SYCLOPS

Deliverable 3.2 – EMDC v2.0 with RVV accelerator release

GRANT AGREEMENT NUMBER: 101092877

Project acronym:	**SYCLOPS**
Project full title:	**Scaling extreme analYtics with Cross architecture acceLeration based on OPen Standards**
Call identifier:	**HORIZON-CL4-2022-DATA-01-05**
Type of action:	**RIA**
Start date:	**01/01/2023**
End date:	**31/12/2025**
Grant agreement no:	**101092877**

D3.2 – EMDC v2.0 with RVV accelerator release

Executive Summary: This deliverable documents the progress made in the SYCLOPS project's infrastructure layer, specifically focusing on "Task 3.2: RISC-V reference platform" and "Task 3.3: EMDC assembly". The primary objective of these tasks was to deploy EMDC v2.0, a hardware testbed designed to support the evaluation of SYCLOPS use cases using software developed in the platform and libraries layers. This has been achieved by the deployment of a new hardware testbed contains three key components: (i) a proprietary RISC-V platform from CSIP featuring vector extensions, (ii) an open-source RISC-V platform known as SYCLARA developed within SYCLOPS, and (iii) a CXL-enabled EMDC testbed.

WP:	3		
Author(s):	Jan Kastil, Martin Bozek, Mojtaba Rostamibilandi, Fred Buining		
Editor:	Raja Appuswamy		
Leading Partner:	CSIP		
Participating Partners:	HIRO, EUR		
Version:	1.0	**Status:**	Draft
Deliverable Type:	Other	**Dissemination Level:**	PU
Official Submission Date:	30-Sep-2025	**Actual Submission Date:**	06-Oct-2025

Disclaimer

The SYCLOPS consortium consists of the following partners:

No.	Partner Organisation Name	Partner Organisation Short Name	Country
1	EURECOM	EUR	**FR**
2	INESC ID - INSTITUTO DE ENGENHARIADE SISTEMAS E COMPUTADORES, INVESTIGACAO E DESENVOLVIMENTO EM LISBOA	INESC	**PT**
3	RUPRECHT-KARLS-UNIVERSITAET HEIDELBERG	UHEI	**DE**
4	ORGANISATION EUROPEENNE POUR LA RECHERCHE NUCLEAIRE	CERN	**CH**
5	HIRO MICRODATACENTERS B.V.	HIRO	NL
6	ACCELOM	ACC	FR
7	CODASIP S R O	CSIP	CZ
8	CODEPLAY SOFTWARE LIMITED	CPLAY	UK

Document Revision History

Version	Description	Contributions
0.1	Structure and outline	EUR
0.2	Updated description of CSIP RISC-V platform	CSIP
0.3	Updated description of SYCLARA	EUR
0.4	Updated description of HIRO EMDC	HIRO
1.0	Final draft	EUR

Authors

Author	Partner
Jan Kastil	CSIP
Martin Bozek	CSIP
Mojtaba Rostamibilandi	EUR
Fred Buining	HIRO

Reviewers

Name	Organisation
Aleksandar Ilic	INESC
Vincent Heuveline	UHEI
Stefan Roiser	CERN
Nimisha Chaturvedi	ACC

Statement of Originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Table of Contents

1. Introduction 7
2. CSIP Vector-enabled RISC-V Platform 9
2.1 Vector-enabled processor core 9
2.1.1 Overview of RISC-V core 9
2.1.2 Technical Specifications 9
2.1.3 Description of the RVV Extension for A730 10
2.2 FPGA Platform 12
2.3 Operating System 13
2.4 Experimental Results 15
2.4.1 Benchmarking in simulation 15
2.4.2 OpenBLAS evaluation 18
2.4.3 Dot product emulation benchmark 23
2.5 Summary 25
3. SYCLARA Open Source RISC-V Platform 26
3.1 SYCLARA Open Source RISC-V Platform 26
3.2 Evaluation 27
3.3 Summary 27
4. CXL-enabled EMDC Testbed 28
4.1 PCIe CXL Switch Development 28
4.1.1 Architecture and Implementation 30
4.1.2 Prototyping and Validation 31
4.2 CXL-enabled GPU server 33
5. Conclusion 35

Executive Summary

This deliverable documents the progress made in the SYCLOPS project's infrastructure layer, specifically focusing on "Task 3.2: RISC-V reference platform" and "Task 3.3: EMDC assembly". The primary objective of these tasks was to deploy EMDC v2.0, a hardware testbed designed to support the evaluation of SYCLOPS use cases using software developed in the platform and libraries layers. This has been achieved by the deployment of a new hardware testbed contains three key components: (i) a proprietary RISC-V platform from CSIP featuring vector extensions, (ii) an open-source RISC-V platform known as SYCLARA developed within SYCLOPS, and (iii) a CXL-enabled EMDC testbed.

On the RISC-V side, significant work was completed on both the proprietary and open-source RISC-V platforms. The CSIP Vector-enabled RISC-V Platform utilizes the Codasip A730 core, a 64-bit in-order dual-issue processor, integrated with a Vector Processing Unit (VPU) that fully conforms to version 1.0 of the RISC-V "V" Vector Extension specification. This proprietary platform was deployed on the AMD VCU118 FPGA Evaluation Kit. Initial simulation results confirmed performance gains, with speedups ranging up to 9×. Concurrently, the SYCLARA open-source platform was developed by EUR, integrating the CVA6 RISC-V core with the ARA2 RVV accelerator and incorporating an Ethernet module to enable interaction with a SYCL host. Preliminary evaluation using SYCL kernels on SYCLARA showed impressive performance improvements, reaching speedups up to 6.27× for 1D convolution and up to 6× for matrix multiplication. Through these results, we have exceeded the target KPI of 2× speedup we set to achieve with RVV.

On the EMDC side, we have deployed a CXL-enabled EMDC Testbed that integrates switches from the Broadcom PEX90000 family, which supports PCIe Gen6 (64 GT/s data rates) and has roadmap support for CXL 3.0. Prototyping and validation efforts confirmed reliable 64 GT/s signaling with PAM-4, ensuring the fabric is ready to adopt CXL 3.0 when the ecosystem matures. Additionally, a stand-alone CXL 2.0 server was established in collaboration with Supermicro and Micron to allow SYCLOPS partners to investigate performance tradeoffs between local DRAM and CXL memory.

1. Introduction

Figure 1 shows the SYCLOPS hardware-software stack consists of three layers: (i) infrastructure layer, (ii) platform layer, and (iii) application libraries and tools layer.

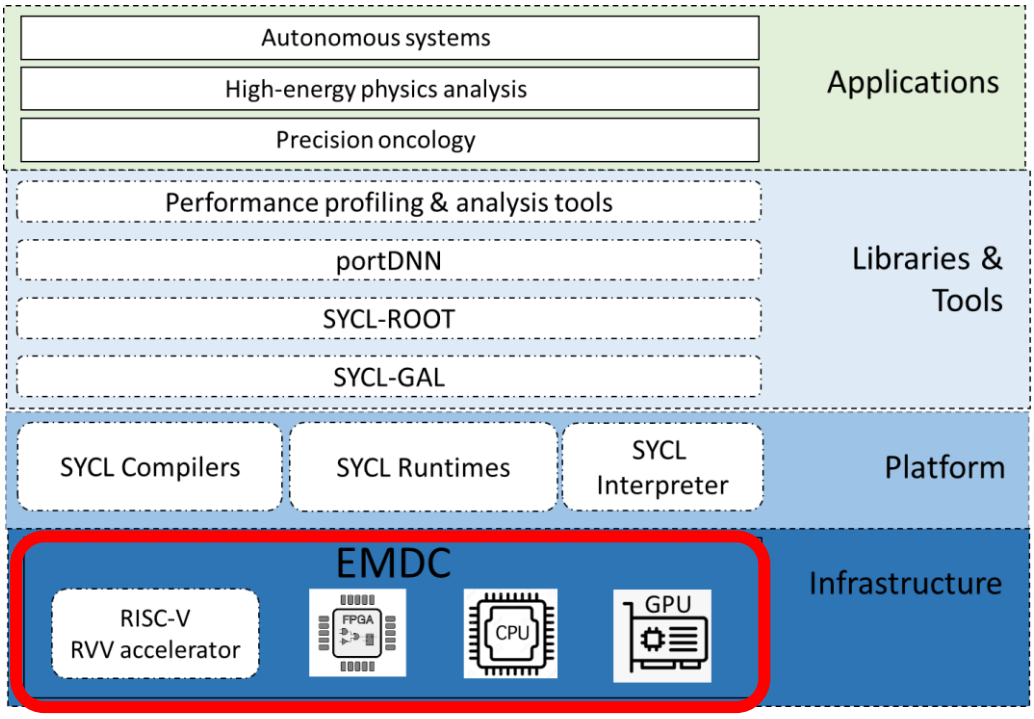


Figure 1. SYCLOPS architecture

1. **Infrastructure layer:** The SYCLOPS infrastructure layer is the bottom-most layer of the stack and provides heterogeneous hardware with a wide range of accelerators from several vendors.
2. **Platform layer:** The platform layer, provides the software required to compile, execute, and interpret SYCL applications over processors in the infrastructure layer.
3. **Application libraries and tools layer:** The libraries layer enables API-based programming by providing pre-designed, tuned libraries for various deep learning methods for the PointNet autonomous systems use case (SYCL-DNN), mathematical operators for scalable HEP analysis (SYCL-ROOT), and data parallel algorithms for scalable genomic analysis (SYCL-GAL). This layer also contains conversion tools, to facilitate porting of CUDA applications into SYCL, and profiling tools to enable the analysis of cross-architecture SYCL applications.

This deliverable presents the work carried out in the **infrastructure layer** on **EMDC v2.0** as highlighted in Figure 1 in the context of “Task 3.2: RISC-V reference platform” and “Task 3.3: EMDC assembly ” of WP3 in the SYCLOPS project. The goal of these tasks was to build an RVV-capable, RISC-V reference platform and deploy it together with CPU, GPU, and FPGA accelerators from other vendors in a hardware testbed that will be used to support evaluation of SYCLOPS use cases using software developed in platform and libraries layers.

At M18, we had developed and deployed v1.0 of the SYCLOPS EMDC which was described in deliverable D3.1. At M33, we have developed and deployed v2.0 of the SYCLOPS hardware testbed containing three key components: (i) A proprietary RISC-V platform from CSIP with support for vector extensions, (ii) an open-source RISC-V platform called SYCLARA we have developed in SYCLOPS, and (iii) a CXL-enabled EMDC testbed.

This deliverable is structured as follows. Section 1 of this deliverable provides a high-level overview of the overall SYCLOPS architecture and positions this deliverable with respect to both components in the

SYCLOPS stack and WP/tasks in the work plan. Sections 2 and 3 provide an overview of the work done on the proprietary CSIP RISC-V core with vector extensions and the open-source SYCLARA RISC-V platform in the context of “Task 3.2: RISC-V Reference Platform”. Section 4 provides an overview of the work done in CXL-enabled PCIe 6.0 switching for HIRO’s next generation EMDCs in the context of “Task 3.3: EMDC Assembly”.

2. CSIP Vector-enabled RISC-V Platform

This section describes the RISC-V processor with Vector extension designed by CSIP. The Vector extension allows RISC-V processor to us the SIMD parallelism to speed up the SYCL computation kernels.

First, we describe the RISC-V core itself. The partner CSIP provided the Vector extension for the A730 application core. A730 is in-order dual issue processor with 9-stage pipeline. Then, we introduce the FPGA platform called Hobgoblin. Hobgoblin was developed for the emulation of the Codasip processor cores. It provides all peripheral necessary to execute binaries on the A730 as well as ethernet interface used to connect with the outside environment. We also briefly describe the Linux operating system used to enable OneAPI construction Kit server. The provided Linux distribution enabled the vector instruction to the user space applications but does not use them itself. Any Linux binary compiled for the RISC-V with Vector extension can be executed on the platform. Finally, we present an evaluation of the platform using several benchmarks.

2.1 Vector-enabled processor core

2.1.1 Overview of RISC-V core

The foundation of this RVV integration into EMDC v1 and v2 is the Codasip A730, an advanced, customizable RISC-V core. The Codasip A730 is a versatile mid-range 64-bit dual-issue in-order application core, well-suited for a broad spectrum of applications due to its significant performance improvements over previous generations. Its compliance with the RVA22 RISC-V profile further solidifies its position as a robust and standard-compliant foundation for Vector extension. The CSIP's development work in the SYCLOPS project contributes to the larger activity focused on the implementation of the Vector extension and its integration in the A730.

2.1.2 Technical Specifications

- **Architecture:** 64-bit in-order dual-issue RISC-V core.
- **ISA:** Supports base integer set, RV64I.
- **Microarchitecture:** In order, 9 stages, dual issue pipeline.
- **Configurations:** Available in single- and multi-core configurations, with a cluster featuring full coherency, shared L2 cache, and an interrupt controller compliant with the RISC-V standard.

- **Performance**: 2x the performance of previous generations, designed for complex compute tasks in power-constrained devices.

- **Customization:** Flexible design with options to configure the number of cores (one to four), memory protection, branch prediction, cache sizes, and TLBs.

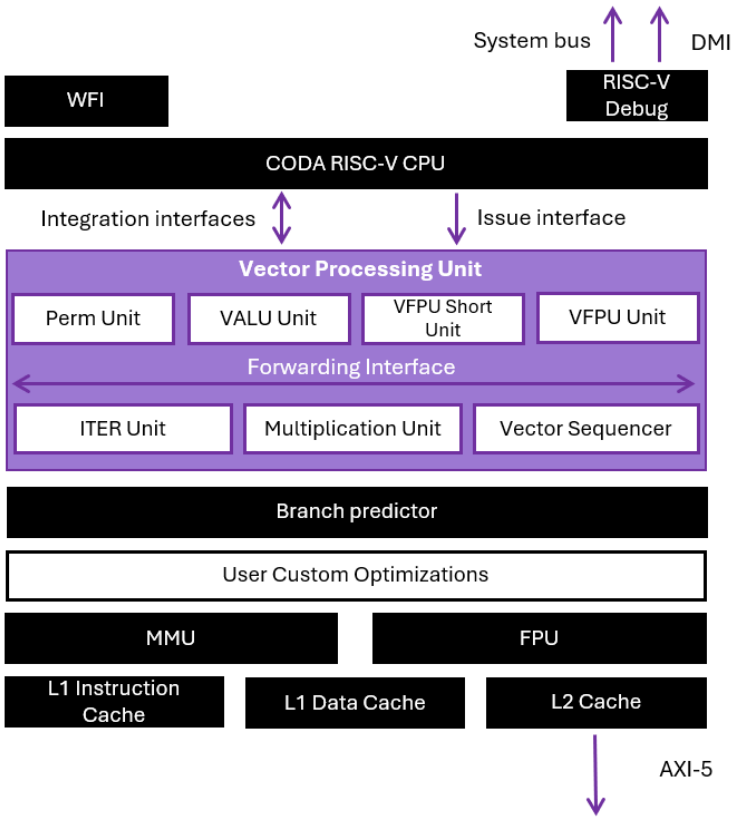


Figure 2. High-level overview of the A730 RISC-V core and Vector Processing Unit

2.1.3 Description of the RVV Extension for A730

The Vector Processing Unit, representing the RVV accelerator, is designed for full conformance with version 1.0 of the RISC-V "V" Vector Extension specification, which is a foundational document for vector processing in the RISC-V ecosystem. Section 18.3, specifically titled "V: Vector Extension for Application Processors," outlines the stringent requirements for vector units designed for general-purpose computing environments. The official reference for this section can be found at: https://lists.riscv.org/g/tech-vector-ext/attachment/686/0/riscv-v-spec.pdf#page=94.

The design of the CSIP's vector unit incorporates several key attributes and design choices that balance performance, area and complexity:

- **Memory Access Limitations**: The vector unit intentionally does not support access to non-idempotent and/or non-cached memory. However, it fully supports misaligned memory access for cached, idem-potent memory during vector load/store operations.
- **Vector Length (VLEN)**: For the Codasip A730 processors, the planned VLEN is 128 bits. This length is consistently twice the XLEN (eXtension Length) and representing the maximum data size the memory subsystem can transfer in a single cycle.
- **Implementation Model**: The vector unit is realized as a series of specialized vector execution units that are tightly integrated with the scalar processor.
- **Issue Model**: The vector unit operates as a single-issue pipeline, meaning it cannot simultaneously issue instructions to multiple execution units within the same clock cycle. This design choice prioritizes simpler control logic and potentially faster time-to-market over maximizing instruction-level parallelism, which is a common trade-off in processor design.
- **Memory Coherence**: All memory access for the vector unit is handled by the scalar machine's L1 cache, ensuring inherent data coherence between the scalar and vector processing elements.

- **Pipeline and Register Management**: Instructions and their constituent micro-operations (μOPs) are issued and complete in-order, without the use of register renaming. The width of the vector datapaths is matched to the VLEN, and the memory interface width also corresponds to VLEN.
- **SIMD Operation**: Arithmetic operations within the vector unit (excluding memory access operations) function in a simple Single Instruction, Multiple Data (SIMD) manner. This allows all elements within a VLEN-sized vector register to be processed in parallel. For a VLEN of 128 bits, this translates to 16 operations on 8-bit elements, 8 operations on 16-bit elements, 4 operations on 32-bit elements, or 2 operations on 64-bit elements.

EEW (Effective Element Width)	Description (VLEN = 128)
8	16 operations of 8-bits each
16	8 operations of 16-bits each
32	4 operations of 32-bits each
64	2 operations of 64-bits each

- **No Chaining**: The design explicitly avoids chaining, meaning that results are forwarded as a complete vector register rather than individual elements. This simplification in forwarding logic may introduce stalls if dependent operations require only a subset of elements, representing another design consideration that balances complexity with potential performance gains.
- **LMUL Implementation**: The vector register grouping (LMUL) feature is implemented through a sequencer that orchestrates the issuance of multiple micro-operations to complete the required architectural instruction.
- **Multi-cycle Operations**: The vector sequencer is also responsible for managing vector instructions that require multiple cycles to complete, such as vector reduction operations.
- **Vector Register File**: The vector register file is equipped with three read ports and a single write port. The v0 mask register is shadowed in flip-flops, which eliminates the need for an additional read port specifically for v0.
- **Execution Unit Sharing**: There is no attempt to share execution units between the scalar and vector machines; for instance, the Floating-Point Unit (FPU) is kept separate.
- **Pipelined ALUs**: Arithmetic execution units are generally pipelined, allowing them to initiate or complete a new micro-operation every clock cycle, irrespective of the element size. Exceptions to this are the divide and square root operations, which are iterative and cannot be pipelined.
- **Byte-Lane Cross-Bar**: The implementation assumes that a full byte-lane cross-bar switch can be achieved within a single clock cycle. This implies that for a VLEN of 128 bits, each of the sixteen destination bytes can arbitrarily select data from any of the sixteen input bytes.
- **Vector Memory Load/Store**: These operations are executed as a series of MEM_WIDTH-aligned accesses to memory. Single-cycle access is achieved when EMUL=1 and memory addresses are word-aligned. If addresses are not aligned, additional MEM_WIDTH-aligned accesses are required.
- **Interrupts and Exceptions**: For vector instructions requiring multiple micro-operations, interrupts are generally disabled within the architectural instruction. However, a select few very long-running operations (exceeding 100 cycles) permit interrupts at specific points. Similarly, certain exceptions, such as page faults, may be taken during vector load/store instructions, utilizing the vstart mechanism to enable instruction restart.

2.2 FPGA Platform

To be able to emulate the core the FPGA emulation platform is provided. The purpose of the platform is to run the core in FPGA fabric and provide all peripherals necessary to run operating system. The ethernet was selected to facilitate connection with outside environment.

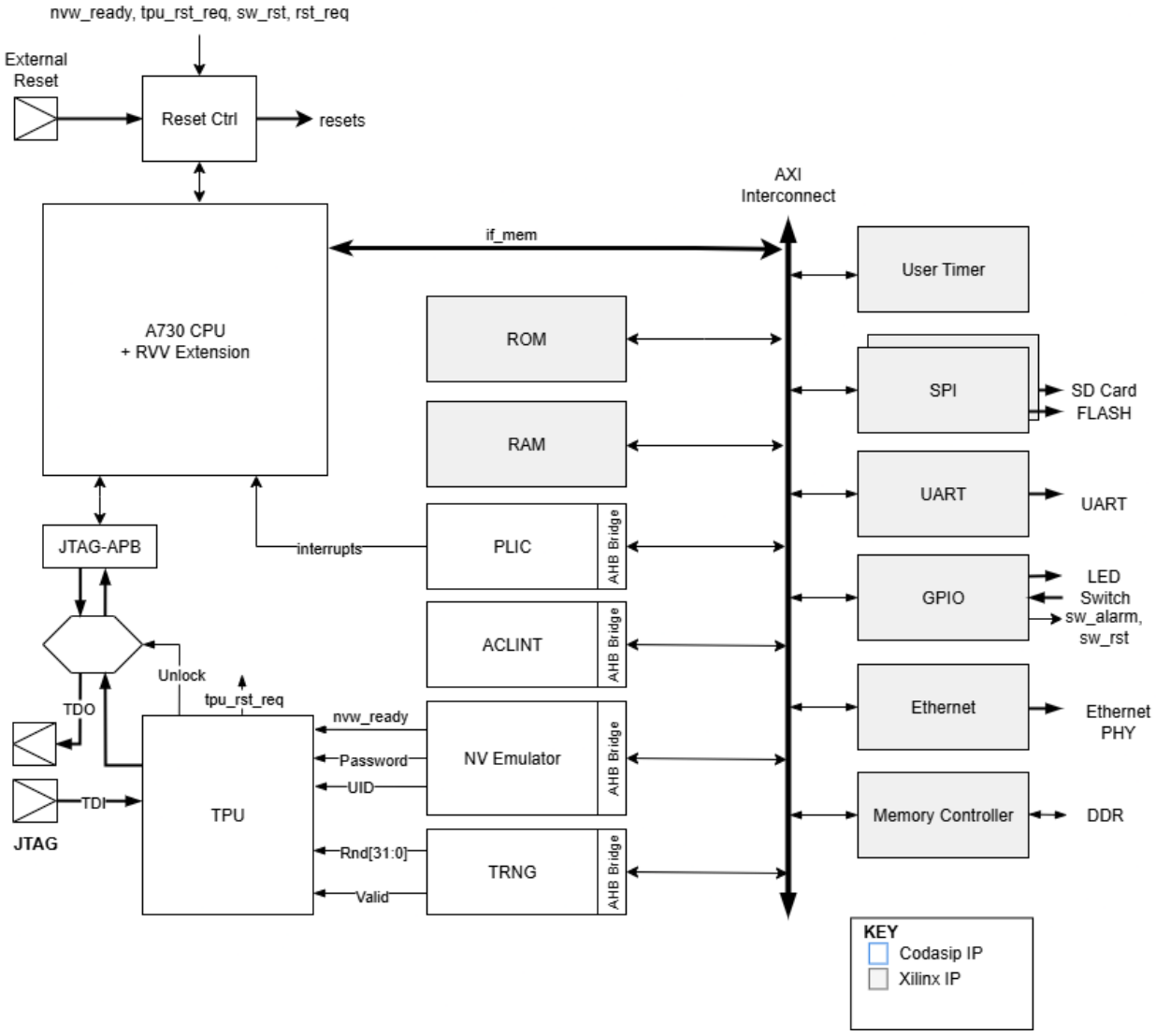


Figure 3. The FPGA Platform High Level Architecture for EMDC v2.0

The Figure 2 shows the block diagram of the FPGA platform. For the EMDC v2.0 release, the FPGA platform was strategically upgraded to the AMD VCU118 Evaluation Kit, featuring the Virtex UltraScale+ XCVU9P FPGA. This transition was necessitated by the escalating logic and memory capacity requirements of the integrated RVV accelerator and the broader EMDC v2.0 architecture. The VCU118 provides a substantial increase in programmable logic resources, a significantly more robust memory subsystem (2.5 GB DDR4 SDRAM), and enhanced high-speed I/O capabilities, all critical for accommodating the expanded design complexity, memory-intensive workloads, and high-throughput data processing demands of the advanced edge microdatacenter. Key features and specifications of the board include:

- **FPGA**: Virtex UltraScale+ XCVU9P FPGA

- **Logic Cells**: 2.5 million
- **DSP Slices**: 1,248
- **Transceiver Speeds**: Up to 40 GB/s
- **Memory**: 2.5 GB of DDR4 SDRAM, 4 GB of QSPI flash
- **High-Speed Connectivity**: 1000Mbps Ethernet ports
- **Connectivity and On-board I/O**:
    - FMC Connectors: FMC, FMC+
    - Pmod Connectors
- **Programming Flexibility**: Vivado Design Suite and PetaLinux

Unfortunately, the VCU118 does not have the SD Card slot. As such, the FMC extension board was provided by CSIP. This board contains another 1000Mbps ethernet port together with the SD Card connector.

The SDCard from this port is used to boot up the software the FPGA bitstream is programmed. The FPGA platform support the secure boot process which allows to run only the cryptographically signed binaries. The started binaries may be the final application or the operating system bootloader.

Unfortunately, unlike the Genesys 2 board used for the EMDC v2.0 release, the VCU118 does not support the loading of the bitstream from the SDCard itself. As such, the bitstream must be stored on the QSPI flash or programmed into FPGA directly by JTAG programming interface.

Therefore, platform contains two JTAG chains. The first chain contains the FPGA itself and is used to program the bitstream into the FPGA through Vivado tool. The second JTAG chain contains only the A730 core. The second JTAG chain is used to connect the debugger to the core itself.

2.3 Operating System

The software environment supporting the EMDC v2.0 with the RISC-V platform has been prepared to ensure a robust and flexible development ecosystem.

- **Linux Distribution**: The Poky distribution, a reference embedded Linux distribution, has been specifically modified to support the FPGA Platform.
- **Yocto Manifest**: The yocto manifest for this modified distribution is publicly available on GitHub, promoting transparency, collaboration, and reproducibility of the software environment.
- **System Access**: The system offers versatile access methods, including the UART console and secure shell (SSH) over Ethernet.

The final Linux distribution is based on the poky distribution which is further modified to meet all the SYCLOPS requirements.

The first important modification is the enablement of the Vector extension. To enable the use of vector instruction in the user space application, vector extension must be enabled in the linux kernel configuration file.

```
CONFIG_RISCV_ISA_V=y
CONFIG_RISCV_ISA_V_DEFAULT_ENABLE=y
CONFIG_RISCV_ISA_V_UCOPY_THRESHOLD=25600000000
```

As can be seen from previous figure, the Vector ISA was enabled, and it use was allowed by default. However, the use of vector instruction for the memory copy was allowed only for the prohibitively large memory blocks. This configuration was selected to ensure that while the vector instructions and registers are handled correctly by the kernel, they are not used for the kernel operation.

Even if the kernel is configured to support vector extension by default, it will not use vector instruction if they are not available in the HW. Therefore, the device tree for the platforms must contain the vector support as well.

To ensure the correct comparison with the previous version of the platform. The rest of the Linux binaries were not recompiled with vector support. Therefore, only application that uses the vector instruction will be the evaluated user space application downloaded over SSH interface. This is to ensure that the measured speed up is cause by the parallelization in the evaluated application instead of more efficient operating environment.

```
root@hobgoblin-a730:~# openblas_utest_ext

<around 1500 Lines removed>

TEST 1520/1522 csbmv:upper_k_2_inc_b_1_inc_c_1_n_100 [OK]
TEST 1521/1522 csbmv:upper_k_1_inc_b_1_inc_c_1_n_100 [OK]
TEST 1522/1522 csbmv:upper_k_0_inc_b_1_inc_c_1_n_100 [OK]
RESULTS: 1522 tests (1522 ok, 0 failed, 0 skipped) ran in 75662 ms
root@hobgoblin-a730:~#
```

For validation that the user space application can access the vector instruction, openblas extended regression test binary (*openblas_utest_ext)* was added into the Linux distribution. When the Linux OS boots up, user can execute this binary. If the VPU is working and available, all tests will pass without any issues. Otherwise, the error messages will appear.

The test binary may be executed through console access over UART or over the SSH connection. To run the secure shell connection, the platform must be connected to the ethernet. By default, the Linux boot up with static IP (10.15.51.61). The Linux distribution contain also dhcp client so it is possible to change the network configuration to accept the IP address from the DHCP server.

To change network setting, the file */etc/network/interfaces* on SDCard must be modified. The SDCard contains two partitions, one with the bootloader and kernel image. The second partition contain the Linux filesystem. This partition is mounted and accessed from the Linux on the FPGA. After the network is correctly set up, the board is available through secure shell and as such can be used to run arbitrary SW compiled for the RiscV with Vector extension.

4.1 Integration with oneAPI Construction Kit

Integration with the oneAPI Construction Kit is a critical aspect of the software environment, enabling broader industry adoption and heterogeneous computing paradigms. This integration involved three key areas:

- **Platform Preparation:** Ensuring the FPGA board is fully prepared to interface with the oneAPI construction kit, which includes integrating the RISC-V core with the necessary software environment.

- **Software Environment Setup**: Establishing a complete development toolchain, including compilers and debuggers, that are fully compatible with the RISC-V architecture. This also encompasses

the deployment of a rich Linux operating system, which is fully supported by the A730 RISC-V core and the RVV Extension.

- **Testing and Validation**: Conducting comprehensive testing to verify the compatibility and performance of the integrated system. This phase leverages instruction-accurate and cycle-accurate models, which are key for early software development and debugging, ensuring the system meets its performance and functional targets.

2.4 Experimental Results

2.4.1 Benchmarking in simulation

The first analysis of the designed VPU was done in the simulation. We run the VPU core in the simulation and executed small kernels for the basic vector and matric operations.

2.4.1.1 Experimental Setup

To evaluate the VPU-extended core, we used Codasip's internal verification environment, coreTB. This environment runs RTL simulation of the processor in a selected third-party simulator, automatically generates instruction traces from the simulated core, and compares them with traces from a golden model (e.g., the SAIL model).

The benchmark C code was compiled with a standard toolchain, and the resulting binaries were loaded into simulated memory within coreTB. The simulation then accesses memory and boots the processor as real hardware or an FPGA would.

2.4.1.2 Benchmark evaluation

Because simulation is significantly slower than emulation, we did not run a Linux OS. Instead, the benchmarks were executed as bare-metal applications. This initial experiment used four vector algorithms with different degrees of inherent parallelism. Each algorithm used single-precision floats, so the measured VPU processes four elements at a time. The benchmark marks the start and end of each function, allowing exact measurement of the number of clock cycles spent in computation from the simulation logs.

2.4.1.3 Benchmark results

Table 1 summarizes the simulation measurements. Observed speedups range from 1.5× to 5×, depending on the algorithm and input size.

The SYCLOPS KPIs target a 2× speedup. Vector addition and vector multiplication meet this target even for the smallest datasets. The matrix-vector product reaches the target for sizes 32×32 and above. The vector dot product does not meet the target - in the measured implementation, a scalar unit performs the final accumulation after the multiplications, which limits the achievable speedup.

Overall, when sufficient parallelism is available, the vector extension delivers speedups above 4× in this simulation study.

Operation	Task Size	Scalar Avg. Cycles	Vector Avg. Cycles	Speedup (Scalar/Vector)	Scalar Cycles/Elem	Vector Cycles/Elem
Vector Addition	64	501	170	2.95	7.83	2.66
Vector Addition	128	1053	263	4	8.23	2.05
Vector Addition	512	3891	1002	3.88	7.6	1.96
Vector Addition	1024	7640	1892	4.04	7.46	1.85
Vector Multiplication	64	639	164	3.9	9.98	2.56

Vector Multiplication	128	1239	261	4.75	9.68	2.04
Vector Multiplication	512	4625	998	4.63	9.03	1.95
Vector Multiplication	1024	9538	1890	5.05	9.31	1.85
Vector Dot Product	64	415	260	1.6	6.48	4.06
Vector Dot Product	128	792	488	1.62	6.19	3.81
Vector Dot Product	512	3150	1898	1.66	6.15	3.71
Vector Dot Product	1024	6209	3701	1.68	6.06	3.61
Matrix-Vector Product	8x8	958	636	1.51	119.75	79.5
Matrix-Vector Product	16x16	2963	1493	1.98	185.19	93.31
Matrix-Vector Product	32x32	11103	3972	2.8	346.97	124.12
Matrix-Vector Product	64x64	41952	15245	2.75	655.5	238.2

Table 1. Benchmarking time in clock cycles

Tab 2. column descriptions:

1. **Operation**
   - Name of the computation being benchmarked (e.g., Vector Addition, Vector Dot Product, Matrix-Vector Product). Serves as the row label.

2. **Task Size**
   - Problem size for the row.
   - For vector ops: the vector length n (e.g., 64, 128, …).
   - For Matrix-Vector Product: the matrix dimension n×n (e.g., 32×32), which produces n output elements.

3. **Scalar Avg. Cycles**
   - Average number of clock cycles to complete the operation using the scalar (non-vectorized) implementation.
   - Unit: cycles.
   - If repeated runs were used, this is the mean across runs.

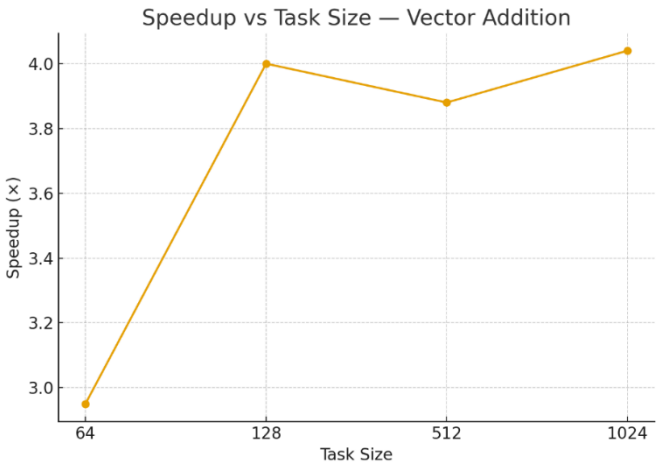
4. **Vector Avg. Cycles**
   - Average number of clock cycles to complete the operation using the vectorized implementation.
   - Unit: cycles.
   - Mean across runs if repeated.

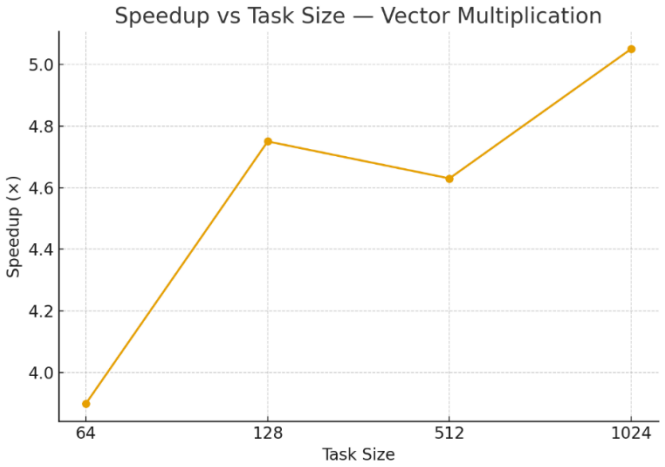
5. **Speedup (Scalar/Vector)**
   - How many times faster the vector version is compared to scalar.
   - Formula: `Speedup = (Scalar Avg. Cycles) / (Vector Avg. Cycles)`
   - Interpretation: values > 1.0 indicate the vector version is faster; = 1.0 no change; < 1.0 regression.

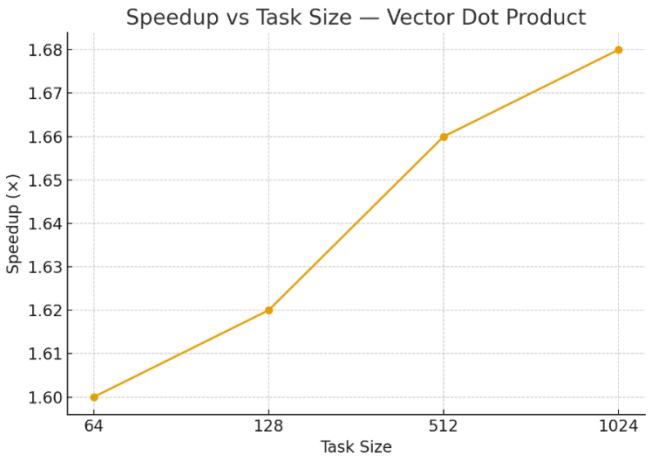
6. **Scalar Cycles/Elem**
   - Per-element cost for the scalar implementation.
   - Formula (vector ops): `Scalar Avg. Cycles / n`
   - Formula (M×V): `Scalar Avg. Cycles / n` (since an n×n matrix–vector produces n outputs)
   - Interpretation: lower is better; shows efficiency independent of total size.

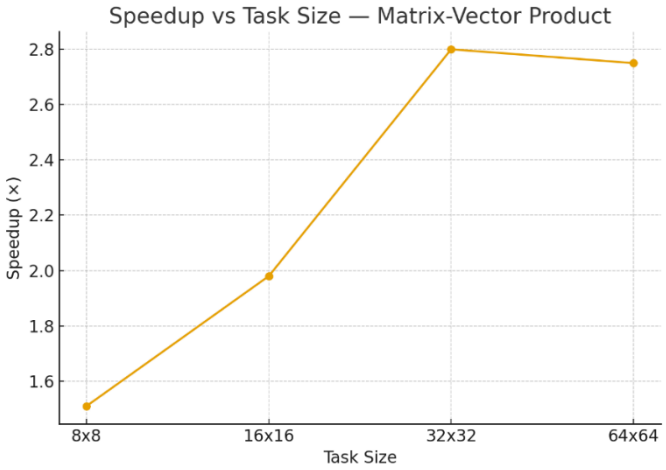
7. **Vector Cycles/Elem**
   - Per-element cost for the vector implementation.
   - Formula (vector ops): `Vector Avg. Cycles / n`
   - Formula (M×V): `Vector Avg. Cycles / n`
   - Interpretation: lower is better; directly comparable to *Scalar Cycles/Elem* to see per-element gains.

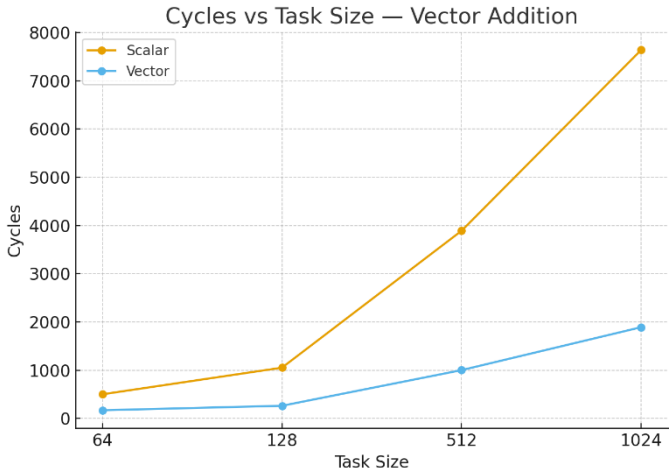
Note: All cycle counts are hardware-clock cycles; convert to time with `time = cycles / frequency`.

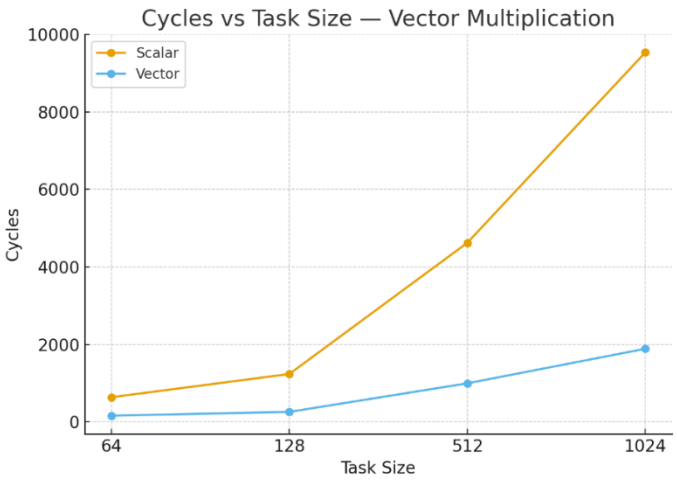


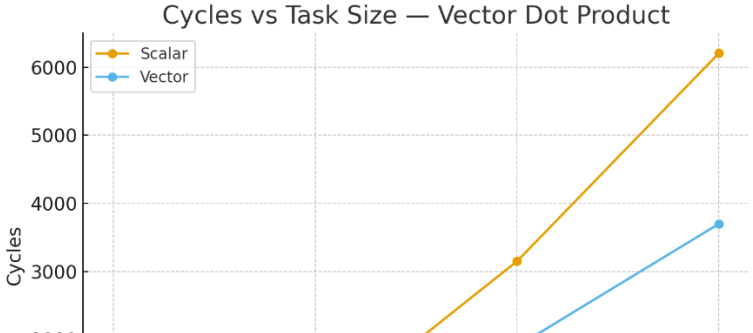


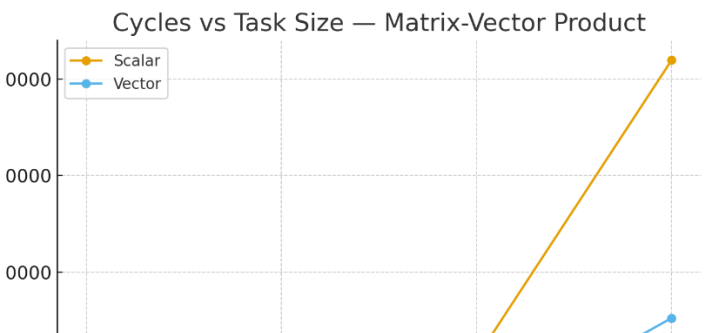












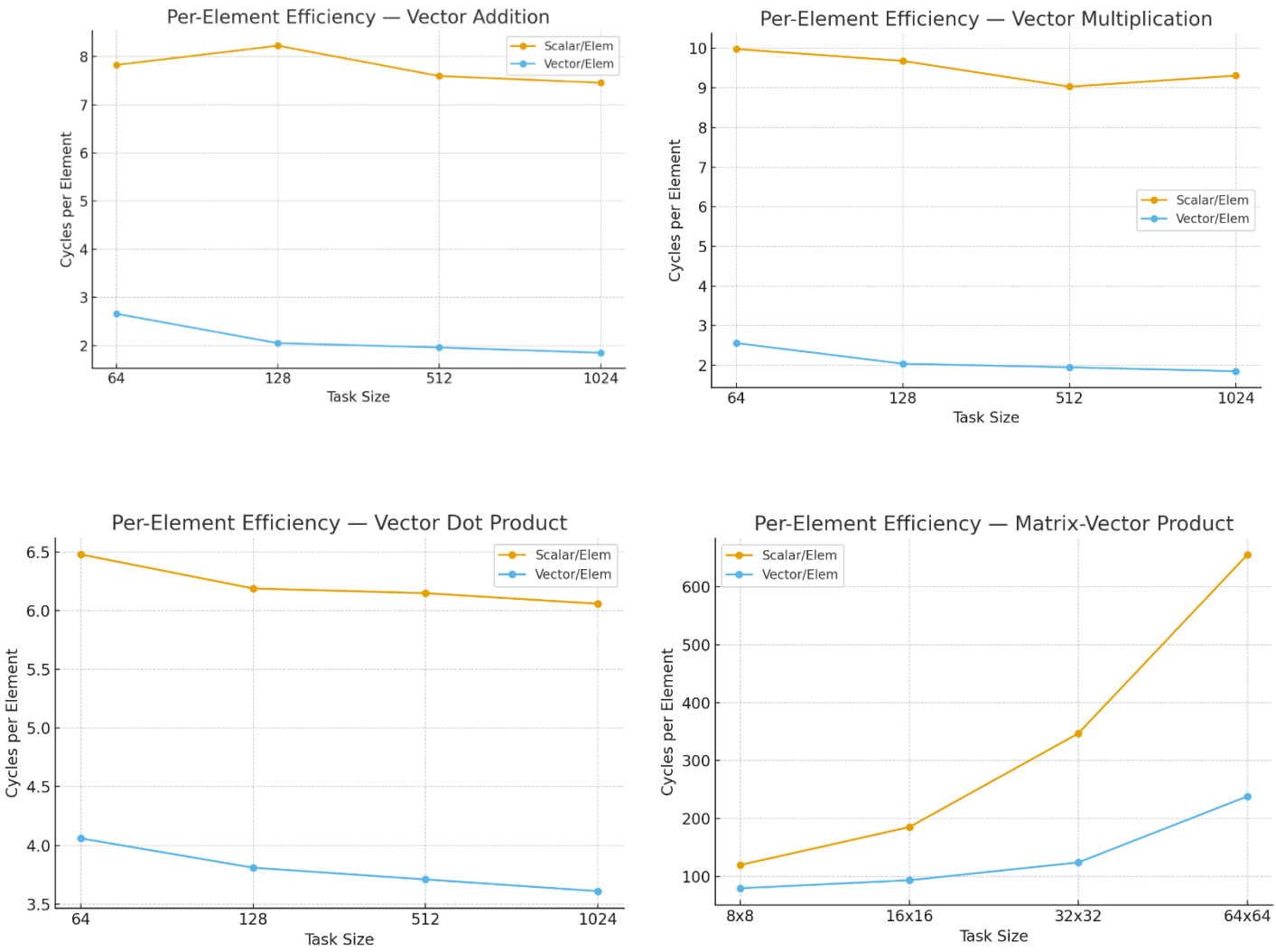


Figure 4. Plots based on Table. 1, showing speedup, absolute cycles, and per-element efficiency.

2.4.2 OpenBLAS evaluation

To ensure the compliance of the RISCV Vector extension Codasip run multiple tests during the development of the VPU in the FPGA emulation. While these tests are not benchmark on their own, they provide the interesting and important view of the capabilities of the Developed HW.

2.4.2.1 Experimental Setup

The experiments were run on the Linux emulation platform described in this document. The tests were compiled as a Linux user space applications and they were executed from console. Each test was responsible for measuring its own execution time, which was reported at the end of the test.

2.4.2.2 Benchmark evaluation

The openBLAS unit Test are testing a large part of the openBLAS capabilities and as such we used them to evaluate the correctness of our VPU prototype together with the extensive verification in the simulation.

The openBLAS test measures the time needed to execute the given test set. Therefore, it is possible to use them as crude benchmarks. We compiled the two version of the openBLAS library. Scalar version was compiler for the *GENERIC RISCV* target and generated test binaries were copied on the file system of the emulation platform with `suffix _scalar`.

```
make TARGET=RISCV64_GENERIC CFLAGS="-DTARGET=RISCV64_GENERIC"
BINARY=64 ARCH=riscv64 NOFORTRAN=1 CC=riscv64-unknown-linux-gnu-gcc
AR=riscv64-unknown-linux-gnu-ar AS=riscv64-unknown-linux-gnu-as
LD=riscv64-unknown-linux-gnu-ld FC=riscv64-unknown-linux-gnu-gfortran
HOSTCC=gcc HOSTFC=gfortran -j32
```

The second version was compiled for the RISCV64_ZVL128B target, which is the GENERIC RISCV target with Vector extension support. The names of vector binaries were given `suffix _vector` and copied to the same location as their scalar counterparts.

```
make TARGET=RISCV64_ZVL128B CFLAGS="-DTARGET=RISCV64_ZVL128B"
BINARY=64 ARCH=riscv64 NOFORTRAN=1 CC=riscv64-unknown-linux-gnu-gcc
AR=riscv64-unknown-linux-gnu-ar AS=riscv64-unknown-linux-gnu-as
LD=riscv64-unknown-linux-gnu-ld FC=riscv64-unknown-linux-gnu-gfortran
HOSTCC=gcc HOSTFC=gfortran -j32
```

The openBLAS version v0.3.30 source codes were compiled by the GNU Compiler Collection version 15.1.0

The benchmarking is controlled by the script that can be run from command line. This script executes each `testset` in the `utest` and `utest_ext` and forwards `stdout` of the tests into the log file. Every command is run two times and only the second time results are reported. This ensures that every test binary is available in the filesystem cache and therefore the runtime is not affected by the time needed to access SD Card.

2.4.2.3 Benchmarking results

Every of the openBLAS tests reports the time spend in the data preparation and the computation itself. The following table summarizes the time for the tests run on the FPGA platform. Due to the nature of the emulation, the Idefix core was running on 40MHz while the final ASIC is expected to run in the range of 1 to 1.7 GHz. Therefore, if run on the final product, the tests would run at least 40 times faster.

Test	Scalar [ms]	Vector [ms]	Speedup [x]	Delta Time [%]	Share [%]	BLAS Level	Datatype
fork	515435	295913	1.741846421	-42.58965728	83.48788504	1	
zgemm	12923	10872	1.188649742	-15.8709278	2.093210469	3	z (complex128)

cgemm	11196	7256	1.542998897	-35.19113969	1.813478636	3	c (complex64)
ztrsv	7989	7598	1.051460911	-4.894229566	1.294022939	2	z (complex128)
ztrmv	7906	7596	1.040810953	-3.921072603	1.280578966	1	z (complex128)
zgemmt	7202	5662	1.271988697	-21.38294918	1.166548155	3	z (complex128)
ctrmv	6482	5680	1.141197183	-12.37272447	1.049925734	1	c (complex64)
ctrsv	6420	5616	1.143162393	-12.52336449	1.039883248	2	c (complex64)
cgemmt	6196	4088	1.515655577	-34.02194964	1.003600717	3	c (complex64)
dgemmt	3662	2867	1.277293338	-21.70944839	0.593154588	3	d (float64)
sgemmt	3224	2011	1.603182496	-37.62406948	0.522209282	3	s (float32)
kernel_regress	2790	2298	1.214099217	-17.6344086	0.451911879	1	
zimatcopy	2557	2426	1.053998351	-5.12319124	0.414171568	1	z (complex128)
cimatcopy	2179	1976	1.102732794	-9.316200092	0.352944797	1	c (complex64)
zgeadd	2096	2193	0.955768354	4.627862595	0.339500824	1	z (complex128)
zgemv	2022	1962	1.03058104	-2.96735905	0.32751463	2	z (complex128)
cgeadd	1833	1651	1.11023622	-9.929078014	0.296901245	1	c (complex64)
cgemv	1707	1546	1.104139715	-9.431751611	0.276492322	2	c (complex64)
zomatcopy	1536	1498	1.025367156	-2.473958333	0.248794497	1	z (complex128)
comatcopy	1309	1250	1.0472	-4.507257448	0.212026039	1	c (complex64)
dimatcopy	1282	1230	1.042276423	-4.056162246	0.207652698	1	d (float64)
dgeadd	1168	1085	1.076497696	-7.106164384	0.189187482	1	d (float64)
simatcopy	1084	1030	1.052427184	-4.981549815	0.175581533	1	s (float32)
sgeadd	929	864	1.075231481	-6.996770721	0.150475317	1	s (float32)
zsbmv	880	791	1.112515803	-10.11363636	0.142538514	1	z (complex128)
csbmv	774	677	1.143279173	-12.53229974	0.125369102	1	c (complex64)
domatcopy	713	678	1.051622419	-4.908835905	0.115488591	1	d (float64)
zspmv	704	596	1.181208054	-15.34090909	0.114030811	2	z (complex128)
cspmv	602	457	1.317286652	-24.08637874	0.097509301	2	c (complex64)
somatcopy	590	556	1.061151079	-5.762711864	0.095565594	1	s (float32)
zgbmv	173	168	1.029761905	-2.89017341	0.028021776	2	z (complex128)
cgbmv	144	139	1.035971223	-3.472222222	0.023324484	2	c (complex64)
zrot	132	122	1.081967213	-7.575757576	0.021380777	1	z (complex128)
crot	110	107	1.028037383	-2.727272727	0.017817314	1	c (complex64)
zaxpby	107	115	0.930434783	7.476635514	0.017331387	1	z (complex128)
caxpby	101	98	1.030612245	-2.97029703	0.016359534	1	c (complex64)
daxpby	74	70	1.057142857	-5.405405405	0.011986193	1	d (float64)
saxpby	70	67	1.044776119	-4.285714286	0.011338291	1	s (float32)
idamin	60	45	1.333333333	-25	0.009718535	1	
zaxpyc	53	54	0.981481481	1.886792453	0.008584706	1	z (complex128)

caxpyc	49	55	0.890909091	12.24489796	0.007936804	1	c (complex64)
isamin	46	47	0.978723404	2.173913043	0.007450877	1	
izamin	45	44	1.022727273	-2.222222222	0.007288901	1	
icamin	44	43	1.023255814	-2.272727273	0.007126926	1	
samin	44	40	1.1	-9.090909091	0.007126926	1	s (float32)
scsum	44	42	1.047619048	-4.545454545	0.007126926	1	s (float32)
crotg	43	40	1.075	-6.976744186	0.00696495	1	c (complex64)
damin	43	39	1.102564103	-9.302325581	0.00696495	1	d (float64)
dsum	43	41	1.048780488	-4.651162791	0.00696495	1	d (float64)
dzsum	43	42	1.023809524	-2.325581395	0.00696495	1	d (float64)
ssum	43	42	1.023809524	-2.325581395	0.00696495	1	s (float32)
drotmg	42	36	1.166666667	-14.28571429	0.006802975	1	d (float64)
srotmg	41	39	1.051282051	-4.87804878	0.006640999	1	s (float32)
zscal	41	50	0.82	21.95121951	0.006640999	1	z (complex128)
cscal	40	45	0.888888889	12.5	0.006479023	1	c (complex64)
dzamax	40	42	0.952380952	5	0.006479023	1	d (float64)
zrotg	40	41	0.975609756	2.5	0.006479023	1	z (complex128)
dzamin	39	42	0.928571429	7.692307692	0.006317048	1	d (float64)
scamax	37	44	0.840909091	18.91891892	0.005993097	1	s (float32)
scamin	37	40	0.925	8.108108108	0.005993097	1	s (float32)
potrf	22	21	1.047619048	-4.545454545	0.003563463	1	
axpby	16	17	0.941176471	6.25	0.002591609	1	
amax	15	5	3	-66.66666667	0.002429634	1	
dsdot	14	4	3.5	-71.42857143	0.002267658	1	d (float64)
sscal	5	16	0.3125	220	0.000809878	1	s (float32)
amin	4	4	1	0	0.000647902	1	
axpy	4	5	0.8	25	0.000647902	1	
dgemv	4	15	0.266666667	275	0.000647902	2	d (float64)
dnrm2	4	4	1	0	0.000647902	1	d (float64)
dscal	4	16	0.25	300	0.000647902	1	d (float64)
ismax	4	14	0.285714286	250	0.000647902	1	
rot	4	5	0.8	25	0.000647902	1	
swap	4	5	0.8	25	0.000647902	1	s (float32)
ismin	3	3	1	0	0.000485927	1	
max	3	16	0.1875	433.3333333	0.000485927	1	
min	3	4	0.75	33.33333333	0.000485927	1	
sgemv	3	3	1	0	0.000485927	2	s (float32)
zdotu	3	4	0.75	33.33333333	0.000485927	1	z (complex128)

Table 2. OpenBLAS unit tests execution times

Table 2. column descriptions:

1. **Test**
   - Name of the benchmarked kernel or operation (e.g., cgemm, dgemv).
   - Used as the row identifier.
2. **Scalar**
   - Baseline runtime of the kernel in milliseconds when compiled/executed without vectorization.
   - Unit: ms.
3. **Vector**
   - Runtime of the same kernel in milliseconds using the vectorized implementation.
   - Unit: ms.
4. **Speedup**
   - How many times faster the vector version is versus scalar.
   - Formula: `Speedup_x = Scalar_ms / Vector_ms.`
   - Interpretation: >1.0 = vector is faster; <1.0 = regression.
5. **Delta Time**
   - Percent change in runtime from scalar to vector.
   - Formula: `DeltaTime_pct = ((Vector_ms − Scalar_ms) / Scalar_ms) × 100.`
   - Interpretation: negative = improvement (time decreased); positive = slower.
6. **Share**
   - The kernel's share of the total scalar runtime, showing its weight in the overall workload.
   - Formula: `ShareTotalScalar_pct = (Scalar_ms / Σ Scalar_ms over all rows) × 100.`
   - Interpretation: higher values indicate greater impact on end-to-end time.
7. **Level**
   - BLAS-level categorization inferred from the operation name:
     - Level-1 (vector ops),
     - Level-2 (matrix-vector),
     - Level-3 (matrix-matrix).
   - Helps group performance characteristics.
8. **Datatype**
   - Numeric type inferred from the test name prefix:
     - s (float32),
     - d (float64),
     - c (complex64),
     - z (complex128).
   - Useful for spotting precision- or type-specific trends.

Table 2 shows that not all tests benefit strongly from the vector extension; the aggregated speedup across all tests is 1.6×, below the 2× target. This is expected because the reported times include setup and test-selection overheads that make little use of vector instructions. In addition, the unit tests are written to run as fast as possible and therefore use very small inputs. As seen in the simulation study, larger inputs tend to increase speedup, and the same should hold for these tests.

Timing was recorded in milliseconds using `getCurrentTime`. On the 40 MHz emulation platform, 1 ms ≈ 40,000 cycles, which is a coarse resolution for short test cases and may distort measurements of small runtimes. A cycle counter or higher-resolution timer would provide more accurate results.

Overall, the 1.6× speedup on the OpenBLAS unit tests indicates that the library does make substantive use of the VPU and should yield gains in realistic workloads. However, these unit tests are not designed to explore the VPU's performance limits, so they do not by themselves establish its practical ceiling.

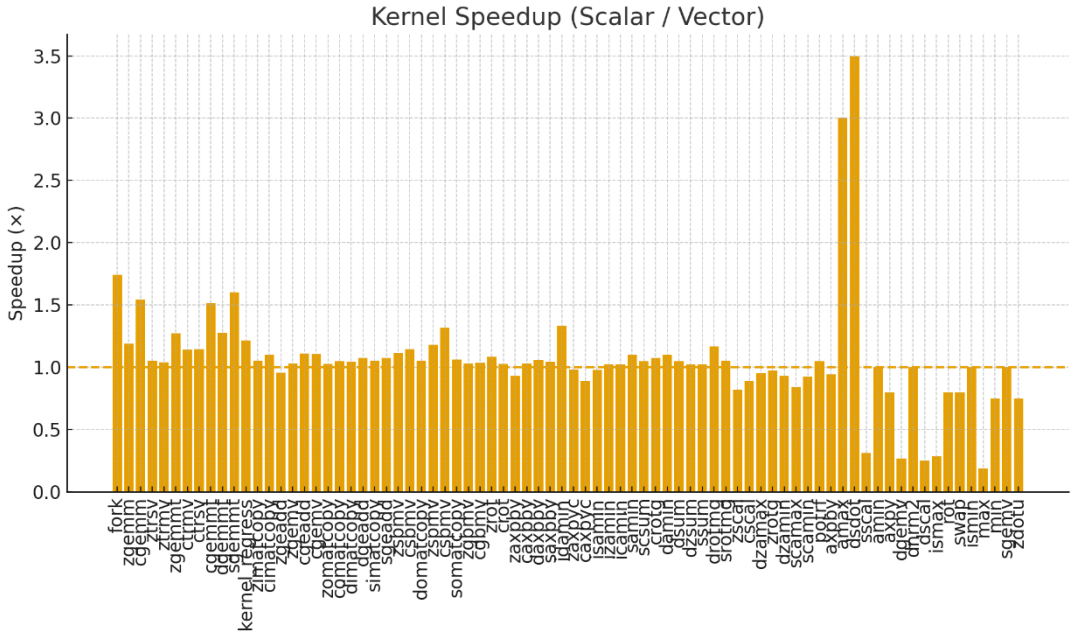


Figure 5 - Kernel-level Speedup (Scalar ÷ Vector), overview of Tab. 2 results.

2.4.3 Dot product emulation benchmark

The benefits of the vectorization become apparent for the problems with larger datasets. To evaluate behaviour of the VPU on the large problem, we prepared simple dot product test.

2.4.3.1 Experimental Setup

The experiments were run on the Linux emulation platform described in this document. The tests were compiled as a Linux user space applications and they were executed from console. Each test was responsible for measuring its own execution time, which was reported at the end of the test.

2.4.3.2 Benchmark evaluation

The test runs three implementations of the dot product written in the combination of C code and RISCV assembly. The C code handles the data preparation, time measurement and printing the results. The scalar version of the dot product is also implemented in C. Both vectorized implementation of dot product are written in the assembly and linked with main C function in the linker phase. This is different from the simulation tests where all functions were written in C and vectorized part were introduced by the inline assembly.

The C code generates two random vectors of the length 512000 elements. Each element is of float type. The dot vector function is called inside for loop which ensures that every operation is repeated 1000 times. The number of clock cycles necessary to compute all iterations are measured by calling `clock()` function. The selected time measurement method relies on the Linux OS and its handling of time. It is not as precise as the clock cycle measurement performed in the simulation environment. Therefore, the experiments were run on the large amount of data. Moreover, the obtained data are used to compute speed up between different implementation of the same function. Since all variants were measured by exact same technique on the same emulation, the conclusion of speed ups remains valid, even if there are systematic error in the measurements. The clock cycle values from this benchmark should not be compared with the clock cycles obtained from the different benchmarks.

The vectorized implementation relies on the assumption that the order of the summation can be changed. Therefore, instead of adding results of the vector multiplication in one floating point register, we add the

results in the vector register and the final result is computed by the reduction after both input vectors are processed. The LMUL setting allows the VPU to group several vector registers into one. The grouping of registers does not increase number of the operation computed in one clock cycle, but it prolongs each vector instruction.

2.4.3.3 Benchmark results

The scalar implementation finished in 533 197 968 clock cycles. The Vector variant with LMUL equals one computed result in 99 020 236 clock cycles which indicates 5.3x speed up. Such speed up proves that VPU is fully utilized during the dot product computation. The vector registers are 128bit wide which means that they can execute operation on 4 elements simultaneously. Moreover, number of iterations of the computation look is reduce four times, which means that the overhead of the computation loop is also reduced.

The vector tests for LMUL8 provides further speed up. Increasing the LMUL causes further reduction of number of iterations of the computation loop eight times. However, for every vector instruction, the VPU runs small loop in hardware to work on all vector registers. The observed speed up point to the effective design of the VPU sequencer, which generates sequences of micro-operation.

Test Run	Clock Cycles	Speedup vs Scalar
Vector result (LMUL8)	62 150 645	8.59677419354838
Vector result (LMUL1)	99 020 236	5.38383838383838
Scalar result	533 197 968	1

Table 3. Dot-Product Macro Benchmark: Clock Cycles and Speedup

On the measured dot product, the VPU obtained speed up of 8.58x. This proves that the VPU meets the KPI targeted in the SYCLOPS project.

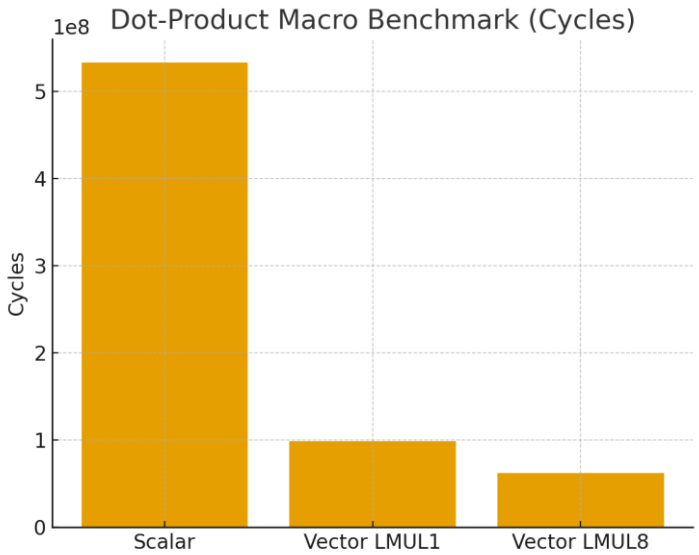


Figure 6. Dot-Product Macro Benchmark (Clock Cycles)

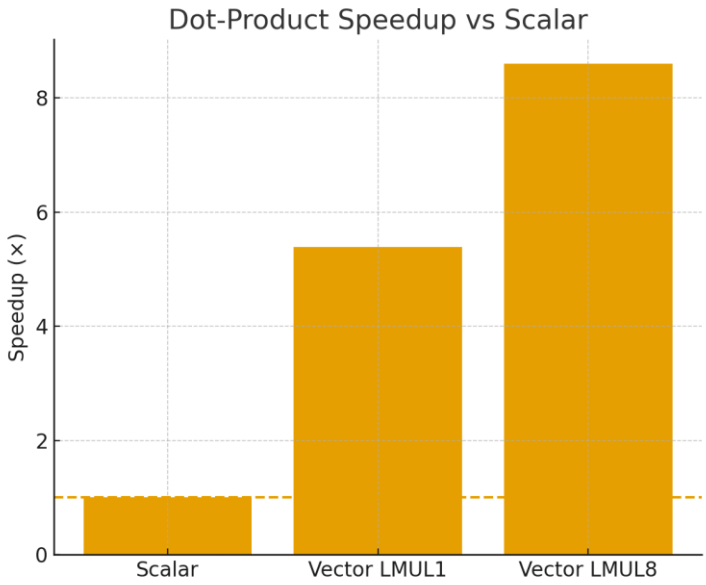


Figure 7. Dot-Product Speedup vs Scalar

2.5 Summary

The VPU experiments indicate that, with appropriate software, the platform meets the project's KPI. These measurements are an initial evaluation; more precise and robust benchmarking is planned as the platform matures in the final phase of SYCLOPS. All tests used single-precision floats, so repeating the experiments with other data types will be needed for a fuller assessment. Broader testing and performance evaluation will form part of the future work to raise the prototype VPU's TRL.

3. SYCLARA Open Source RISC-V Platform

In parallel with the CSIP RISC-V platform, EUR worked on developing an open-source RISC-V vector extension platform called SYCLARA. SYCLARA is an end-to-end, hardware–software platform that can be used as a testbed to evaluate SYCL and RISC-V implementations together. In particular, we built upon the recent integration of the CVA6 RISC-V core [1] with ARA2 RVV accelerator [2] and extended it by adding Ethernet capability and enabling Linux boot from an SD card to bring up an RVV accelerator on the VCU118 platform deployed as a part of EMDC v2.0 at EUR.

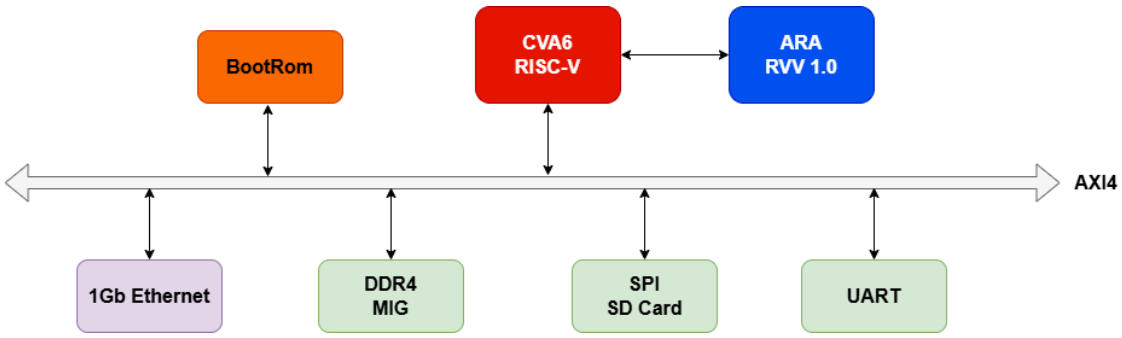


Figure 8. SYCLARA RISC-V platform

3.1 SYCLARA Open Source RISC-V Platform

Figure 8 shows a high-level overview of the architecture of the SoC we have adapted and brought up. It is based on the Cheshire SoC developed by the PULP platform and integrates CVA6, an open- source, high performance RISC-V processor core that implements the RV64GC ISA, with ARA, an open-source implementation of the RVV. The first integration of ARA with CVA6 was not capable of running Linux applications because ARA lacks an MMU. To overcome this limitation, we adopted a patched version of ARA2 and CVA6 where the MMU was shared between CVA6 and ARA. We brought up this integration on VCU118 by further adapting the DDR4 memory controller and using an SD card to hold the Linux image.

In order to offload SYCL kernels to our hardware platform, we rely on DPC++ and the recently open-sourced OCK. In particular, we used the OCK remote HAL, which provides a server and an OpenCL client that can be run on any standard Linux installation, and use socket connections to communicate with each other. This HAL relies on the accelerator being equipped with an Ethernet interface. Thus, we integrated an Ethernet module into the SoC to enable interaction with CVA6 and ARA from a SYCL host. In a study by Chaoqun Liang et al.[1], Gigabit Ethernet was implemented and integrated into the Cheshire SoC and tested on Genesys2 and VCU118. However, they used an external Ethernet adapter with an Ethernet PHY and an RGMII interface. In contrast, we used the Xilinx 1G/2.5G BASE-X PCS/PMA or SGMII core, which converts the GMII interface to SGMII, the interface supported by the Ethernet PHY chip on the VCU118. For the MAC layer, we used an existing MAC 2. To overcome the clock domain crossing between the AXI interconnect and the Ethernet IP, a FIFO was used to bridge the AXI data. The Ethernet driver was adapted from the lowRISC Ethernet driver. We installed the driver and its dependencies on Linux using Buildroot, incorporating patch files for the CVA6-SDK and enable SSH support.

We cross-compiled the HAL server for execution on CVA6. We compiled the HAL client to run on a local x86-64 machine. The default HAL client setup (targeting RV64GC) does not generate any vector instructions. Thus, we configured the HAL client manually by enabling the vector extension in the HAL

[1] Chaoqun Liang, Alessandro Ottaviano, Thomas Benz, Mattia Sinigaglia, Luca Benini, Angelo Garofalo, and Davide Rossi. 2024. A Gigabit, DMA-enhanced Open-Source Ethernet Controller for Mixed-Criticality Systems. In Proceedings of the 21st ACM International Conference on Computing Frontiers: Workshops and Special Sessions. 55–58

device during compilation. Additionally, the vectorization factor can be adjusted using the environment variable CA_RISCV_VF. This variable acts as a multiplier for vectorization levels, allowing control over the degree of vectorization.

3.2 Evaluation

Having described the SYCLARA hardware–software stack, in this section, we present our experimental evaluation. The SoC has been implemented on the VCU118 evaluation board, which features a Xilinx Virtex UltraScale+ FPGA. CVA6 and ARA were clocked to run at 50 MHz. A higher clock rate was tested, but we encountered timing violations in CVA6 and ARA. The ARA configuration includes 2 lanes, and for the experiments reported here, we set the vector length (VLEN) to 2048. Table below presents the implementation results for the SoC, CVA6, ARA and Ethernet, showcasing resource usage such as LUTs, BRAMs, and DSPs.

Resource usage	LUT	BRAM	DSP
SOC	318884	154	132
CVA6	47114	60	27
ARA	129512	32	102
Ethernet	1652	10	0

Table 4. SYCLARA Implementation results

To evaluate the performance improvement of RVV compared to non-RVV, we tested two SYCL kernels: 1D Convolution (conv), where we convolve two arrays with 4k entries, and matrix multiplication (matmul) of (150, 300) × (300, 600) matrices, as shown in the table below. We executed the two kernels on CVA6 both in scalar mode and vector using ARA. Clearly, these results demonstrate that SYCL compiler and runtime toolchains are capable of exploiting RISC-V vector accelerators to improve performance, as the 1D convolution execution time improves up to 6.27×, and matrix multiplication improves up to 6×. However, increasing the vectorization factor (CA_RISCV_VF) does not have a uniform impact across all applications. For instance, conv achieves the best performance with a VF of 16, but matmul does not improve beyond VF of 8.

	Scalar		VF=4		VF=8		VF=16	
	Conv	Matmul	Conv	Matmul	Conv	Matmul	Conv	Matmul
int32	26775	139909	19276	40008	10918	23351	6322	145803
float	28975	141210	21542	44683	11796	25508	6897	145619
double	47499	187428	22943	51325	12881	29528	7571	191197

Table 4. SYCLARA evaluation results

3.3 Summary

In this work, we provided an overview of our SYCLARA platform that builds on OCK to offload SYCL computations on the ARA2 RVV accelerator integrated with the CVA6 RISC-V CPU. Using SYCLARA, we presented a preliminary evaluation of a few kernels to demonstrate that SYCL compiler toolchains developed in SYCLOPS can exploit RVV to improve performance on real RISC-V hardware. Given the lack of RVV implementations, we believe that SYCLARA provides a valuable framework for furthering research on open, standards-based hardware acceleration of analytics and AI.

4. CXL-enabled EMDC Testbed

While sections 2 and 3 described the work done in the context of Task 3.2, this section describes the work done in the context of “Task 3.3: EMDC Assembly”. HIRO has developed a CXL-enabled EMDC testbed in SYCLOPS project. Figure 9 shows the hardware architecture of the EMDC, and Figure 10 is a preliminary 3D rendering of a possible module arrangement showing key components, starting with (1) multiple COM-HPC module that provides the central compute foundation. It is interconnected with (2) an Ethernet switch and (3) a PCIe switch to enable high-bandwidth connectivity and data routing. Power conversion for 12V only modules is handled by (4) a 48V to 12V converter, allowing low power modules to integrate. Compute acceleration or processing flexibility is achieved through (5) PIC64-HPSC or Nvidia Orin modules. For networking and centralized management interfaces, the design incorporates (6) OCP-NIC 3.0 and (7) OCP-BMC DC-SCM 2.0 standard modules. Storage requirements are met with (8) EDSFF E1.S SSDs, while CXL memory expansion is supported by (9) EDSFF E3.S CXL memory modules.

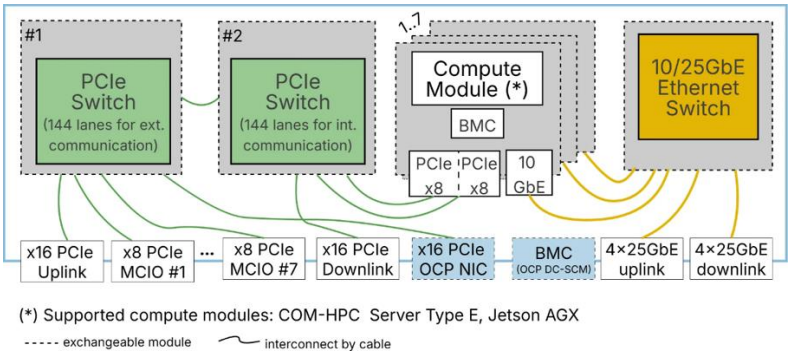


Figure 9: Overview of the EMDC architecture, based on custom form factors for latter adoption to established form factors like COM-HPC, COM Express, Orin AGX.

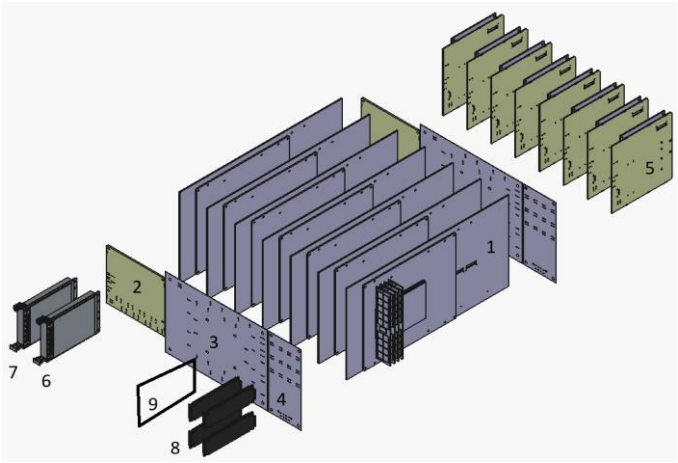


Figure 10: Preliminary 3D-rendering of the EMDC platform.

4.1 PCIe CXL Switch Development

The PCIe/CXL switch is a cornerstone of the project providing the high-bandwidth fabric required to interconnect heterogeneous compute, storage, and accelerator resources. Its role is to replace rigid, monolithic server architectures with a composable infrastructure where modules can be flexibly pooled, reassigned, and orchestrated according to workload demand.

While traditional PCIe deployments have been limited to host-to-device topologies, the rise of large-scale data fabrics and AI workloads requires far greater flexibility. The evolution to PCIe Gen6 and Compute Express Link (CXL) 3.0 represents a decisive step. PCIe Gen6 doubles the raw lane speed to 64 GT/s,

delivering up to 128 GB/s per x16 slot in one direction and ~256 GB/s bidirectional throughput, a performance class aligned with 800 Gb Ethernet and state-of-the-art accelerators. This is achieved not by merely pushing clock speeds, but by adopting PAM-4 modulation, forward-error correction (FEC), and a fixed 256-byte FLIT framing model. These innovations keep error rates under control at multi-tens of gigahertz signaling, while preserving software transparency: the same drivers that worked on PCIe Gen3 continue to function unmodified.

CXL builds on the PCIe physical layer, but extends it with cache-coherent protocols. In addition to CXL.io for I/O compatibility, CXL.cache and CXL.mem allow hosts and accelerators to share memory directly, eliminating the latency and complexity of software-based emulation layers. This makes it possible to build memory pools accessible across multiple nodes, a key enabler for disaggregated infrastructures.

4.1.1 Architecture and Implementation

The EMDC integrates switches from the Broadcom PEX90000 family, currently the most advanced PCIe Gen6 silicon available and the first with roadmap support for CXL 3.0. Devices like the PEX90144 offer up to 144 lanes, sufficient to tie together CPU modules, GPU accelerators, FPGA boards, NVMe SSDs, and network cards within a single composable chassis. Connectivity is realized through MCIO cabling, which supports PCIe Gen6's signal integrity requirements while remaining mechanically flexible. An x8 MCIO link can be split into two x4s for SSDs, or combined into an x16 link for high-throughput accelerators. The switch supports partitioning and multi-host operation, allowing independent root complexes to coexist while still sharing selected resources.

Figure 11 shows the hardware architecture of the PCIe switch module that enable high speed data transfer and module interconnectivity between CPU nodes and devices. Figure 12 shows the preliminary 3D rendering of the PCIe switch.

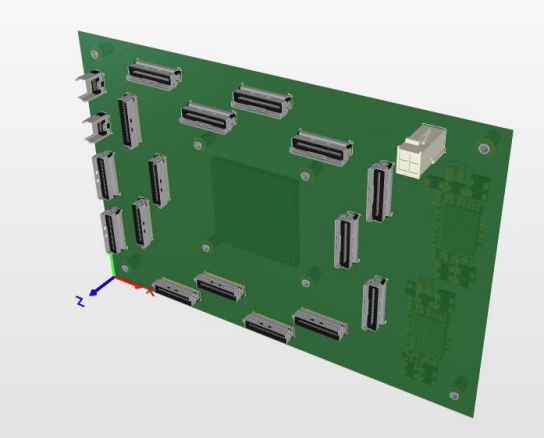


Figure 11: Preliminary 3D of the PCIe/CXL switch board.

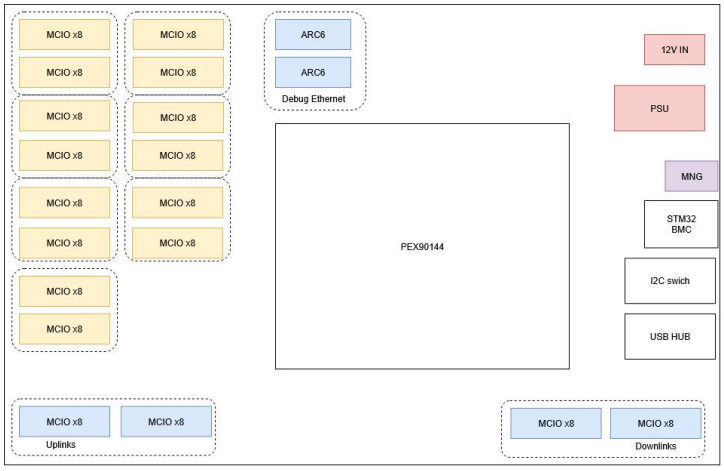


Figure 12: Block Diagram of the PCIe/CXL switch board.

4.1.2 Prototyping and Validation

The project's PCI Express (PCIe) evaluation platform is built around Broadcom's PEX 90000 switch family, one of the earliest silicon implementations to natively support the full PCIe 6.0 specification, including 64 GT/s data rates, FLIT-mode packetization, and PAM-4 signaling. By standardizing on this switch family, every hardware substrate developed in the project, both the Embedded Micro Data Center (EMDC) sleds and the embedded High-Performance Server (eHPS) blades, shares an identical, forward-compatible PCIe backbone.

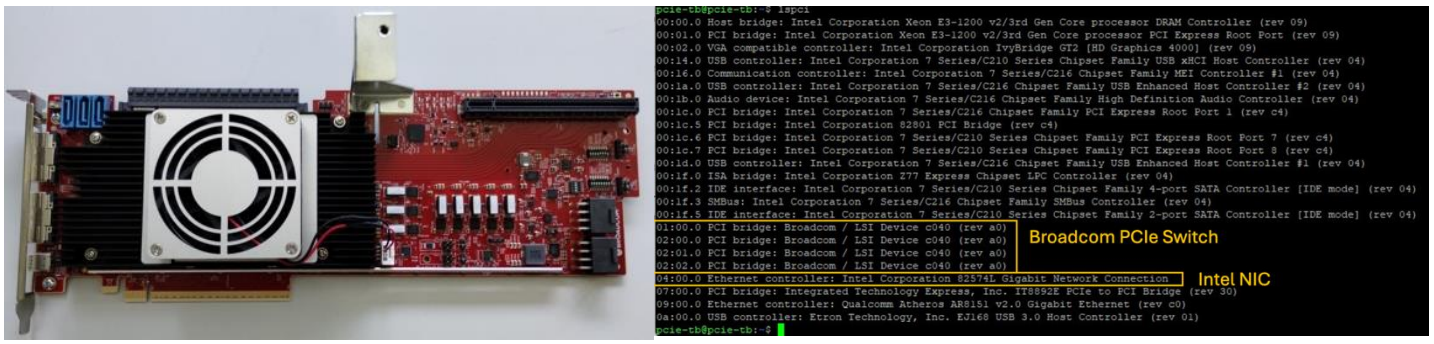


Figure 13: Testbed photos

4.1.2.1 HIB and RDK architecture

Prior to embedding the switches on production boards, a multi-stage testbed was assembled to validate the switch architecture before embedding into production EMDC sleds as shown in Figure 13. The setup combines Host Interface Boards (HIBs) that connect a server root complex to the switch, and Rapid Development Kits (RDKs) that expose downstream ports for GPUs, FPGAs, and NVMe devices as shown in Figure 14. In more advanced topologies shown in Figure 13, three PEX90144 switches are interconnected, enabling multi-host stress tests and exploration of complex partitioning scenarios.

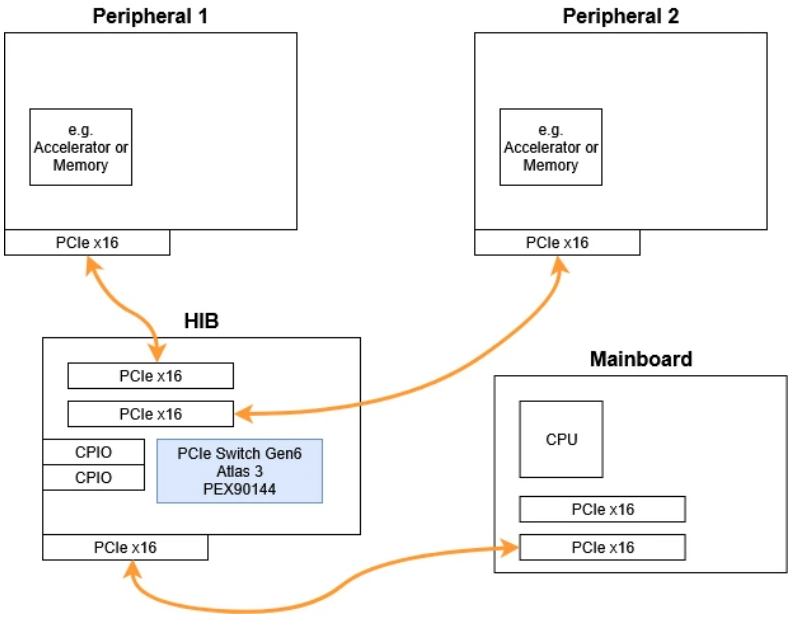


Figure14: PCIe testbed with basic functionality

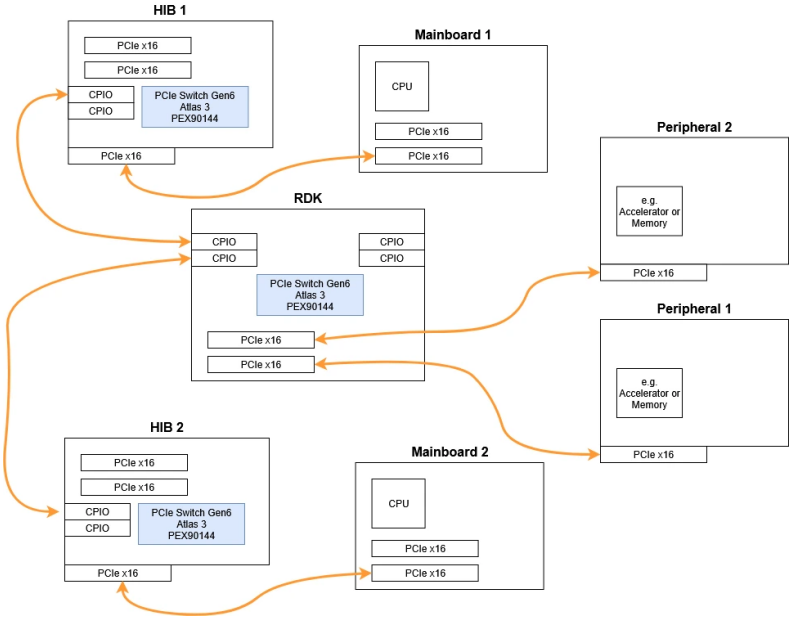


Figure 15: Multi-host PCIe testbed with HIB and RDK for full functionality

Validation efforts confirmed reliable 64 GT/s signaling with PAM-4, proper operation of FEC/CRC retry mechanisms, and stable link margining across high-loss channels. Host-to-host NTB communication was successfully tested within Linux, while early experiments with CXL decoders and ACPI table integration are underway. These steps ensure that when full CXL 3.0 support becomes available in silicon, the EMDC fabric will be ready to adopt it.

4.1.2.2 Production board manufacturing and integration

Based on the successful validation effort, we have already started the manufacturing of the actual PCB with the switch. We expect the actual hardware to be ready for bring up and testing by the end of October. We plan to test the hardware as follows. The validation of the PCIe/CXL switch module is designed to ensure both the physical performance of Gen6 signaling and the functional capabilities required for the EMDC fabric. Testing begins at the physical layer, where each port is powered and trained to 64 GT/s. Signal integrity is confirmed through on-die eye-diagram monitoring and PRBS stress patterns, while live lane-margining commands verify that all three PAM-4 “eyes” remain open under load. Forward-error

correction counters and CRC retries are observed during induced noise conditions to prove the robustness of the FLIT-mode encoding.

Once physical stability is secured, the switch is exercised in managed firmware mode. Using Broadcom's SDK, we access detailed packet analyzers and performance counters to validate FEC efficiency, port bifurcation, and Dynamic Port Reconfiguration. A key step is the demonstration of flexible MCIO connectivity, where ×8 links can be bifurcated into dual ×4s for EDSFF storage or combined into full ×16 paths for accelerators.

Throughput and latency are characterized with mixed device classes. Standard NVMe drives are benchmarked with fio and SPDK, while GPUs and FPGAs validate peer-to-peer transfers through the switch. In Gen6 mode the target envelope is ≈121 GB/s one-way on ×16 ports and up to ≈256 GB/s bidirectional, with host-to-device and device-to-device paths both measured. Latency overhead is expected in the tens of nanoseconds per hop, well within HPC and AI accelerator requirements.

Beyond raw bandwidth, the switch enables multi-host partitioning and Non-Transparent Bridging (NTB). Partitioning tests create multiple isolated root complexes with dedicated endpoint visibility, confirmed by configuration-space scans and controlled failover. NTB is validated by mapping memory windows and doorbells between two hosts, exposing the link as a virtual Ethernet device in Linux. These host-to-host transfers are expected to deliver tens of GB/s at microsecond-scale latency, demonstrating that the switch can serve as a low-latency inter-CPU fabric.

Finally, using both the testbed and the manufactured switch hardware, we plan to conduct integration tests that demonstrate the potential of PCIe v6.0 in accelerating SYCLOPS use cases.

4.2 CXL-enabled GPU server

The goal of our work on CXL-enabled PCIe switching is to research the next generating switching technology for our EMDC. Due to the low-level nature of this work, and the fact that while PCIe Gen6 switches are shipping, CXL 3.0 support is still limited in silicon and ecosystem software, the work in this task is not meant to be directly integrated with higher-level software.

However, there are already CXL 2.0 solutions available in the market, and the Linux kernel already provides a CXL subsystem to integrate such solutions. In order to provide SYCLOPS partners the ability to work on current CXL solutions, we also developed a stand-alone CXL server using COTS components in collaboration with Micron and Supermicro. The server is based on the Supermicro SSG-121E-NE3X12R platform, which is a dual-CPU server with 128 cores and 256GB of DRAM. The system incorporates a Micron CXL 128G DDR4 PCIe5 E3.S CXL module. This CXL device registers a physical size of 128 GB. The server is also equipped with a L40S GPU, making it possible for SYCLOPS partners to execute their cross-architecture application on both CPU and GPU, and on local DRAM and CXL memory, and investigate tradeoffs.

We worked together with Micron in configuring the server to get CXL memory recognized and supported for use by applications. The Micron device includes a mailbox and, crucially, operates in system-ram mode by default, which allows it to be used directly via `numactl`. In the dual-CPU environment, the server presents three NUMA nodes: nodes 0 and 1 correspond to the two CPUs, while the CXL device is mapped as NUMA node 2. The total memory capacity detected by the OS is 257,315 MB, with Node 0 having 63,853 MB, Node 1 having 62,390 MB, and Node 2 (CXL) initially reporting 131,072 MB.

A critical difficulty encountered during the initial configuration was the inability to reconfigure the CXL memory to work in devdax mode. The command `sudo daxctl reconfigure-device --mode=devdax dax0.0 --force` failed, reporting "Device or resource busy," thereby making it impossible to run benchmarks specifically designed for DAX mode. Furthermore, this unsuccessful reconfiguration attempt resulted in unexpected behaviour, where a portion of the memory blocks was put offline, causing the reported size of NUMA node 2 to drop from 131,072 MB to 94,208 MB. Another infrastructure issue involved the toolkit provided by Micron; the latest versions of `mxcli` did not return any output or function

at all, while the older version available in the `cxl-reskit` repository had limited functionality, only allowing connection to the module without the ability to pass commands or acquire logs. We worked with Micron on solving these issues to get CXL memory operational.

We also did preliminary performance testing that revealed varied results across different benchmarks. The Memory Latency Checker (MLC) showed mixed findings; latency between CPU node 0 and the CXL device (node 2) was surprisingly low, almost matching the latency between the two CPU nodes (0 and 1). However, the latency between CPU node 1 and CXL node 2 was significantly higher, recorded at almost 3.5 times longer than typical inter-CPU communication. The pointer chasing benchmark, Multichase, indicated that CXL performance was only slightly slower than DRAM (about 2 nanoseconds difference), confirming that the CXL memory was functional. In stark contrast, the STREAM benchmark results demonstrated that CXL memory exhibited approximately 10 times less memory bandwidth compared to DRAM, and average running times were 15 times longer for CXL.

The CXL server has been installed at the HIRO datacentre in Budapest, and we have made it available to SYCLOPS consortium members for research and development activities.

5. Conclusion

This deliverable concludes the work done in "Task 3.2: RISC-V reference platform" and "Task 3.3: EMDC assembly" of WP3 in the SYCOPS project, and documents the successful deployment and initial evaluation of the EMDC v2.0 infrastructure. Experiments on the proprietary CSIP Vector Processing Unit (VPU) confirmed that the platform meets the project's Key Performance Indicators (KPIs) by demonstrating significant speedups on vector and matrix operations. In parallel, using the SYCLARA open-source platform, we demonstrated that the SYCL compiler toolchains developed in WP4 of the SYCLOPS project are capable of leveraging RVV technology to enhance performance on real RISC-V hardware.

The infrastructure development focused on establishing a future-proof backbone through the PCIe/CXL switch, which is critical for constructing composable infrastructures. This foundational work ensures the EMDC fabric is prepared for the maturation of CXL, which will unlock capabilities such as true memory disaggregation, coherent accelerators, and dynamic orchestration across fabrics. This strategic development positions the HIRO EMDC at the forefront of the transition between current PCIe ecosystems and future coherent CXL datacenters.

In the final phase of the project, we are working on integrating the software developed in SYCLOPS for each use case, deploying them on the hardware described in this deliverable, performing end-to-end performance analysis, and demonstrating concrete improvements bought about by the SYCLOPS hardware—software as a whole to each application vertical.