icBLM Firmware - Software Interface

Version History

Version	Full name & Date	Note
0.1	Wojciech Jamuna, 31.01.2019	Initial draft
0.2	Wojciech Jamuna 4.02.2019	Unification with nBLM document

Table of content

Abbreviations

BPM	Beam Position Monitor
EPICS	Experimental Physics and Industrial Control System
ESS	European Spallation Source
FPGA	Field Programmable Gate Array
FW	FirmWare
MPS	Machine Protection System
LLRF	Low Level RF
QW	Quadword (64 bits)
DQW	Double Quadword (128 bits)

Introduction

Overview

The primary goal of the Beam Loss Monitoring (BLM) system is to monitor beam losses of the ESS accelelator and detect abnormal beam behavior and promptly inhibit beam production in case of beam failures to keep the machine safe from beam-induced damage.

Hardware Platform

The hardware platform for the system is based on uTCA standard. The hardware consists of all elements required by uTCA

• uTCA chassis with power supply	mechanical frame with backplane and all connections defined by the standard together with managed power supply unit
• MCH	main management platform for uTCA systems. It executes power-supply negotiations and provides diagnostic and control information for all devices in the chassis

and additional custom boards dedicated for actual functionality:

• (AMC) Concurrent CPU	x86 based processing platfrom running Linux operating system. It can access all devices in a chassis using either PCIe or GbE interface. This is main execution unit for software components of nBLM/icBLM systems.

• (AMC) IFC1410	Kintex Ultrascale FPGA based uTCA FMC carrier equipped with 2 HPC slots and embedded PowerPC system for data readout and platform management. The PowerPC system can be alternative place to run all software components of nBLM/icBLM systems.
• (FMC) PICO4	FMC extension card used with IFC1410 equipped with 4 1 MHz isolated channels for micro-current sampling
• (AMC) EVR	FPGA based timing receiver, which provides timing information for all other boards in the chassis

In the current configuration, the system is not using any RTM modules.

IFC1410

The IFC1410 is powerful FMC carrier board in uTCA standard. It provides enough high-performance resources to interface to fast ADC modules and be able to process several 250 MHz data streams. For theses reasons it has been selected as a main processing unit for nBLM and icBLM implementations. The board is presented in Figure 1.

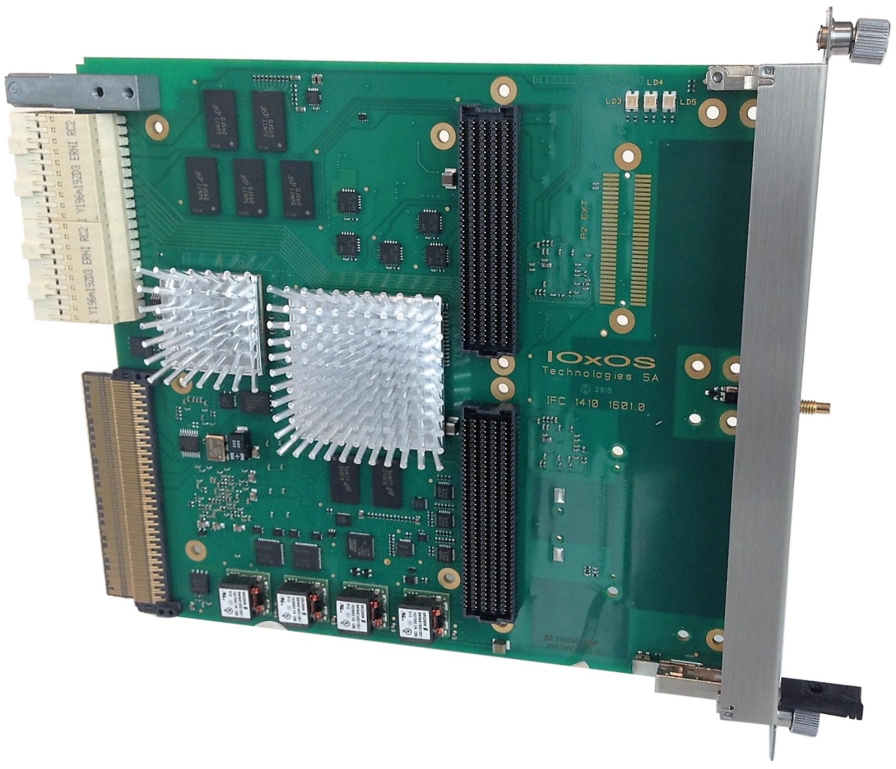


Figure 1. IFC 1410

Main Features of IFC1410:

- FPGA Processing Unit
  - High performance Xilinx Kintex UltraScale KU040 or KU060 FPGA
  - 1024MB dual channel DDR3L-1066 SDRAM (2x 256M x 16)
  - Configuration from on-board SPI flash, or with remote configuration file fetched by the processor through Ethernet
- Processor Unit
  - High-performance Freescale/NXP QorIQ T2081 processor, featuring the e6500 power architecture which includes Altivec (was not available on the P2020 implemented on the IFC_1210)
  - On-board 2GB DDR3L 1866 SDRAM
  - Non-volatile boot memories: dedicated 512MBit (64MB) SPI flash
  - Non-volatile storage memory: dedicated 4Gbit (512MB) NAND flash
  - Powered by U-Boot/Linux and able to run EPICS-based applications
- FMC Interfaces
  - Dual HPC VITA-57.1-compliant FMC slots
  - 80 LVDS channels

- 4 differential clocks (CLK0, CLK1, CLK2, CLK3)
    - 1x GTH x4 channel
    - Programmable VADJ power supply (1.5V – 1.9V)
- AMC Interface
    - Port 0: AMC.2-compliant gigabit Ethernet link with the processor
    - Ports 4 to 7: AMC.1-compliant PCI express x4 Gen3 link with the FPGA Processing Unit
    - Ports 12 to 15: point-to-point LVDS links with the FPGA Processing Unit
    - Ports 17 to 20: shared bus M-LVDS links with the FPGA Processing Unit
    - Telecom clock TCLKA and TCLKB used for ultra low jitter clock

PICO4

The FMC-Pico-1M4 is a standard FPGA Mezzanine Card (FMC - VITA 57.1) Low Pin Count (LPC) board that allows monitoring bipolar currents up to 10 mA with high sampling rate and high resolution.The board is presented in Figure 2.

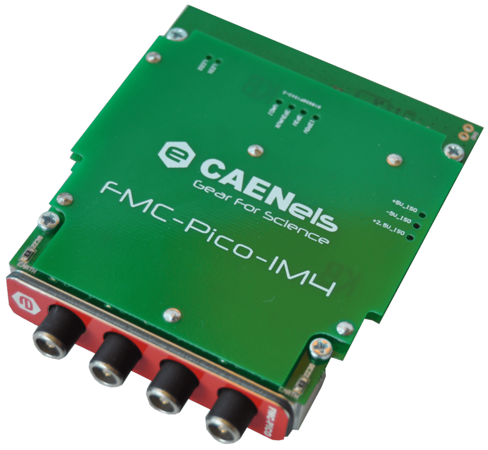


Figure 2. PICO4 board

Main Features of PICO4 board are:

- FPGA Mezzanine Card
- VITA 57.1 Standard
- High Resolution multi-channel current measurements up to ±10 mA (different versions available)
- 4 Current-Input Channels - Bipolar
- 1 MSPS simultaneous and independent sampling
- 20-bit Resolution
- Floating up to 300 V

Firmware description

General Overview

The general structure of the whole FPGA implementation is enforced by TOSCA framework, which provides access channels to PCIe, DDR4 resources and defines interfaces between FMC modules and user logic. Simplified diagram is presented in the Figure 3. The parts implemented during icBLM work are marked with green.

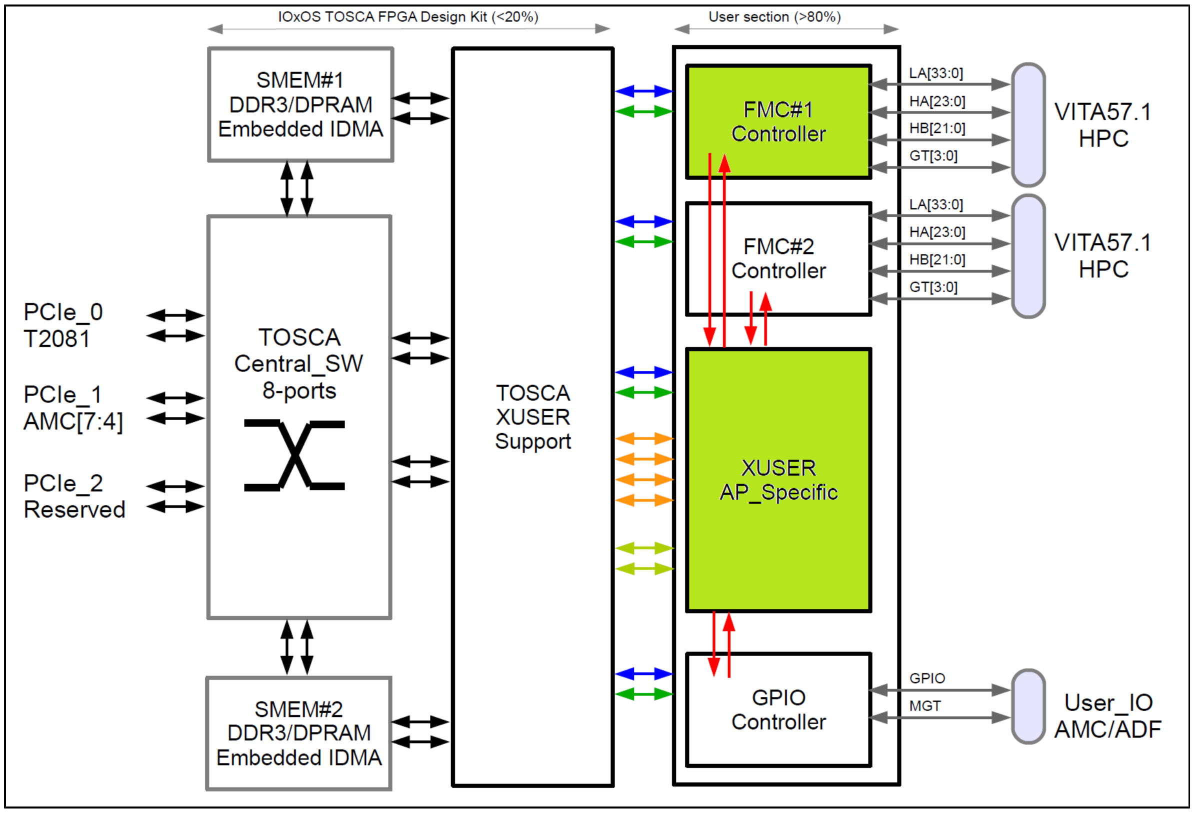


Figure 3 Tosca Framework Diagram.

icBLM Overview

icBLM implementation is placed in the xuser application part of the framework and FMC1 module support. it is using several interfaces provided by TOSCA:

- TSCR bus - for register access (parameters for all components of the app)
- TMEM bus - for dedicated memory access (upload of test patterns)
- SDMA interface - for DDR4 external memory access (DAQ data from framers/circular buffers)

All the processing results are stored in two banks of DDR4 memory on the IFC1410 board, logically treated as 4 independent data streams (called channels) and are available to the software via DMA through the PCI Express interface, both to the internal POWER CPU and the external CPU in MTCA chassis. The data flow diagram in the system is presented below.

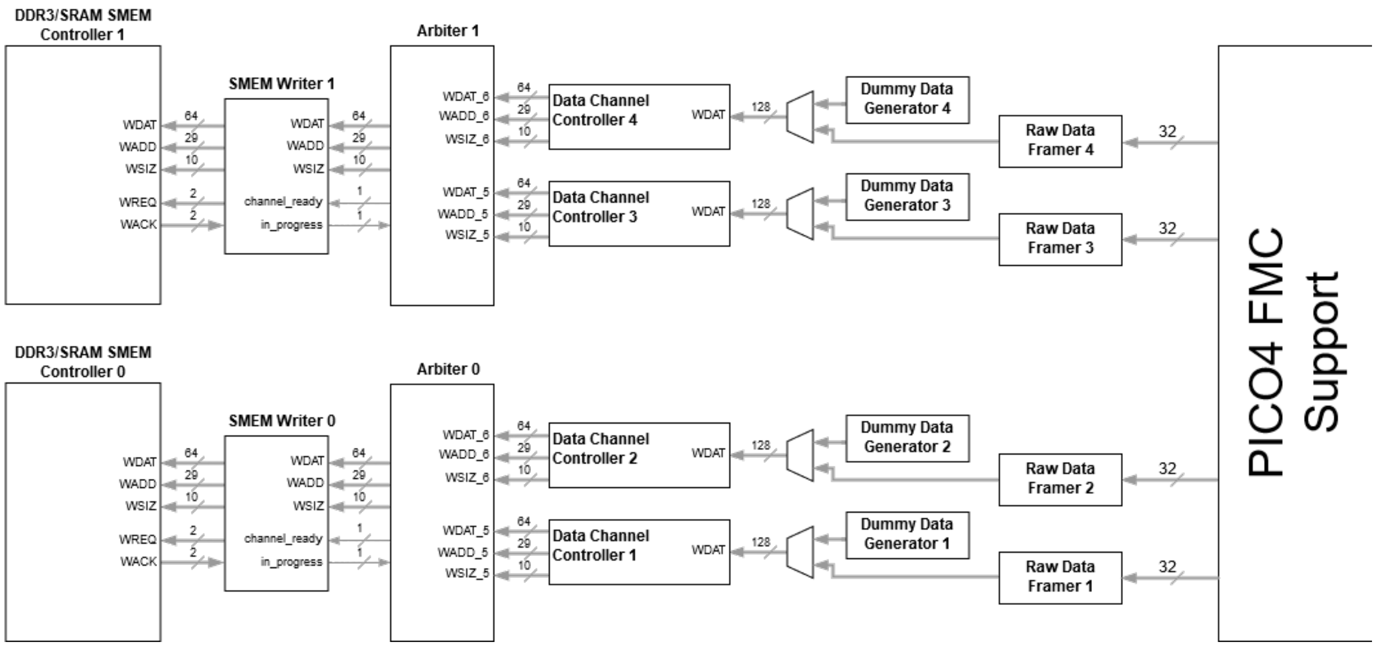
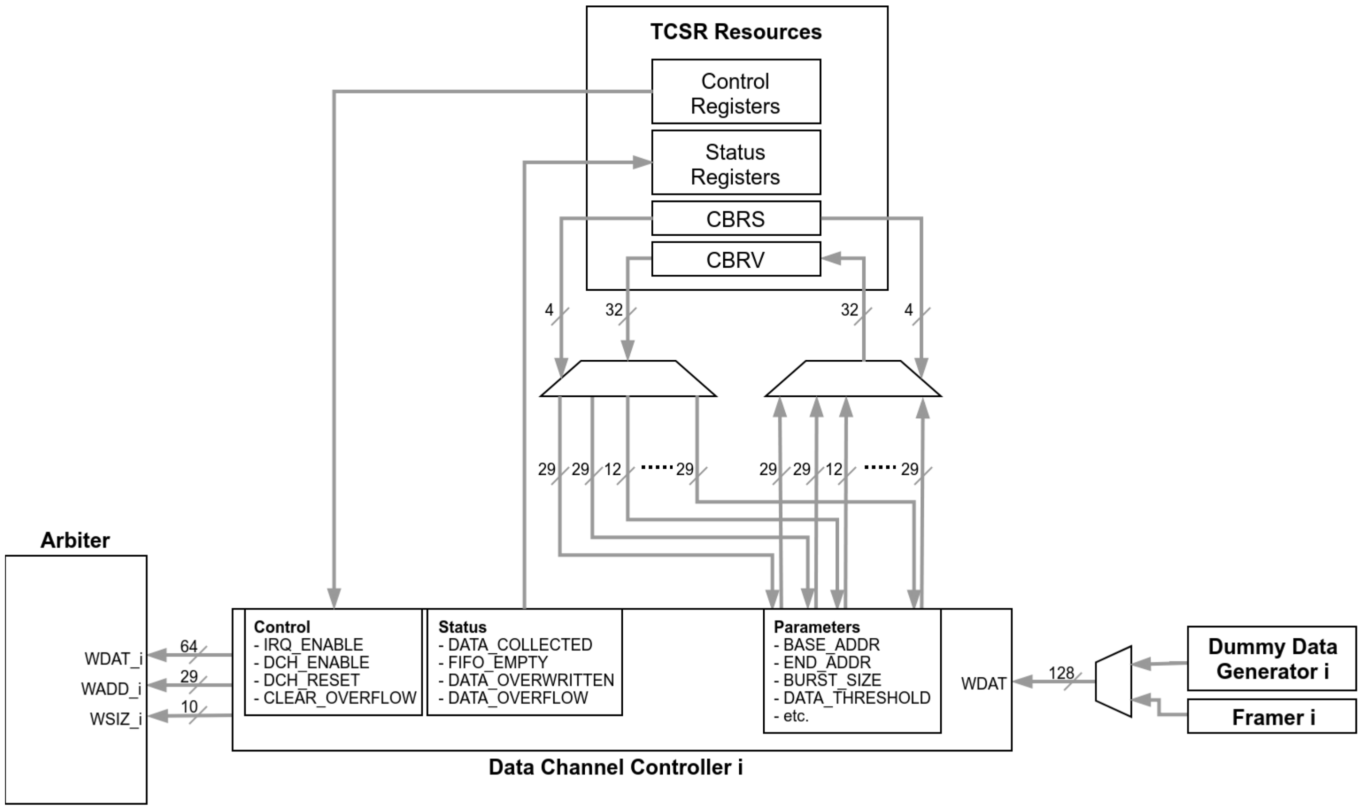


Figure 4. Data flow in a icBLM system.

The key elements of the system are the *Data Channel Controllers*. The number of these controllers is statically configurable by means of constant NUM_OF_CHANNELS. Each channel controller is attached to a *Framer*. For the verification purposes the *Framer* can be replaced with *Dummy Data Generator*. The *Data Channel Controllers* are connected to *Arbiters*, which selects the controller from which the data will be transferred to the DDR3 memory via *SMEM Writer* and *DDR3/SRAM SMEM Controller*. For each memory bank separate *Arbiter* is used.

The *Arbiters* uses Round-robin scheduling algorithm. When the given channel is selected, the *channel_ready* flag is asserted and *SMEM Writer* st arts the negotiations with the *SMEM Controller* in order to start the data transfer. When this transfer is completed, *SMEM Writer* deasserts the *in_progress* flag and *Arbiter* checks if next *Data Channel Controller* is ready for the transmission.

The state and the most parameters of the Circular Buffer firmware can be checked and modified at run time by means of reads and writes to the appropriate registers, as it is schematically presented in figure below. The most crucial (from the point of view of system management efficiency) status and control bits can be accessed directly. The parameters of the system (e.g. base address, end address, etc) and some state variables (write pointer, read pointer), which do not need such an immediate access, can be read and written indirectly, in two steps. In the first step the proper parameter or state variable selector must be written to *CBRS (Circular Buffer Register Selector)* register. Than the selected value can be read or written through the access to *CBRV (Circular Buffer Register Value)* register. Such an approach minimizes the usage of limited *TCSR Resources* while still allows efficient system management.



The internal architecture of *Data Channel Controller* is presented in figure below. Since it interconnects 2 subsystems synchronized by different clocks (250 MHz and 125 MHz), *Data Channel Controller is based on 3 FIFOs:*

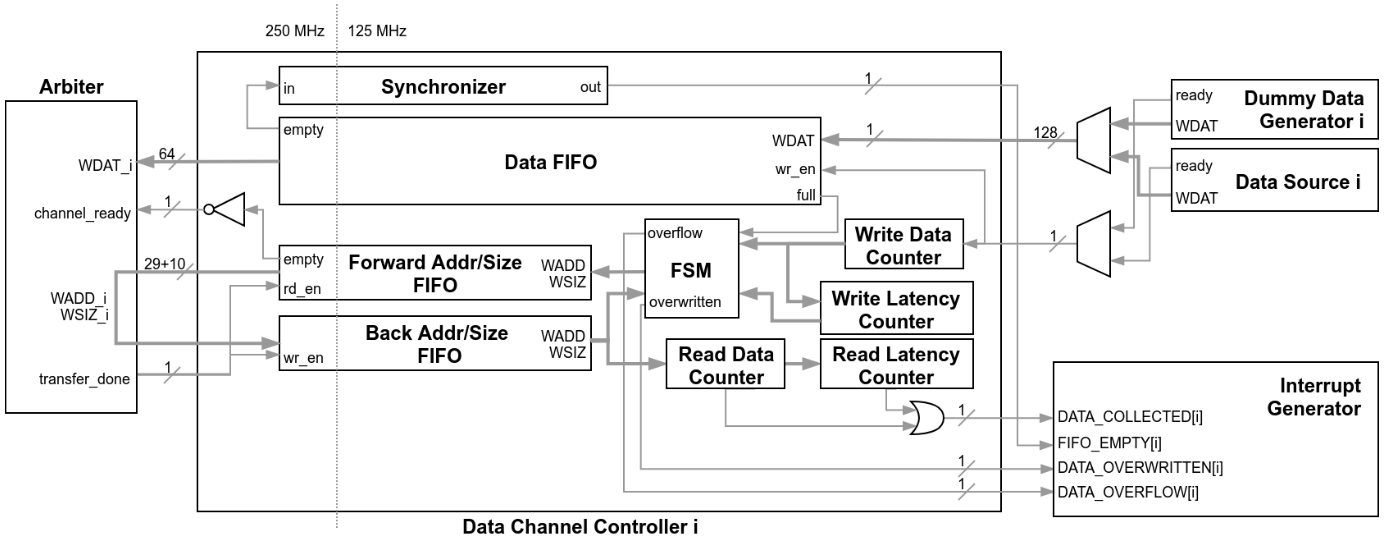
- *Data FIFO*, which except transferring data from one clock domain to another, changes the data width from DQW (Double QW, 16B, 128 bits) to QW (8 B, 64 bits).
- *Forward Address and Size FIFO*, which transfers between clock domains the start addresses and the sizes of the bursts scheduled to be stored in memory.
- *Back Address and Size FIFO,* which transfers between clock domains the start addresses and the sizes of the bursts already stored in memory.

The data flow in the *Data Channel Controller* is monitored by 2 data counters:

- *Write Data Counter*, which counts the bytes inserted into *Data FIFO* and waiting for the burst scheduling - when the number of these bytes exceeds assumed level (BURST_SIZE), the new burst is scheduled (i.e. the start address and the size of this burst is inserted into *Forward Address and Size FIFO*) and the *Write Data Counter* is decremented (by the size of the burst just scheduled).
- *Read Data Counter*, which counts the bytes already stored in memory - when the number of these bytes exceeds assumed level (DATA_THRESHOLD), the interrupt is generated.

The operation of the *Data Channel Controller* is also monitored by 2 latency counters:

- *Write Latency Counter*, which counts the milliseconds from the last burst scheduling - when the value of this counter exceeds assumed level (LATENCY_THRESHOLD), the new burst is scheduled, even if the value of Write Data Counter does not exceed the BURST_SIZE. *Write Latency Counter* is reset if there is no data to be scheduled (*Write Data Counter* = 0) or when the new burst is scheduled.
- *Read Latency Counter*, which counts (in milliseconds) how long the data is waiting in memory for being read by processor - when the value of this counter exceeds assumed level (LATENCY_THRESHOLD), the interrupt is generated. *Read Latency Counter* is reset if there is no data in memory (*Read Data Counter* = 0) or if there is enough data for the interrupt to be generated by the *Read Data Counter* (*Read Data Counter* >= DATA_THRESHOLD).



All the FIFOs and counters of *Data Channel Controller* are managed by *Finite State Machine (FSM)*, which has 5 possible states: *WAIT_FOR_RESET, IN_RESET, WAIT_FOR_NO_RESET, DISABLED, ENABLED*. The state of this *FSM* can be set by means of appropriate writes to control registers. The parameters of the *Data Channel Controller* (BASE_ADDR*,* END_ADDR*,* BURST_SIZE, DATA_THRESHOLD, LATENCY_THRESHOLD) can be set only, when the *Data Channel Controller* is disabled (FSM is in *DISABLED* state, DCH_ENABLE is deasserted). Furthermore, the reset is required to apply the new values of these parameters. Otherwise these values will be ignored.

The *Data Channel Controller* can be reset only after it was disabled first. Otherwise the reset command will be ignored. In some situations disabling the *Data Channel Controller* can take some time since this controller in the same time can be serviced by the Arbiter and this operation must be finished before disabling. Therefore, before issuing the reset command it must be checked if the *Data Channel Controller* is already disabled (deasserting DCH_ENABLE flag by writing to control register has not immediate result - this flag is cleared when the channel is really disabled).

The data in the streams are divided into frames, allowing timestamping and integrity checking. The general structure of the data frame is presented below:

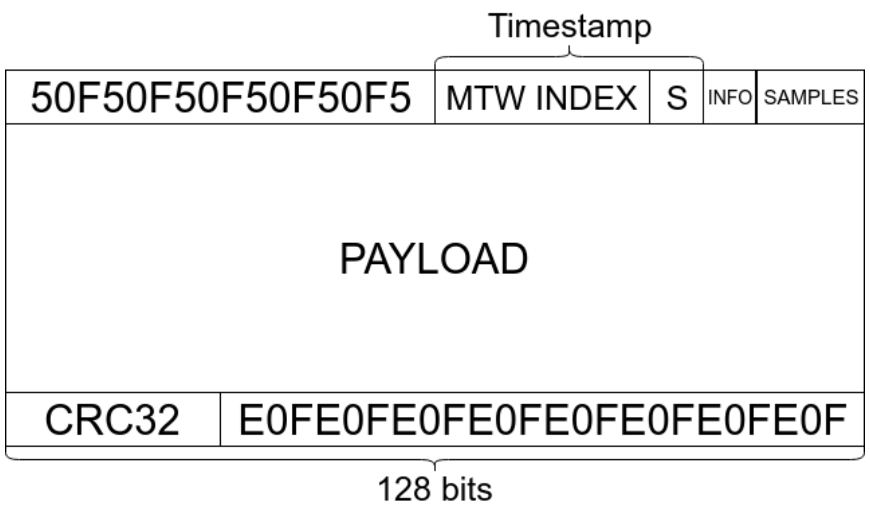


Figure 5. Structure of a data frame

The interface width on the writer side is 128 bits, therefore the frame consists of an integer number of 128-bit words. The first word in the frame contains the following fields:

- Start-of-frame pattern (50F50F50F50F50F5)
- Timestamp consisting of a serial number of the 1-microsecond algorithm window (MTW INDEX) and the number of the sample within the window (S). The timestamp can denote the time when data have been generated (in channels 0-7) or time when the data have been inserted into the buffer (in channel 8)
- 1 generic information byte (INFO, depending on the data channel)
- 16-bit number of samples in the frame (SAMPLES)

The payload consists of several data samples, usually having several fields each, packed back-to-back on the bit level without any additional padding.

The last word in a frame contains a 32-bit CRC of all the previous words in the frame (CRC32) followed by the End-of-frame pattern (E0FE0FE0FE0FE0FE0FE0FE0F).

By default the frame contains SAMPLE_THRESHOLD samples. If there is no possibility to flush the sample buffer due to lack of available bandwidth downstream, larger frame size will be used. Latency timer is used in order to prevent holding data in buffer indefinitely. When LATENCY_THRESHOLD elapses or framer is disabled while the sample buffer is not empty, shorter frame will be sent.

PICO4 Support Module

To handle ADCs on FMC module, the dedicated component was implemented. It manages all actions related to data readout and clock domain crossing. Its structure is presented in the figure 6.

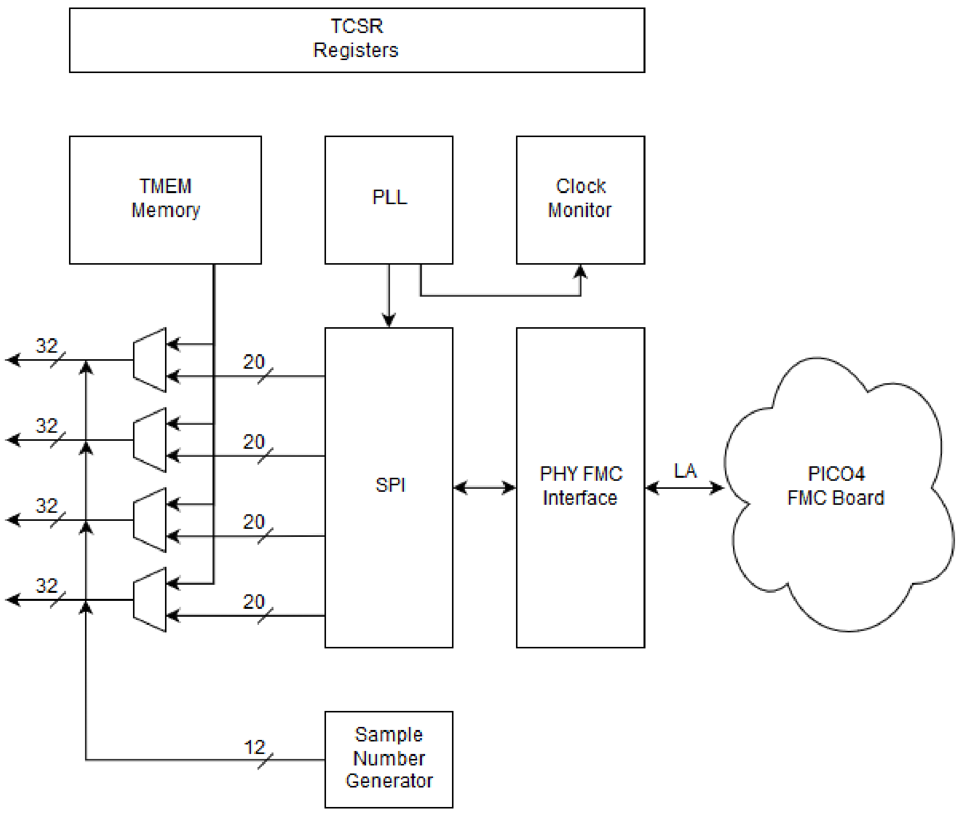


Figure 6. Structure of PICO4 VHDL Module

The following sections describe individual components.

TCSR Registers

The component is responsible for handling of Tosca TCSR bus used for register access. Its primary operation clock is xuser_CLK (provided by TOSCA). No clock domain crossing is needed neither on input nor outputs of the module.

TMEM Memory and Mux

The module is implementing TMEM memory, which can be used to upload test patterns to be used instead of real ADC data. It is connected to Tosca TMEM bus and operates on xuser_CLK. The main module outputs are connected to the Mux, which selects either real ADC signal or test pattern.

PLL and Clock Monitor

The PLL is used to synthesize 75 MHz clock for SPI interface. It is synchronized with xuser_CLK and used internally by SPI module to generate transactions on SPI bus. Clock Monitor module is measuring clock frequencies of all used clocks using xuser_CLK as its reference. The results are provided to TCSR registers.

SPI and Sample Number generator

The SPI is main execution module which performs data readout from SPI devices. It performs all needed clock domain crossings from SPI clock to xuser_CLK, so its external interface is synchronous to xuserCLK. The samples from ADCs are 20-bit wide, the remaining 12 bits are appended by Sample Number Generator, which helps to check data consistency.

PHY FMC

The module maps interface signals into physical connections on FMC connector (LA Bus). It contains instances of all needed IO buffers.

Low Level Interfaces

The following interfaces are present on VHDL level. They must be connected to TOSCA infrastructure or application part (as indicated in a table)

Name	Direction	Interface	Function
xuser_RESET	IN	TOSCA	Reset signal from TOSCA Infrastructure
xuser_CLK	IN	TOSCA	~125 MHz clock from TOSCA - its frequency depends on TOSCA settings
fmc_TCSR_*	IO	TOSCA	Register Access Bus
fmc_TMEM_IF_*	IO	TOSCA	Memory Access Bus
pad_FMC_*	IO	HW	Connections to on-board FMC slot - must be directly connected to FPGA pins
axis_aclk	IN	APP	(NOT USED - kept for compatibility)
axis_tdata	OUT	APP	Contains 4x 32 bits of data for each ADC
axis_tvalid	OUT	APP	Indicates that data on axis_tdata is currently valid
axis_tready	IN	APP	Indicates if APP is ready to receive data (FIFO interface)
p_o_clk_adc	OUT	APP	Output Clock which can be used by APP (copy of xuser_CLK)

Clocking and reset

The icBLM custom logic has two clock domains

- TOSCA_CLK125, the 122 MHz clock defined by the Tosca framework.
  - the clock is used for:
    - register components (for parameter upload)
    - framers and circular buffers
- TOSCA_CLK250, the 275 MHz clock defined by the Tosca framework.
  - It is used by the part of circular buffer controller interfacing to the SMEM Direct interface

The clock domain crossing between the TOSCA_CLK125 and TOSCA_CLK_250 takes place inside circular buffer components.

Software interfaces

Bit width and number representation

All signals in the custom logic implementation are digital binary signals. To represent other values than binary, two or more binary signals are combined into one value. The number of binary signals used to represent a value is referred to as bits.

Value representation is by default unsigned integer numbers. Values that contain signed or fractional values will be marked with a **Signed (int_bits, frac_bits)** or **Unsigned(int_bits, frac_bits)**. All signed numbers are using two's complement number representation.

Examples:

0x7FFF0000 Signed(16,16)	=> 32767.0 decimal
0x80000000 Signed(16,16)	=> -32768.0 decimal
0xFFFF0000 Signed(16,16)	=> -1.0 decimal
0xFFFF8000 Signed(16,16)	=> -0.5 decimal
0xFFFF0000 Unsigned(32,0)	=> 4294901760 decimal
0x0000C000 Signed(16,16)	=> 0.75 decimal

Register map

Registers can be accessed in four different ways, as shown in the table below.

Access method	Abbreviation	Function
Read	R	Read 32-bits, where bits exceeding the size of the register are filled with zeroes.
Write	W	Write 32-bits, where bits exceeding the size of the register are ignored.
Set	S	Set 32-bits, where bits exceeding the size of the register are ignored. Writing to the set interface of a register will change the value of the register to one for all bits that are one in the write and leave the rest of the bits as they were, i.e. New_Reg_val = Old_Reg_val or Write_val.
Clear	C	Clear 32-bits, where bits exceeding the size of the register are ignored. Writing to the clear interface of a register will change the value of the register to zero for all bits that are one in the write and leave the rest of the bits as they were, i.e. New_Reg_val = Old_Reg_val and not(Write_val).

Directly accessible registers

Overview

Offset (base = 0x100)	**Register**	**Access**	**Shadow register**	**Function**
0x98	IRQ_ENABLE	R/W	No	Enable interrupts from *Data Channel Controllers*
0x9C	DCH_ENABLE	R/W	No	Enable *Data Channel Controllers*
0xA0	DCH_RESET	R/W	No	Reset *Data Channel Controllers*
0xA4	CLEAR_OVERFLOW	W	No	Clear overflow flags
0xA8	DATA_COLLECTED	R	No	*Data Channel Controller* data collected flags
0xAC	FIFO_EMPTY	R	No	*Data Channel Controller* FIFO empty flags
0xB0	DATA_OVERWRITTEN	R	No	*Data Channel Controller* data overwritten flags

0xB4	DATA_OVERFLOW	R	No	*Data Channel Controller* data overflow flags
0xB8	RESERVED			
0xBC	RESERVED			
0xC0	CBRS	R/W	No	Circular Buffer Register Selector
0xC4	CBRV	R/W	No	Circular Buffer Register Value
0xD0	AMRS	R/W	No	Algorithm Module Register Selector
0xD4	AMRV	R/W	No	Algorithm Module Register Value
0xD8	RAW_DATA_SELECTOR	R/W	No	Selector of raw data source
0xDC	DECIMATOR_PARAMETERS	R/W	No	Raw data decimator period and duty cycle

DCH_ENABLE (0x9C)

Command for *Data Channel Controller*: enable.

	Default Value	Function
13-0	0x0000 unsigned(14,0)	Writing '1' on given position enables corresponding *Data Channel Controller*. For data consistency reasons, disabling *Data Channel Controller* is blocked if the data transfer from this controller is in progress. Thus, writing '0' on given position does not immediately disable corresponding *Data Channel Controller*. This operation is postponed until the potential data transfer from this controller is finished.
Register default (32-bits): 0x00000000		

Back to register map overview

DCH_RESET (0xA0)

Command for *Data Channel Controller*: reset.

	Default Value	Function
13-0	0x0000 unsigned(14,0)	Writing '1' on given position resets corresponding *Data Channel Controller*. For data consistency reasons this command is ignored when the controller is enabled. In order to reset the *Data Channel Controller,* it must be disabled first.
Register default (32-bits): 0x00000000		

Back to register map overview

CLEAR_OVERFLOW (0xA4)

Command for *Data Channel Controller*: clear overflow flags.

	Default Value	Function

13-0	0x0000 unsigned(14,0)	Writing '1' on given position clears corresponding flag in DATA_OVERFLOW register.
Register default (32-bits): 0x00000000		

Back to register map overview

DATA_COLLECTED (0xA8)

State of *Data Channel Controller*: data collected flags.

	Default Value	Function
13-0	0x0000 unsigned(14,0)	'1' on i-th position means that the required (specified by DATA_THRESHOLD register for i-th data channel) amount of data was transferred to DDR3 memory via *DDR3/SRAM SMEM Controller*. The amount of data collected in memory for i-th data channel is calculated using corresponding W_POINTER and R_POINTER registers. If this amount is grater than DATA_THRESHOLD for i-th channel, appropriate DATA_COLLECTED flag is set and also the interrupt is generated (if enabled by IRQ_ENABLE register). If the DATA_THRESHOLD is set to 0, each burst issued to *DDR3/SRAM SMEM Controller* generates the interrupt and sets the appropriate DATA_COLLECTED flag. The flag is cleared automatically when the amount of unread data in DDR3 memory is reduced below the DATA_THRESHOLD (i.e. when R_POINTER is appropriately modified).
Register default (32-bits): 0x00000000		

Back to register map overview

FIFO_EMPTY (0xAC)

State of *Data Channel Controller*: FIFO empty flags.

	Default Value	Function
13-0	0x0000 unsigned(14,0)	'1' on i-th position means that the output FIFO of i-th *Data Channel Controller* has no data to be transferred to the DDR3 memory via *DDR3/SRAM SMEM Controller*.
Register default (32-bits): 0x00000000		

Back to register map overview

DATA_OVERWRITTEN (0xB0)

State of *Data Channel Controller*: data overwritten flags.

	Default Value	Function
13-0	0x0000 unsigned(14,0)	'1' on i-th position means that the data collected by i-th *Data Channel Controller* was overwritten. These flags can be cleared by reading corresponding R_POINTER_OVERWRITTEN register.

Register default (32-bits): 0x00000000

Back to register map overview

DATA_OVERFLOW (0xB4)

State of *Data Channel Controller*: data overflow flags.

	Default Value	Function
13-0	0x0000 unsigned(14,0)	'1' on i-th position means that the data overflow was detected by i-th *Data Channel Controller*. These flags can be cleared by writing corresponding CLEAR_OVERFLOW register.
Register default (32-bits): 0x00000000		

Back to register map overview

CBRS (0xC0)

Due to the limited register addressing capabilities, the algorithm parameters for specific data processing channels are set in two steps. First, the specific register and channel has to be selected by writing to the AMRS register. Then the selected register is accessible via the AMRV register.

	Default Value	Function
31-16	0x0000	Selects the data channel (0-8)
15-0	0x0000	Selects the specific register (it is an 'Index' in Circular buffer parameters table)
Register default (32-bits): 0x00000000		

Back to register map overview

CBRV (0xC4)

Due to the limited register addressing capabilities, the algorithm parameters for specific data processing channels are set in two steps. First, the specific register and channel has to be selected by writing to the AMRS register. Then the selected register is accessible via the AMRV register.

	Default Value	Function
31-0	0x0000	Depends on the value of the CBRS register
Register default (32-bits): 0x00000000		

Back to register map overview

AMRS (0xD0)

Due to the limited register addressing capabilities, the algorithm parameters for specific data processing channels are set in two steps. First, the specific register and channel has to be selected by writing to the AMRS register. Then the selected register is accessible via the AMRV register.

	Default Value	Function
31-16	0x0000	Selects the data processing channel (0-5)
15-0	0x0000	Selects the specific register
Register default (32-bits): 0x00000000		

Back to register map overview

AMRV (0xD4)

Due to the limited register addressing capabilities, the algorithm parameters for specific data processing channels are set in two steps. First, the specific register and channel has to be selected by writing to the AMRS register. Then the selected register is accessible via the AMRV register.

	Default Value	Function
31-0	0x0000	Depends on the value of the AMRS register
Register default (32-bits): 0x00000000		

Back to register map overview

RAW_DATA_SELECTOR (0xD8)

Selector of raw data source.

	Default Value	Function
2-0	0x0 unsigned(3,0)	Data source for CB channel 0
5-2	0x0 unsigned(3,0)	Data source for CB channel 1
8-6	0x0 unsigned(3,0)	Data source for CB channel 2
11-9	0x0 unsigned(3,0)	Data source for CB channel 3
Register default (32-bits): 0x00000000		

Back to register map overview

DECIMATOR_PARAMETERS (0xDC)

Parameters of raw data decimator.

	Default Value	Function
31-16	0x0000 unsigned(16,0)	Value (in us) of decimator period (DECIMATOR_PERIOD).
15-0	0x0000 unsigned(16,0)	Time (in us) for which the decimator enables raw data framer (DECIMATOR_DUTY). Decimator enables raw data framer for DECIMATOR_DUTY us, every DECIMATOR _PERIOD us.
Register default (32-bits): 0x00000000		

Back to register map overview

Circular buffer and framer parameters (register map)

Overview

Index	Register	Access	Shadow register	Function
0x0000	BASE_ADDR	R/W	No	The address of the first DDR3 page to store the data.
0x0001	END_ADDR	R/W	No	The address of the DDR3 page that follows the last page to store the data.
0x0002	BURST_SIZE	R/W	No	Preferred burst size.
0x0003	DATA_THRESHOLD	R/W	No	The amount of data to be announced when collected.
0x0004	LATENCY_THRESHOLD	R/W	No	The time after which the new transfer should be scheduled.

0x0005	R_POINTER	R/W	No	Read pointer.
0x0006	W_POINTER	R	No	Write pointer.
0x0007	R_POINTER_OVERWRITTEN	R	No	Pointer to overwritten memory.
0x0008	GENERATOR_PARAMETERS	R/W	No	Generator data rate multiplier and divider.
0x0009	SAMPLE_THRESHOLD	R/W	No	Number of samples in frame.

Back to register map overview

BASE_ADDR (0x0000)

Data channel controller parameter: base address.

	Default Value	Function
28-12	0x00000 unsigned(17,0)	The address of the first DDR3 page to store the data. It is aligned to 4kB page boundary (bits 11-0 are ignored).
Register default (32-bits): 0x00000000		

Back to circular buffer and framer parameters

END_ADDR (0x0001)

Data channel controller parameter: end address.

	Default Value	Function
28-12	0x00000 unsigned(17,0)	The address of the DDR3 page that follows the last page to store the data. It is aligned to 4kB page boundary (bits 11-0 are ignored).
Register default (32-bits): 0x00000000		

Back to circular buffer and framer parameters

BURST_SIZE (0x0002)

Data channel controller parameter: burst size.

	Default Value	Function
11-4	0x00 unsigned(8,0)	Preferred burst size. It will be used under the condition that the transfer does not cross the DDR3 page, the data overflow was not detected and LATENCY_THRESHOLD did not elapse. It is aligned to DQW boundary (bits 3-0 are ignored).
Register default (32-bits): 0x00000000		

Back to circular buffer and framer parameters

DATA_THRESHOLD (0x0003)

Data channel controller parameter: data threshold.

	Default Value	**Function**
28-4	0x0000000 unsigned(25,0)	The amount of data (in B) which - when collected in DDR3 memory - should generate the interrupt and set the appropriate DATA_COLLECTED flag in CBS register. if DATA_THRESHOLD is set to 0, each burst generates the interrupt and sets the appropriate DATA_COLLECTED flag. It is aligned to DQW boundary (bits 3-0 are ignored).
Register default (32-bits): 0x00000000		

Back to circular buffer and framer parameters

LATENCY_THRESHOLD (0x0004)

Data channel controller and framer parameter: latency threshold.

	Default Value	**Function**
15-0	0x0000 unsigned(16,0)	For Data Channel Controller: Time (in ms) elapsed (from the previous transfer) after which the new transfer should be scheduled, even if there is not enough data for nominal (determined by means of BURST_SIZE and DATA_THRESHOLD) transfer. Two independent timers control the appropriate transfers: 1. Write latency timer controls the time after which the data (if present in FIFO) should be scheduled for writing to memory. Such scheduling does not generate an interrupt. 2. Read latency timer controls the time after which the data (if present in memory) should be read by processor. When this timer reaches the LATENCY_THRESHOLD value, an interrupt is generated and an appropriate DATA_COLLECTED flag is set. Both timers are limited by the same LATENCY_THRESHOLD. When LATENCY_THRESHOLD is 0, no time-driven transfers are scheduled (the data for full burst must be collected for the new transfer to be scheduled). For Framer: Time (in ms) elapsed from the previous transfer after which the new transfer should be scheduled, even if there is less than SAM PLE_THRESHOLD samples accumulated in buffer. When LATENCY_THRESHOLD is 0, no time-driven transfers are scheduled (the data for full frame must be collected for the new transfer to be scheduled).
Register default (32-bits): 0x00000000		

Back to circular buffer and framer parameters

R_POINTER (0x0005)

Data channel controller parameter: read pointer.

	Default Value	Function
28-4	0x0000000 unsigned(25,0)	The pointer to the DDR3 memory region that follows the one which was already read and can be overwritten (the pointer to the first word (DQW) which was not read yet). It is aligned to DQW boundary (bits 3-0 are ignored).
Register default (32-bits): 0x00000000		

Back to circular buffer and framer parameters

W_POINTER (0x0006)

Data channel controller parameter: write pointer.

	Default Value	Function
28-4	0x0000000 unsigned(25,0)	The pointer to the DDR3 memory region that follows the one which was already written and can be read (the pointer to the memory which will be written during next transfer). It is aligned to DQW boundary (bits 3-0 are ignored).
Register default (32-bits): 0x00000000		

Back to circular buffer and framer parameters

R_POINTER_OVERWRITTEN (0x0007)

Data channel controller parameter: pointer to overwritten memory.

	Default Value	Function
28-4	0x0000000 unsigned(25,0)	The value of R_POINTER register for which the part of the memory was overwritten. It is aligned to DQW boundary (bits 3-0 are ignored).
Register default (32-bits): 0x00000000		

Back to circular buffer and framer parameters

GENERATOR_PARAMETERS (0x0008)

Data channel controller parameters: generator data rate multiplier and divider.

	Default Value	Function
31-24	0x00 unsigned(8,0)	Generator data rate multiplier (GENERATOR _MULTIPLIER).
23-0	0x000000 unsigned(24,0)	Generator data rate divider (GENERATOR_ DIVIDER). Data generator produces GENERATOR_MULTIPLIER words (DQWs) every GENERATOR_DIVIDER clock cycles.

		E.g.: when GENERATOR_MULTIPLIER = 1 and GENERATOR_DIVIDER = 3, only 1 DQW is generated every 3 clock cycles. When GENERATOR_MULTIPLIER >= GENERATOR_DIVIDER, 1 DQW is generated every clock cycle. If GENERATOR_MULTIPLIER = 0 or GENERATOR_DIVIDER = 0, generator is disabled (other data source is used, see Data flow in a nBLM system).
Register default (32-bits): 0x00000000		

Back to circular buffer and framer parameters

SAMPLE_THRESHOLD (0x0009)

Framer parameter: number of samples in frame.

	Default Value	Function
15-0	0x0000 unsigned(16,0)	Nominal number of samples in frame. If there is no possibility to flush the sample buffer due to lack of available bandwidth downstream, larger frame size will be used. When LATENCY_THRESHOLD elapses or framer is disabled while the sample buffer is not empty, shorter frame will be sent.
Register default (32-bits): 0x00000000		

Back to circular buffer and framer parameters

PICO4 Module Registers

Overview

Offset (base =)	**Register**	**Access**	**Function**
0x80	ID	R/W	ID register
0x81	RST	R/W	*Reset register*
0x82	RESERVED	-	*RESERVER*
0x85	PATTERN_MASK	R/W	MUX control
0x86	CLK_MON0	R	Clock Monitor Channel 0
0x87	CLK_MON1	R	Clock Monitor Channel 0

ID (0x80)

ID of the FMC module

	Default Value	Function
31-0	0xDEADBEE2	The register contains ID of the FMC module. It can be used to verify if proper FW is present in FPGA.
Register default (32-bits): 0xDEADBEE2		

RST (0x81)

Control of Reset Signal for all submodules

	Default Value	Function
0	0	'0' - all submodules are in reset state - no DAQ is done '1' - normal operation
Register default (32-bits): 0x00000000		

PATTERN_MASK (0x85)

Control of the mux for individual ADC channels

	Default Value	Function
3-0	0000	each bit of the register corresponds to single ADC channel: '0' - normal operation - ADC data is passed to module output '1' - pattern memory is routed to module output
Register default (32-bits): 0x00000000		

CLK_MON0 (0x86)

Output 0 of clock monitor

	Default Value	Function
31-0	0x00000000	Supplies measured frequency of channel 0 of clock monitor module. In this case xuser_CLK is used as both reference and channel 0, so it will always report 122000000
Register default (32-bits): 0x00000000		

CLK_MON1 (0x87)

Output 1 of clock monitor

	Default Value	Function
31-0	0x00000000	Supplies measured frequency of channel 0 of clock monitor module. the channel is connected to SPI clock.
Register default (32-bits): 0x00000000		

Data frames

The icBLM design is using only Raw Data Framers - all acquired data are in the following format:

Raw Data (CB channel 0-3)

Raw samples from one of the channels:

- INFO: data channel
- SAMPLES: number of 32 bit samples

Firmware usage

Board tests

The program to test the board operation and perform data acquisition can be found in the sw/blm_driver_icblm subdirectory.

Usage example:

```
./mem_dma_reader_icblm -c <bitmask> -e 10
```

<bitmask> allows enabling individual channels

bits 0-3 - ADC channels raw data

Firmware limitations and known issues

Loss of data during simultaneous DMA transfers from both DDR banks

There is a bug in the TOSCA firmware or device driver causing data loss when the DMA transfer is performed simultaneously from two different memory banks using different DMA channels from different threads. The workaround is protections of the DMA transfer function in the userspace program by mutex, but this solution reduces performance.

Tsc driver not supporting scatter-gather operations and prone to resource leaks

Tsc driver has high performance, but requires allocation of contiguous physical memory buffers for operation. When memory is fragmented, it is not possible to allocate these memory buffers and a CPU reboot is required. Tosca driver did not have this problem, but it offered much lower performance.

These memory buffers have to be manually allocated and are not freed automatically when the device file is closed. It requires appropriate signal handlers in the userspace program to avoid resource leaks.

Problems with PCIe Gen3 link to Concurrent CPU

The PCIe link speed between Concurrent CPU and the IOxOS board alternates between Gen1 and Gen3 after each system reboot.

No separate interrupt support for both memory banks

Only one interrupt, common for both memory banks, is supported. It is a firmware issue and will be fixed

Tsc Driver cannot access PON space when run on Concurrent CPU

TscMon (and custom software applications) fail to access registers placed on PON configuration space (such as Power On for FMC cards) when run on Concurrent CPU. The same software and TscMon commands can be successfully run on embedded IFC1410 processor. The bug prevents proper behaviour of the software when only one PCIe endpoint (to Concurrent) is present in the design.