 : 32-bit header format address in bit				
31 28	**27 16**	**15 8**	**7 4**	**3 0**
Protocol Version (4 bits)	**Words (12 bits)**	**Transaction ID (8 bits)**	**Type ID (4 bits)**	**Info Code (4 bits)**

Table 16: IPbus header format

The definition and meaning for the above fields are following in Table 17

Protocol Version first four bits (MSB 31 - 28)	Protocol version field. Although actual version of protocol is the 1.4, this field must be **set to 2**
Words next twelve bits (bits 27 - 16)	Number of 32-bit words that are interacted with (written/altered/read) within the addressable memory space of the bus
	Defines read/write depth of blocks to be read/written
Transaction ID eight bits (bits 15 - 8)	Transaction identification number, client could track each transaction
Type ID four bits at (7 - 4)	Defines the request/response type of the IPbus transaction
Info Code last four bits (LSB 3 - 0)	Defines the direction and error state of the transaction request/response
* see the following section Transaction Types for the different ID codes ** see the following section Info Codes for the different codes	

Table 17: definition and meaning of header fields

2.9.2 IPbus Info Codes

Within the IPbus header, the field named Info Code consists of 4 bits which encode direction and error state for the transaction. All allowed code are listed in Table 18 while main purposes are summarized:

a. All requests (for instance client to host) must have an Info Code of 0xf; in effect this is the only code allowed for requests;

b. Requests which are successfully served by the host have, in the response, an Info Code of 0x0;

c. All other values are response error codes that detail how a request failed to be served by the host.

Please notice that some of Info Codes are reserved and have neither useful nor specified meaning in the present version of the protocol.

Info Code	Direction	Meaning
0x0	Response	Request handled successfully by host
0x1	Response	Bad header
0x2	Response	Bus error on read
0x3	Response	Bus error on write
0x4	Response	Bus timeout on read
0x5	Response	Bus timeout on write
0x6	Response	Overflow of the Buffer
0x7	Response	Underflow of the Buffer
0x8	not applicable	Reserved
0x9	not applicable	Reserved
0xa	not applicable	Reserved
0xb	not applicable	Reserved
0xc	not applicable	Reserved
0xd	not applicable	Reserved
0xe	not applicable	Reserved
0xf	Request	Outbound request

Table 18: Info Codes for IPbus header.

2.9.3 IPbus Transaction Types

All transactions have a specific 32-bit IPbus header and some can have a payload. Possible transactions types (either request or response) are identified by a specific code referred to as Transaction Type. The Transaction Type is a 12-bit field in the IPbus header that encodes the direction of a transaction and its error status. Identifying codes and their description are listed in following Table 19.

Transaction Types	Type ID
Byte-order/Idle Transaction	0xf
Read Transaction	0x0
Non-incrementing Read Transaction	0x2
Write Transaction	0x1
Non-incrementing Write Transaction	0x3
Read/Modify/Write Bits (RMWbits) Transaction	0x4
Read/Modify/Write Sum (RMWsum) Transaction	0x5
Get Reserved Address Information Transaction	0xe

Table 19: IP bus Transaction Types

3.0 GETTING STARTED

3 GETTING STARTED

This section provides a first walkthrough MOSAIC setup from initialization to a starting test of functionality of the system. Assuming that you have already installed the board in a crate and plugged all cables needed for your specific setup, first thing you need is to set network connection for communications with the MOSAIC system.

3.1 IP address configuration

The mosaic board has a network IP address stored inside a EEPROM. It can be changed using the following procedure.

On a Linux workstation connected on the same LAN where is connected the mosaic board, start the configuration server utility:

```
cfgsrv IP_start_address [mask]
```

e.g.:

```
$ ./cfgsrv 192.168.168.250
```

Or, if the network mask has to be different from default 255.255.255.0:

```
$ ./cfgsrv 192.168.168.250 255.255.0.0
```

During the configuration user will be requested to press the push button for one second. Then the CPU LED (part n.13 in Figure 1) starts to flash at higher rate, specifically 0.1s on / 0.1s off. During that time the board sends a broadcast request and the server responds with the new IP address and mask. As the board receives the configuration data, the CPU LED lights for 2 s before to return to the normal operation blinking mode.

The configuration utility reports the MAC address of the board requesting configuration data and the IP address assigned:

```
Mosaic IP configuration server
Received Request from MAC address: fc:d4:f2:0c:0a:6e
Assigned address 192.168.168.250 mask 255.255.255.0
```

Since the handshake is done using broadcast UDP messages, the computer has to be able to receive messages on UDP port 10067 and send messages on UDP port 10068. If the machine has an active firewall, configure it to open these ports or temporary disable it. Similarly the presence of a switch in between can hinder a proper connection. In that occurrence, in order to prevent erroneous filtering, you can connect the MOSAIC board directly to the Ethernet port on the PC.

Moreover some configuration of SELinux (Security Enhancement to Linux) inhibit configuration utility to be properly executed. In that case please use the dedicated version of the same tool under Windows ®Microsoft.

3.2 Firmware upgrade

As we said MOSAIC is based on a programmable logic component, namely an FPGA. Hence at startup, MOSAIC board loads its firmware configuration from an EEPROM (a FLASH).

In particular, the flash memory on the MOSAIC board stores two firmware, so called: the working and the "golden" images. For the sake of clarity the current version of the firmware, namely the firmware that will be loaded and used for normal operation of the FPGA is referred to as the working image. Instead the golden image is a minimal version of the firmware needed just for a minimal startup and the loading of the working image.

The golden image is loaded if the loading of the main working image fails for any reason.

Considering what we have said about firmware, MOSAIC has been provided with a software utility, named *artix_fw* and dedicated to upgrade the onboard FPGA firmware.

To upload a new firmware:

```
$ ./artix_fw –file work_image.bin 192.168.168.250
```

It can also read the current firmware version using the command:

```
$ ./artix_fw 192.168.168.250
```

As response the software will display information like the following example:

```
BOARD INFO
Firmware version: 1.1
Flash ID: 0x20 (Manufacturer), 0xba (Type),0x19 (Capacity)
Flash Status Register: 0x00

Software Identity: 05/02/2015  9.13.11,90
Firmware Identity: pALPIDE-2 11/02/2015 14:30
```

Please notice that in order to provide a good protection against programming interruption and error the firmware is update following a secure procedure. In practice: the boot record into the flash is switched to force the loading of the golden image, before to erase the working image. After a correct programming of the working image, the boot record is restored to enable the loading of the new firmware.

3.3 Network considerations

Communication between MOSAIC board and an external controlling PC take place under Transmission Control Protocol (TCP), which provides host-to-host connectivity at the Transport Layer of the Internet model. Generally, over most of the Internet, default value for the Maximum Segment Size (MSS in the following) is limited to a 1460-byte packet, the largest allowed by Ethernet at the network layer.

As we know, a larger MSS brings greater efficiency, first because each network packet carries more user data under the same protocol headers and underlying per-packet delays. Besides a larger MSS also requires the CPU (internal to MOSAIC) processing of fewer packets for the same amount of data. Considering the CPU frequency (around 50 MHz), per-packet-processing would have been a critical performance limitation.

For this reason it is recommended to increase MSS up to 32000 Bytes, applying under Linux, the route command as in following:

```
ip route add 192.168.168.250 advmss 32000 dev eth0
```

Thus, taking the maximum advantage of having implemented MOSAIC Ethernet interface with segmentation in hardware, the resulting improvement in bulk protocol throughput means that the MOSAIC board can stand a throughput of about 120 MB (Gb Ethernet).

It is worth to notice that, in some cases, user will need to set the amount of memory available for IP reconstruction of fragmented packet, in the kernel of the operating system in use.

Suggested commands for the purpose are the following:

```
echo "1000000" > /proc/sys/net/ipv4/ipfrag_high_thresh
```

and for setting the maximum amount of memory available during TCP reception/acceptance:

```
echo “16777216” > /proc/sys/net/core/rmem_max
echo “4096 87380 16777216” > /proc/sys/net/ipv4/tcp_rmem
```

Last consideration is that, due to the small size of the network receiving buffer, if the MOSAIC card is working in a busy network, where there is a lot of broadcast traffic, packet loss may occur. Therefore it is always advised to connect the MOSAIC card on a private subnet in which broadcast traffic is not forwarded.

3.4 Launch the readout test program

It is provided a simple readout test software to check basic functionality of the just settled system.

This **Simpl**e **R**ead-**O**ut **T**est program is written in file named simplereadouttest.cpp.

Basically, the simple readout test program will perform in the order:

- Set up in the data generator the rate of generation, having set the delay between two subsequent events equals to 800 in Wishbone bus clock period and the event size of each event, namely 100000 data (32-bit) for a single event;
- Start the Run;
- Check that received data match expected values.

More in detail:

During operation, please consider that, at the very beginning, the execution of the program will stop the current Run in case the system is already running.

Following procedure, the registers with previous errors are reset while latency time is set to zero, which is to say that the latency mode is fixed at End of Event. Moreover, the threshold for DDR3 memory almost full is set to 512000 Bytes. Then the Data Generator is setup with default values for Event Size and Event Delay respectively equals to 100000 and 800; by the way the data rate is fixed accordingly. To follow, a TCP connection is established (open) and afterward the Run starts.

The function takes one arguments which is the IP address of the employed MOSAIC board.

```
simplereadouttest 192.168.168.250
```

If user does not give any argument, the software will display the following message:

```
usage: readout <IP address>\n
```

After a basic starting of simplereadouttest, namely the software as is, the RUN LED lights up while BUSY LED turns off. After a few seconds, seeing as the data generator writes data in the DDR3 memory faster than the reading speed of Ethernet, BUSY LED will begin blinking. Afterwards, the RUN LED switch off and the software ends execution reading entirely memory contents, within 10 seconds the DDR3 should be empty.

After that, if the test is successfully, the strings shown below will be displayed, in order:

```
Reading data
Stopping run
Skipping data block from unregistered source
Read 3371772736 bytes in 102699 blocks
```

Where "`Skipping data block from unregistered source`" is negligible for the purposes of the present test, due to missing parsing of data from pAlpide readout interface.

Executing the test program under the linux “time” utility it is possible to evaluate the time spend to move the reported amount of data from the board to the PC. Namely after the command:

```
time ./simplereadouttest 192.168.168.250
```

the strings shown below will be displayed (as in previous case if the test is successfully):

```
Reading data
Stopping run
Skipping data block from unregistered source
Read 3371772736 bytes in 102699 blocks
0.226u 4.041s 0:29.45 14.4%     0+0k 656+0io 5pf+0w
```

Otherwise, depending on the specific condition under which the procedure fails, the following error messages will be displayed:

```
Error register:_____________________
```

At last, please notice that the above information are referred to specified software and hardware release, please verify you are working with the following:

Software Identity: 27/04/2015 13.01.14,61

Firmware Identity: pALPIDE-2 05/05/2015 12:11

4.0 USING THE SYSTEM

4 USING THE SYSTEM

As we said MOSAIC system provides an IP Bus transactor for the communication from/to external. Software developer should consider how to build IPbus packets for addressing commands to the various blocks and which will be transmitted through a Wishbone Bus. More over the user interface software has to handle response packet coming back from the various part of the MOSAIC system.

As we previously said the Address Space through Wishbone bus, is structured in two fields, which can be said as BUS ADDRESS = BASE ADD + RELATIVE ADD. The first Field (BASE ADD) consists of the high part, namely the more significant 8 bits are used to indicate the target sub-system block. Besides, the other 24 less significant bit (RELATIVE ADD) are used to indicate registers for data control and functions inside the target device.

It can be useful to remember that the list of relative addresses (offset) is in and that absolute addressing could be different in final release, as a matter of fact the running arrangement is established in the package file **`mapping.pkg`** and can be modified from there.

At the moment there are, still under development, few routines intended to allow the user to access different blocks and perform basic operation with targeted hardware. Therefore users and developer of software interface for MOSAIC can take advantage of classes and headers (undergoing review) written in c/c++, in order to exploit facilities of the system without concern Wishbone and IP buses protocol. On the other hand, technical specifications of internal architecture are useful to a deep understanding of the system and definitely needed for future development. This will be the case for maintenance, upgrading and for coding new routines to perform other tasks on MOSAIC board.

In any case and to be thorough, a list of the blocks as addressable by Wishbone bus, hence through IP bus transactor is in the following Table 20.

Sub system blocks accessible to reference: access routines and Base Address for Wishbone Bus slaves		
Part Name	Reference Class	Base Address (hex)* only 8 MSB
Controller of Run	mrunControl	00
Controller of Trigger	triggerControl	01
Generator of data for test/debug sys communication	dataGenerator	02
I2C bus (Master) Controller	i2cMaster	03
Data (Interface) Controller	controlInterface	04
Pulse Generator to readout device	pulser	05
***Please Note: this is a provisional mapping of Base Address which can change in final release. For the correct mapping (in use) it is mandatory refer to the current version of file mapping.pkg.**		

Table 20: Sub system logic blocks accessible from external

4.1 Classes and Records Access Methods

As we said, commands to different blocks are sent through a Wishbone bus by means of IP Bus control packets going from a software (UDP) client to the hardware target. However thanks to software routines user does not really need to write Ethernet packets (which will contain proper Wishbone bus command wrapped in IP/UDP packet). Those routines are actually C++ classes.

Evidently, to work, every new instance of the classes, operating on the Wishbone bus, needs a pointer to the Wishbone bus interface and the base address of the block on it.

As general remark in the following: we refer to standard integer types and exact-width integer types, for instance like uint32_t = unsigned integer type with width of exactly 32 bits, according to definitions from standard headers <cstdint> and <stdint.h> since C99 [ISO/IEC 9899:1999 p. 264, § 7.18 Integer types].

Ultimately documentation of reference classes is generated by Doxygen software version 1.8.9.1 and supplied aside the present document as HTML file. More details about Doxygen software can be found on line (http://www.stack.nl/~dimitri/doxygen/) as in reference item number [11] at page 1-13.

The doxy-generated file is useful to illustrate in web pages, detached for classes, functions for addressing and performing tasks, either on MOSAIC infrastructure or on its single part (sub-systems) beside to visualize structure and relations in between.