Project:	ISOLDE: customizable Instruction Sets and Open Leveraged Designs of Embedded riscv processors
Reference number:	101112274
Project duration:	01.05.2023 - 30.04.2026
Work Package:	WP2: Open-Source Foundation Cores
Deliverable	**D2.2**
Title	**Intermediate report on foundational IP cores**
Type of deliverable:	Report
Deadline:	31.10.2024
Creation date:	31.10.2024
Authors:	Nils Wessman, GSL Rafael Tornero, UPV Carles Hernández, UPV André Sintzoff, TDIS Holger Blasum, SYSGO Esther Soriano, FEN Mihai Munteanu, Honorius Galmeanu (FotoNation, former XPERI) Matteo Perotti, ETHZ Catalin Ciobanu, Alexandru Puscasu (IMT) Sylvain Girbal, Daniel Gracia Pérez (TRT) Mladen Berekovic (UZL)
Involved grant recipients:	Frontgrade Gaisler AB (GSL) Universitat Politècnica de València (UPV) Thales DIS France SAS (TDIS) SYSGO GmbH (SYSGO) Fent Innovative Software Solutions S.L. (FEN) FotoNation SRL (former XPERI) Eidgenoessische Technische Hochschule Zuerich (ETHZ) National Institute for Research and Development in Microtechnologies (IMT) Thales Research & Technology (TRT) Universität zu Lübeck (UZL)
Contacts:	André Sintzoff, TDIS, andre.sintzoff@thalesgroup.com

Table of Contents

1 Executive Summary

This document describes the work performed within ISOLDE WP2 Open-Source Foundation Cores. It is the update of document deliverable D.2.1 (M6). WP2 is developing IPs that will be delivered through internal repositories during the work and finally through the ISOLDE virtual repository. WP2 progress will be reported through update of this report that will be issued as document deliverable D2.3 (M33).

The requirements and specifications for the work to be performed is established in WP1. At the time of issuing this report, much of the work to be carried out in WP2 is now in the implementation stage.

2 Introduction

2.1 Scope

This document provides a description and progress report on the work performed in WP2. The development in WP2 is based on the specifications and requirements generated in WP1. Some parts of the technical descriptions from document D1.2 [AD02] and document D1.4 [AD03] are reused in the work descriptions in this report to make this document self-contained.

The document organised with main headings with the same names and in the same order as the tasks within WP2 [AD01]. The current development status is provided as subsections for each task with an overview of the work to be performed and the current status.

Progress will continue to be reported in updates of this document that will be issued as the D2.3 deliverables.

2.1.1 Applicable documents

Internal code / DRL	Reference	Issue	Rev.	Title
AD01				Grant Agreement Project 1011112274 - ISOLDE
AD02	D1.2	1	0	Requirements and specifications on architecture, hardware and software modules and IPs
AD03	D1.4	1	0	Consolidated requirements and specifications on architecture, hardware and software modules and IPs

2.1.2 Reference documents

Internal code / DRL	Reference	Issue / Rev.	Date	Title
RD1	CV-X-IF	1.0		Core-V eXtension interface (CV-X-IF) specification: https://github.com/openhwgroup/core-v-xif/releases/tag/v1.0.0

2.1.3 Definitions and Acronyms

Acronym	Definition
AHB	Advanced High-performance Bus
AIA	Advanced Interrupt Architecture
APB	Advanced Peripheral Bus
APLIC	Advanced Platform-Lecel Interrupt Controller
AXI	Advanced eXtensible Interface
BSP	Board Support Package
CFI	Control Flow Integrity
CLIC	Core-Local Interrupt Controller
COP	Call-Oriented Programming
CSR	Control and Status Register
CVA6	Core-V Application 6 stages processor
CV-XIF	Core-V eXtension InterFace
EDA	Electronic Design Automation
JOP	Jump-Oriented Programming
L2C	Level 2 Cache
MSI	Message-signaled interrupt
NoC	Network on Chip
NOEL-V	(Not an acronym)
NTT	Number Theoretic Transform
PQC	Post Quantum Cryptography
PWM	Pulse Width Modulation
ROP	Return-Oriented Programming
RTL	Register-Transfer Level
SoC	System on Chip
XNG	XtratuM Next Generation

3 Processor IP: Performance analysis, safety, and improvements

In this section, we describe the work on processor cores improvements, including extensions (e.g. for safety and security purposes) and core configurability (to support the various demonstrator needs). For each task, a short description and the status about the work already done is provided.

3.1 NOEL-V processor extensions (GSL)

The NOEL-V processor will be extended with security features defined in the Control-Flow Integrity (CFI) RISC-V extension. Work will also be performed towards supporting a trusted execution environment and the cryptography Extensions.

3.1.1 Support for shadow stack (Zicfiss) and Landing pad (Zicfilp) RISC-V extensions.

The RISC-V CFI extension has been identified as a desirable security feature. This extension helps defend against both Return-Oriented Programming (ROP) and Call/Jump-Oriented Programming (COP/JOP) attacks, when code is compiled with tools that support these extensions.

ROP protection relies on special instructions to store return addresses in a protected shadow stack and checking these before using them. Currently the shadow stack protection is done via page tables, so it requires the availability and use of an MMU. During the development of Zicfiss, protection on the PMP (physical memory protection) level was considered, which would enable shadow stack protection in machine mode and on systems without an MMU - this may possibly return in a future update to the standard.

When COP/JOP protection is enabled, indirect jumps/calls require a matching "landing pad" as the first instruction at the destination. In its simplest form, all legal "landing pads" can look the same, but it is also possible to make use of a call graph to only allow specific caller/callee combinations. Unlike Zicfiss, Zicfilp is usable in all processor modes.

Current status

The Zicfilp and Zicfiss extensions were ratified in June 2024. Testing has not shown any problems with the current implementation in NOEL-V.

3.1.2 Support for RISC-V Cryptography extensions.

To support the increased demand for security, the RISC-V standard cryptography extension Zk is being implemented in the NOEL-V processor (Figure 1). This extension enables fast cryptographic operations by leveraging specialized hardware to accelerate key cryptographic algorithms, making it ideal for embedded systems with high security requirements. The development plan includes the implementation of the following sub-extensions within the Zk standard:

- NIST Algorithm (Zkn):
  - AES Encryption (Zknd)
  - AES Decryption (Zkne)
  - SHA-2 Functions (Zknh)
  - Bit Manipulation (Zbkb) – already implemented in NOEL-V
  - Carry-less Multiplication (Zbkc) – already implemented in NOEL-V
  - Cross-bar Permutation (Zbkx) – already implemented in NOEL-V
- Entropy Source (Zkr)
- Data Independent Execution Latency (Zkt)

Non-standard Zk instructions, such as those related to SM3 and SM4, have been excluded from the implementation plan.

To support these cryptographic instructions, a dedicated Crypto Unit has been integrated into the NOEL-V processor. This unit works in parallel with other functional units, such as the ALU and MUL, without stalling the pipeline. Crypto Unit is shared between both lanes, so issuing two cryptographic instructions in the same cycle is not allowed.

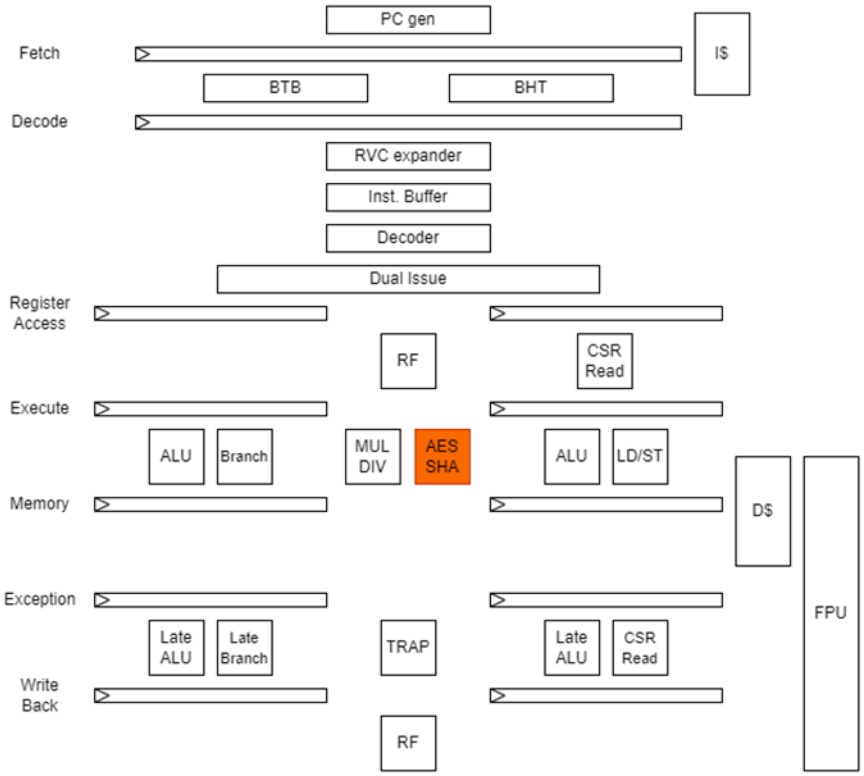

Figure 1: NOEL-V Pipeline Diagram Highlighting the Integrated Crypto Unit

To verify the implementation of this extension, official RISC-V Architectural Tests will be used to ensure full compliance and functionality of the cryptographic instructions implemented within NOEL-V. These tests cover:

- Register and immediate field coverage.
- Input patterns: single-bit, random, and byte-count.
- S-Box testing for AES.
- Forwarding and hazard checks in real-world sequences.
- Entropy source testing.
- Constant-time execution.

Current status

The control logic required for decoding, exception handling, and data forwarding has been added into the NOEL-V pipeline, ensuring smooth integration of the Crypto Unit into the processor architecture. Within the Crypto Unit, the logic for processing all SHA instructions has been fully implemented, while the logic for processing AES instructions is still under development.

To early test the SHA instructions implemented in the NOEL-V processor, C++ tests have been developed: one for the RV32 configuration and another for RV64. These tests execute each SHA instruction and compare the results against manually calculated expected values.

3.1.3 Extend NOEL-V processor system to support trusted execution environment.

In the context of the ISOLDE project the RISC-V standardization of “World Guard” will be followed and evaluated. Following this, NOEL-V should be updated to support this.

Current status

Currently no work has yet been done for this task.

3.1.4 Generate IP-XACT description for the NOEL-V subsystem.

To support integration of the NOEL-V processor system into the demonstrator design an IP-XACT description will be generated. This will make it easier to integrate the processor using EDA tools like Xilinx Vivado.

Current status

This work has been descoped due to less need for the demonstrator development then anticipated and custom IP-XACT definitions used by EDA tools.

3.2 CVA6 processor (TDIS)

The CVA6 processor will be extended to support post quantum cryptography extensions. To fulfil the different requirements expressed by the ISOLDE demonstrators, the CVA6 RTL will be configurable by enable/disable features.

3.2.1 Support for post quantum cryptography extensions

To support the increased demand for security due to quantum computing threat, support of post quantum cryptography is a must for the future. Additional instructions will be useful to support PQC algorithms standardised by NIST.

Current status

Since previous year, RISC-V International is no more primarily considering standardization of scalar instructions for PQC.

This task addresses Number Theoretic Transform (NTT) which is part of PQC Dilithium algorithm standardized by NIST.

The implementation will be scalar one. The specification of scalar instructions has been completed. Implementation in CVA6 pipeline is started.

3.2.2 Configurable RTL

The CVA6 RTL configurability is a mean to have cores adapted to the demonstrator needs. It will be possible to enable/disable and/or configure different features: e.g., privilege modes, extensions, cache configuration. By disabling some unused features, the resulting area and power consumption are reduced.

Current status

To manage RTL configurability, TDIS implemented a semi-formal flow that derives various configuration files and human-readable documentation from a single central specification of a system configuration (Figure 2).

Each specification is validated against formal and hand-coded consistency rules before generating the final configuration and documentation files.

The outputs of the flow are:

- an RTL parameter file that controls the CVA6 RTL configuration.
- a human-readable design specification of the instruction set and the CSR registers of the RTL configuration.

- a configuration file for the Spike simulator used in step-and-compare simulations of the given CVA6 configuration.

The flow consists of two stages:

- validation of a system configuration specification against the rules set forth in official RISC-V specifications. The output of this first stage is a consolidated specification containing reset values and documentation strings. It is also guaranteed as consistent with the official RISC-V ISA specifications.
- generation of RTL/simulator configuration files and generation of design documentation files from the consolidated specification. Currently, the CVA6 RTL configurability work is started, and some features are already made configurable.

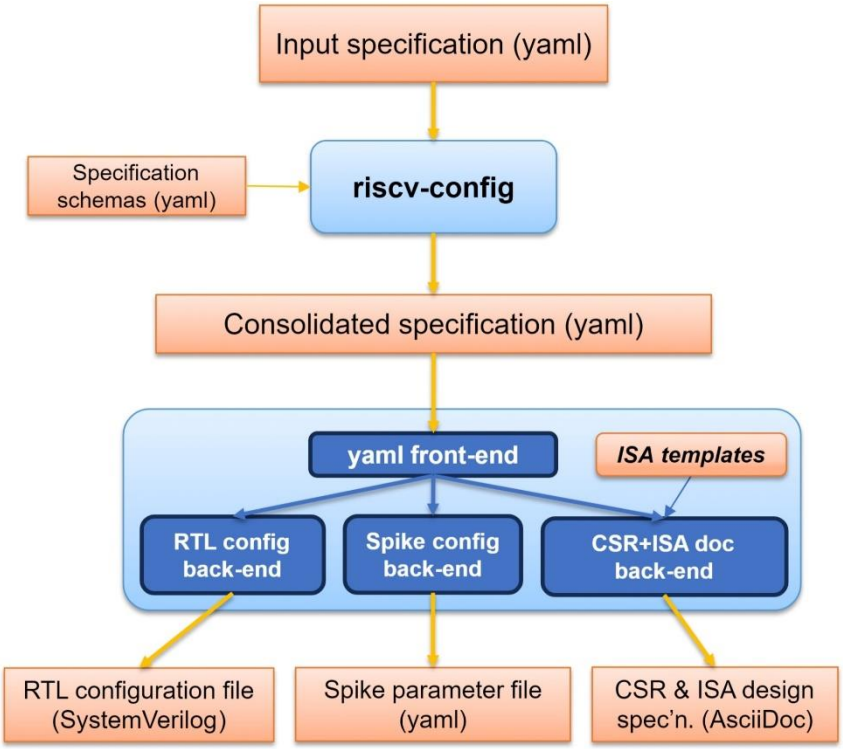

Figure 2: RTL Configuration Flow

Specification Representation and Validation

System configurations are described formally and validated using the `riscv-config` tool (https://github.com/riscv-software-src/riscv-config/) endorsed by RISC-V International.

Configuration specifications are expressed in Yaml and must comply with document schemas supplied with `riscv-config`.

The specification of a system configuration consists of four components:

- an ISA specification that defines the instruction set architecture, the supported privilege levels, the CSR set, any standard ISA extensions and any implementation-defined properties defined in the official RISC-V ISA specifications.
- a platform specification that captures the information about memory-mapped CSRs and execution entry point.
- a debug specification that captures the properties of the debug unit (if present).
- a custom specification that can capture any features outside the scope of RISC-V ISA specification documents.

The ISA specification format supports the description of an arbitrary number of harts, thus enabling the representation of heterogeneous multi-core configurations.

Document schemas of the four specifications provide the sets of valid values and the default values for all properties that are expressible according to the given schema. The sets of legal values in the schemas strictly follow the official RISC-V ISA specifications, including the applicable versions of the specification where needed. The default documentation strings of all properties are an integral part of the schemas.

Each input specification can restrict the sets of valid values and can override the default values from the schema to match the needs of the system being described. Default documentation strings can also be overridden to tailor their content to the needs of a configuration.

The `riscv-config` tool transforms a set of input specifications into a set of consolidated output specifications in which:

- all properties that were implicitly left at default values in the input specification are explicit and have a defined value.
- reset values of all CSR registers are defined and guaranteed compliant with the input specification and the corresponding schema.
- valid value ranges of properties are guaranteed compliant with the input specification and the corresponding schema.

To validate an input specification, the tool checks the values and value ranges in that specification against the corresponding schemas and checks compliance to explicitly coded rules such as dependences between standard ISA extensions. If any of the checks fail, the nature of the failure is reported to the user and the tool stops without generating output specifications.

To fully capture RTL configurations used in the ISOLDE project, the original platform specification schema of `riscv-config` was extended to support arbitrary memory maps, including memory region properties such as idempotence and cachability. The modified `riscv-config` tool is maintained in a forked copy of the official repository. It is integrated into the CVA6 repository by means of "vendorization" (see https://opentitan.org/book/util/doc/vendor.html) under https://github.com/openhwgroup/cva6/tree/master/vendor/riscv/riscv-config.

The custom specification component is used to capture micro-architecture properties of system configuration. While there is a generic support for architectural signals in the corresponding `riscv-config` schema, it is expected that new, dedicated schema elements will be needed to provide a higher-level abstraction of microarchitectural properties. The study of the appropriate abstractions is currently under way.

A public example of input and output specifications for a simple CV32A65X system configuration, together with a matching Makefile is available in the CVA6 GitHub repository under https://github.com/openhwgroup/cva6/tree/master/config/riscv-config.

Generation of design documentation and configuration files

In the ISOLDE project TDIS developed a set of scripts which take the set of consolidated specifications as input and produce:

- a SystemVerilog RTL parameter file that is an integral part of the CVA6 RTL instantiation.
- a Yaml configuration file that controls the operation of the Spike simulator to reflect the RTL configuration.
- a design specification document covering all instructions supported (legal) in the current RTL configuration.
- a design specification document covering all CSR registers present in the input specification.

Each of these outputs requires a combination of information from several specification components. In addition, instruction set documentation uses templates providing a natural language description for all instructions supported in the RISC-V specifications. These templates can be tailored to the needs of specific projects. The scripts producing the various kinds of outputs share a common Yaml front-end library in charge of loading consolidated specifications but use separate backends, each tailored towards the type of information to be produced.

Current scripts and public examples of the output generated from CV32A65X specifications mentioned above is available in the CVA6 GitHub repository under https://github.com/openhwgroup/cva6/tree/master/config/gen_from_riscv_config/.

3.3 Testing Design Parameters for CVA6 (UZL)

Creation of CVA6 of different multicore architectures by system-on-chip generator tools. For this purpose, various design parameters are systematically created according to the specifications of T1.3/T1.4 and compared with each other using various design metrics (computing power, power consumption, etc.) and the suitable design parameters are identified, targeting different use cases.

3.3.1 Setup of SoC Generator and CVA6 Configuration

The CVA6 RISC-V Core is embedded in a system-on-chip design, which is generated by specialized tools (OpenHW Group / OpenPiton). UZL has examined different FPGA target platforms (Digilent Gensys2, AMD/Xilinx VCU118), based on the requirements of the demonstrators. The system-on-chip consists of the CVA6, bus-infrastructure, memory-controller and peripheral IPs. The evaluation consists of two main parts. First, it focuses on CVA6 RISC-V processor alone. Second, it evaluates the CVA6 Core together with the system-on-chip architecture, especially memory controller.

Current status

UZL selected Digilent Gensys2 and AMD/Xilinx VCU118 as the target FPGA platform and adapted the system-on-chip generator configuration accordingly. The platform setup is completed which builds the base for the benchmarks of Section 3.3.2. Based on the results of Section 3.3.2, the design parameters might be extended or updated to achieve better results.

3.3.2 Choice of design parameters and benchmarking

Various design parameters (clock frequency, ISA-Extensions, number of cores) for the CVA6 are chosen and tested according to the specifications of T1.3/T1.4. UZL uses well established benchmark suites, such as Dhrystone and CoreMark. The benchmarking process includes a design space exploration of various design metrics (computing power, power consumption, etc.) and the suitable design parameters are identified for the different use cases.

Current status

UZL currently focuses on CVA6-only benchmarks on the Digilent Gensys2 board building up a design space, which is further extended by the experiments with system-on-chip benchmarks and the other FPGA target platform (AMD/Xilinx VCU118).

4 Peripheral and interconnect IP cores

Here, we provide the description of the main IP cores, including their firmware, that compose the catalogue of components that are usually required to build a System-on-Chip (SoC) suc-

cessfully. This catalogue contains infrastructure IP cores mainly. Thus, compute power (processor IPs) is beyond the scope of it. We have structured the catalogue in two groups: peripherals and interconnects.

4.1 Peripherals

4.1.1 GRLIB peripheral IPs (GSL)

GSL will provide the GPL version of the GRLIB IP library to support building the demonstrator SoCs. Required updates or extensions of the IP cores necessary for the demonstrators will be assessed in terms of feasibility and handled in support of the demonstrator platforms developed within ISOLDE project.

Current status

Latest release of the GPL version of GRLIB is available.

4.1.2 Improved L2C-lite (UPV)

GRLIB library from GSL includes a share-level cache module (L2C-lite) licensed as GPL to improve the performance of the open-source multicore architectures that can be built with this library. This cache module has an AHB interface as the front-end (to connect the cores) and supports both AXI and AHB as a back end to interconnect with memory or upper cache levels.

In the context of ISOLDE, the L2C-lite module has been improved to increase the level of parallelism supported. Thus, increasing the performance of the open-source multicores that can be built with GRLIB. At the same time, we have also extended the replacement policies of the L2C-lite module to support cache partitioning. This additional support will benefit those use-case needing to have time-predictable behaviour since it removes the inter-task interferences that will otherwise occur in the L2C-lite module.

Current status

Functionality implemented and verified. The modifications are already available in the L2C-lite module of the SELENE platform (https://gitlab.com/selene-riscv-platform/selene-hardware).

4.1.3 Timer (IFX)

The timer module shall support the following features

1. Watchdog
2. Clock
3. Time interval measurement
4. PWM signal generation

The interface may be alternatively via memory mapped registers or via CSRs or both. In that sense the timer may be an own IP-module or a RISC-V sub-module, later potentially merged with existing performance counter functionality.

Memory mapped registers may be accessible via AHB or via APB (decision pending). The provisioning – e.g. multiplexing – of control signals shall be done outside the timer.

The timer should have several timer channels, each with a size constrained counter, an optional number of same size capture compare units (CCUs) and a control module. Mapping of counter, CCU and control bit-fields to registers is not yet defined.

Current status

Requirements capture and concept for timer finished. Second version of prototype ready, simulated and validated on FPGA. Formal verification ongoing.

4.1.4 Interrupt Controller (IFX)

The interrupt controller shall support priorities, priority groups and the passing of the active/newly requesting interrupt number to the RISC-V core.

The interface may be alternatively via memory mapped registers or via CSRs or both. In that sense the timer may be an own IP-module or a RISC-V sub-module, later potentially merged with RISCV CLIC.

Memory mapped registers may be accessible via AHB or via APB (decision pending). The provisioning – e.g. multiplexing – of control signals shall be done outside the timer.

The number of interrupts supported – and thus implicitly the maximum number of priorities is under discussion. Mapping of configuration bit-fields and the interrupt status bitfields to registers is not yet defined. Direct provisioning of interrupt number to exception handler is in discussion, whereas the CLIC way of doing is preferred.

Current status

Requirements capture and concept for timer finished. Second version of prototype ready, simulated and validated on FPGA. Formal verification ongoing.

4.2 Interconnects

4.2.1 Wormhole NoC (UPV)

For SoC designs based on a Network-on-Chip (NoC) we envision a Tile-oriented design approach. Here, a Tile is an abstraction that holds one or more Intellectual Properties (IPs) and provides a common access interface and protocol to them. Through the NoC, the components located at different tiles will exchange messages in a unified manner. These IPs will interface to the NoC by means of a Network Interface (NI), which will expose different types of signal interfaces to satisfy the typical connections between two given components, like Initiator and Target. Thus, the NI will comply with AXI interfaces to connect external IP Cores to the network. The NoC router will support 2D mesh topologies. Overall, the NoC will provide the next features:

- Scalable on-chip architecture.
- Deadlock avoidance guarantee.
- Freedom of data losses.
- Broadcast and point-to-point communication primitives.
- Traffic partitioning.
- Quality of Service.
- High throughput.

Current status

We have completed the design of the architecture of the NI and the NoC Router which is available at https://github.com/UPV-GOS-NOC/euros2pronoc.The activities for implementing the AXI4 interface in NI have just recently started.

4.2.2 AXI traffic sniffer (UPV)

To enable traffic monitoring in a wide variety of SoCs, we provide and integrate a hardware module that monitors the network activity in AXI interconnects. This module is written in VHDL, and its function is to leverage source information in the network packets to obtain contention information between sources. Thus, this module tracks the timing contention between the different traffic sources on an AXI network for each given destination.

In some cases, such as traversing a cache, initiator information is obscured on the AXI network, thus making fine-grained contention monitoring impossible. To solve this issue, the AXI traffic monitor relies on request initiator information propagated using the 4 bits available in the AXI QoS field.

The contention tracked by this module is propagated with specific interfaces to a timing interference hardware monitor using the safeSU's CCS signalling mechanism developed by BSC in the context of WP3.

The AXI4 contention monitoring infrastructure monitors read and write contention by blaming the head of its respective channel queues for the delay it causes to the tails of the queues. It can also monitor cross-channel contention when one channel's requests are blocked by another channel's requests being processed.

Current status

This module has been completed and verified. A paper describing its functionality and integration with the SafeSU from BSC in the NOEL-V based SELENE SoC has been published at (https://www.sciencedirect.com/science/article/pii/S0167739X24004825).

4.2.3 Context-Aware BUS (CA-BUS) - TRT

The context-aware bus (CA-BUS) extends existing AXI-buses so they can integrate the context information associated to the requests for their processing by the context-aware performance monitoring counters (CA-PMCs) developed in WP3.

Furthermore, as part of the CA-BUS development, this work will study the transmission of the context-aware augmented requests through IPs in the path between buses, like bridges and caches that can transfer requests between buses.

Figure 3 shows an integration example of the CA-BUS alongside the other related IPs for the context-aware monitoring developed in WP2 (CA-BUS, CA-CORE) and WP3 (CA-PMC, CA-PMC-IF). In Figure 3 the CA-BUS BUS enriches the system bus with context-aware information about the running software.

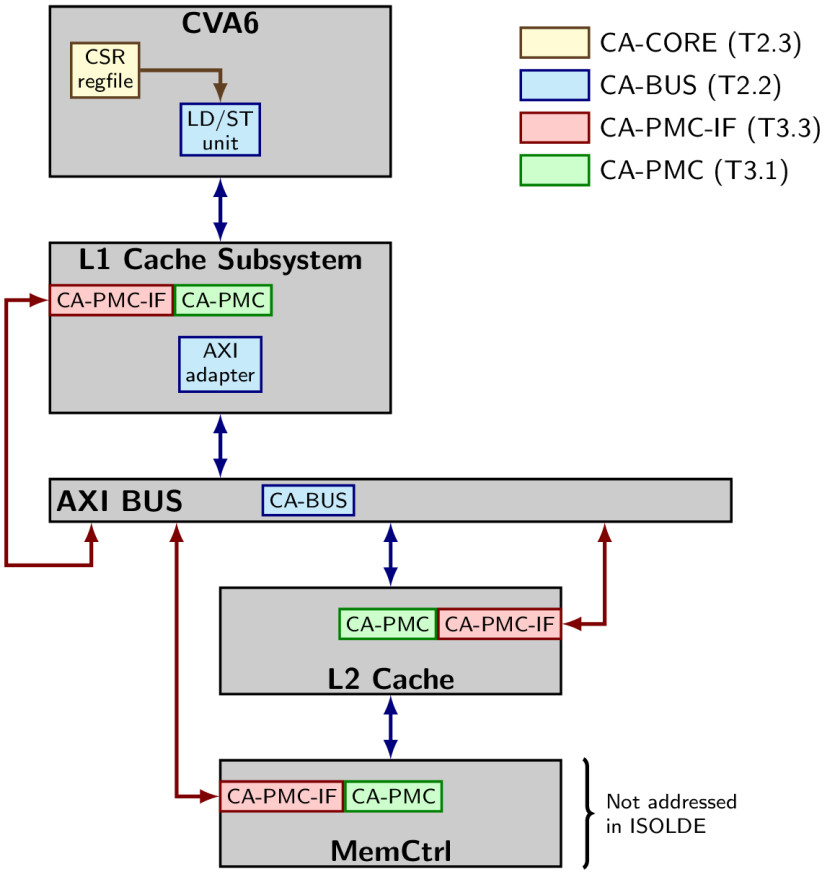

Figure 3: context-aware monitoring infrastructure

Current status

The current context is stored by the CA-CORE extension (from T2.3) as a specific CSR as part of the CVA6 CSR Register File. The purpose of the CA-BUS extension is to propagate this information alongside memory requests.

There are two level of memory requests, first the memory requests that are generated by the pipeline to the L1 cache subsystem, and second the memory requests that correspond to L1 cache misses and that are sent through the AXI bus toward the memory hierarchy (L2 caches and memory controllers).

The CA-Bus extension has therefore been implemented as two sub-extensions: The **ca-bus/core** extension responsible for forwarding the context information towards the L1 cache subsystem, and the **ca-bus/axi** extension transferring the context information alongside the AXI bus.

CA-BUS/core extension:

A pair of common interfaces `dcache_req_i_t` and `dcache_req_o_t` are already defined in cva6.sv for data request to/from the L1 cache subsystem. We extended this interface with a `cam_context` signal.

In the execution stage of the CVA6, the context information is obtained from the CSR register file as part of the Address Generation Unit of the Load/Store Unit, and transferred to the `cam_context` signal as part of both the load unit and the store unit.

CA-BUS/AXI extension:

Upon a cache miss in the L1 cache subsystem, a memory request is sent alongside the AXI bus towards the memory controller. The corresponding dcache_request is converted into its AXI counterpart as part of the AXI adapter module.

We modified the AXI adapter module so that the cam_context is propagated as part of the AXI user interface (`axi_wr_user` and `axi_rd_user`) for write and read requests respectively, so far avoiding modifying the AXI API.

4.2.4 AHB Bridge (IFX)

The AHB crossbar prioritizes and multiplexes different initiators' and responders' read and write requests. Burst transfer will not be supported. *Please note "master" and "slave" has been widely used, fortunately more and more these terms are replaced. "initiator" and "responder" are used here instead. As these terms are not fully aligned, we strive for Isolde consistent names.*

Current status

Requirements capture and concept for timer started. Second prototype ready, simulated and validated on FPGA. Complete formal verification ongoing.

5 Common extension interfaces

In this section we describe the work on extending the processor cores to integration accelerators and support context aware performance counters. A short description on the different tasks will be provided together with an update on the progress currently made.

5.1 Collaboration on a common coprocessor/accelerator interface

Within the scope of the project, ISOLDE partners together with partners from the TRISTAN project and OpenHW Group has put effort into updating the Core-V eXtension interface (CV-X-IF) specification. This has resulted in the release of a ratified version 1.0. This specification defines a RISC-V extension interface that provides a generalized framework suitable to implement custom coprocessors and ISA extensions for existing RISC-V processors. It features independent channels for accelerator-agnostic offloading of instructions and writeback of the result(s). The ratified CV-X-IF release [RD1] is available at https://github.com/openhw-group/core-v-xif/releases/tag/v1.0.0.

Both foundational cores, CVA6 and NOEL-V, has been extended to implement version 1.0 of the CV-X-IF. This enables the use of a common extension interface for tightly coupled accelerators. Multiple ISOLDE accelerator development has evaluated and selected to adapt the processor interface to CV-X-IF. This will simplify the integration of the accelerator with the processor core in the demonstrator designs.

5.2 RISC-V accelerator interface for RISC-V core (IMT)

We design a tightly coupled accelerator and analyses multiple possibilities for this task. But after the research we decided to use COREV-Extension-Interface (CV-X-IF). We decided to use this interface because it is RISC-V standard interface, and it is supported by foundation cores (CVA6 and NOEL-V). Besides that, this interface allows access to internal registers, this allows small configuration data to accelerator.

To unload the core from memory operations we decided to use a dedicated AXI interface for the task. This interface is connected to the central bus and allows direct access to main memory.

Both interfaces we use are standard. In Figure 4 is presented the internal organization of our accelerator.

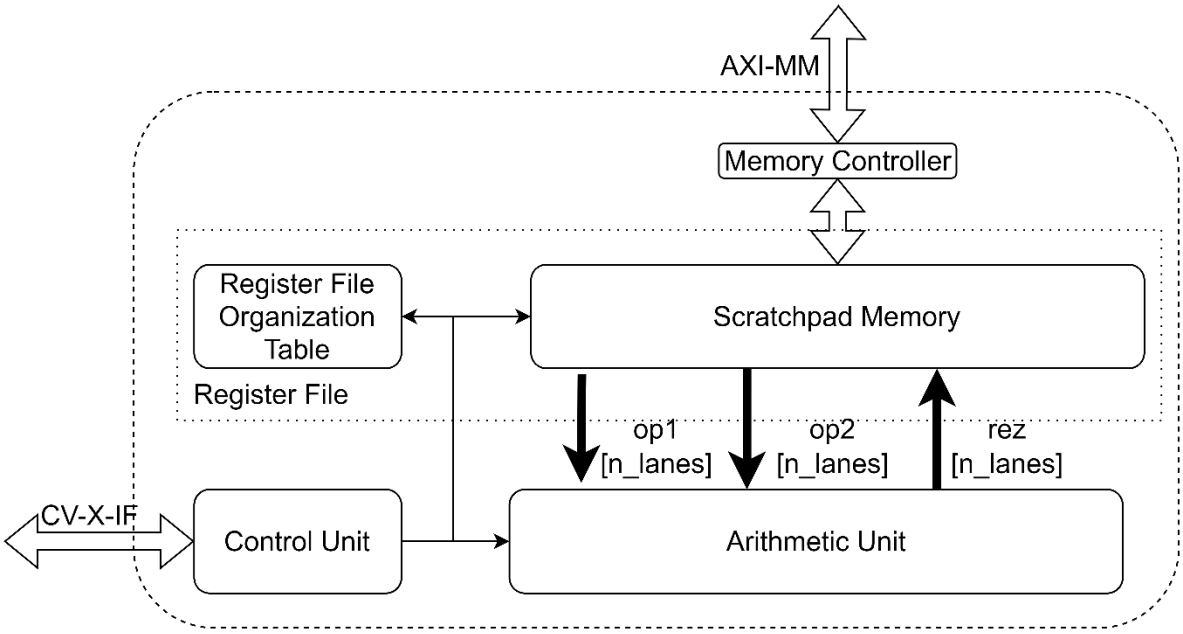

Figure 4: Internal organization of IMT accelerator

Current status

We defined RISC-V extension for the accelerator. The instructions were added to riscv-gnu-gcc and tested on riscv-isa-sim.

We planned the design of accelerator and define milestone and deadlines for development. The work for designed started for an initial version, that will support a reduced number of features.

5.3 Memory bank accelerator interface for RISC-V core – XPERI

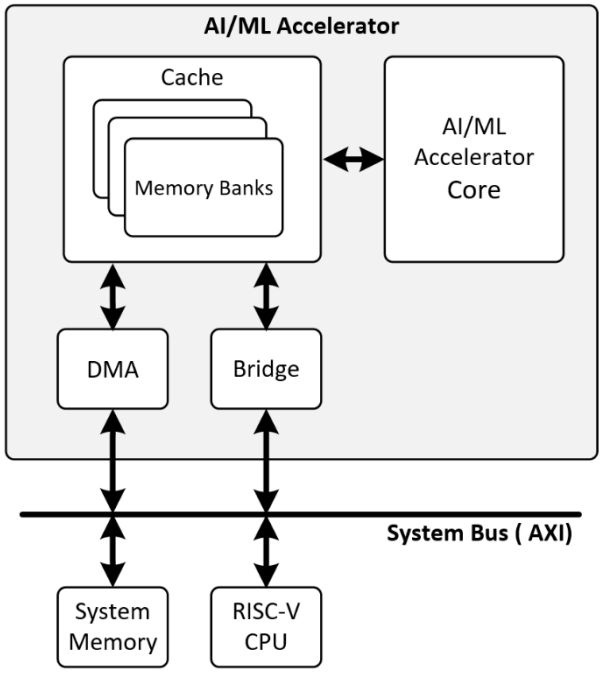

Figure 5: Memory bank accelerator interface

The purpose of the AXI to Memory banks bridge is to bypass costly transactions to and from the system memory. Many operations could be performed by RISC-V accessing directly the memory banks, without the need to transfer the data through the system memory (Figure 5).

For example, we consider a SoftMax operation that must be applied on the results of a previous operation. This result resides in the memory banks.

1. Without the bridge processing has the following steps:
    a. Use DMA to transfer data to the system memory. Transfers to and from the system memory can be very expensive operations in clock cycles and power, especially if an off-chip DDR memory is used.
    b. The RISC-V processes the data from the system memory and writes the results back to the system memory.
    c. Data is transferred back to the memory banks using DMA, so the accelerator can process it further.
2. The optimized approach uses the bridge and requires only two transfers (Memory Banks → RISC-V and then RISC-V → Memory Banks), instead of four transfers used previously.
    a. The CPU reads data directly from the memory bank and processes it.
    b. The CPU writes the results back to the memory banks.

The Bank Bridge allows the system CPU to directly access the memory banks of the AI/ML accelerator, by mapping the bank's address space into the processor address space. In the system the Bridge is an AXI slave, providing one read and one write AXI lite channels.

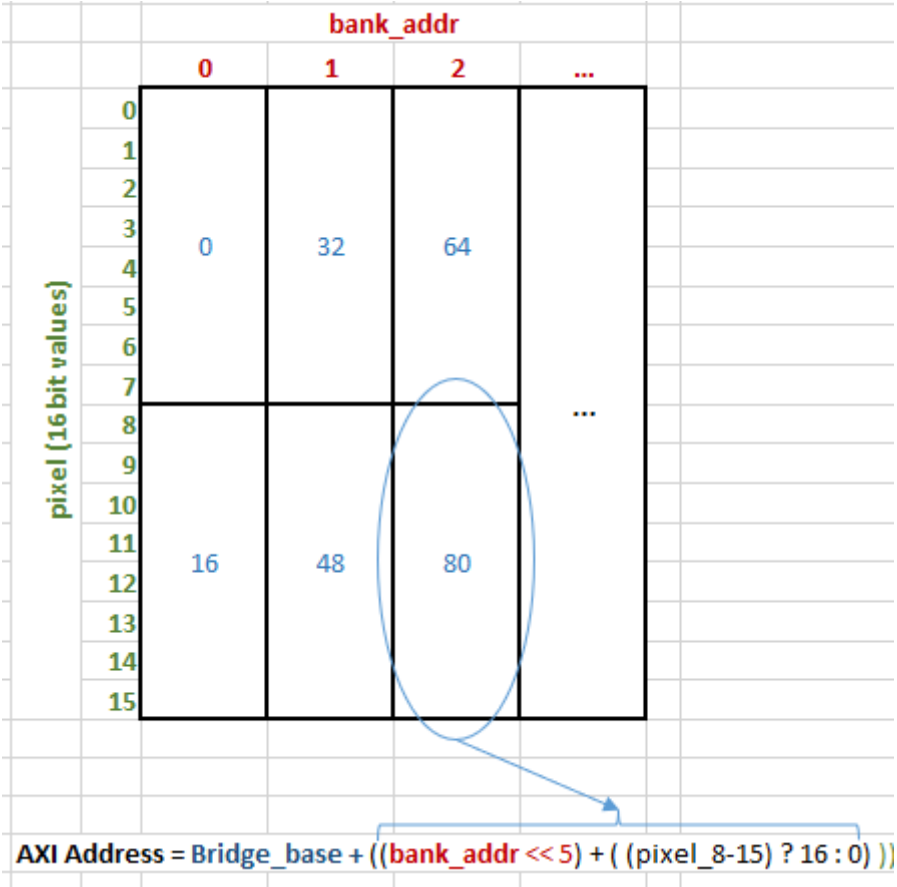

Figure 6: The AI/ML banks' addressing structure

The AI/ML Banks store 16 values (pixels), each one on 16 bit at each address. This means that 256 bits are stored at each bank address. The system AXI bus is 128-bit meaning that two

AXI transfers are needed to access all 16 values stored at one bank address. This is illustrated in Figure 6.

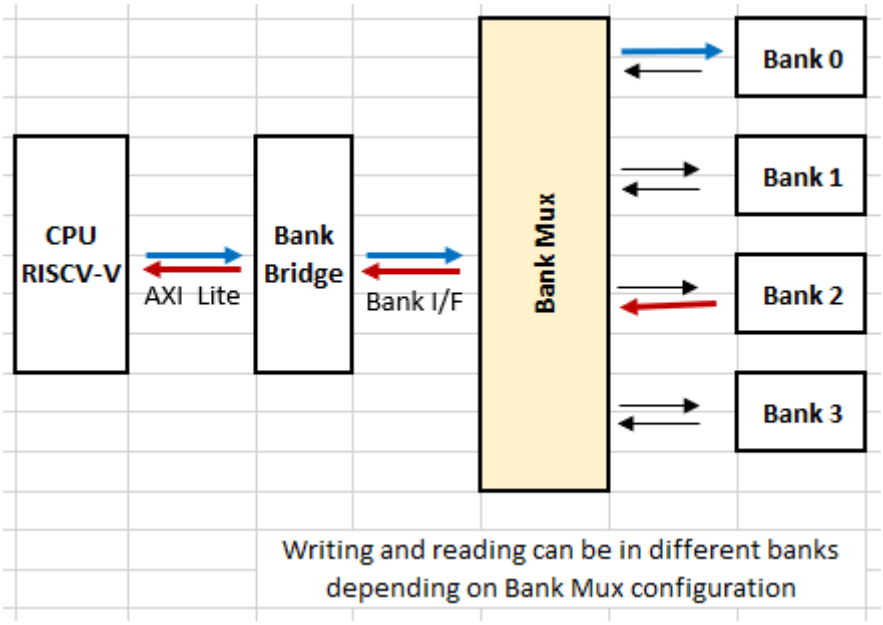

Figure 7: Memory bank accelerator interface

Figure 7 shows the system integration of the bridge. Only one bank address space is mapped into the CPU address space. A configurable bank multiplexer selects which bank is connected to the CPU. As the Bridge has two independent AXI channels, one for read, one for write, the bank multiplexer can be configured so that the CPU writes and reads from different banks.

Current status

Specification completed; RTL implemented. Verification ongoing (20%).

5.4 Context-Aware Core Extension (CA-CORE) - TRT

The context-aware core extension (CA-CORE) will upgrade the RISC-V CVA6 core to provide the context information required for the exploitation of the context-aware performance monitor counters (CA-PMC) developed in WP3.

The CA-PMCs will enable fine-grained monitoring controlled by the software, typically the OS or the Hypervisor, which will be capable of programming the CA-PMCs to monitor the incoming request based on their context. The CA-CORE extension extends the RISC-V CVA6 with the required mechanisms to define the context of the software currently executing in the core.

The CA-CORE extension adds a dedicated CSR to keep the context of the software currently executing in the core. The OS or hypervisor will change the value of the context CSR depending on the running application or VM. The CA-CORE extension uses the value in the CSR context to associate the context value in the CSR with all the load/store requests that core will issue. For that purpose, we modify load/store-unit to create requests with the context information. These context-aware requests are transmitted through the system with the CA-BUS (see Section “Peripheral and interconnect IP cores” - CA-BUS) to enable their processing in the CA-PMCs developed in WP3.

Figure 3 (see section 4.2.3 “Context-Aware BUS (CA-BUS) - TRT”) shows an integration example of the CA-CORE alongside the other related IPs for the context-aware monitoring developed in WP2 (CA-BUS, CA-CORE) and WP3 (CA-PMC, CA-PMC-IF).

Current status

We implemented the CSR register CSR_CAM at address offset 0x810 and a reset value of zero as part of the CSR register file in csr_regfile.sv, implementing both the read and write access to this special context register, as well as providing direct output logic to the context value.

The context has been parameterized as part of the configuration file, with CAM being the enable bit of context availability, and CAMContextWidth indicating the number of bits composing the context.

Rather than having some dedicated bits in the software context indicating the current OS at hypervisor level, the current process at system level and the current thread/function at user level. We are likely in the future to split the context register into several registers so that the write access could be restricted to the proper level.

5.5 Integration and test of accelerator cores with NOEL-V (GSL)

The NOEL-V processor will be updated to support an extension (co-processor/accelerator) interface. In the context of the ISOLDE project, standard extension interfaces will be evaluated (in collaboration with OpenHW group and the TRISTAN project) and finally demonstrated the integration of the ETHZ vector accelerator developed within WP3 (and potentially other accelerators).

During the project the OpenHW group CV-X-IF interface will be evaluated for integration with NOEL-V. A test implementation will be made to determine if some aspects of the specification should be updated. We will work towards an updated specification more suitable to be implemented in different processor architecture supporting a variety of accelerators.

The final stage in this development is to integrate the vector accelerator form ETHZ with the NOEL-V processor and show vector performance improvements.

Current status

Gaisler has implemented the latest CV-X-IF specification (1.0.0) into the NOEL-V processor, though with some limitations—most notably, the compressed interface is not yet integrated.

Integrating the compressed interface into the NOEL-V presents challenges due to the processor’s pipeline structure. Specifically, in NOEL-V, inputs to the register file are assigned during the decode stage. This does not pose a challenge for the issue interface, as the register file sources rs1 and rs2 are always encoded in the same instruction bits, even for CV-X-IF custom instructions.

However, the situation becomes more complex with compressed instructions, which do not carry register file source information. This information only becomes available after the instruction is decompressed by the compressed interface. To address this limitation, a bubble can be systematically inserted after the decode stage for illegal compressed instructions, allowing time for decompression. While this approach resolves the issue, it incurs a one-cycle performance penalty and increases hardware complexity. For this reason, Gaisler's current strategy is to implement and verify the CV-X-IF interface without the compressed interface. The compressed interface will be added once the current implementation is fully functional and tested.

At present, the implementation is in the verification phase. A co-processor supporting the M extension (for multiplication and division) has been developed, and current efforts are focused on generating randomized test cases targeting Mul/Div instructions.

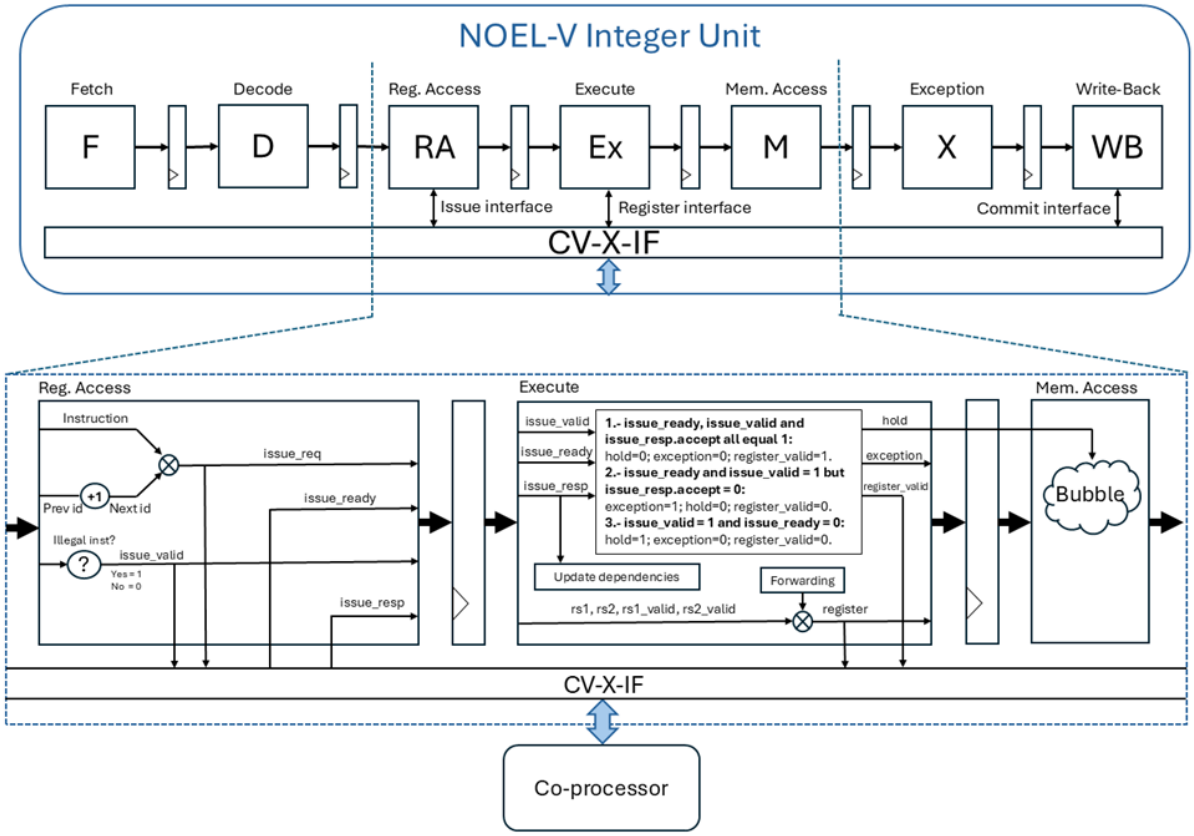

Figure 8: CV-X-IF Interface integrated into NOEL-V

Figure 8 presents a simplified block diagram of the CV-X-IF within the NOEL-V integer unit. The register access stage implements the issue interface, where a new request is initiated if the current instruction is illegal. The response from the issue interface is registered along with the issue_valid signal, and based on these signals, different decisions are made during the Execute stage.

- In the case that the request is accepted, a register interface transaction begins.
- In the case that the request is rejected, the exception signal is set and propagated to the next stage.
- If the request from the issue interface is not ready, a bubble (pipeline stall) is inserted into the next stage until the response is available.

The commit interface resides in the write-back stage, where the instruction is either killed or committed depending on its valid flag. While the response interface is not depicted in the figure, when the result is ready and a new transaction begins, the pipeline holds for one cycle, and the result is written into the register file. This allows the pipeline to continue execution, as long as there is no dependency on the operands of an instruction routed through the CV-X-IF.

5.6 Common coprocessor interface for CVA6 (TDIS)

The first implementation of the CV-X-IF on CVA6 processor was compliant to version 0.2. It has been revisited to support CV-X-IF version 1.0. The specification of this version is the outcome of a joint work with OpenHW Group members and TRISTAN project partners.

The CV-X-IF ratified release is available at https://github.com/openhwgroup/core-v-xif/releases/tag/v1.0.0

CVA6 modifications will be evaluated during the CV-X-IF specification development to ensure the needs of the different possible coprocessors (e.g., post quantum cryptography coprocessors, vector coprocessor, security coprocessor) will be met.

The CV-X-IF on CVA6 processor will be verified using UVM.

Current status

The CV-X-IF version 1.0 is currently implemented on CVA6 (Figure 9).

Notable parameters of the interface on CVA6 are:

- X_DUALREAD = 0
- X_DUALWRITE = 0
- X_ISSUE_REGISTER_SPLIT = 0
- X_NUM_RS = 2 (default) or 3 (supported)

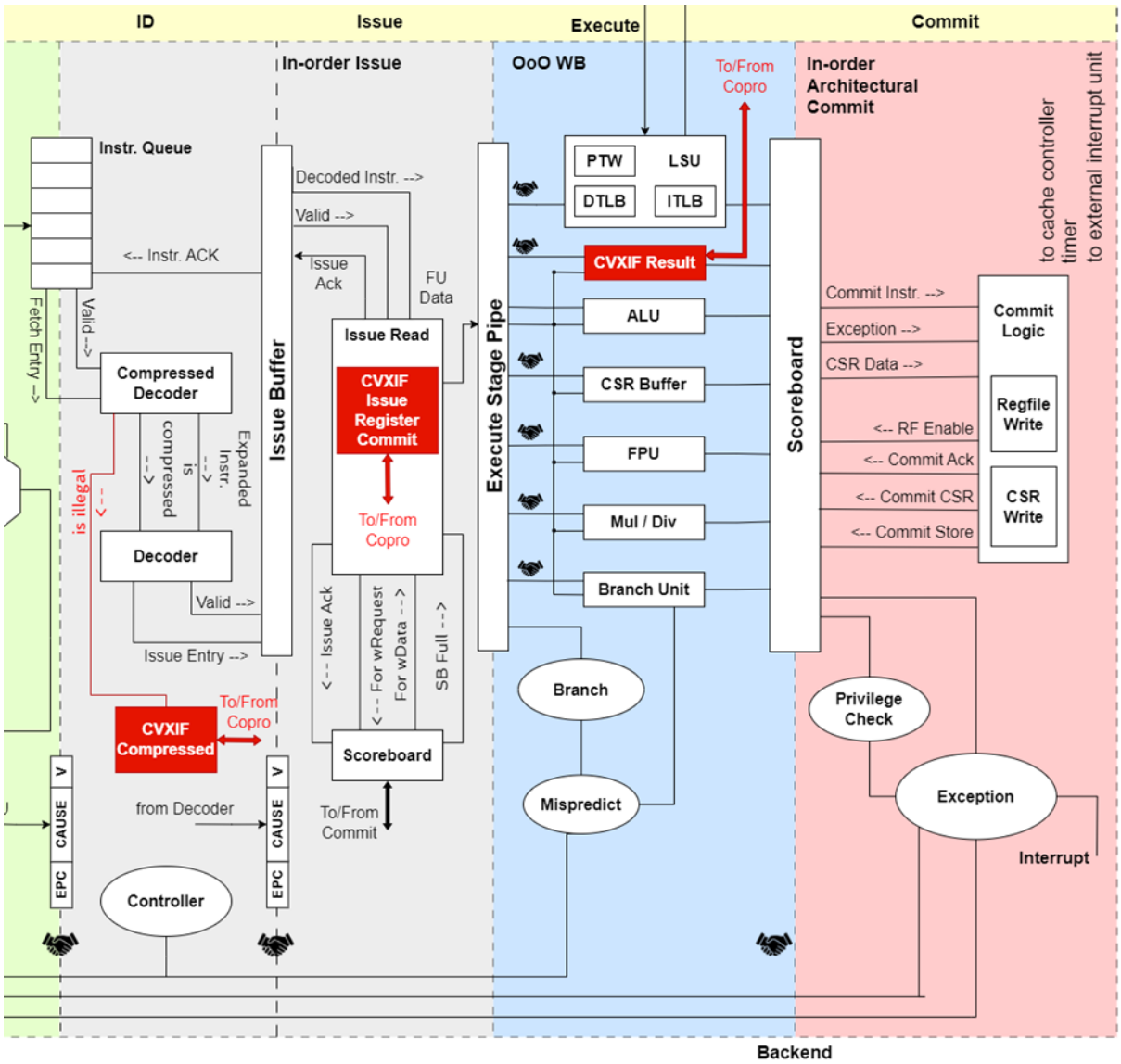

Figure 9: CV-X-IF interface integrated in CVA6

The compressed interface is implemented in the decode stage. The raw instruction passes through the compressed decoder and the macro decoder. If it is considered illegal at this point it is offloaded through the compressed interface. It can be accepted by the coprocessor which

will give back a 32-bit instruction illegal to the main decoder. In case the coprocessor does not accept the compressed illegal instruction, the 16-bit instruction will be associated to an “illegal instruction” exception by the main decoder.

All 32-bit illegal instructions are offload in the issue stage via the issue interface to the coprocessor. In the meantime, the CPU will resolve read-after-write and write-after-write dependencies to give the pre-decoded source registers to the register interface. The instruction is accepted once all required registers are ready, and the 32-bit instruction is legal to the coprocessor. The commit interface is coupled to the issue interface (and register interface). All offloaded instructions are committed to their execution if accepted. Speculation is handled internally in the CVA6. This means that no issue transaction on the issue interface is speculative and there is no need to kill them via the commit interface.

Results from the coprocessor are recovered in the execute stage via the result interface. CV-X-IF instructions have their own writeback bus to the scoreboard making the CV-X-IF interface an independent functional unit.

The CV-X-IF interface can be enabled/disabled via a parameter in the CVA6 configuration.

5.7 XIF for CVA6 and Vector Accelerator (ETHZ)

The current specifications of the CV-X-IF (1.0.0) have never been tested on RISC-V V vector co-processors. After a first analysis of the possible blockers that would prevent the CV-X-IF to work with a RISC-V V accelerator (developed by ETHZ within ISOLDE), a full implementation of the CV-X-IF on CVA6 and a RISC-V V 1.0 coprocessor will be provided together with an IPC and PPA analysis.

This will allow implementing the CV-X-IF in a WP5 demonstrator if the CV-X-IF interface does not negatively impact the PPA and IPC metrics of the non-CV-X-IF architecture, and a valuable feedback of the effects of the CV-X-IF on a RISC-V V architecture, together with what the next CV-X-IF specifications will need to support in order to provide the required functionality.

Current status

ETHZ has successfully merged the custom interface in CVA6 to have a first working interface that can act as a baseline during the CV-X-IF evaluation. ETHZ also participated in the meetings for CV-X-IF 1.0.0 ratification, providing a first feedback on the specifications, and working towards a first version of the CV-X-IF (1.0.0) implementation between CVA6 and the multi-precision vector coprocessor developed within WP3 (T3.4).

This first version of the architecture with CV-X-IF is currently being streamlined not to impact the PPA metrics of the architecture.

Moreover, ETHZ will provide feedback on the CV-X-IF to the interested TRISTAN/ISOLDE partners.

5.8 OS support for WP3 Vector Accelerator (ETHZ)

The CVA6 core supports Linux and a custom accelerator port for the vector coprocessor developed within WP3 (T3.4) by ETHZ. Nevertheless, when the vector coprocessor is enabled, CVA6 can execute vector code only in bare-metal mode, limiting the flexibility of the vector coprocessor.

ETHZ will make the necessary architectural modifications to support an operating system and the vector coprocessor concurrently, e.g., by enabling MMU address translation for vector memory operations.

Current status

ETHZ has almost completed the implementation of an MMU sharing port for the WP3 vector coprocessor to enable vector code execution on Linux.

6 Software interfaces to general purpose cores

6.1 XNG RISC-V BSP to support new NOEL-V features (FEN)

The XNG (Xtratum Next Generation) RISC-V BSP (Board Support Package) will be updated to support the new standards of RISC-V architecture implemented by NOEL-V. In the context of the ISOLDE project, this new XNG RISC-V BSP will also support the new advanced interrupt architecture (AIA) extension that will be developed inside the scope of this project. This extension implements two new devices, the Advanced Platform-Level Interrupt Controller (APLIC), which replaces the old PLIC, and the incoming MSI controller (IMSIC). These devices allow the support of a new type of interrupts, the message-signalled interrupts (MSIs), and both controllers will be supported by XNG.

Current status

After the analysis of the requirements of the new features to be supported and the reception of the NOEL-V bitstream with the AIA extension, the software development has started. In particular, the first implementations have been focused on supporting the wired interrupts, for which a bare metal driver for the APLIC has been firstly created. Then, the integration of this driver in the XNG hypervisor has been started, in order to manage this kind of interrupts directly at the hypervisor level.

6.2 NOEL-V software tools (GSL)

GSL will update and provide bare-metal and Linux RISC-V tool chains for NOEL-V. Work will also be performed to extend NOEL-V software support.

Current status

Current versions of the NOEL-V tool chains are available for download.

6.3 System software support (SYSGO)

Test hardware developments; produce open-source hardware demonstrations / device drivers.

Current status

We have set up SYSGO's embedded Linux ELinOS on top of PikeOS in a RISC-V environment, and thus demonstrated the feasibility of paravirtualisation of ELinOS running on top of PikeOS. We have also demonstrated the ability to install the Docker virtualization environment on top of ELinOS. Moreover, we have configured CVA6 support for PikeOS in the context of the smart home demonstrator in WP5.

7 Conclusion

The Intermediate report on foundational IP cores deliverable provided intermediate results developed within WP2 (Open-Source Foundation Cores) of the ISOLDE project, building on the requirements defined in deliverables D1.3 and D1.4.

First architecture and implementation results are described. For the next period, the focus will be on completing implementations and on test and verification activities.

A final version of this report will be provided in Deliverable D2.3 to be due in M33.