Project:

MNEMOSENE

(Grant Agreement number 780215)

"Computation-in-memory architecture based on resistive devices"

Funding Scheme: Research and Innovation Action

Call: ICT-31-2017 "Development of new approaches to scale functional performance of information processing and storage substantially beyond the state-of-the-art technologies with a focus on ultra-low power and high performance"

Date of the latest version of ANNEX I: 11/10/2017

D3.2 –Initial Communication Protocols and Infrastructure

Project Coordinator (PC):	Prof. Said Hamdioui Technische Universiteit Delft - Department of Quantum and Computer Engineering (TUD) Tel.: (+31) 15 27 83643 Email: S.Hamdioui@tudelft.nl
Project website address:	www.mnemosene.eu http://www.leo-fp7.eu/
Lead Partner for Deliverable:	Eidgenössische Technische Hochschule Zürich (ETHZ)
Report Issue Date:	03/04/2019

Document History *(Revisions – Amendments)*	
Version and date	**Changes**
1.0 02/04/2019	First version

Dissemination Level		
PU	Public	**X**
PP	Restricted to other program participants (including the EC Services)	
RE	Restricted to a group specified by the consortium (including the EC Services)	
CO	Confidential, only for members of the consortium (including the EC)	

European Commission

The MNEMOSENE project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant

The MNEMOSENE project aims at demonstrating a new computation-in-memory (CIM) based on resistive devices together with its required programming flow and interface. To develop the new architecture, the following scientific and technical objectives will be targeted:

- Objective 1: Develop new algorithmic solutions for targeted applications for CIM architecture.
- Objective 2: Develop and design new mapping methods integrated in a framework for efficient compilation of the new algorithms into CIM macro-level operations; each of these is mapped to a group of CIM tiles.
- Objective 3: Develop a macro-architecture based on the integration of group of CIM tiles, including the overall scheduling of the macro-level operation, data accesses, inter-tile communication, the partitioning of the crossbar, etc.
- Objective 4: Develop and demonstrate the micro-architecture level of CIM tiles and their models, including primitive logic and arithmetic operators, the mapping of such operators on the crossbar, different circuit choices and the associated design trade-offs, etc.
- Objective 5: Design a simulator (based on calibrated models of memristor devices & building blocks) and FPGA emulator for the new architecture (CIM device combined with conventional CPU) in order demonstrate its superiority. Demonstrate the concept of CIM by performing measurements on fabricated crossbar mounted on a PCB board.

A demonstrator will be produced and tested to show that the storage and processing can be integrated in the same physical location to improve energy efficiency and also to show that the proposed accelerator is able to achieve the following measurable targets (as compared with a general purpose multi-core platform) for the considered applications:

- Improve the energy-delay product by factor of 100X to 1000X
- Improve the computational efficiency (#operations / total-energy) by factor of 10X to 100X
- Improve the performance density (# operations per area) by factor of 10X to 100X

Table of Contents

1. Introduction

In this report, we describe the circuits and methods to allow efficient data transfer between tiles and the outside world. These form the basis of an initial memory-centric architecture. More specifically, the focus of D3.2 is on the description of the interface and communication protocol between the CIM Macro Block and the rest of processing chains in the conventional processing units. The heart of our architecture is based on Parallel Ultra Low Power (PULP) cluster that efficiently integrates one or many CIM Macro Blocks with conventional processing cores and shared data memory. PULP provides all necessary interfaces to program CIM Macro Blocks and allows their fast accesses to data reside in shared memory thereby eliminating data transfer overheads.

In the rest of this report, we first provide an overview of PULP architecture in Section 2. In Section 3, we detail the CIM Macro Block interface, followed by CIM accelerator interface in Section 4.

2. Architecture Overview

In this section, we first provide a top down description of the toplevel system architecture and the flow of data between the conventional CPU cores, the shared memory and CIM accelerator. A more detailed description of the interface signals, protocols and configuration registers is provided in section 3 and 4.

2.1 SoC Architecture

Figure 1 depicts the architecture of the SoC. The system is split into two different power/clock domains: The Cluster domains contains several general-purpose processing cores, CIM Accelerators and shared memory. The SoC domain on the other hand contains IO blocks for communication with off-chip peripherals and external memory, a power management unit and clock generators. All these components as well as the cluster domain is controlled by a tiny processor, the so called fabric controller.

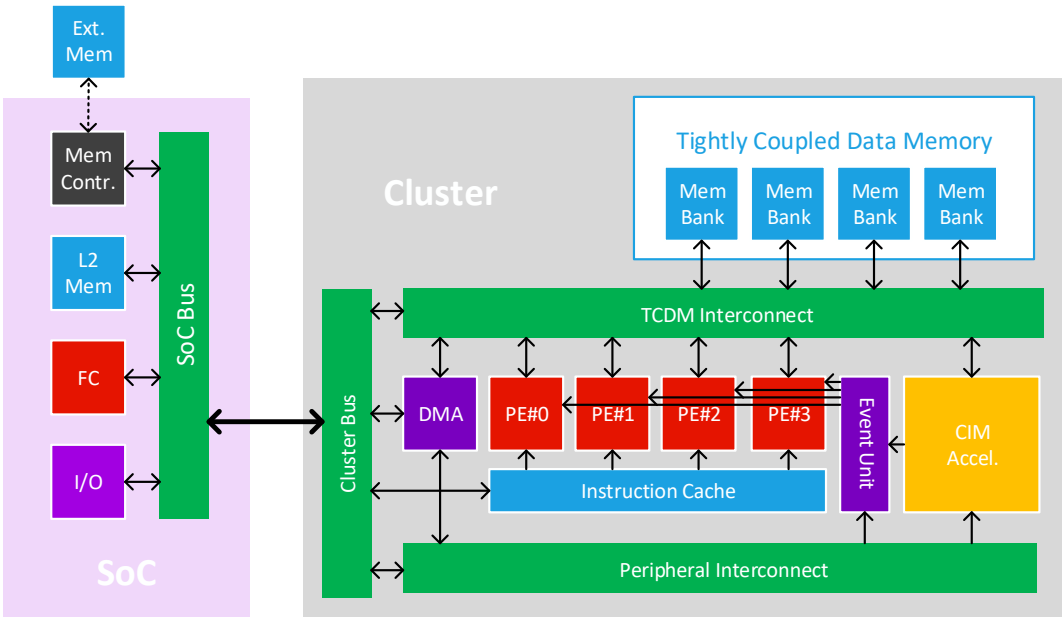


Figure 1 SoC Architecture

2.1.1 Fabric Controller

The fabric controller, in previous PULP architectures based on a very lightweight RISC-V core (zeroRISCY), is responsible for orchestrating I/O peripherals the external memory controller as well as the cluster. To process data on the high-performance cluster the fabric controller can fork execution to the general purpose cores within the cluster and can dynamically adjust the clusters frequency for optimal performance and energy efficiency.

2.2 Cluster Architecture

The cluster consists of several general purpose processing elements (PE) based on ETH's RI5CY RV32IMF core and one or multiple CIM Accelerators. All processing elements share access to the tightly-coupled data memory, a multi-banked software managed scratchpad memory for data exchange between the processing elements. The DMA module within the cluster can be programmed by the Processing elements to transfer data between the larger L2 memory in the SoC domain and an event unit provides the means for synchronization between the processing elements. Each processing element is connected via two interconnects: The peripheral interconnect is a low bandwidth bus based on AMBA APB[1] used for configuration of the DMA, event unit and most importantly the CIM Accelerators. The TCDM interconnect provides low latency and high-bandwidth access to the shared TCDM memory and is based on the so-called Logarithmic Interconnect, a forest of arbitration trees providing parallel single cycle access to the multi-banked memory with transparent arbitration in the case of bank conflicts between different processing elements.

[1] Advanced Peripheral Bus

2.3 CIM Accelerator

The CIM Accelerator contains a single CIM Macro Block and interface logic that provides arbitration and synchronization of requests from multiple cores to the same CIM Accelerator and hides the tasks of address generation and data realignment from the CIM Macro Block.

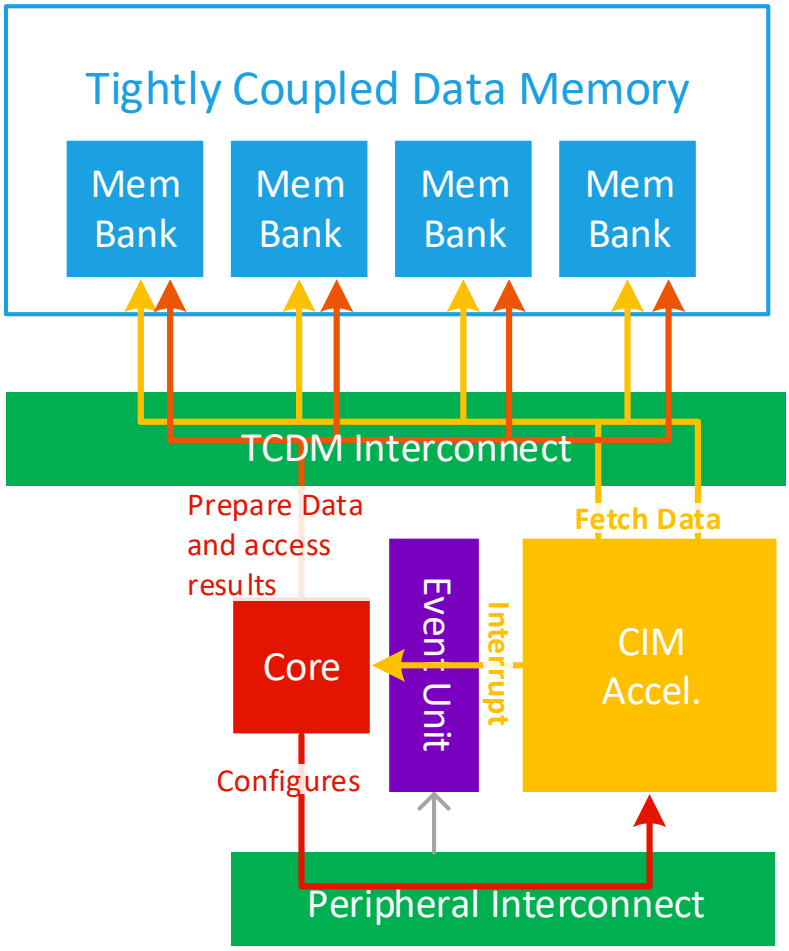


Figure 2 Data flow between Core and CIM Accelerator

Figure 2 illustrates how a conventional processing core can offload a micro-ISA instruction to the CIM Accelerator. Software running on a conventional CPU core pre-processes and prepares the data that is to be processed by the CIM Accelerator by writing it to the shared memory. Then the core writes the micro-ISA instruction and its corresponding arguments (e.g. addressing scheme, matrix dimensions etc.) to memory mapped configuration registers within the CIM Accelerator via the *Peripheral Interconnect*[2]. After the core is done configuring the accelerator it triggers the execution of the micro-ISA instruction. The CIM Accelerator will then independently fetch the input data (e.g. weights to writ to the non-volatile memory or vector elements to process) from the shared memory and write back the results to the shared memory according to the memory access scheme configured by the

[2] From a software point of view there is no difference between accessing the TCDM and the Peripheral Interconnect (except for latency). They are just mapped to different memory regions.

core. Once the CIM Accelerator finished the execution of the micro-ISA instruction it sends an event signal to the event unit which in term interrupts the core that offloaded the instruction. Finally, the core can then post-process the results, start another micro-ISA instruction or issue a DMA transfer to move them out of the cluster.

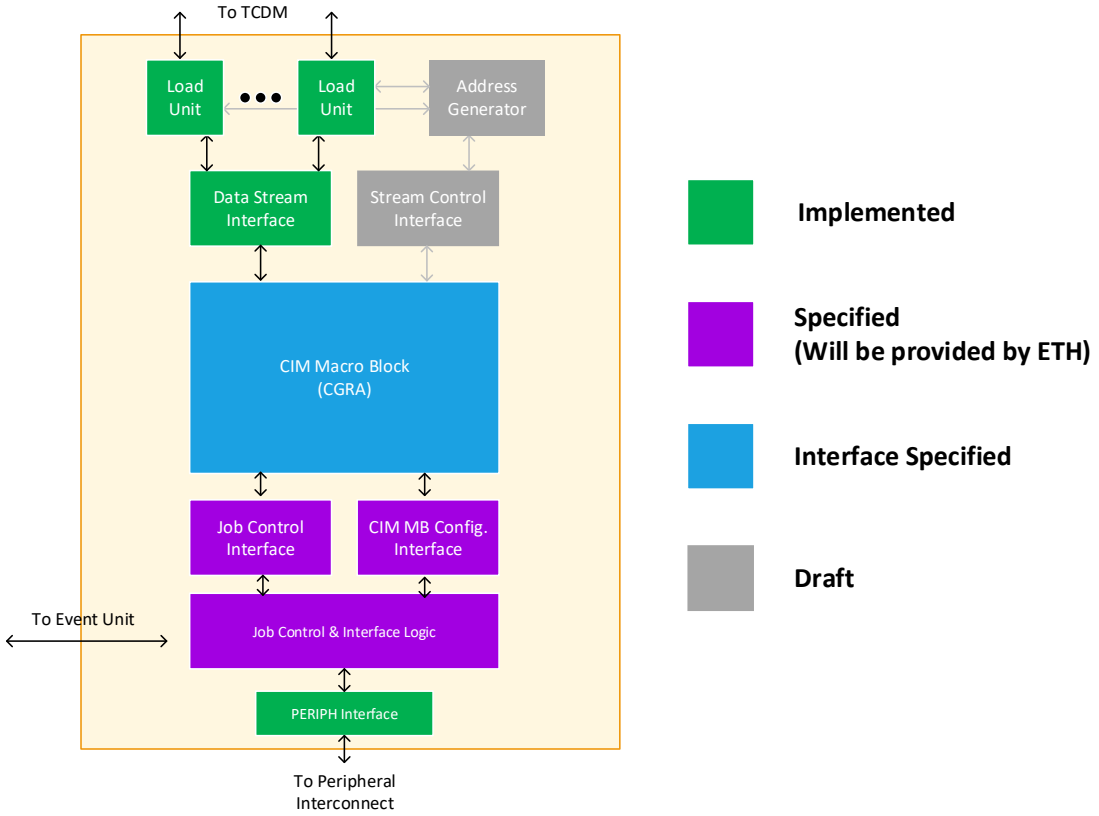


Figure 3 CIM Accelerator Architecture

Figure 3 depicts the structure of the CIM Accelerator block and color codes the readiness and responsibility of the individual IPs with the CIM Accelerator. Together with the *Address Generator* blocks the *Load- and Store-Units* issue read- and write-transactions to the logarithmic interconnect, potentially handling misaligned access.

There are four different interfaces between the CIM Macro Block and the interface logic:

- **Instruction Offload Interface** - The Instruction Offload Interface notifies the CIM Macro Block when a new micro-ISA instruction was offloaded by one of the cores. It exposes the configuration registers related to the execution of a single micro-ISA (e.g. OP code, matrix dimensions, addressing schemes etc.) to the CIM Macro Block. The interface logic between the CIM Macro Block and the Peripheral Interface contains multiple copies of this register file to support multiple outstanding micro-ISA instructions while the CIM Macro Block is only exposed to one of them corresponding to the micro-ISA instruction currently processed.

- **Static Configuration Interface** - The Static Configuration Interface is used to change long-lasting configuration of the CIM Macro Block (e.g. offset correction values for the memristive devices, status and control registers specific to the CGRA or debug functionality). In contrast to the micro-ISA instruction specific registers read- or write-accesses to these registers are handled within the CIM Macro Block. The Static Configuration Interface bypasses requests from the Peripheral Interconnect directly to CIM Macro Block.
- **Data Stream Interface** - The data to be processed enters the CIM Macro Block via the Data Stream Interface. The interface hides the complexity of the individual transactions to the Logarithmic Interconnect from the CIM Macro Block and instead exposes contiguous streams of data through a simple ready/valid handshaking interface.
- **Stream Control Interface** - The Address Generator and the load and store units are controlled by the CIM Macro Block through this interface. The CIM Macro Block can setup new streams by providing the parameters for a strided 3-dimensional memory access scheme.

Since TUE also has load- and store-units available for their CGRA architecture it is at this point not yet decided whether the Data Stream Interface and Stream Control Interface with the external load- and store-units and corresponding address generators will be used. The other possible solution would be to remove these blocks from the CIM Accelerator and directly expose the TCDM interface to the CGRA (CIM Macro Block).

2.4 Terminology

Here, we provide a list of terms and their descriptions that are used extensively throughout this document:

- **CIM Macro Block** *CIM-MB*: The macro architecture which combines multiple CIM-tiles to perform more complex operations. This block will be based on the Coarse Grain Configurable Array (CGRA) architecture from TUE.
- **micro-ISA**: The instruction set architecture used by the CIM Macro Block. A single micro-ISA instruction represents a macro operation (e.g. matrix vector multiplication) that is atomicly processed on the memristive crossbars within the CIM Macro Block.
- **CIM Accelerator**: The combination of a single CIM-MB and the logic for micro-ISA instruction queueing, address generation and configuration interface. This is the entity that will be instantiated one ore multiple times inside the PULP cluster alongside the General Purpose Cores.
- **General Purpose Core *GP core***: Conventional Von-Neumann architecture-based processor cores. They perform conventional computation and offload micro-ISA instructions to the CIM-Accelerators.
- **Tightly-coupled-data-memory *TCDM***: A high-bandwidth, low-latency memory based on multiple SRAM banks which is efficiently shared between all the GP cores and all the CIM-Accelerators thanks to a logarithmic interconnection and simple memory protocol.
- **Cluster**: A system consisting of several GP cores, CIM-Accelerators, TCDM and interconnection fabric.

3. CIM Macro Block Interface

In this section the interface signals to the CIM Macro Block and the corresponding handshaking protocols are described in detail. All signal directions are stated from the perspective of the CIM Macro Block. Therefore, *input-signals* are signals driven by the interface logic around the CIM Macro Block while *output-signals* are driven by the CIM Macro Block. **If not otherwise indicated all signals are active-high.**

3.1 Clock & Reset

All the interface signals of the CIM Macro Block are synchronized to the rising edge of the reference clock `REF_CLK`.

Signal	Width [bits]	Direction	Description
REF_CLK	1	*input*	The reference clock with which the interface is synchronized.
RSTN	1	*input*	Asynchronous active-low reset.
SOFT_CLEAR	1	*input*	Synchronous active-low soft reset.

Table 1 Clock and reset signals

The asynchronous reset signal `RSTN` is asserted during system power-up and completely resets the CIM Macro Block. The synchronous soft clear signal `SOFT_CLEAR` on the other hand is used by the interface logic to reset the CIM Macro Block to a known good state in the case of a *non-recoverable error* (see 3.2.1) or when a Soft Clear was requested by one of the cores (see 4.3.1). A soft clear does not necessarily re-initialize the content of the non-volatile memory or long-lived configuration but must unconditionally and immediately put the CIM Macro Block into a known good idle state where it is ready to accept a new micro-ISA instruction. Upon completion of the soft clear the `STATUS` signal[3] must be `IDLE`.

3.2 Instruction Offload Protocol

Table 2 gives an overview of all signals related to this interface. As already explained in subsection 2.3 the interface logic around the CIM Macro Block contains multiple identical register file for micro-ISA related configuration to allow cores to already start configuring the next micro-ISA instruction while the CIM Macro Block is still busy processing the previous one and allows to hide the latency of the *Peripheral Interconnect*. The Instruction Push Interface exposes the register file corresponding to the micro-ISA instruction that is currently being processed to the CIM Macro Block. The configuration registers within these register file remains to be specified and depends on the operations supported by the CIM Macro Block.

[3] See 3.2.2 CIM Macro Block Status

Signal	Width [bits]	Direction	Description
INSTR_REQ	1	*input*	Requests the CIM-MB to process a new micro-ISA instruction
INSTR_ACK	1	*output*	Indicates that the CIM-MB finished processing the micro-ISA instruction.
INSTR_PARAM_ADDR	8	*output*	Instruction Parameter Register Address.
INSTR_PARAM_DATA	32	*input*	Instruction Parameter Register Data.
STATUS	8	*output*	The current status of the CIM-MB. See 3.2.2 for details.

Table 2 Instruction offload and CIM Macro Block status signals

Figure 4 illustrates the handshaking between the interface logic and the CIM Macro Block. As soon as the `INSTR_REQ` is asserted the CIM Macro Block can start reading from the register file for the new micro-ISA instruction to be processed. Access to the configuration registers is read-only and single cycle. During the processing phase of the micro-ISA instruction the `STATUS` signal may change several times indicating the current *Busy State* of the CIM Macro Block. Upon successful completion of the micro-ISA instruction the CIM Macro Block must change the `STATUS` signal back to `IDLE` and must assert the `INSTR_ACK` line for a single clock cycle. The data in the currently exposed register file is guaranteed to remain valid until next rising clock edge after `INSTR_ACK` was asserted.

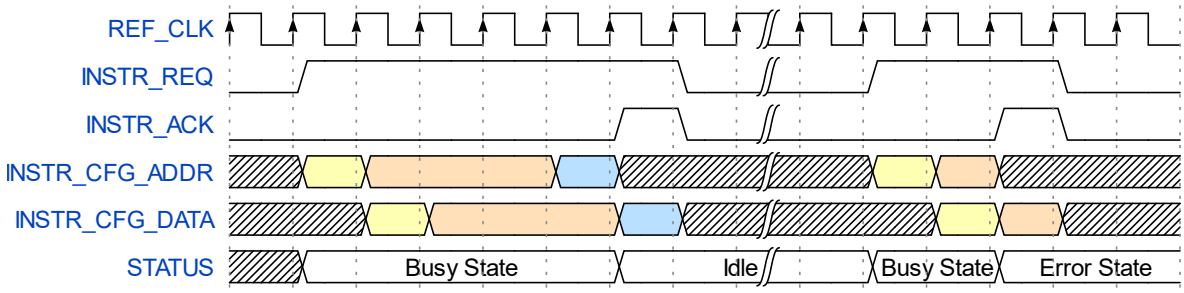


Figure 4 Instruction offload protocol handshaking with two micro-ISA instructions processed.

3.2.1 Error Conditions

There are two kinds of errors the CIM Macro Block can indicate through the `STATUS` line:

- *Recoverable Errors* - These kind of error codes indicate an error during the execution of the current instruction (e.g. malformed instruction parameters). The CIM Macro Block can however accept a new instruction without problems. The interface logic will continue issuing new instructions and expose the error to the core that offloaded the instruction via the STATUS register. The CIM Macro Block must still assert the `INSTR_ACK` line for single cycle to notify the interface logic about the unsuccessful completion of the micro-ISA instruction.

- *Non-recoverable errors* - These errors indicate that the CIM Macro Block is in an erroneous state where it cannot accept any new instructions. A status code of this kind will cause the interface logic to immediately issue as soft clear upon the next rising edge of the reference clock regardless the state of the `INSTR_ACK` signal, causes all remaining instructions in the pipeline to fail while notifying the respective cores and resets the Input and Output Stream interfaces.

3.2.2 CIM Macro Block Status

The status signal `STATUS` indicates the current status of the CIM-MB. The status codes are defined as follows:

Status Code	Mnemonic	Description
`0x00`	`IDLE`	Indicates that the CIM-MB is ready to process a new micro-ISA instruction.
`0x01 to 0x2f`	*CIM-MB dependent*	**Busy states:** Status codes indicating different states of the CIM-MB while processing a micro-ISA instruction.
`0x30 to 0x4f`	*CIM-MB dependent*	**Recoverable error states:** Status codes that indicate the failure of a micro-ISA instruction. A new micro-ISA instruction may be issued to the CIM-MB without reset.
`0x50 to 0x6f`	*CIM-MB dependent*	**Non-recoverable error states:** Status codes that indicate the failure of a micro-ISA instruction. The CIM-MB needs to be reset before a new micro-ISA instruction may be issued.
`0x70 to 0xff`	*Reserved*	*Reserved for future use*

3.3 Data Stream Protocol

Data Stream Protocol sinks or sources streams of data from or to the shared memory to provide the CIM Macro Block with input data to process or store in non-volatile memory or to write back the results of an instruction. Table 3 and 5 list all the signals for input and output streams. Multiple Input and Output streams can be instantiated and the datawidth of the individual instances can be adjusted to any multiple of 32-bits according to the input- and output-bandwidth requirements of the CIM Macro Block. At this time, these design parameters remain to be evaluated.

Signal	Width [bits]	Direction	Description
`INPUT_DATA`	Multiple of 32	*input*	The data stream from the TCDM to be processed by the CIM-MP.
`INPUT_STRB`	width(`INPUT_DATA`)/8	*input*	Optional. Indicates which bytes in the data payload are valid (1=valid byte)
`INPUT_VALID`	1	*input*	Handshake valid signals (1=asserted).
`INPUT_READY`	1	*output*	Handshake ready signals (1=asserted).

Table 3 Data input stream signals

Each Interface instance corresponds to one *Stream Control Interface* instance which enables the CIM Macro Block to start new streams by providing the desired memory access scheme (see 3.4 for details).

Signal	Width [bits]	Direction	Description
OUTPUT_DATA	Multiple of 32	*output*	The data stream from CIM-MP to be stored in the TCDM.
INPUT_STRB	width(INPUT_DATA)/8	*input*	Optional. Indicates which bytes in the data payload are valid (1=valid byte)
OUTPUT_VALID	1	*output*	Handshake valid signals (1=asserted).
OUTPUT_READY	1	*input*	Handshake ready signals (1=asserted).

Table 4 Data output stream signals

The stream protocol uses a simple ready/valid handshaking protocol according to the following rules:

1. **A handshake occurs in the cycle when both INPUT_VALID and INPUT_READY are asserted.** The handshake is the "atomic" event after which the current payload is considered consumed by the consumer at the sink side of the Stream interface.
2. INPUT_DATA and INPUT_STRB can change their value either **a)** when INPUT_VALID is deasserted, or **b)** in the cycle following a handshake, even if INPUT_VALID remains asserted. **In other words, valid data payloads must stay on the interface until a valid handshake has occurred.**
3. **The assertion of INPUT_VALID (transition 0 to 1) cannot depend combinationally on the state of INPUT_READY**. On the other hand, the assertion of INPUT_READY (transition 0 to 1) can depend combinationally on the state of INPUT_VALID. This rule, which is modeled around the similar behavior used by TCDM memories is meant to avoid any deadlock in ping-pong logic.
4. The deassertion of INPUT_VALID (transition 1 to 0) can happen only in the cycle after a valid handshake. In other words, valid data produced by a source must be correctly consumed before INPUT_VALID is deasserted.

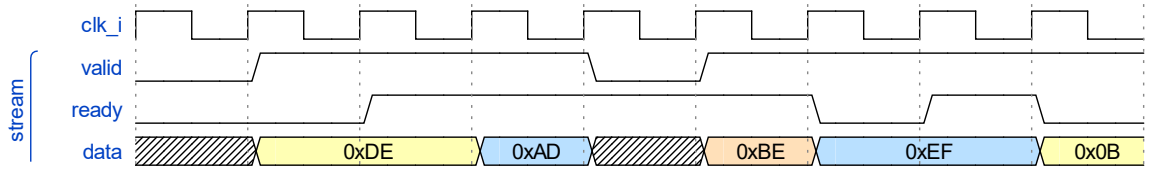


Figure 5 Example of a stream with a datawidth of 8bit. Valid handshakes happen in cycles 3,4,6, and 8

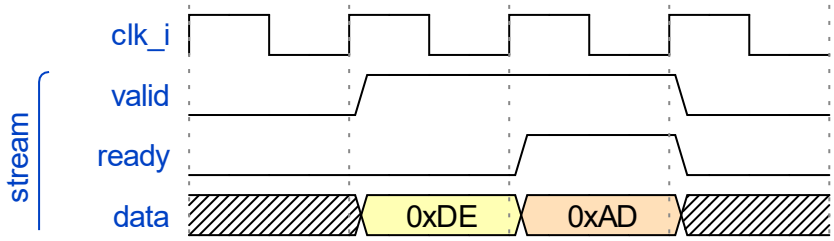


Figure 6 Incorrect handshake, not satisfying rule 2

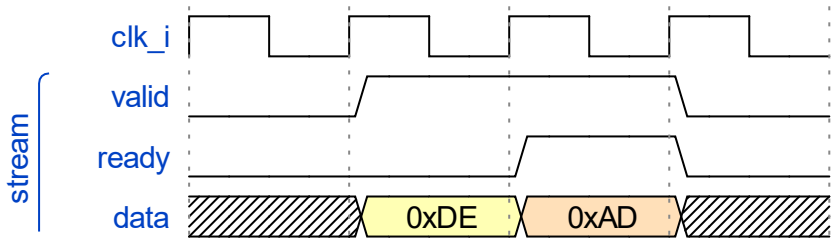


Figure 7 Incorrect handshake, not satisfying rule 4

3.4 Stream Control Protocol

The description of the Stream Control Protocol in this sub-section should be considered as a initial sketch rather than a specification. The details of the addressing schemes the CIM Accelerator will support are not fully developed yet. This interface is used to control a corresponding input or output stream interface by providing the parameters for a simple three-dimensional strided memory access scheme.

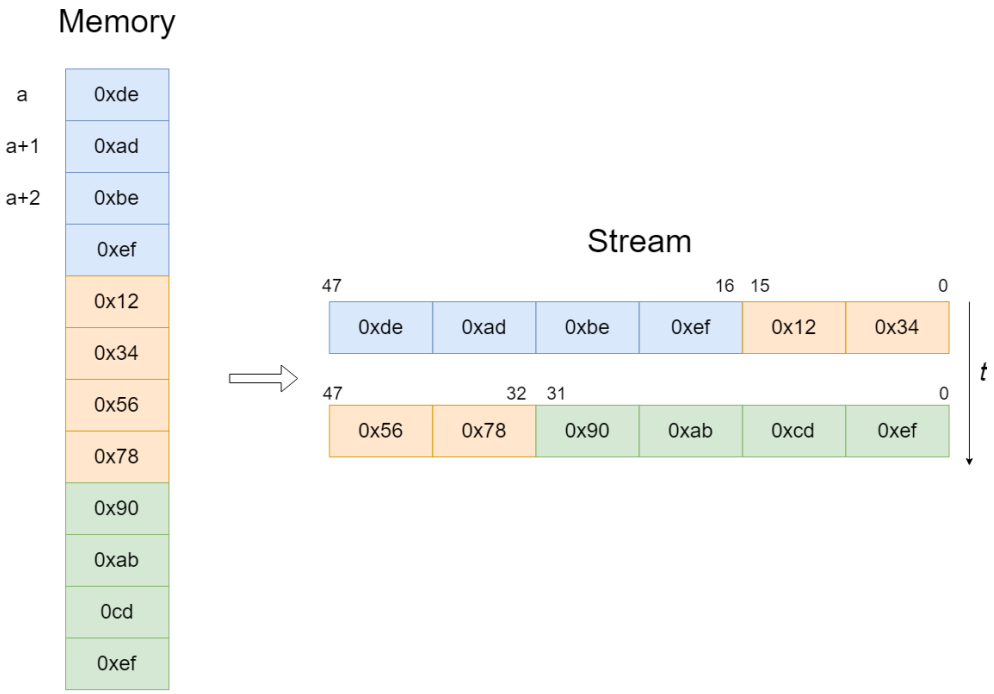


Figure 8 Byte order within a data stream of width 48 bit and an element size of 4 bytes.

The elements of the tensor are accessed sequentially starting from a *base-address*. Each element consisting of *element-size* number of bytes is separated by an *element-stride*, each line of *line-length* number of elements separated by a *line-stride* and each *feature* of *feature-size* number of lines separated by a feature stride. To initiate a new input- or output-stream the CIM Macro Block applies the new parameters (see Table 5) and pulls the `STREAM_CONFIG_WEN` line high for a single cycle. From the next rising edge of `REF_CLK` on the corresponding stream interface handles new values according to the new memory access scheme regardless whether the previous stream was fully consumed. This means that the CIM Macro Block is not required to consume the whole stream programmed and is free to configure new parameters at any time. The data is presented in little-endian byte order as indicated in Figure 8.

Signal	Width [bits]	Direction	Description
STREAM_CONFIG_WEN	1	*output*	Active-low write enable for the stream config parameters.
STREAM_BASE_ADDR	32	*output*	The start address of the stream. Does **not** have to be word-aligned.
STREAM_ELEM_SIZE	2	*output*	Each element consists of n number of bytes, where $n = 2^{STREAM\_ELEM\_SIZE}$
STREAM_ELEM_STRIDE	16	*output*	The distance between two elements within the same line in bytes.
STREAM_LINE_LENGTH	16	*output*	Number of elements in a line.
STREAM_LINE_STRIDE	16	*output*	Distance between two adjacent lines in bytes.
STREAM_FEATURE_SIZE	16	*output*	Number of lines in a feature.
STREAM_FEATURE_STRIDE	16	*output*	Distance between two adjacent features in bytes.
STREAM_FEATURE_COUNT	16	*output*	Number of features to process.

Table 5 Stream Control Signals

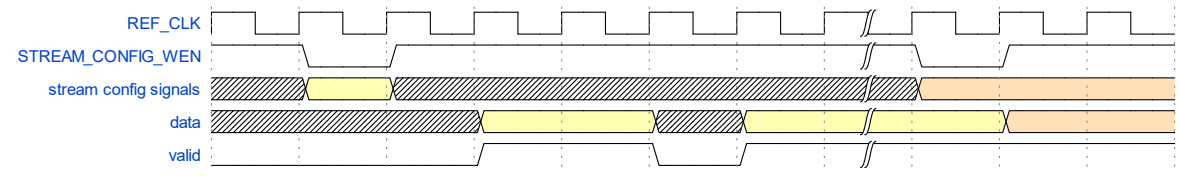


Figure 9 Configuration of three streams. Note that the next valid data word of the stream after STREAM_CONFIG_WE has been asserted already belong to the next stream.

3.5 Static Configuration Protocol

Configuration or interaction with the CIM Macro Block that is not related to the execution of a micro-ISA instruction is performed via the static configuration interface. This interface directly routes read- and write-requests from a core to the CIM Macro Block. Each consecutive address relates to a single 32-bit register. This contrasts with the CIM Accelerator's Peripheral Interface where byte-level addressing with word-aligned addresses are used.

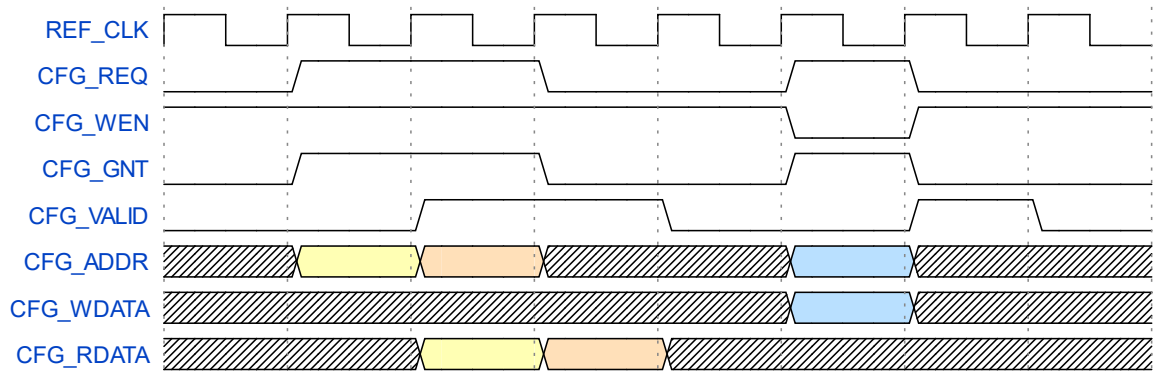


Figure 10 Two consecutive read- and a write-request on the static configuration interface

Table 6 lists all signals related to the interface and figure TODO illustrates the handshaking. For write-requests the CIM Macro Block must assert `CFG_GNT` signal as soon as it latched `CFG_ADDR` and `CFG_WDATA`. The interface logic is then free to change the address and write data with the next rising edge of `REF_CLK`. The interface is very similar to the TCDM Interface and the same handshaking rules apply. Especially the rule that **all transactions must be closed by asserting `CFG_VALID` for a single cycle**.

Signal	Width [bits]	Direction	Description
`CFG_REQ`	1	*input*	Handshake request signal (1=asserted).
`CFG_WEN`		*input*	Active-low read enable (1=read, 0=write).b
`CFG_GNT`	1	*output*	Handshake grant sinals (1=asserted).
`CFG_VALID`	1	*output*	For reads: rdata is valid. For writes: wdata was written to destination.
`CFG_ADDR`	8	*input*	Address of the targeted configuration register.
`CFG_WDATA`	32	*input*	Data to write to the configuration register.
`CFG_RDATA`	32	*output*	Read data from the configuration register.

Table 6 Signals of the static configuration interface

4. CIM Accelerator Interface

This section describes the interface between the Cluster and the whole CIM Accelerator.

4.1 TCDM Interface

CIM-Accelerators are connected to external L1/L2 shared-memory by means of a simple memory protocol, using a request/grant handshake. The protocol used is called Tightly-Coupled Data Memory (TCDM) protocol, and it is the same as the one used by cores and DMAs operating on memories. It supports neither multiple outstanding transactions nor bursts, as the accelerators are designed to be closely coupled to memories.

The TCDM protocol is used to connect a master to a slave. Table 7 reports the signals used by the TCDM protocol.

Signal	Width [bits]	Direction	Description
req	1	*Master → Slave*	Handshake request signal (1=asserted).
gnt	1	*Slave → Master*	Handshake grant signal (1=asserted).
add	32	*Master → Slave*	Word-aligned memory address.
wen	1	*Master → Slave*	Write enable signal (1=read, 0=write).
be	4	*Master → Slave*	Byte enable signal (1=valid byte).
data	32	*Master → Slave*	Data word to be stored.
r_data	32	*Slave → Master*	Loaded data word.
r_valid	1	*Slave → Master*	Response valid (1=asserted).

Table 7 TCDM protocol signals

The handshake signals req and gnt are used to validate transactions between masters and slaves. Transactions are subject to the following rules:

5. **A valid handshake occurs in the cycle when both req and gnt are asserted**. This is true for both write and read transactions.
6. **Every transaction is completed with the `r_valid` signal being asserted for one cycle**. In the case of read requests, the asserted `r_valid` indicates that the requested data is now provided at `r_data`. For write transactions the asserted `r_valid` signal indicates the completion of the write request. In this case `r_data` contains invalid data.
7. **The assertion of `req` (transition 0 → 1) cannot depend combinationally on the state of `gnt`.** On the other hand, the assertion of gnt (transition 0 → 1) can depend combinationally on the state of req (and typically it does). This rule avoids deadlocks in ping-pong logic.

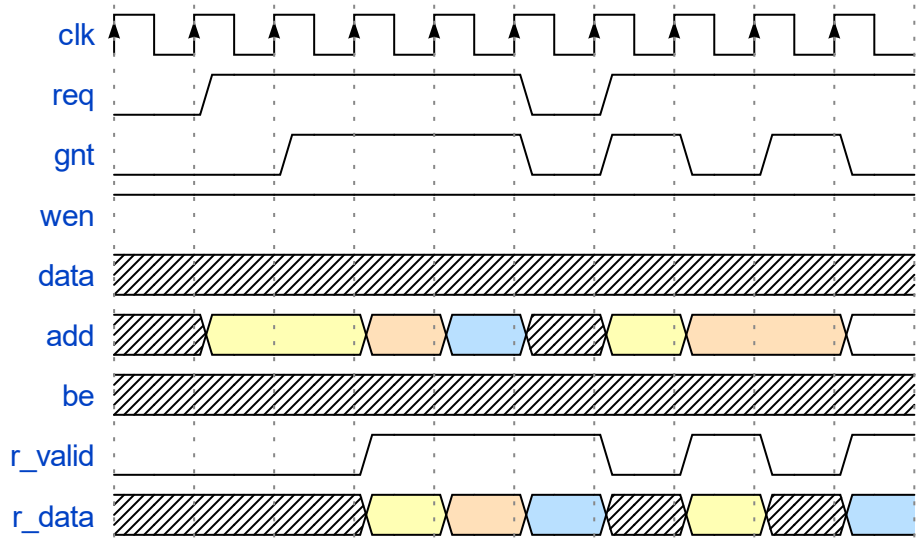


Figure 11 Multiple TCDM read-requests

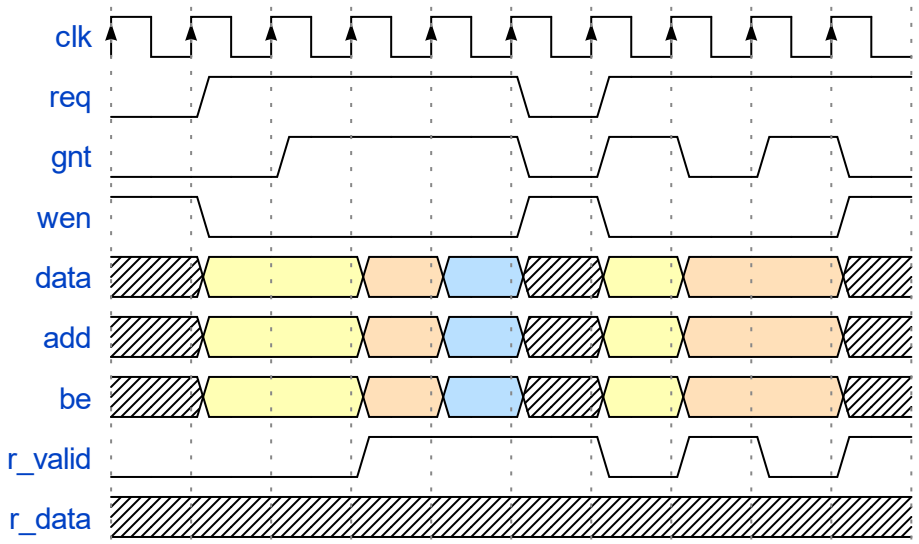


Figure 12 Multiple TCDM write requests

4.2 Peripheral Interface

To enable control of the CIM Accelerators, they typically expose a slave port to the peripheral system interconnect (see 2.2). The slave port follows an extension of the TCDM protocol which we can call PERIPH. The PERIPH protocol is the same exposed by most peripherals in a PULP system and used by the GP cores to communicate with them.

Signal	Width [bits]	Direction	Description
req	1	*Master → Slave*	Handshake request signal (1=asserted).
gnt	1	*Slave → Master*	Handshake grant signal (1=asserted).
add	32	*Master → Slave*	Word-aligned memory address.
wen	1	*Master → Slave*	Active-low write enable signal (1=read, 0=write).
be	4	*Master → Slave*	Byte enable signal (1=valid byte).
data	32	*Master → Slave*	Data word to be stored.
id	ID_WIDTH	*Master → Slave*	ID used to identify the master (request)
r_data	32	*Slave → Master*	Loaded data word.
r_valid	1	*Slave → Master*	Valid loaded data word (1=asserted).
r_id	ID_WIDTH	*Slave → Master*	ID used to identify the master (reply).

Table 8 PERIPH protocol

The PERIPH protocol is distinguished by the TCDM protocol by the `id` and `r_id` side channels. They are used in load operations issued through a PERIPH interface: the `id` identifies the master during the request phase, is buffered by the slave peripherals and accompanies the response phase as `r_id`. In this way, multiple masters can distinguish which traffic is related to themselves.

4.3 Configuration Registers

The CIM Accelerator exposes a set of configuration registers via the Peripheral Interface to the GP cores. The configuration registers can be separated into three different categories:

- **status and control registers** The status and control registers are used for synchronizing the accesses from different GP cores to the same CIM accelerator and also allow to track the progress of a micro-ISA instruction. These registers are not managed by the CIM-MB but by the interface logic around it.
- **Instruction Dependent Registers** These registers contain all the necessary information for the CIM-MB to process the next micro-ISA instruction. Even though the CIM-MB defines the configuration registers needed for micro-ISA instruction processing the register file is managed by the interface logic which queues multiple micro-ISA instructions internally and exposes one set of the *Instruction Dependent Registers* at a time to the CIM-MB with the Instruction Offload Protocol.
- **static CIM configuration registers** These registers contain long-lived configurations and are completely managed by the CIM-MB. The interface logic forwards requests to these registers to the CIM-MB via the Static Configuration Protocol.

Figure 13 illustrates the structure of the configuration register addresses. Each address is word aligned, therefore the 2 least significant bits are always zero.

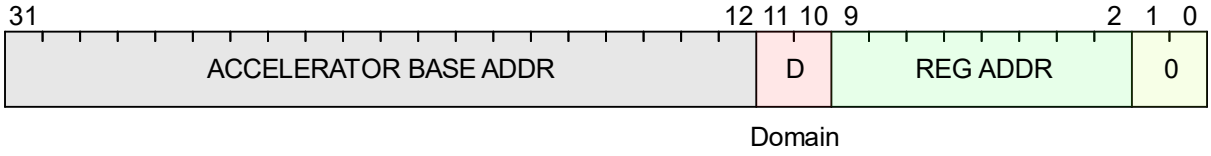


Figure 13 Address structure for communication with a CIM Accelerator

- **ACCELERATOR BASE ADDR:** Unique address assigned to each accelerator.
- **Domain (D)**:

Value	Description
0b00	Status Control Register Domain
0b01	Instruction Dependent Register Domain
0b10	Static Configuration Register Domain
0b11	*Reserved for future use*

4.3.1 Status and Control Registers

These registers are used by the GP cores to lock the CIM Accelerator during the instruction offloading sequence to queue a new micro-ISA instruction, track the status of the CIM Accelerator or in order to reset the accelerator to a known good state.

Table 9 lists all registers that are currently defined:

REG ADDR	Name	R/W	Description
0x00	TRIGGER	wo	Offload the prepared micro-ISA instruction to the micro-ISA instruction queue.
0x01	ACQUIRE	ro	Acquire the lock to prepare and offload a new micro-ISA instruction.
0x02	FINISHED_INSTRUCTIONS	ro	Returns the number of concluded jobs since last read.
0x03	STATUS	ro	Returns the status of the CIM Accelerator. See 3.2.2 for details.
0x04	RUNNING_INSTRUCTION	ro	Returns the ID of the currently running micro-ISA instruction.
0x05	SOFT_CLEAR	wo	Resets the CIM Accelerator to a known good idle state.
0x06 to 0xff	*reserved*		*Reserved for future use*

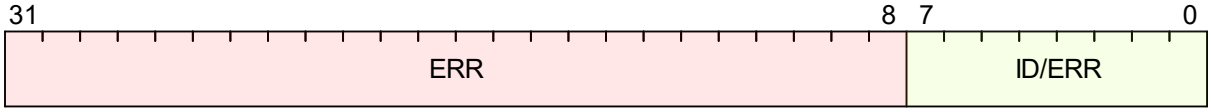
Table 9 Currently defined status and control registers

TRIGGER Register

Write-only register; any write to this register will close the current offload phase by releasing the micro-ISA instruction offload lock and inserting the currently offloaded micro-ISA instruction in the control queue.

ACQUIRE Register

Read-only register; any read to this register has the “side effect” of initiating an offload sequence by acquiring the micro-ISA instruction offload lock. Until the offloading core releases the lock by writing to the TRIGGER register, no other core can start a micro-ISA instruction offload.



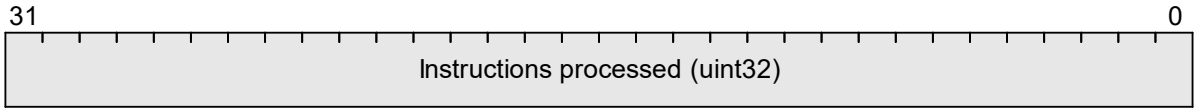
A read to the ACQUIRE register can return:

- the id of the micro-ISA instruction to be offloaded (a number from 0 to 255)
- an error code if one of the following conditions apply:
  a. the micro-ISA instruction offload lock has already been acquired:
  answer: 0xfffffffe (-2)
  b. the micro-ISA instruction queue is full:
  answer 0xffffffff (-1)

Bitfield	R/W	Description
31:8	ro	Error code
7:0	ro	ID of the offloaded micro-ISA instruction or part of the error code.

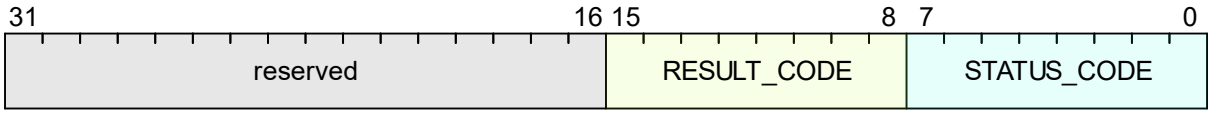
FINISHED_INSTRUCTIONS Register

Read-only register containting the number of micro-ISA instructions that the CIM Accelerator finished processing since the last read to this register or the last reset/soft clear.



STATUS Register

The STATUS_CODE field within the read-only STATUS register contains the status currently reported by the CIM Macro Block and the status code corresponding to the most reccent processed micro-ISA instruction. Cores interrupted by the CIM Accelerator should consult the RESULT_CODE field of this register to verify the successful completion of the micro-ISA instruction it issued.



Bitfield	R/W	Description
31:16	ro	*reserved*
15:8	ro	Status code reported with the (possibly unsuccessful) completion of the last micro-ISA instruction
7:0	ro	Status code currently reported by the CIM Macro Block.

PENDING_INSTRUCTION Register

Read-only register containing the ID (see ACQUIRE Register) of the instruction that is currently processed.

SOFTCLEAR Register

Write-only register. Any write to this register regardless the actual data written causes the CIM Accelerator to Soft Clear. See 3.1 for details on the Soft Clear functionality.

4.3.2 Instruction Dependent Registers

The micro-ISA instruction dependent configuration registers are managed by the interface logic around the CIM-MB. The register file is replicated `INSTRUCTION_QUEUE_DEPTH` times and the CIM-MB has read access to the register file corresponding to the currently processed micro-ISA instruction. This makes it possible to prepare the next instruction while the CIM Accelerator is busy processing the previous one.

REG ADDR	Name	R/W	Description
`0x00 to 0xff`	*reserved*	-	*To be specified*

Table 10 Instruction Dependent Registers

4.3.3 Static Configuration Registers

Long-lived configuration data (e.g. offset correction data, debug mode etc.) is stored in the CIM Configuration registers within the CIM-MB. This registers TODO

REG ADDR	Name	R/W	Description
`0x00 to 0xff`	*reserved*	-	*To be specified*

Table 11 Static Configuration Registers