MEEP

MareNostrum Experimental Exascale Platform

D6.4 -Full Emulation prototype release

Version 1.1

Document Information

Contract Number	946002
Project Website	https://meep-project.eu
Contractual Deadline	30/06/2023
Dissemination Level	Public (PU)
Nature	Others
Author	Teresa Cervero (BSC)
Contributors	Alexander Kropotov (BSC), Francelly Cano Ladino (BSC), Elias Perdomo (BSC), David Castells (BSC), Blanca Sabater (BSC), , José Oliver (BSC), Teresa Cervero (BSC)
Reviewers	Behzad Salami (BSC), John Davis (BSC)

The MEEP project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 946002. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Croatia, Turkey.

Change Log

Version	Description of Change
v0.1	Initial structure
v0.2	Completing all content
v0.3	FPGA-cluster section is extended and internal review
v1.0	Version after internal review
v1.1	Second internal review: Sections reorganization, extending FPGA Cluster section, results and conclusions.

ACKNOWLEDGEMENTS

The MEEP project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 946002. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Croatia, Turkey.

The partners in the project are BARCELONA SUPERCOMPUTING CENTER - CENTRO NACIONAL DE SUPERCOMPUTACION (BSC), FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING, UNIVERSITY OF ZAGREB (UNIZG-FER), & THE SCIENTIFIC AND TECHNOLOGICAL RESEARCH COUNCIL OF TURKEY, INFORMATICS AND INFORMATION SECURITY RESEARCH CENTER (TÜBITAK BILGEM).

The content of this document is the result of extensive discussions within the MEEP © Consortium as a whole.

Contents

Executive Summary

This document completes the descriptions of the whole Hardware Stack and presents the status of the MareNostrum Experimental Exascale Platform (MEEP) as a digital laboratory, in which all the activities that have been developed during the project, converge.

In accordance with the DoA, this deliverable demonstrates the readiness of all the activities developed in the different technical work packages (WP) during the lifetime of the MEEP project. On one hand, the baseline Accelerated Compute and Memory Engine (ACME) accelerator, developed in work package 4 (WP4), is implemented and validated as an emulated accelerator on MEEP. For this, different targeted platforms have been used:

1) the ones available in Phase 1 (a set of 6 servers, where 4 of them include a Xilinx Ultrascale U280 FPGA each, and 2 more a Xilinx Ultrascale U55c each), and

2) the one available in Phase 2 (the FPGA-cluster) to be used as digital laboratory.

On the other hand, the implementation of the different flavors of the ACME emulated accelerator have been done by using the tools developed in WP6:

1) the MEEP FPGA Shell. This tool provides a seamless communication wrapper to any design, in this case the ACME accelerator, for interacting with the host and/or any other FPGA,
2) the FPGA flow. A mechanism developed to automate the bitstream generation in a way that can be used as a pipe-clean process from RTL team, but also to schedule periodic releases of the RTL designs to guarantee the readiness and compatibility of all the programmed designs even after those have been updated; and
3) the FPGA tools. This is a set of tools specifically developed to facilitate scalability, reusability and maintenance of all the different activities required to deploy and use a bitstream: programmability, configurability and operability.

Finally, after the deployment of WP4 designs, using WP6 tools and infrastructure, WP5 developments are executed, starting from booting the image of the Operating System, and continuing with the execution of different applications. This relationship between WPs is shown in Figure 1.

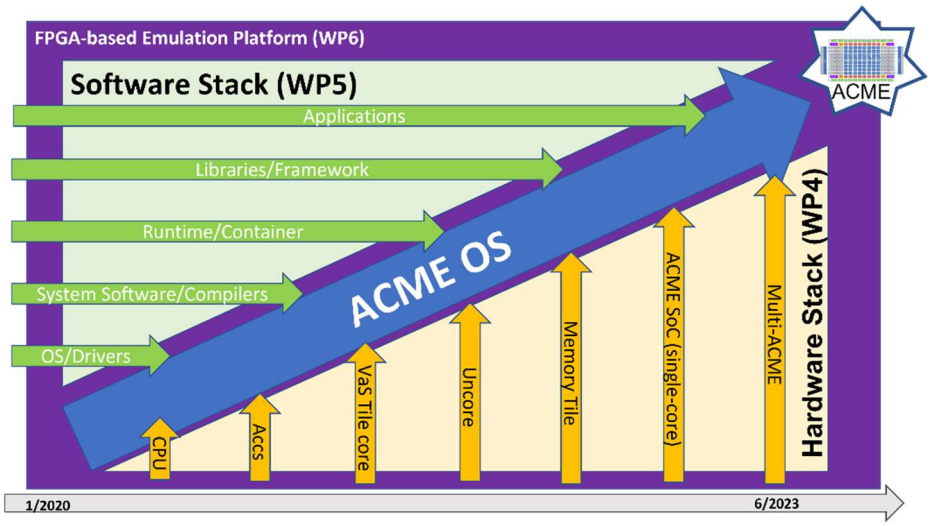


Figure 1. Relationship among technical work packages (WP4, WP5 and WP6)

This **D6.4 Full emulation prototype release (M42)** complements the previous deliverables:

- *D6.1 Emulation Platform specification (M6).*
- *D6.2 Emulated accelerator initial release (M18).*
- *D6.3 Emulated accelerator second release with full capability of inter-accelerator communication (M36).*

D6.4 Full emulation prototype release (M42) deeply explores and extends the communication capabilities between emulated accelerators implemented on the MEEP FPGA Shell (FPGA Shell from now on). This document describes a more advanced version of the second release of the FPGA Shell. This new version extends communication capabilities for any targeted emulated accelerator, although the scope of this document is constrained to the ACME accelerator developed in MEEP project.

In accordance with the objectives of the work package 6 (*WP6: FPGA programming/Tools support and Emulation Integration*):

- Aligned with the platform definition document (*D6.1 Emulation platform specifications*), this document presents the status of the Emulation Platform infrastructure, and the list of the tools associated to it (i.e., FPGA flow and FPGA tools). With respect to the FPGA flow, in this period MEEP project has maintained, and updated the workflow, refining the continuous integration and continuous development (CICD) infrastructure. Now more stages of the FPGA flow have been added to the process, and it is able to target different FPGAs (U280 or U55C) in different infrastructures (*Phase 1 or Phase 2*).
- The FPGA Shell has been extended, with respect to the previous versions by adding support for: 1) point-to-point communication using Aurora with DMA, 2) collecting information of the generated design, and making accessible to any user through the read of the InfoROM, and 3) maximizing the access of any accelerator to all the HBM channels.
- Regarding the ACME as an emulated accelerator, by using different configurations of the RTL design from WP4 as an input (including a many-core system), different versions of it have been implemented and deployed on two different Xilinx Ultrascale FPGA cards (i.e., U280 and U55C), targeting two different infrastructures (the one available in Phase 1, and the last one available in Phase 2). More information about the final ACME release is available in the deliverable *D4.3 Full RTL for FPGA final release*.

Introduction

In this deliverable **D6.4 Full emulation prototype release (M42),** *the full prototype will be made ready, running some instances with many scalar processors, other instances with scalar+vector. Second and final release of the FPGA Shell. The applications from WP5 will be executed in the emulation environment. And an evaluation of the self-hosted accelerator will be performed with a comparison to CPU-only, and when possible, GPU execution environments. The evaluation criteria will include performance, ease of programming and tuning.*

Moreover, this document completes the information related to the completion of the emulated platform bring-up; which includes completing the assembly, configuration, setup and running basics tests to guarantee the usability and operability of the machine as digital laboratory. The deployment of the ACME design as a Proof of Concept (PoC) has served as a mechanism to stress and tune some of the features of the system.

The document is structured as follows. Section 1 provides an overview of all the activities developed in the WP6 during the life of the MEEP project, showing a clear roadmap of all the activities from the moment the project started to its end.

Section 2 describes the updates on the FPGA Shell. It includes improvements on the point-to-point communication using Aurora, and addition to support multiple channels for accessing the HBM memory.

In Section 3 is reported the current status of the FPGA Tools developed during the lifetime of the project, paying special attention to the updates on the FPGA flow.

Section 4 provides an analysis of the different flavors of the ACME as an emulated accelerator deployed on the emulation platform.

Section 5 describes the final MEEP FPGA-cluster, and demonstrates its status after checking all its components, features and characteristics.

Section 6 summarizes the achievements of this technical workpackage at the end of the MEEP project. It also compares the results with the expected ones at the beginning of project, based on the proposed Key Performance Indicators (KPI).

1. Work Package 6 overview

MareNostrum Experimental Exascale Platform (MEEP) is intended to become a Digital Laboratory for exploring hardware/software co-design activities for European-developed IPs targeting the design for Exascale Supercomputers. Thus, two main functionalities are pursued:

- Being used as an evaluation platform for pre-silicon validation of IPs and ideas at speed and scale.
- Being used as a software development vehicle to enable software readiness for new hardware, whereas the architecture is not ready in silicon.

Within this vision, the mission of this WP6 is to guarantee the proper delivery of the FPGA-based emulation platform, which means the hardware infrastructure (racks, nodes, FPGAs…), and the associated tools for allowing accessing, operating with it and take advantage of its features.

At the moment of testing the correctness of any design, one of the main challenges that all hardware developers face is having to deal with topics that are beyond their expertise. In many cases the complexity relies on the communication IPs. All elements necessary to communicate with the proper design, this is the case of IPs like PCIe, or Ethernet among others. Despite there being some solutions in the market tackling this issue, when this MEEP project started they lacked flexibility and did not incorporate modern drivers to exploit more advanced features. Good examples of this lack of flexibility are the shells offered by Vitis[1] and Amazon Web Service (AWS)[2]. In both cases these shells are completely static, which means that all the offered IPs must be present all the time, no matter what the needs of the emulated design are. On top of that, it is not clear how to interact with those IPs, or how to exploit advanced features, like the case of the QDMA driver for PCIe. In this sense the FPGA Shell is proposed as a flexible, scalable, extensible, and easy to use option to hardware developers.

Since the emulation platform is based on FPGAs, MEEP project wants to offer a mechanism to let users operate in a seamless way with the infrastructure, just being focused on those parts of the design that they are interested in. This is the aim of the FPGA flow developed during the project.

[1] Vitis: https://www.xilinx.com/products/design-tools/vitis.html
[2] AWS: https://github.com/aws/aws-fpga

2.Updates on the MEEP FPGA Shell

From D4.3 deliverable, the FPGA Shell has been extended by adding the following capabilities:

- Related to main memory, HBM: 1) support of multiple memory controllers (Multi-MC), and 2) support of HBM characterization and exploration with the Memory Sandbox tool.
- Related to FPGA-to-FPGA communication: support for Aurora design, with DMA capabilities, to enable slow FPGA to FPGA communication.
- Related to emulated accelerators: 1) support for providing information about a design by using the InfoROM module, and 2) support for being compatible with AIT, and support its designs as emulated accelerators.

2.1. HBM improvements

HBM is an important module on the FPGA. It plays the role of main memory and enables the self-hosting capability for the ACME accelerator. The Alveo FPGAs U280 and U55C include two stacks, each of them with 8 memory controllers. The following improvements contribute to better understanding HBM capabilities and to exploit the maximum bandwidth of the HBM.

2.1.1. Support of Multiple Memory Controllers (Multi-MC)

HBM consists of multiple SDRAM cores each controlled by its own MC. The newest version of FPGA Shell supports an arbitrary number of AXI-MM channels for connecting an accelerator to multiple Memory Controllers (MC). Therefore, this feature is beneficial for enabling the high bandwidth nature of HBM due to the fact. Thus, utilization of multiple MC connections potentially increases concurrency of data exchange in a multi-core system through distribution of concurrent memory accesses over multiple MCs. Moreover, this improvement unlocks the Multi-MC option already available in ACME accelerator, ensuring fully compatibility from both sides, the FPGA Shell and the accelerator.

This Multi-MC feature extends the previous capabilities of the FPGA Shell, and consequently two options of Multi-MC usage can be used with any emulated accelerator, being used ACME as a PoC: 1) using only one MC for accessing to cacheable data in HBM, and 2) using multiple MCs.

The former case is shown in Figure 2, which presents a legacy way of connecting system memory in the OpenPiton framework (used for building ACME) - through the single up-left corner tile (as an example). For internal needs ACME requires two types of HBM connection: cached access for normal computations and non-cached for sharing data with DMA.

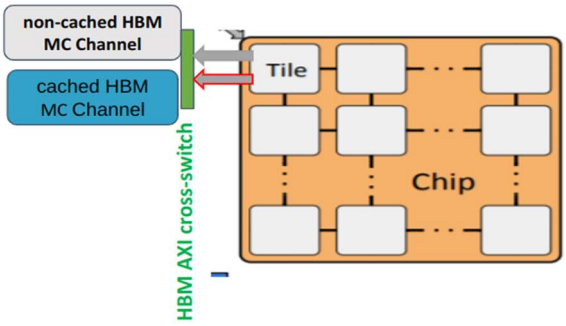


Figure 2: Connection of ACME single corner tile to the HBM of FPGA shell using two fixed MC channels.

In Figure 3 cached access is extended to multiple connections utilizing arbitrary configurable number of available edge tiles of ACME mesh. If multi-MC feature is enabled in ACME, "Minimum Manhattan distance" policy is used for routing memory accesses from any computing tile to an edge tile having MC connection. The FPGA Shell provides connection of those tiles to HBM within a physical limit of the number of HBM channels. The Multi-MC option is also reused in extension of ACME with "Memory Tiles". Extra required MC channels should be enabled in the main FPGA Shell configuration file, where all required interfaces are configured: *acceleartor_def.csv*.

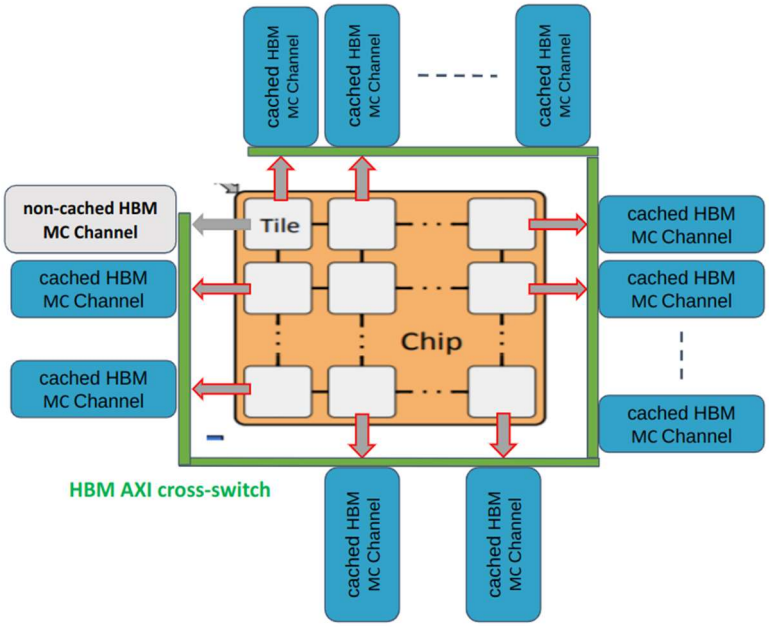


Figure 3: Connection of ACME multiple edge tiles to the HBM of FPGA Shell using an arbitrary number of MC channels.

More details about the impact of this feature on ACME are provided in Deliverable 5.4 *Final release of the software stack*, Section 3.1 *Hardware Co-design: HMB Tests with Stream*.

2.1.2. Support for Memory Sandbox Tool

An initial analysis of the performance characteristics of typical memory access patterns simplifes us to implement benchmarks to reveal the subjacent characteristics of HBM in FPGAs.

For this purpose, we emulate the typical sequential accesses with the widely used in FPGA programming: Repetitive Sequential Traversal (RST) access pattern and sparse accesses with pseudo-random accesses. Since DDR4 is the most popular memory used in computer architectures with FPGAs, the following sections present a comparison.

The Memory Sandbox tool is a tool conceived to help developers to better understand the intrinsic details of HBM on the FPGA. It provides higher configurability, insights, and control over measurements (e.g., clock cycles of each memory transaction). The tool is a configurable environment composed of two components: 1) a user interface for setting-up the experiments (front-end), and 2) a set of hardware IPs to run the experiments in the FPGA, according to the data introduced in the front-end (back-end). A block diagram of our tool is shown in Figure 4.

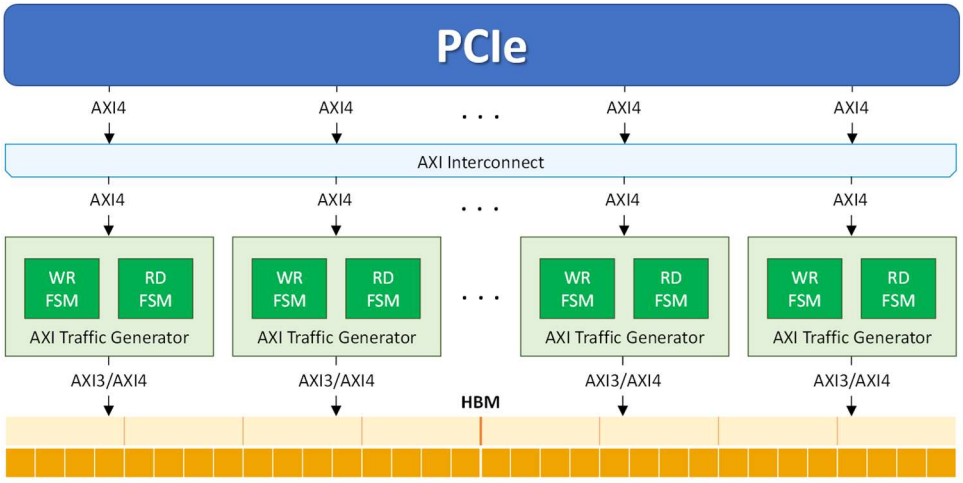


Figure 4. Hardware architecture of the Memory Sandbox Tool.

The front-end provides flexibility in terms of configuring run-time parameters. These parameters reduce the FPGA reconfiguration when a memory analysis is performed.

The back-end mimics processor threads data requests with sequential, or pseudo-random memory access patterns. This is done by the highly configurable AXI Traffic Generator IP, which minimizes its impact on the measured performance (throughput and latency) and guarantees that the data packages are only relative only to memory, whether it is HBM or DDR.

Our Memory Sandbox tool enables software readiness for new hardware and can be used to emulate a diverse set of architectures with different kinds of processing threads, and different access patterns.

The Memory Sandbox tool does not need to regenerate a bitstream to do explore the different features of the HBM, nor recode the IPs. This capability saves time on memory exploration. Therefore, with the same Memory Sandbox configuration (FPGA bitstream), a user might enable different experiments by changing parameters' values. The number of experiments for the same bitstream can be calculated based on the parameters goes to 2^{70}.

To validate the functionality of the Memory Sandbox tool, several experiments were conducted. The secondary goal of the experiments is to better understand the main differences between DDR4 memory and HBM on the FPGA. These experiments were performed on a Xilinx Alveo U280, since includes both kinds of memories, DDR4 and HBM. Table 1 summarizes the nature of those experiments, and the conclusions. More details about the experiments in Appendix A.

Table 1 Throughput analysis comparison between DDR4 and HBM, based on the address mapping policies.

Experiment	Description	Conclusions
Baseline throughput & address mapping policies	All pseudo-channels are accessed simultaneously. Sequential access. $Throughput = \sum_{1}^{16} PseudoChannelThroughput_i$ Address mapping policies (Table 2) DDR4 results: Figure 5 HBM results: Figure 6	• HBM memory controller handles two pseudo-channels without losing performance. • Best throughput = default policy. • Read and write transactions follow the same trend, regardless of the policy. • Best throughput: DDR4 = 6.18% <BW HBM = 0.01% <BW. • Worst throughput: DDR4: 92.02%<BW HBM: 56.39%<BW • At max. throughput: DDR4= 19.2 GB/s / bank HBM = 14.4GB/s / bank • DDR4 max throughput ≤ 4x HBM_banks.
Micro-switches cross-domain	Understanding how HBM behaves when accessing different memory regions across pseudo-channels and micro-switches. Sequential access. Single thread PE. Burst size = 16, RBC=true. HBM results: Figure 7 (baseline) 8** *WR: Write RD: Read **A: different pseudo-channel emulating a single-thread PE connected to AXI0	• Pseudo-channels' throughput, on the same micro-switch, is the same. • Throughput decreases 50% when PE leaves the micro-switch to which is connected. •
Burst impact on performance	Understanding impact of sparse memory accesses (data in non-consecutive addresses in the same bank). Single thread PE. Baseline comparison: Sequential access. Burst size= 1 element= 1 beat= 256 bits= 32 Bytes. Results: Figure 8.B	• 2≤Burst≤8 ⇒ No throughput difference. • No Burst ⇒ Throughput decreases 30%, when AXI port access its micro-switch.
Randomizing inside a pseudo-channel	Understanding impact of sparse memory accesses (data in different banks). Single thread PE.	• Throughput: No difference within the same HBM stack. • First stack: write transactions, all AXI

	Results: Figure 8.C	ports throughput decreases 44%. • First stack: write transactions, vertical accesses, throughput reduces 65%.
Randomizing across different pseudo-channels	Understanding impact of sparse memory accesses (randomizing address inside pseudo-channel and the pseudo-channel to be accessed) Single thread PE. Results: Figure 8.D	• Throughput decreases to 0.61% for write transactions. • Throughput decreases to 0.17% for read transactions.
Simultaneous accesses to the same pseudo-channel	Multiple Traffic-Generators accessing the same pseudo-channel 0. Sequential access. Results: Figure 9	• All Pes connected to the same micro-switch have similar behavior. • Throughput is reduced 50% when PE=2 (Figure 9.B). • Throughput is reduced to 32% when PE=3 (Figure 9.C). • Throughput is reduced to 25% when PE=4 (Figure 9.D). • Figures 9.E/F/G represents PE=8 in different micro-switches.

Table 2 Address Mapping policies for HBM and DDR4. Default Policies Are Marked in Bold

Policy	HBM (app addr[27:5])[21]	DDR4 (app addr[33:6])[22]
RBC	14R-2BG-2B-5C	17R-2BG-2B-7C
RCB	14R-5C-2BG-2B	**17R-7C-2B-2BG**
RCBI	*N/A*	17R-6C-2B-1C-2BG
BRC	2BG-2B-14R-5C	2BG-2B-17R-7C
RGBCG	**14R-1BG-2B-5C-1BG**	*N/A*
BRGCG	2B-14R-1BG-5C-1BG	*N/A*

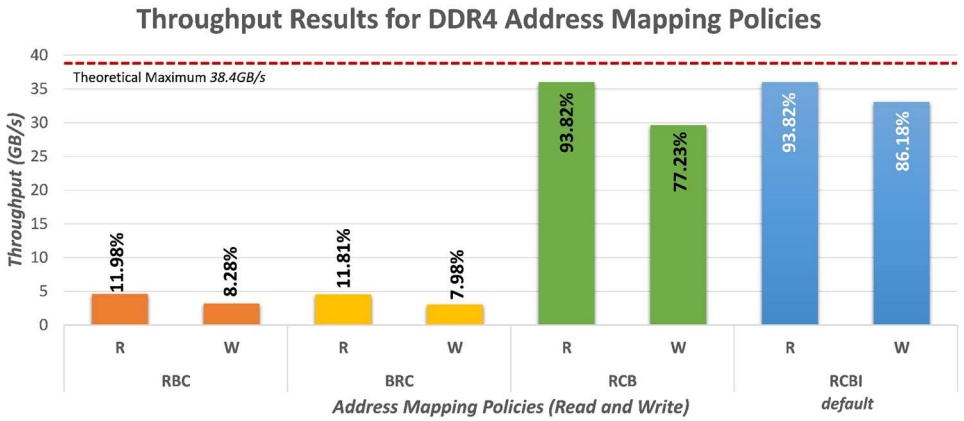


Figure 5. Throughput for DDR4 Address Mapping Policies.

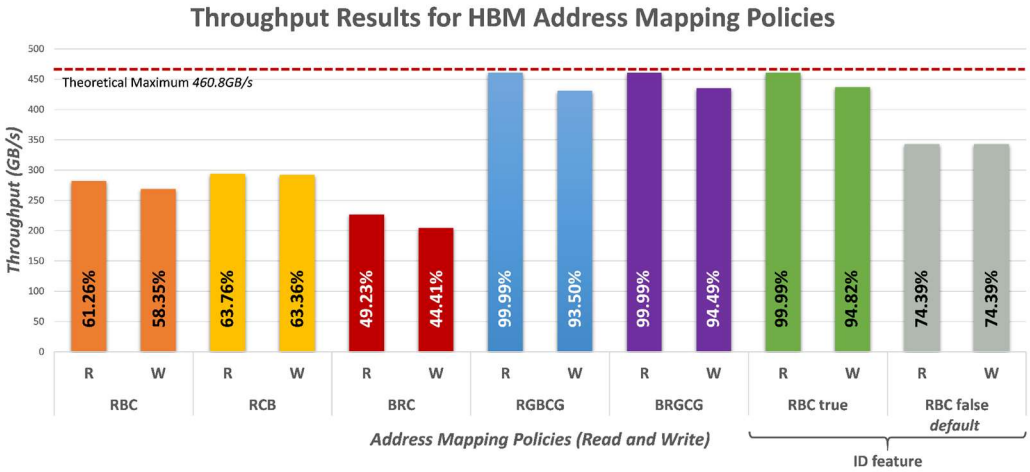


Figure 6. Throughput for HBM Address Mapping Policies.

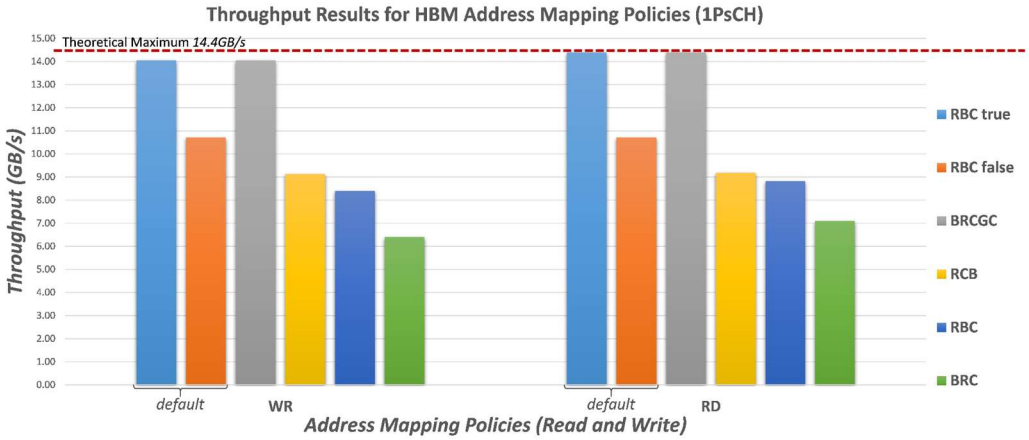


Figure 7. HBM pseudo-channel Throughput with Address Mapping Policies.

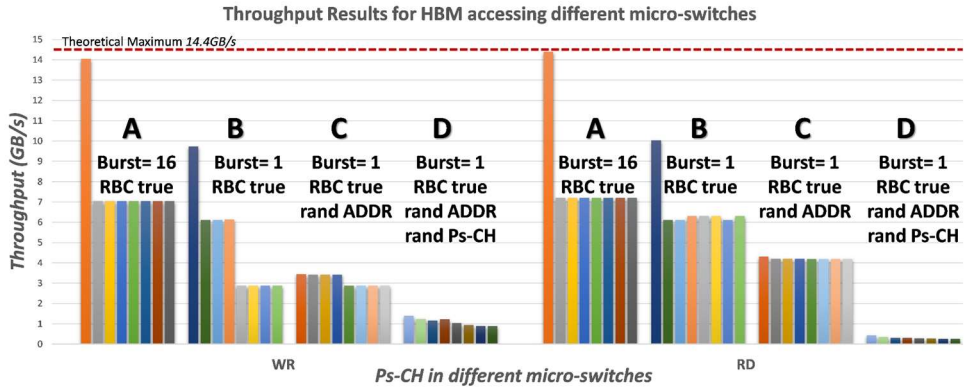


Figure 8. HBM Throughput in different micro-switches.

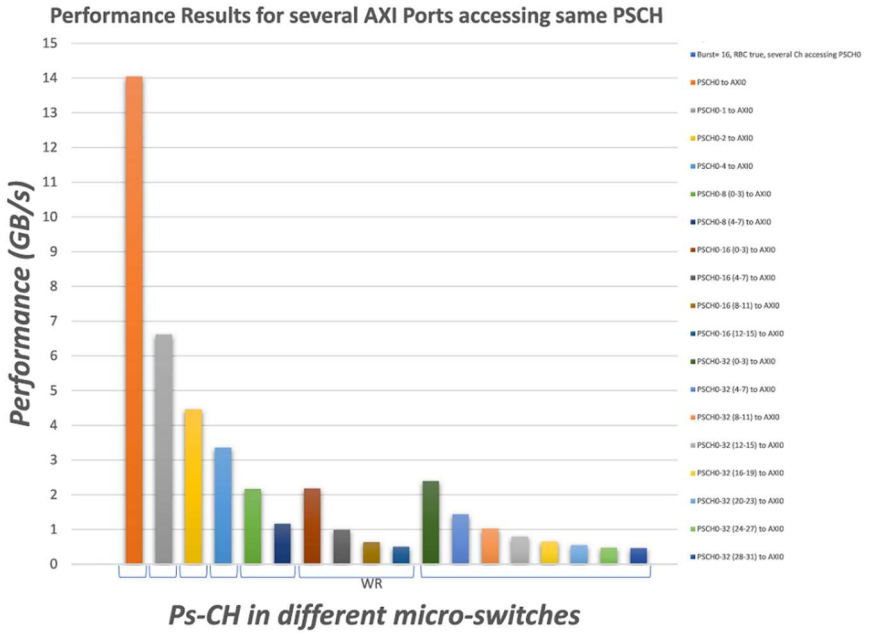


Figure 9. Simultaneous access for different micro-switches.

As expected, the throughput performance was more than 12 times better when using all 32 pseudo-channels in the HBM in parallel than when using the 2 memory banks present in the DDR. The different address mapping policies, the burst size, accesses within a micro-switch or external ones and the randomization of the address can have a huge impact on the HBM throughput.

This analysis makes an important contribution to the state of the art regarding the impact in HBM performance of pseudo-random accesses across and within the pseudo-channels and the concurrent access to a same memory region. In addition to that, the Memory Sandbox tool is valid not only to the provided test scenarios, but designers and researchers can also create their own.

2.2. FPGA-to-FPGA communication: Adapting Aurora designs

The new version of FPGA Shell supports Xilinx proprietary light-weight high-speed protocol for external communications - Aurora. Aurora is a relatively simple protocol that has been

designed to allow other protocols such as TCP/IP to ride on top of it. It uses one or more high-speed serial GT lanes. It can be referenced to a second or Data Link Layer of the OSI model (the layer where data packets are encoded and decoded into bits). Xilinx provides an IP core for implementation of this protocol: Aurora 64B/66B[3]. This IP core optionally includes SerDes hard-macro around FPGA GT differential pin pairs. On the other side Aurora 64B/66B provides receive and transmit 256-bit wide AXI4-Stream channels. Initial experiments with this IP were reported in deliverable *D6.2 Emulated accelerator initial release,* section *4.3.2 Aurora*.

Full Aurora subsystem instantiated in the FPGA Shell also includes AXI DMA engine for providing communications over Aurora by a software. Figure 10 presents the internal structure of the Aurora DMA subsystem.

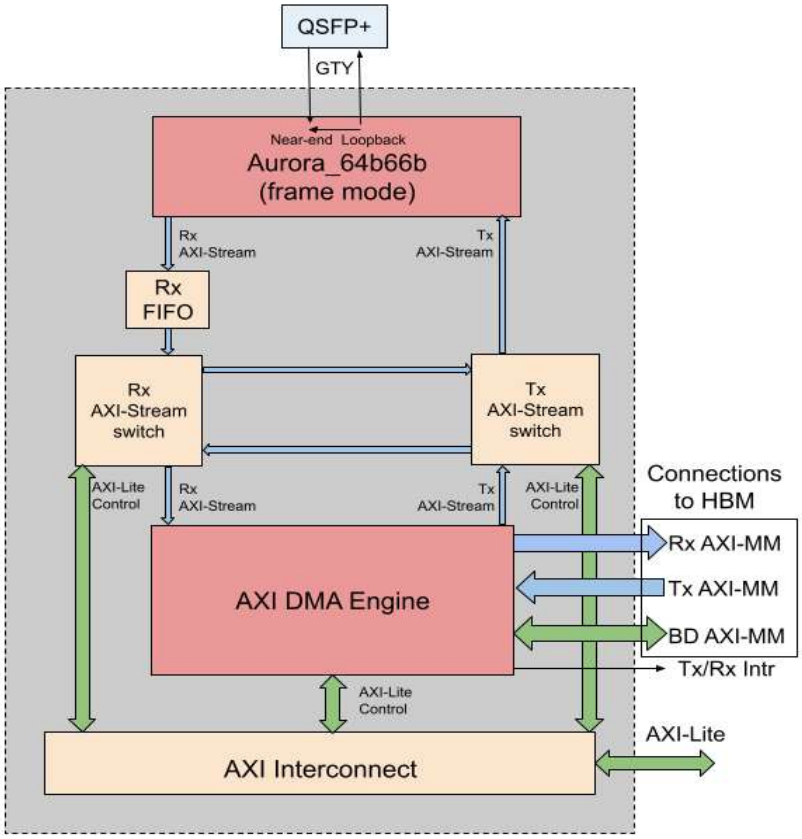


Figure **10**: Structure of Aurora DMA subsystem compliant with MEEP FPGA Shell

In addition, the Aurora subsystem contains logics for control, buffering and diagnostic purposes. By default Aurora IP is configured to use all 4 GTY lanes connected to an optical QSFP+ connector of Alveo FPGA boards, 10 Gb/s per each lane. The performance results, measured in terms of bandwidth, obtained for the Aurora DMA solution under Buildroot Linux are measured by using the test application running on a single-core ACME system. Non-cached region of HBM has been used for configuration of Xilinx DMA. The achieved bandwidth of data exchange between two boards is around 3.5 Gb/s. In addition, our test includes, checking the integrity of the exchanged data, and exchanging the ICMP (Internet Control Message Protocol) packets on the basis of standard PING command, as an example of running IP protocol packets over the Aurora link layer.

[3] Aurora 64B/66B: https://www.xilinx.com/products/intellectual-property/aurora64b66b.html

The resource utilization report for the Aurora DMA solution is shown in Figure 11. As seen, the main source of resource consumption is the Xilinx DMA, which uses around 9K LUTs, whereas the Aurora IP uses around 1.7K LUTs.

Name	CLB LUTs (1303680)	CLB Registers (2607360)	CARRY8 (162960)	F7 Muxes (651840)	F8 Muxes (325920)	CLB (162960)	LUT as Logic (1303680)	LUT as Memory (600960)	Block RAM Tile (2016)
system_top	147364	133646	2537	6244	1602	30637	137919	9445	161
meep_shell_inst (meep_shell)	73244	84621	737	2040	348	16437	65771	7473	104.5
APB_rst_or (meep_shell_APB_rst_or_0)	1	0	0	0	0	1	1	0	0
AuroraSyst_dma_hbm (meep_shell_AuroraSys	17887	25665	132	97	0	4911	15858	2029	29.5
inst (meep_shell_AuroraSyst_dma_hbm_0_	17887	25665	132	97	0	4911	15858	2029	29.5
aur_sysrst_inv (meep_shell_AuroraSyst_	0	0	0	0	0	0	0	0	0
aurora_64b66b_0 (meep_shell_AuroraSy	1689	6663	24	0	0	1073	1642	47	4
axi_reg_slice_rx (meep_shell_AuroraSyst	897	1150	0	0	0	402	387	510	0
axi_reg_slice_tx (meep_shell_AuroraSyst	667	1251	0	0	0	347	340	327	0
axi_timer_0 (meep_shell_AuroraSyst_dm	292	240	40	0	0	72	292	0	0
concat_intc (meep_shell_AuroraSyst_dm	0	0	0	0	0	0	0	0	0
dma_connect_rx (meep_shell_AuroraSys	673	668	8	0	0	130	428	245	0
dma_connect_sg (meep_shell_AuroraSy	1384	1534	16	32	0	251	915	469	0
dma_connect_tx (meep_shell_AuroraSys	535	568	8	0	0	112	319	216	0
eth_dma (meep_shell_AuroraSyst_dma_	8928	9614	36	62	0	2024	8715	213	21
ext_rst_inv (meep_shell_AuroraSyst_dma	0	0	0	0	0	0	0	0	0
gt_ctl (meep_shell_AuroraSyst_dma_hbm	127	312	0	0	0	66	127	0	0
gt_ctl_3 (meep_shell_AuroraSyst_dma_h	0	0	0	0	0	0	0	0	0
gt_ctl_4 (meep_shell_AuroraSyst_dma_h	0	0	0	0	0	0	0	0	0
gt_ctl_5 (meep_shell_AuroraSyst_dma_h	0	0	0	0	0	0	0	0	0
gt_ctl_6 (meep_shell_AuroraSyst_dma_h	0	0	0	0	0	0	0	0	0
gt_rst_comb (meep_shell_AuroraSyst_dn	1	0	0	0	0	1	1	0	0
GT_STATUS (meep_shell_AuroraSyst_dm	0	0	0	0	0	0	0	0	0
periph_connect (meep_shell_AuroraSyst	696	686	0	3	0	204	695	1	0
rstext_0 (meep_shell_AuroraSyst_dma_h	0	0	0	0	0	0	0	0	0
rstext_1 (meep_shell_AuroraSyst_dma_h	0	0	0	0	0	0	0	0	0
rstint_0 (meep_shell_AuroraSyst_dma_hl	0	0	0	0	0	0	0	0	0
rstint_1 (meep_shell_AuroraSyst_dma_hl	0	0	0	0	0	0	0	0	0
rx_axis_switch (meep_shell_AuroraSyst_	968	1445	0	0	0	383	968	0	0
rx_fifo (meep_shell_AuroraSyst_dma_hbr	50	48	0	0	0	12	50	0	4.5
tx_axis_switch (meep_shell_AuroraSyst_	969	1447	0	0	0	360	969	0	0
txrx_rst_gen (meep_shell_AuroraSyst_dn	17	39	0	0	0	7	16	1	0

Figure **11**: Utilization report highlighting the Aurora DMA subsystem.

2.3. Extending emulated accelerator support

The new version of the FPGA Shell includes two new features related to any emulated accelerator. One of them is a ROM memory with information related to the final design generated. This data is written during the bitstream generation. The other feature is the compatibility with AIT, which allows the generation of bitstreams from application code.

2.3.1 Emulated accelerator design information: InfoROM

The infoROM is a memory hardcoded in the FPGA Shell. It stores and displays basic information about a design generated using the FPGA flow, including the date of the project generation, the SHA values of the Shell and the Accelerator, and the IDs of the active interfaces in the FPGA Shell.

This module plays an important role in the configuration of the FPGA Shell by facilitating the storage, retrieval, and display of essential project information, as it is shown in Figure 12. Its implementation uses a BROM, with a BRAM address width of 13 and a Memory data width of 32. It also includes a single AXI slave interface, shared with the HBM from the QDMA. The ROM has a range of 8K, with the master base address at `0x200000000` and the master high address at `0x200001FFF`.

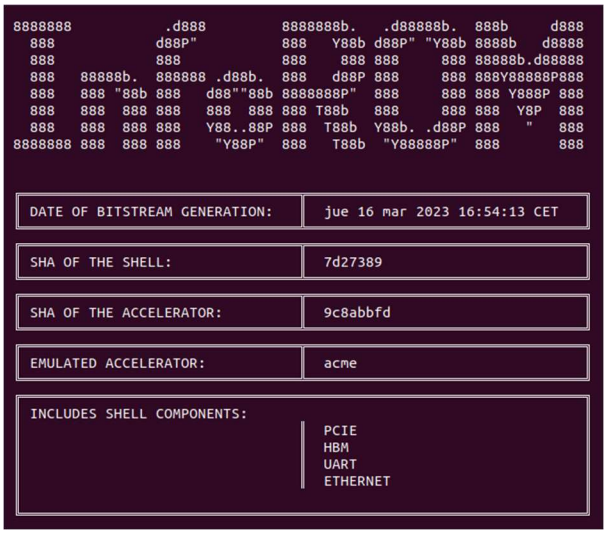


Figure **12**: Output of infoROM

2.3.2. Emulated accelerators from AIT flow

The difference between the FPGA Shell when compared to Amazon F1 (Amazon Web Services, 2023) and the Xilinx Vitis Platform (AMD Xilinx, 2023) lies in its configurable hardware, the use of QDMA, and the Ethernet-over-PCI mechanism. In addition to that, the MEEP FPGA Shell provides hardware/software support for RISC-V-based emulated accelerators, which allows bidirectional communication between the host and the RISC-V emulated accelerator.

Adding support for integrating AIT with the FPGA Shell expands the nature of the supported emulated accelerators, encompassing not only hardware-based designs but also incorporating the possibility to offload software tasks to FPGA devices. AIT flow uses OmpSs@FPGA to create FPGA accelerators from application-level code. The integration with the MEEP FPGA Shell makes AIT design board agnostic and permits to take advantage of all the FPGA Shell features.

A detailed explanation of the AIT + MEEP FPGA Shell integration is provided in deliverable *D6.5 -First AIT release.*

3. MEEP FPGA flow: CI/CD improvements

This section complements the information included in deliverables D6.2 *Emulated Accelerator initial release*, Section 4.4 *CI/CD for the MEEP FPGA Shell*, and D6.3 *Emulated accelerator second release with full capability of inter-accelerator communication*, Section 5 *Continuous integration and Continuous delivery for the FPGA flow*.

Deliverable D6.3 presented the role of CI/CD to support the FPGA flow. It described details about the implementation of the infrastructure used to generate a common tool for supporting development tasks of three different groups (i.e., RTL (WP4), Software (WP5), and FPGA (WP6)), and improving the MEEP project performance.

The improvements on the MEEP FPGA flow are relative to three main areas:

1) CICD implementation
2) FPGA flow automation
3) FPGA flow usability

3.1. CICD implementation

There are significant updates aimed to improve the efficiency of the CICD:

- Docker images: Dockers images has been integrated as runners to improve the working system with the MEEP servers. These images improve the memory space on the servers since the builds are cleaned up automatically once the run has been completed. Moreover, these runners map the Vivado version according to each server where the Docker image runs.
- Licenses: Another important aspect is that some licenses from some specific IP, for example, UltraScale+ Integrated 100G Ethernet Subsystem (AMD Xilinx, n.d.), have been associated to those runners.
- Networking: Docker images have been configured to use the same MAC address as the server they run. This avoids issues with the licenses. This policy is going to be propagated to all the FPGA repositories.
- Dashboard: To simplify data visualization, some time has been devoted to develop an automatic dashboard, based on storing on a database results from the FPGA flow, and representing them using Grafana. Significant progress has been made in adding dockers for MySQL, phpMyAdmin, and Grafana. The idea is to parse the resource files to show using a Python script and create tables to deploy on the database using MySQL commands. PhpMyAdim helps users to work with database information, using a GUI (Figure **13**). This task is not completed, and we plan to continue it in subsequent projects.

Figure **13**. Work in Progress dashboard using Grafana

3.2. FPGA flow automation

As it was presented in deliverable D6.3, the FPGA flow automates the generation of a bitstream, going through the synthesis, implementation, reports, and bitstream generation stages. In addition to that, the FPGA flow is a bridge between RTL (WP4) and software (WP5) teams, which must provide a certain level of reliability on the delivered design. With this purpose, the FPGA flow has improved the validation stage of a bitstream by adding more reliable FPGA tests in the form of Linux booting, execution of bare-metal benchmarks, and Linux distro execution, as it is shown in Figure 14. The drivers, ONIC drivers, for this hardware validation are an essential part of the whole process. More information about these drivers in deliverable D6.3. and D5.4. The flow chart used by MEEP FPGA for the CICD flow is present in Appendix III.

In parallel with the automation of this validation process, most of the FPGA tools (*fpga_tools*) have been updated to be compliant with requirements either the FPGA flow or the Phase_2 infrastructure (MEEP cluster).

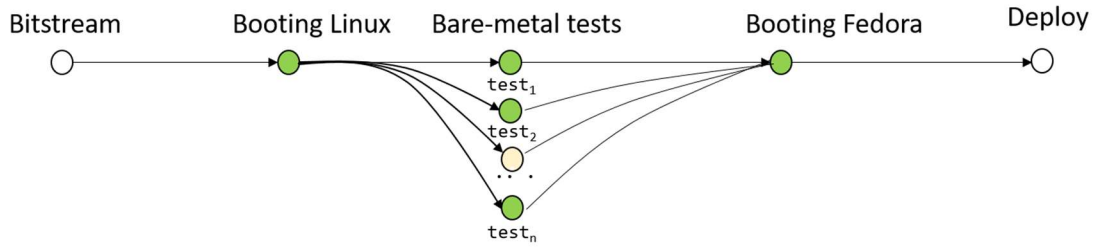


Figure **14**. New FPGA test added to the FPGA flow.

3.2.1. Booting Linux: Buildroot

The Linux test was the first test included. It continues to be part of the flow. It is required to load the bitstream to the FPGA. Then, when we boot, we will validate using the UART output log to find a coincidence that helps validate the test. A detailed explanation can be found deliverable D6.3, Section *Continuous Integration and Continuous Delivery for the FPGA flow*.

3.2.2. Bare-metal Benchmarks

The bare-metal benchmark is a submodule of the ACME_EA repository. The definition of the ACME bitstream determines if the system is single-core or multi-core. The flow will compile the list of tests suitable for each case according to the number of cores. These tests are from repository hosts unit tests for RISC-V processors. (riscv-software-src, 2023). To complete the list, new benchmarks have been included (MEEP, 2023). The list of benchmarks is collected in Table 3, for a single-core design, and Table 4, for a multi-core design.

Table 3. RISC-V Bare-metal Benchmarks for a single core system

Test	RISC-V Benchmark	Operation	Problem Size	Test Name
1	histogram	Integer	1024	histrogram.riscv
2	median	Integer	1024	int-median.riscv
3	multiply	Integer.	1024	int-multiply.riscv
4	qsort	Integer.	2048	qsort.riscv
5	rsort	Integer.	2048	rsort.riscv
6	spmv	Integer.	1000 x 1000 10004 nnz	int-smpv.riscv
7	vvadd	Integer.	1024	int-vadd.riscv
8	matrix mult.	Integer.	64 x 64	int-matrix_mult.riscv
9	fibonacci	Integer.	25	fibonacci.riscv
10	towers	Integer.	7 discs 40 runs	towers.riscv
11	bubblesort	Integer.	1024	int-bubblesort.riscv
12	median	Floating Point.	1024	fd-median.riscv
13	multiply	Floating Point.	1024	fd-multiply.riscv
14	spmv	Floating Point.	1000 x 1000 10004 nnz	fd-smpv.riscv
15	vadd	Floating Point.	1024	fd-vadd.riscv
16	matrix mult	Floating Point.	64 x 64	fd-matrix_mult.riscv
17	bubblesort	Floating Point.	1024	fd-bubblesort.riscv
18	dhrystone	Integer.	500	dhrystone.riscv
19	matrix mult (mm)	Floating Point.	--	mm.riscv

Table 4. RISC-V Bare-metal Benchmarks for a multi-core system

Test	RISC-V Benchmark	Operation	Problem Size	Test Name
1	matrix mult.	Floating Point.	4096	mt-matmul.riscv

2	vvadd	Floating Point.	4096	mt-vvadd.riscv
3	axpy	Floating Point.	16384	mt-axpy.riscv
4	somier	Floating Point.	1024	mt-somier.ricv
5	histogram	Integer.	1024	mt-histogram.riscv
6	stream	Floating Point.	16384	mt-stream-copy.riscv
7	stream	Floating Point.	8192	mt-stream-triad.riscv
8	is	Integer.	1024	mt-int-sort.riscv

3.2.3. Booting Fedora

The third test is booting Fedora distribution. More information about Fedora image in deliverable D5.4 *Final release of the software stack*, Section *4.1 Operating System*. The booting process is the same as the Linux stage (booting Buildroot), but now for a different element, the OpenSBI binary. In both booting processes, Linux and Fedora, the device tree is used to build the image. Consequently, the bitstream must be updated and synchronized with the drivers in the binaries to boot without errors. This is a suitable methodology to prevent hardcoded memory addresses. Everything depends on the device tree data of the ACME_EA. Figure 15 is a screenshot of the Fedora booting process on top of the ACME accelerator.

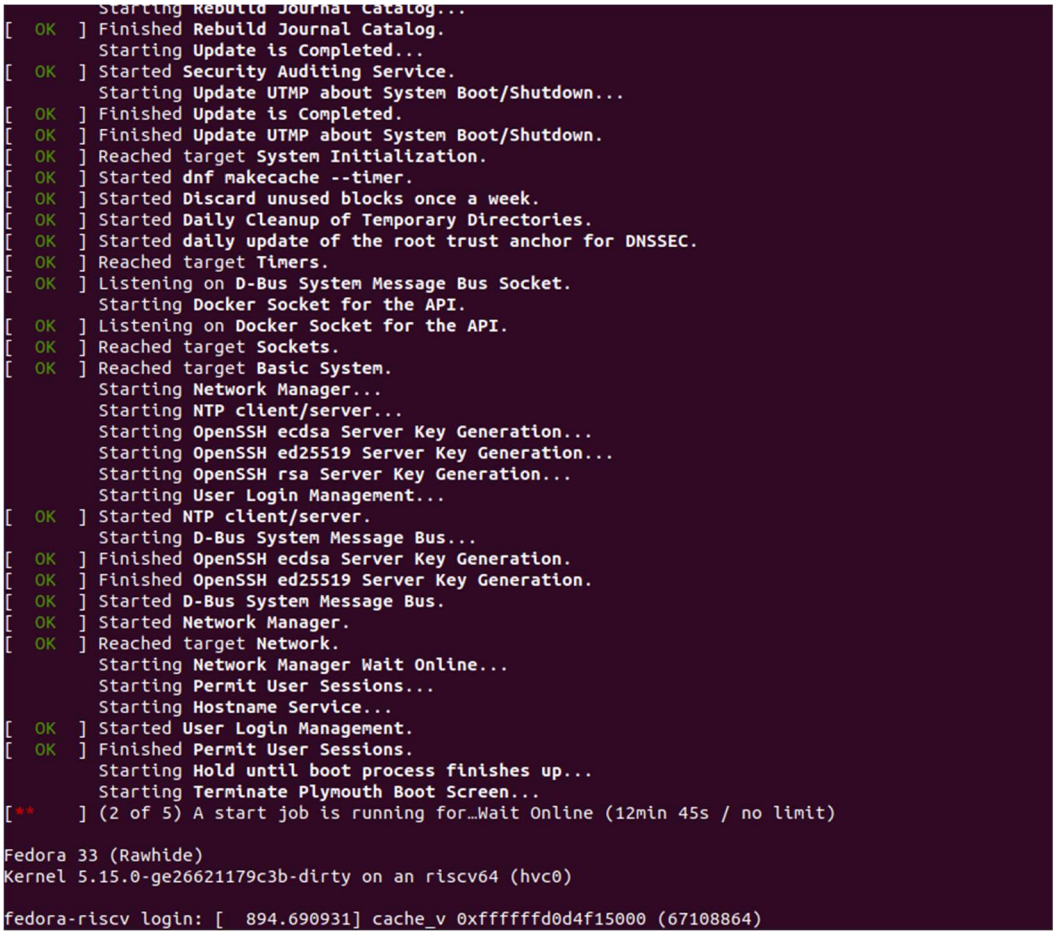


Figure **15**. Booting Fedora output

3.2.4. Deploy

The final deployment contains a structure of folders where the different information is written. The resultant folder structure is shown in Figure 16.

- **Bin**: Includes all the binaries of the benchmarks according to the number of cores of the system.
- **Bitstream**: Contains all the bitstreams generated for each environment. The bitstream name includes the pipeline date, the ACME_EA flavor name, and associated environment used for its generation.
- **Boot**: Includes the OPENSBI binary, and the script to boot any binary for the ACME accelerator.
- **Dcp**: Contains design's checkpoints for each ACME_EA flavor, including information for their synthesis and implementation stages.
- **dts**: This folder keeps the devices tree generated from the OpenPiton framework, for each of the ACME_EA flavor.
- **logs**: Contains the output results of booting Linux/Buildroot and the Fedora stages.
- **project**: This folder contains the most important files of the flow: *system_top.sv* and *gen_system.tcl*.
- **reports**: Includes all the reports generated during the synthesis and implementation.
- **Test_logs**: Store the results of the bare-metal benchmarks execution.
- The final script contains: 1) the commit SHA (*commit_sha.txt*) to identify the FPGA Shell version and the MEEP ACME_EA commit (*EA_info.txt*) version used for a deployment. It also includes the output log (*make_project.log*) when the project was generated (*date.txt*).

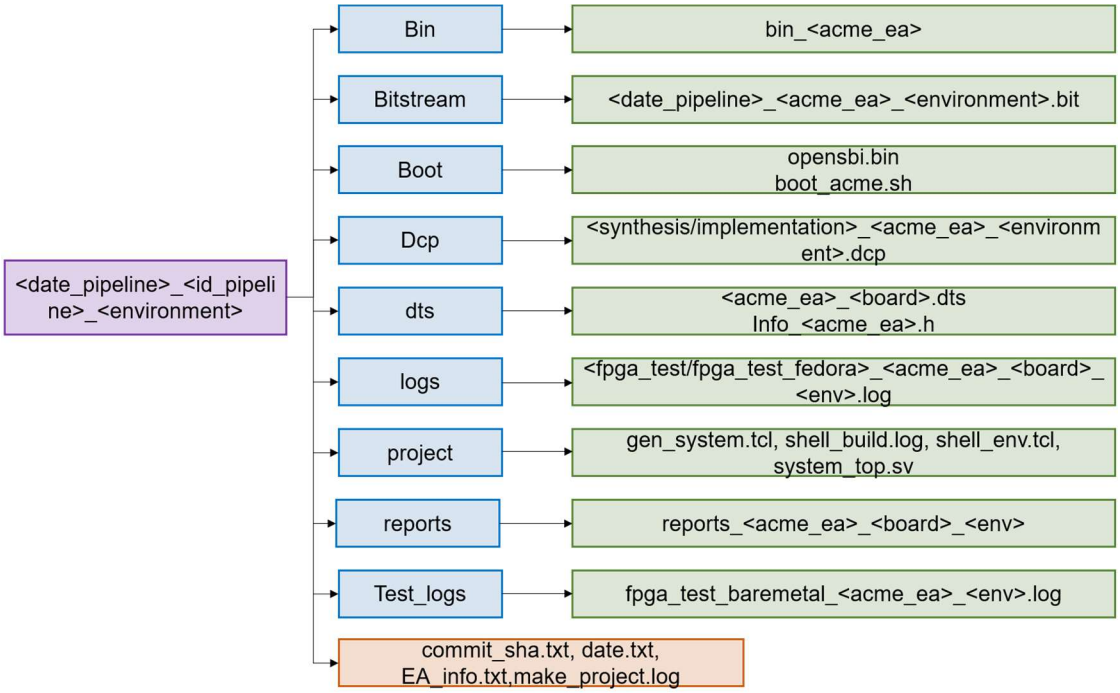


Figure **16**. Deploy structure on MEEP servers

3.3. FPGA flow usability: Working environments

An environment refers to an FPGA flow with a determined configuration and a well-defined purpose. The MEEP CICD has improved its support to the FPGA flow. Therefore, different environments and new tests were included to automate the validation of a hardware design along the different stages of the FPGA flow. This section is focused on the validation of the final emulated design, once the bitstream has been generated.

Thus, two different environments have been set:

- **Production**: Production is the main environment. All the hardware deployed will be accessible to the end user. This environment needs to be stable and reliable. The information is highly available, and there are two places where the data can be found. The first one is the MEEP servers, and there is a special place where all the information is shared among the servers. The second one is the Nexus Repository Manager, http://release.meep-project.eu . Here, it only deploys the production bitstreams.
- **Development**: Two different development or pre-production environments are enabled: 1) Test, and 2) Quick-test. These pre-production environments help developers to test and debug new features for under-development designs, before being deployed to the Production environment.

  The differences between Test and Quick-test environments are: 1) how the design is implemented, 2) the elements involved, and 3) The number of Emulated Accelerators that can be tested at once.

  - Test: We can use the keyword "matrix" to test different configurations in the same pipeline using the GitLab ci tool. In this manner, the FPGA CI flow can ensure that changes are thoroughly tested and validated before they are deployed to Production.
  - Quick-test: The Quick-test environment only can test one EA configuration. Still, it helps the RTL and FPGA developer speed up the process to continue the analysis with the other domains.

Table 5 describes how the developers can use each environment, explaining the main elements involved in the process.

Table 5. FPGA flow environments

Environment	Description	Source	Elements
Production	Help to ensure the stability and reliability of bitstreams	- Merge request - Schedule (monthly)	Design: acme_ea Routers: Pronoc Board: U280, U55C
Test	Help developers to thoroughly test and validate design changes for configurations of emulated accelerators.	Commit message `#TestCICD`	Design: acme_ea Routers: OpenPiton Board: U280, U55C
Quick-test	Help to speed up the analysis for one specific under-development configuration design of an emulated accelerator.	Gitlab web page	Design: any ea Routers: OpenPiton Board: U55C

4. ACME Emulated Accelerator (ACME_EA) designs

This section presents results of the final release of the ACME accelerator, presented in deliverable D4.3 *Full RTL for FPGA release*, as an emulated accelerator (ACME_EA) on the FPGA platform. Different modules have been developed by the RTL team as part of the ACME accelerator: scalar core, vector accelerators (VPU and Systolic Arrays (SA-HEVC, and SA-NN)), and Memory Tile.

Figure 17 summarizes the information of this section, in which information of different ACME_EA configurations is presented. Moreover, this section combines the working environments and the ACME accelerator designs. To clearly identify each of the design's configuration the naming convention agreed by all the technical work packages (WP4-WP6-WP5) is followed. This terminology is explained in deliverable D4.3, section XX *Naming convention*.

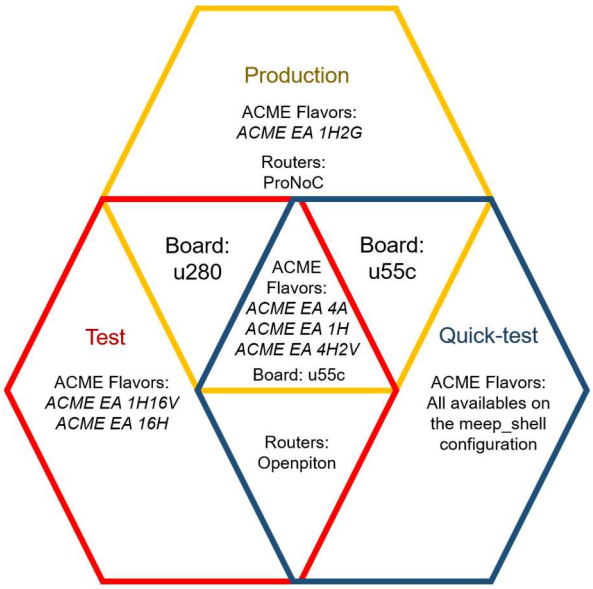


Figure 17. Diagram overview system.

Evaluating and comparing the elements presented here are vital in assessing performance and driving progress. A comparative analysis of two distinct environments will be shown: Production and Test. The MEEP FPGA focus will be on examining the elements within each domain and exploring the results generated over the past three months. Understanding these elements' unique characteristics and outcomes can help gain valuable insights into their performance.

4.1. ACME_EA designs by working environment

4.1.1. Production environment releases

Production releases only include results from stable configurations of the ACME_EA accelerator. That means, those ACME flavors that have passed successfully all the FPGA Flow stages. These emulated accelerators use ProNoC as routers and have been implemented targeting the two different boards available in MEEP infrastructures: U280 and U55C.

Four different configurations have been promoted to Production, and their deployment results are available in Nexus Cloud:

- ACME EA 4A: This design is used as a reference platform by WP5. It is a multi-core system with Ariane as scalar core and no vector accelerator.
- ACME EA 1H: This is the simplest configuration of the ACME accelerator when using Lagarto Hun as scalar core. This is a single core design with no vector accelerators.
- ACME EA 4H2V: This design scales the previous one, to 4 cores. Moreover, each of the cores includes a MEEP.VPU with only 2 vector lanes.
- ACME EA 1H2G: This design is a single core design with the three vector accelerators developed during the project (VPU, SA-HEVC and SA-NN).

Table 6 summarizes the characteristics of each of these previous ACME_EA designs. As it is shown here, any of these ACME_EA accelerators include a Memory Tile in Production. This is because the Memory Tile is not mature enough to be considered stable. As a consequence, these designs only might be set to execute in classic-mode.

Although the Memory Tile has been integrated in ACME design, some problems arose during the FPGA flow, which made no possible to get results, neither under the Test environment. That means that only RTL simulations results are offered for a full design, as it is reported in deliverable D4.3.

In addition to this, evaluation results for ACME EA 4A and ACME EA 4H2V have been performed by WP5, and reported in deliverable D5.4, Section 5 *Evaluated environments*.

Table 6. Production releases 2023

Description		ACME EA 4A	ACME EA 1H	ACME EA 4H2V	ACME EA 1H2G
Architecture		Multi-core	Single core	Multi-core	Single core
NoC		ProNoC			
core	Core	Ariane	ACME VAS Tile (Lagarto Hun + vector accelerators)		
	Scalar Core	RV64GC (6 stages), in-order, double issue	RV64GC (5 stages), in-order, single-issue		
	VPU	No	No	MEEP.VPU	MEEP.VPU
RVV	v0.7.1	No	No	Yes	Yes
	Custom SA	No	No	No	Yes
Config. L2 size		Yes	Yes	Yes	Yes
Config. L1 cache line		No	Yes	Yes	Yes
Number of Tiles		2x2	1	2x2	1
Linux OS support		Buildroot (OpenSBI), Fedora			
Host/Device comm.		PCIe, Ethernet over PCIe			
FPGA Shell support		Yes			
Memory Tiles		No	No	No	No
Clock frequency		50 MHz	50 MHz	50 MHz	50 MHz
exec mode	classic-mode	Yes	Yes	Yes	Yes
	acme-mode	-	-	-	-

4.1.2. Designs under development: Test and Quick-test environments

Test and Quick-test environments use OpenPiton routers for any of the ACME_EA designs, and the targeted boards are: U280 and U55C for Test, and U55C for Quick-test.

Two new designs are run under Test, as it is shown in Table 7, in addition to the ones presented in Production:

- ACME EA 1H16V: This design pursues two things: 1) stress the FPGA flow and check the availability of resources on the FPGA to place and route a system with a single core and a VPU with 16 vector lanes; and 2) demonstrate the functionality of this MEEP.VPU configuration.
- ACME EA 16H: This configuration demonstrates the scalability feature of the ACME design.

One of the problems faced with these designs is that they do not always close timing. These designs are heavy in FPGA resources, and that makes close timing difficult during the place & route stage of the FPGA flow. This instability prevents them from being promoted to Production. However, for testing purposes, WP5 used one of the generated bitstreams that completed the FPGA flow, even under the Test environment, for running some experiments.

Table 7. Test releases 2023

Description		ACME EA 1H16V	ACME EA 16H
Architecture		Single core	Multi-core
NoC		OpenPiton Routers	
core	**Core**	ACME VAS Tile (Lagarto Hun + vector accelerators support)	
	scalar core	RV64GC (5 stages), in-order, single issue	
	VPU	Yes	No
RVV	**v0.7.1**	Yes	No
	Custom SA	No	No
Config L2 size		Yes	Yes
Config L1 cache line		No	No
number of Tiles		No	4x4 mesh
Linux OS support		Buildroot, Fedora	Buildroot, Fedora
Host/Device comm		PCIe, Ethernet over PCIe	PCIe, Ethernet over PCIe
FPGA Shell support		Yes	Yes
Memory Tiles		No	No
Clock frequency		50 MHz	50MHz
exec mode	**classic-mode**	Yes	Yes
	acme-mode	-	-

4.4. ACME_EA resources utilization

This section presents the resource utilization resources for the previous ACME_EA designs.

4.2.1 Production FPGA resources (U280)

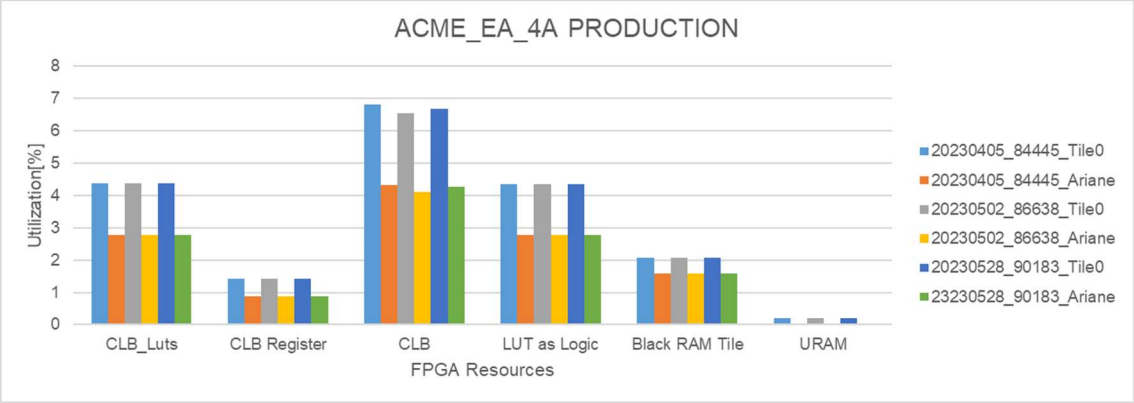


Figure 18. ACME EA 4A Production release for U280 board.

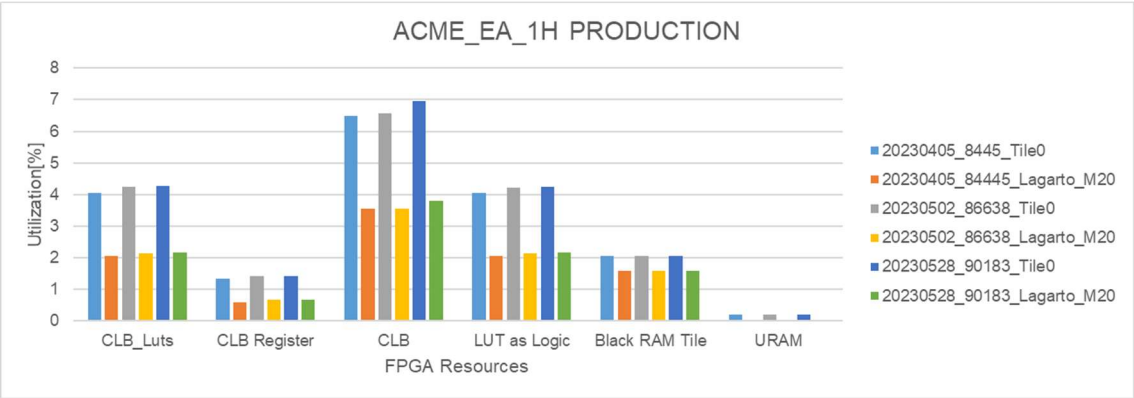


Figure 19. ACME_EA_1H Production release for U280 board.

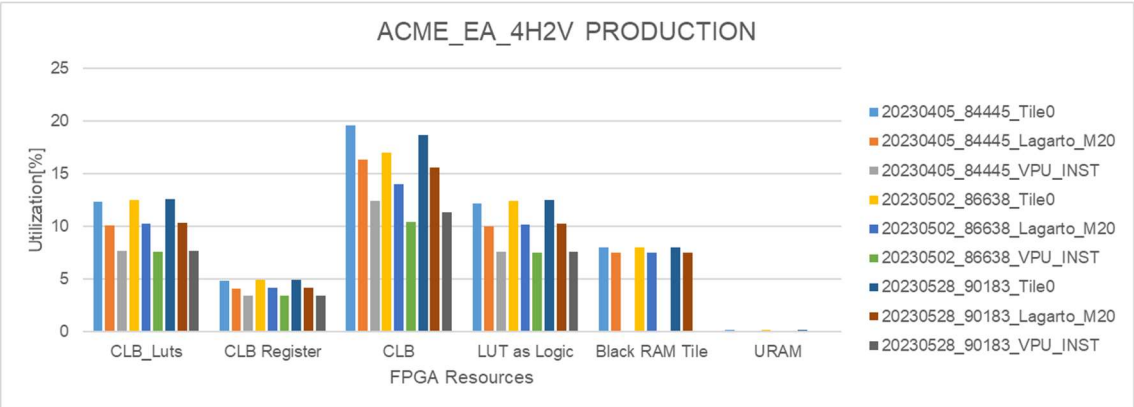


Figure 20. ACME EA 4H2V Production release for U280 board.

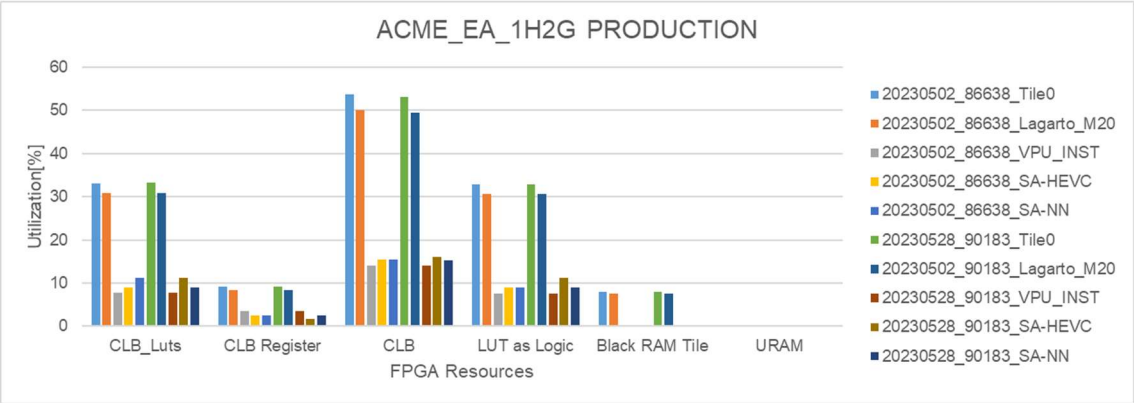


Figure 21. ACME EA 1H2G Production release for U280 board.

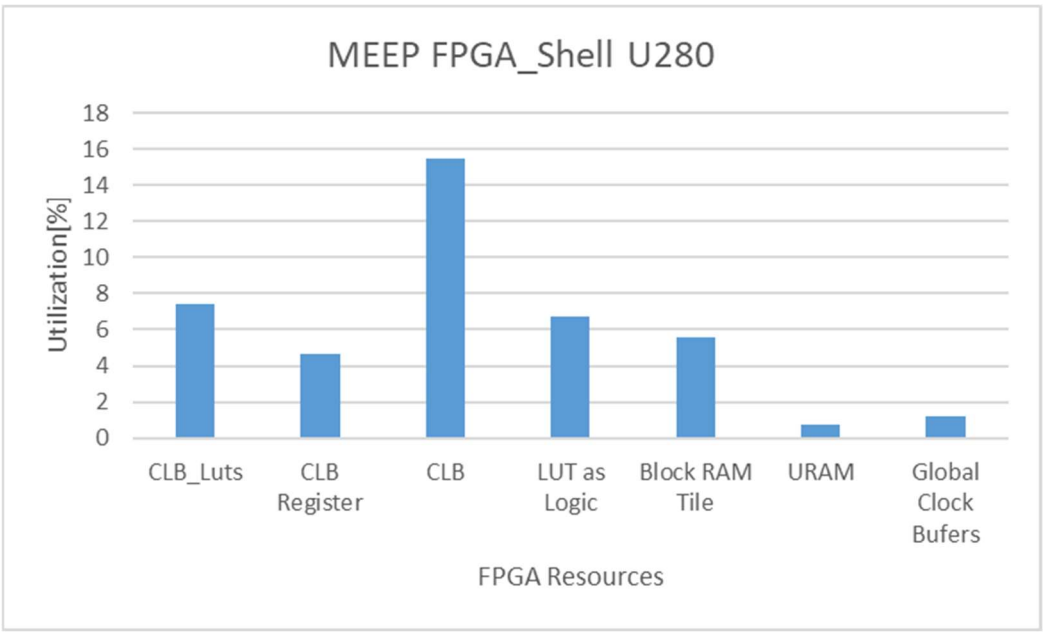


Figure 22. FPGA Shell using U280 board.

These figures (Figure 18, 19, 20 and 21) depict the report utilization using the U280 FPGA board for the Production environment, whereas Figure 22 represents the resource utilization exclusively for the FPGA Shell. Table 8 collects all these results.

Table 8. ACME EA releases resource utilization on U280

Design	CLBs	BRAM	Figure
ACME EA 4A	7%	2%	30
ACME EA 1H	7%	2%	31
ACME EA 4H2V	20% per Tile 80% in total	8% 24% in total	32
ACME EA 1H2G	54% per Tile	8%	33
FPGA Shell	15% in total	6%	34

In the case of the Lagarto Hun core releases, there are three different configurations and different complexities:

- ACME EA 1H is a light system.

- ACME EA 4H2V. The heaviest part of the design is the VPU. Four tiles use 80% of the total CLB available on the U280 board, and including the FPGA Shell, the CLBs reaches around 95% of the total.
- ACME EA 1H2G. This flavor is a single-core system using two vector lanes of VPU and also including the two SAs (SA-HEVC and SA-NN). The CLB utilization for one tile is around 54% of the total. Including the numbers of the FPGA Shell, it can be 70% of the total CLB elements.

4.2.2. Production FPGA resources (U55C)

Similar results have been obtained when targeting the U55C board. More details about the results in Appendix IV.

Here only the ACME EA 1H2G configuration has been included for representing the whole group of designs. (Figure 23 and Figure 24)

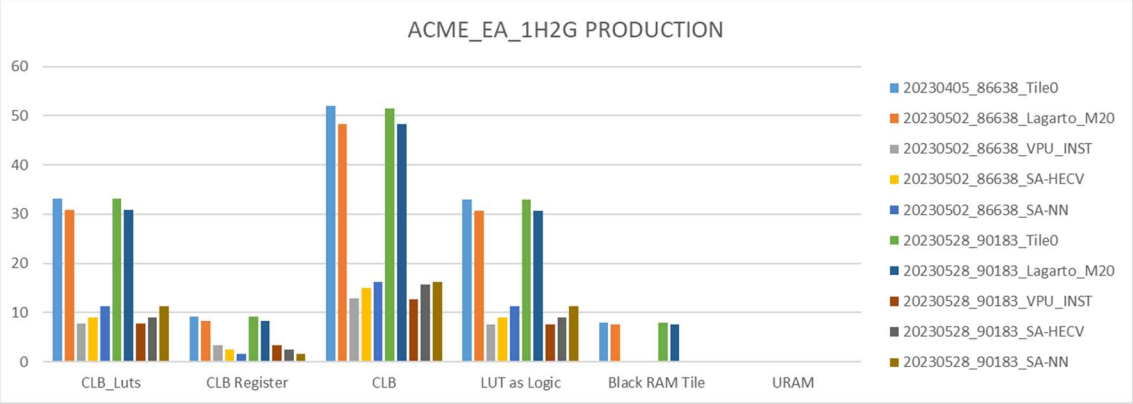


Figure 23. ACME_EA_1H2G Production release for U55C board.

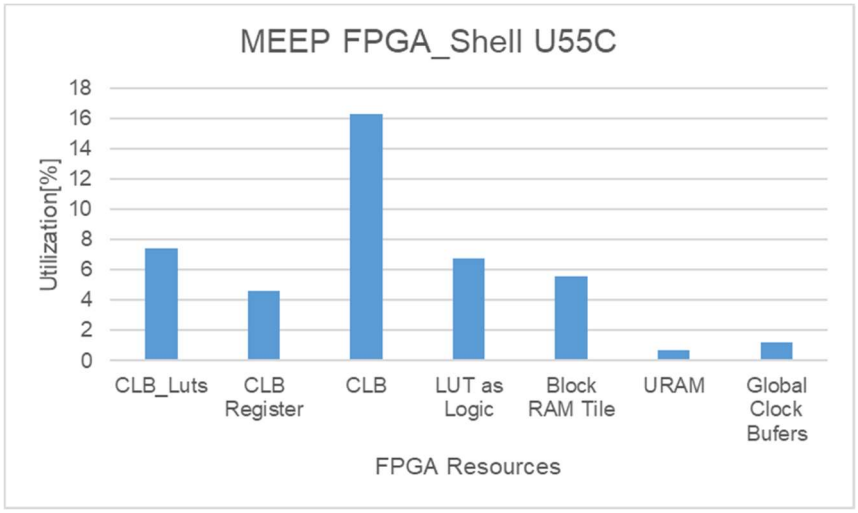


Figure 24. FPGA Shell using U55C board.

As expected, comparing results in both FPGAs, U280 and U55C, there is not a big difference in terms of resources.

4.2.3. Test FPGA resources

As it was previously mentioned, the bitstream generation of *ACME EA 1H16V* and *ACME EA 16H* designs was challenging, due to the place & route stage. Actually, there are situations for the ACME EA 1H16V where a tile uses 85% of total CLBs. A similar situation is faced with ACME EA 16H, in which one tile uses around 5% of the CLBs, and therefore, the 16 tiles require 80% of CLB resources of the total.

FPGA Shell has a technique to improve place & route results using different policies iteratively until closing timing. This feature is available for FPGA developers using the Test environment, and that is how the new methodology includes creating a plan for *place_design* according to the board type and having them in an incremental loop over the strategy for the *route_design* process.

Appendix II shows the last release resources results we have available now. Both boards faced the same issue. The previous release shows that the new methodology works, and we can complete the implementation phase.

4.3. Final emulation FPGA release of the ACME_EA accelerator

Results shown in previous sections on the FPGA have been obtained using the more advanced version of the ACME_EA accelerator, which includes new features with respect to the second release presented in the deliverable D6.3, Section 4 *ACME Emulated Accelerator (ACME_EA)*. For clarity, Table 9 compares the evolution of the ACME accelerator from the first to this final release.

Table 9. Characterization of the first and second FPGA release of the ACME_EA accelerator

Description		ACME_EA FPGA 1st release	ACME_EA FPGA 2nd release	ACME_EA FPGA 3rd release
		M18	M36	M42
Architecture		single core	many-core	
NoC		Bus	NoC (Routers support: OpenPiton and ProNoC routers)	
core	**ACME VAS Tile core**	scalar core + VPU		scalar core + VPU + **SAs** + **LVRF**
	scalar core	RV64IMA (5 stages) In-order	RV64**GC** (5 stages) In-order	RV64GC (5 stages) In-order, with **branch-predictor**
	VPU	MEEP.VPU v1.0 (See Appendix I)	MEEP.VPU v2.2.1 (See Appendix I)	MEEP.VPU v3.0
RVV	**v0.7.1**	Yes	Yes	Yes
	Custom SA	No	No	Yes
Memory Tile		No	No	Yes
Config L2 size		Yes	Yes	Yes

Config L1 cache line		No	No	Yes
Number of Tiles		No	Yes (2x2* 2D-mesh)	
Language		SystemVerilog and Chisel	SystemVerilog	
Linux OS support		Yes (buildroot)	Yes (Fedora 31)	Yes (Fedora 33)
Host/Device communication		PCIe	PCIe, Ethernet over PCIe	PCIe, Ethernet over PCIe, **Ethernet 100GbE**
FPGA Shell support		Yes	Yes	Yes
Memory Controllers (MCs)		N/A	Support for Multi-MCs (from ACME)	support for Mult-MCs (from the Shell)
Clock frequency		50 MHz	50MHz and 100MHz*	50MHz
exec mode	**classic-mode**	Yes (scalar)	Yes (scalar + vector {axpy 2:16 lanes})	Yes (scalar + vector{axpy, spmv, dgemm, fft 2:16lanes})
	acme-mode	No	Yes**	Yes

M36 is the evolution of M18, and it is under development by the RTL team.

*100MHz is only possible when the many-core system only includes the scalar core in each of the Tiles.

Running the system at 50MHz allows the manycore system to close timing with a 4x4 configuration, when the tile includes the scalar core and the VPU.

**Tested with the LVRF isolated (data preloaded in the register), running a few instructions; and no MT in the system

The Emulated Accelerator releases are available in the MEEP Gitlab repository, where the main features of the different releases are also described, and a link to each of the bitstreams is provided: https://gitlab.bsc.es/meep/FPGA_implementations/AlveoU280/fpga_shell/-/wikis/MEEP-FPGA-Releases.

5. MEEP FPGA-Cluster bring-up

This section describes the status of the MEEP FPGA cluster or FPGA-based digital laboratory, not only in terms of the configuration and setup of the infrastructure, but also the set of tools developed around it to supports HW/SW co-design activities of emerging technologies, based on European-developed IPs.

The Digital Laboratory expands the capabilities of a single FPGA platform to this large-scale platform, moving from a single FPGA system into multiple FPGA systems that can be used to look into the future at the system level.

5.1. Infrastructure

As shown in Figure 25, the MEEP system consists of two racks: each one with 6 nodes, and each node with 8 FPGAs. Each rack is fully connected to a switch (direct cabling from each node, but also each of the 48 FPGAs). Then, the racks are connected to each other through the interconnection of both switches.

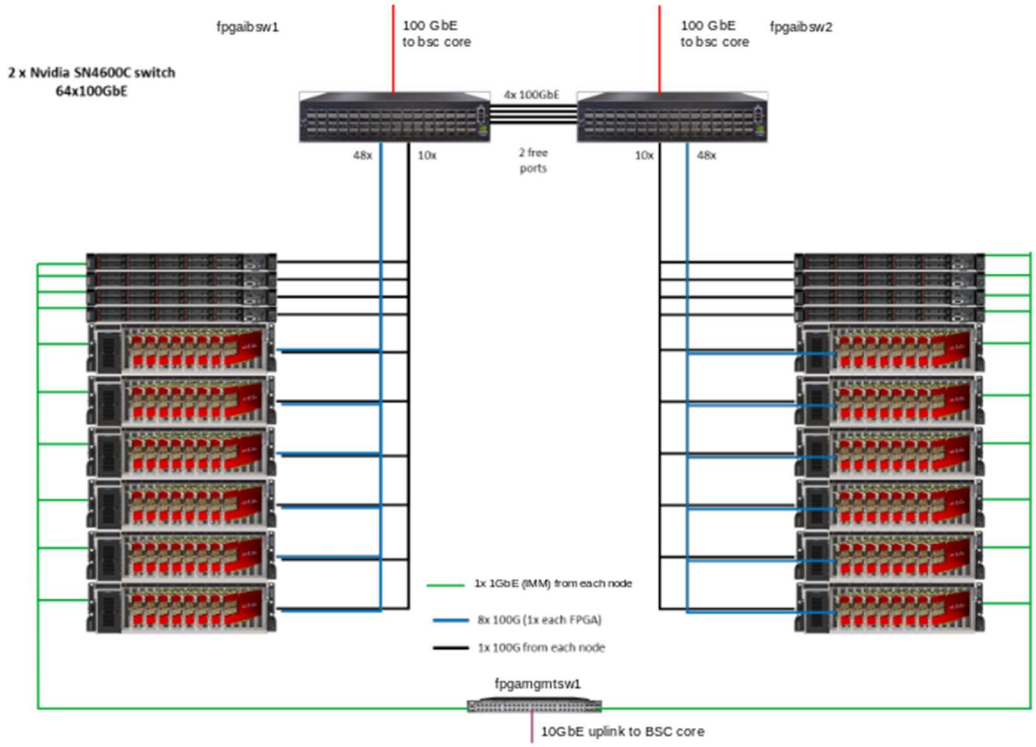


Figure 25. MEEP FPGA cluster connectivity: 2 racks x 6 nodes x 8 Alveo-U55C boards.

Table 10 collects the information for one rack, the same applies for both racks.

Table 10. Large-scale FPGA machine details

Digital laboratory infrastructure details	
Hardware components	
Admin standard nodes	**4** Each one with: 2x intel Xeon Gold 6330 28C 205W 2.0GHz Processor OS: RedHat Enterprise Centos 8.1
Compute nodes	**12** (*fpgan01:fpgan12*) Each one with: 2x Intel Xeon Gold 6330 28C 205W 2.0GHz Processor OS: RedHat Enterprise Centos 8.1 8x FPGAs Xilinx Alveo U55C
100 Gb Ethernet Switch SN4600C (64x100Gb)	**2**
USB Hub	**12** Each one with 10 ports
Hardware connectivity	
PCIe host bridges	Connect each of the host nodes with its 8 corresponding FPGAs
100Gb Ethernet switches	Each of the 96 FPGAs connected through QSFP0 Each of the 12 compute nodes Each of the 8 Admin nodes
Direct FPGA to FPGA connection (no switch)	**(Only available in Rack1)** 48 FPGAs in pairs through QSFP1 (inter-FPGA connectivity per node) FPGA1 – FPGA2 FPGA3 – FPGA4 FPGA5 – FPGA6 FPGA7 – FPGA8
USB Hubs (10 ports)	Connection per node: 8 FPGAs per node The compute nodes

A partial view of the real system is shown in Figure **26**, where only four nodes per rack are visible. The image on the left corresponds to Rack1, and the one on the right shows Rack2. There exist clear differences in the physical cabling between both racks. The 48 FPGA cards in Rack1 have inter-FPGA connectivity via QSFP1 to the FPGA card adjacent to it. This creates a paired grouping of QSFP1 P2P interconnectivity with any intermediate switch. In the case of Rack2, FPGAs are not directly connected in pairs. This means that QSFP1 is unused for now.

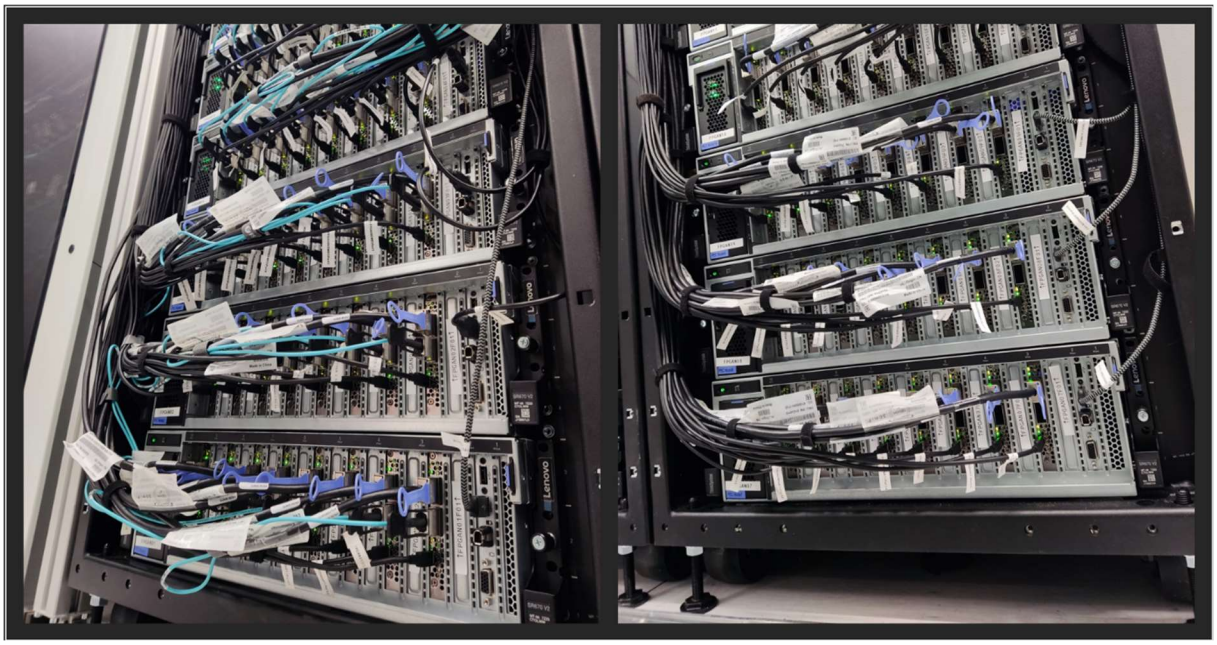


Figure 26. Physical cabling of the large-scale FPGA machine racks. (Rack1-left, Rack2-right)

As Figure **27** depicts, QSFP0 is the top port on the FPGA card, and QSFP1 is the bottom port. Regarding the cables, the black cables are copper QSFP DAC connections between QSFP0 on the FPGA card and the cumulus 100GbE switch. Then, the point to point interlink communication between adjacent FPGAs uses fiberoptic cables.

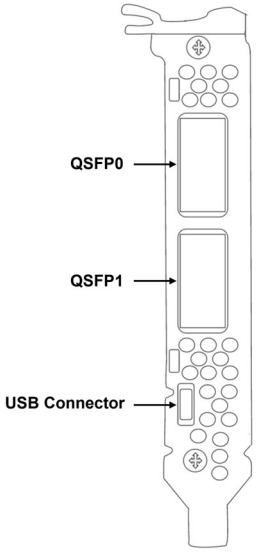


Figure 27. U55C FPGA position in the node

The third port in the FPGA, the USB connector; is used to connect an FPGA of a node to one USB Hub port, by using the cable shown in Figure **28**.

Figure 28. USB cables to connect an FPGA with the USB Hub

Even though initially the USB Hub shown in Figure **29** was installed, it has been finally replaced by the one shown in Figure **30**. A more basic hub model, with 10 ports instead of 16. Two reasons motivated this replacement: 1) smaller dimension, and 2) no stockage issues. The new USB Hub with only 10 ports fulfils the requirements per node, and it is possible to accommodate 6 of those in a rack. The purpose of each of the hubs is facilitating FPGA programmability using JTAG, but also getting output data through the UART.

Figure 29. USB HUB with 16 ports installed initially.

Figure 30. USB HUB with 10 ports finally installed

The assembly and physical installation of this system comprises the execution of several activities, executed over several months. The overview of these activities is shown in Table 11.

Table 11. Assembly and Physical installation process

Activity	Status end of MEEP
1) On-site machine assembly	Completed (Lenovo)
2) Cabling and configuration	Full Rack1 Full Rack2 *fpganode01, fpganode12*

3) Network cabling	Full Rack1 Full Rack2 *fpganode01, fpganode12* System connected to BSC network
4) Electrical installation	Completed
5) Machine OS and cluster installation	Full Rack1 Full Rack2
6) FPGA SW requirements	As per request
7) FPGA-to-FPGA programmability (QSFP+ and JTAG cabling to USB Hubs)	Full Rack1 Full Rack2
8) Tests (Hardware basic checks)	Full Rack1 Full Rack2
	Work done by Lenovo team Work done by BSC team

The assembly and the physical installation took place on-site, at BSC facilities. The *assembly* of all the modules was led by Lenovo team; whereas the rest of the activities relied on BSC team; except for the *electrical installation* that required supervision from Lenovo. In any case, BSC team has access to Lenovo support for all the issues related to the configuration and installation of the machine.

Before facing the installation of the full system, BSC team worked on a small set of it, starting from one node, and once it was stable, adding new nodes in an incremental fashion. This brought the possibility of tunning and debugging the installation, but also adapting the setup accordingly.

5.2. Configuration and setup

The MEEP system configuration and setup has been incrementally implemented, starting with one node, and scaling up to multiple nodes after ensuring functionality and correctness on the small environment.

5.2.1 Basic configuration and setup

Before getting the large-scale FPGA machine accessible to any user, a basic setup was required:

- Preparation of an operating system image and corresponding packages (according to Lenovo indications).
- Installation of some packages and tools required by *Operations team* for monitoring, managing, and guaranteeing good levels of security.
- Connecting the large-scale machine to the BSC LAN using GBIC cables, to simplify networking and software installation activities.

These previous activities are common to most of the systems under Operation team at BSC, since they are executed on the host. The novelty for the teams comes from the fact of having to handle 96 FPGAs, new devices for the Operations team. To deal with the associated complexity,

they were supported by the BSC FPGA developers all team. The working methodology was based on an iterative and incremental learning process consisting of testing, understanding, replicating, validating, and, in the end, automating. The first stage was a completely manual process working on a single node.

From an operational perspective, MEEP system was structured into five different types of nodes, according to their purpose: 1) login, 2) host, 3) head, 4) compute, and 5) FPGA nodes. More details are in Table **12**.

Table 12. Type of nodes as part of the large-scale FPGA machine

Type of node	Amount	Tag	Accessibility
Login node	1	*fpgalogin1*	only Operations
Virtual Machine host node	1 VM/FPGA (96 in total)	*fpgavmhost<node>f<card>* *<node> [01:12]* *<card> [01:08]*	Operations Users
Head nodes	2	*fpgahead<1,2>*	only Operations
Compute nodes	4	*fpgac[01:04]*	Operations Users
FPGA Host nodes	12	*fpgan[01:12]*	only Operations

1) Login node: it is used to get access to the FPGA cluster and being able to operate with one or several FPGA cards.
2) VM host node: it is a node that creates a Virtual Machine (VM) instance for each of the FPGAs of the cluster to allow users access to operate with a specific FPGA without interfere with other users.
3) Head node: There is one head node per Rack, which allows to control overall operation of a rack, and it is only accessible by Operations.
4) Compute node: These are general compute nodes, with no FPGAs associated.
5) FPGA node: Each of these nodes has 8 FPGAs connected to it. The control of the node is under Operations, although working together with the FPGA team, they are allowing certain privileges to users by adding commands to a sudo list.

Regarding the specific software packages required for guaranteeing a basic functionality of the system, the following ones were installed:

- Xilinx Vivado 2023.1
- Xilinx Vitis development environment 2021.2 and 2022.1 (FPGA login node)
- Xilinx XRT environment 2023.1 (FPGA nodes)
- UART clients picocom, microcom (FPGA nodes)
- Software development tools for OmpSs and others (clang, boost-1.66, ninja, lld, hwloc, numactl, gcc-10, gfortran)
- Gitlab-runner for CICD flow
- Slurm and Slurm X11

The last one is not specific to FPGAs, but for SLURM installation. Although SLURM was not configured from the beginning, Operations team installed all the packages required, based on their experience to progress on the configuration of the system.

The above scenarios are currently in use (in part) at the FPGA cluster by several BSC projects (e.g., MEEP, EPI, OmpSs). All projects actively utilize all depicted interfaces (PCIe, Ethernet-over-QSFP, UART, JTAG), thus confirming their proper hardware and software configurations.

Xilinx Vitis environment

The part of standard Alveo packages for the U55C board from AMD Xilinx version 2023.1 responsible for Vitis runtime environment are installed at each of the compute nodes. Besides PCIe drivers providing Vitis based runtime flow (*xocl* OS kernel module) and card management (*xclmgmt* OS kernel module) the packages contain flashable partitions for the cards:

- PCIe XDMA based Xilinx Platform *xilinx_u55c_gen3x16_xdma_base_3* for programming to FPGA after cold reboot and providing Vitis based flow from FPGA side.
- Satellite Controller firmware version 7.1.22 providing management and monitoring of on-board hardware.

Both of the above partitions were flashed to all 96 FPGA boards in order to support standard Xilinx Vitis runtime flow and support hardware management/monitoring through the Satellite Controller.

5.2.2. Advanced configuration and setup

Based on Operations and FPGA team requirements for the envisioned functionality of the MEEP system, as digital laboratory, a more advanced configuration was implemented.

One of the most complex activities has been configuring the PCIe. A very important peripheral for establishing proper communication between the host node and each of the FPGA cards. The complexity relied on two aspects: 1) identifying the physical PCIe port mapping, and 2) adapting the drivers to the needs of present and future users. The former, physical mapping had also two levels: 1) understanding and clearly identifying the mapping between the physical chassis port and the PCIe port slot (Figure 31); and then 2) associating the PCIe and FPGA port mapping (Figure 32).

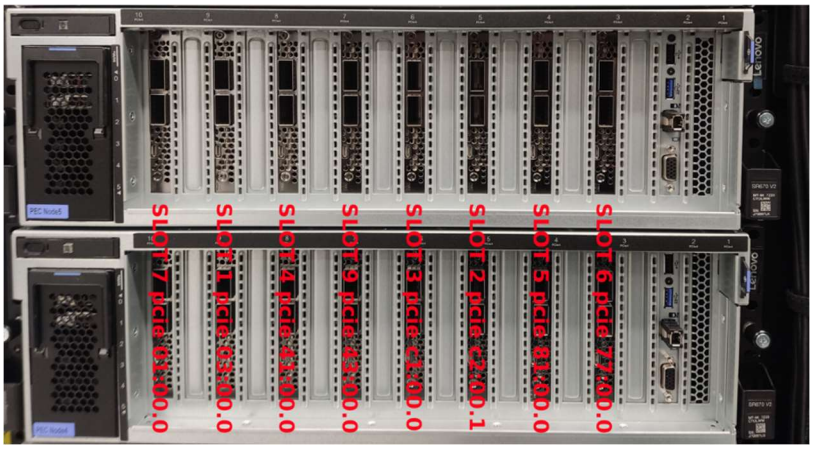


Figure 31. Mapping chassis example

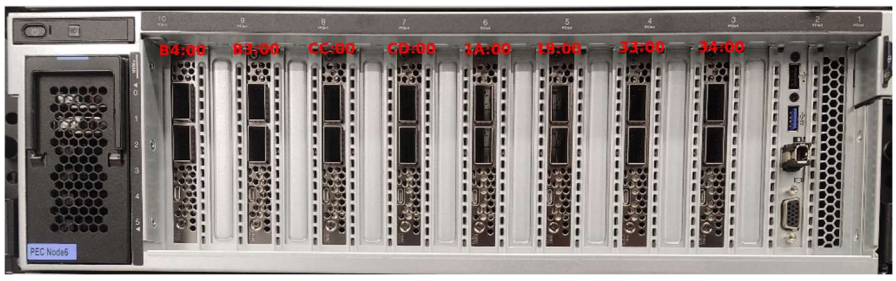


Figure 32. PCIe – FPGA mapping example

At functionality level, another challenge was to adapt the PCIe drivers to two different users's requirements, and then at system configuration level being able to make them compatible. In this sense, the FPGA team used two different working scenarios: one operating with *xdma* drivers, and another with *qdma* drivers. More details are included in Table **13**.

Table 13. More configuration details.

PCIe Drivers
PCI Drivers: • `xocl` (PCIe User Physical Function) Driver Interfaces • `xclmgmt` (PCIe Management Physical Function) Driver Interfaces QDMA: • BSC ONIC version (Ethernet over PCI) XDMA: • BSC version UART and USB: • Picocom (UART client)
Other software packages installed
Installed several software packages: • OmpSs toolchain (boost, boost-devel, hwloc-devel, numactl-devel) • gcc and g++ • ninja, clang and lld • gfortran • More under user's demand

Complementary to the information above, *udev* rules were introduced to allow user access to different devices (DMA, PCIe, and USB).

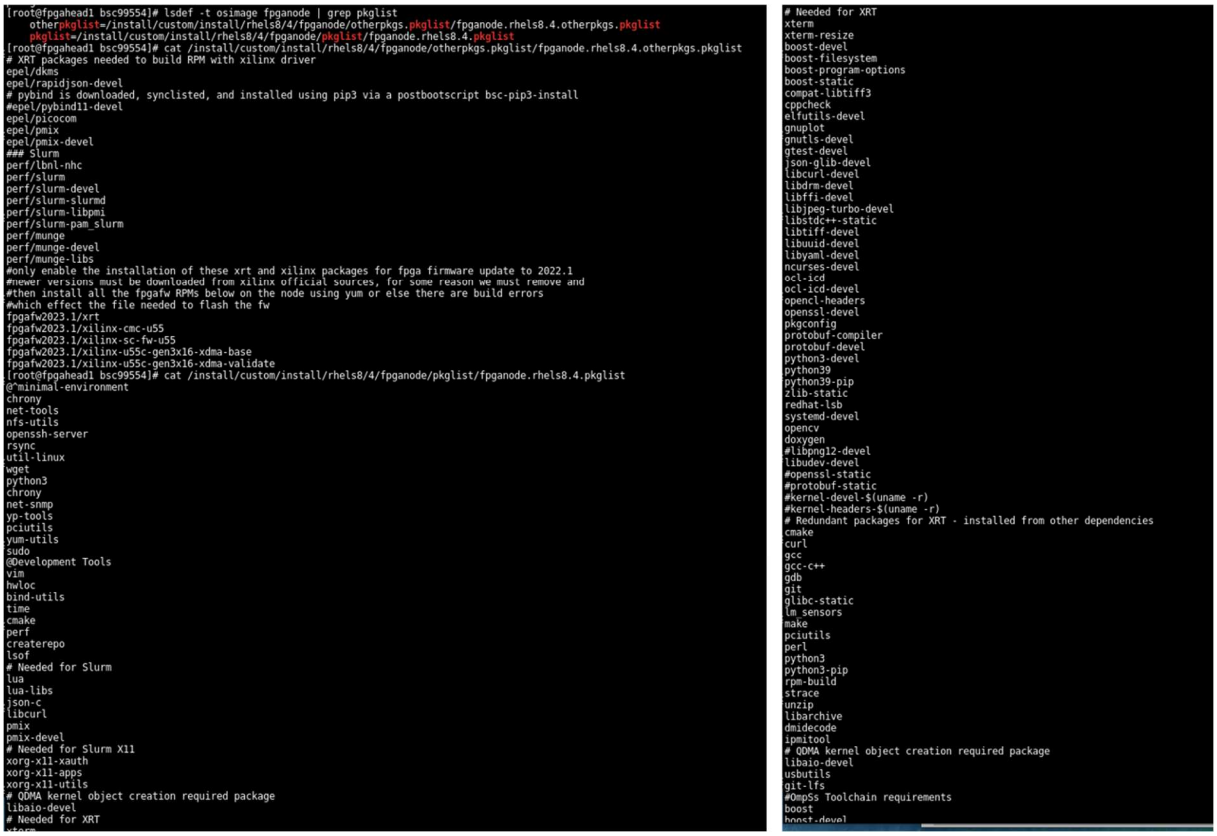


Figure 33. Installed packages on FPGA nodes

An overview of all the packages installed in all FPGA nodes (*fpgan[01:12]*) is shown in Figure **33**.

Custom requirements

Besides the standard Vitis flow, the most basic use case for an FPGA cluster is to use custom bitstreams from scratch, which means having to complete the FPGA programming step. These custom bitstreams intend to have one of two basic options of PCIe configuration: QDMA (used in MEEP project) and XDMA (used in EPI project). Accordingly on the host side two kinds of PCIe drivers should be used to interact with custom bitstreams. For both of them the QDMA and XDMA drivers provided by Xilinx are taken as reference: https://github.com/Xilinx/dma_ip_drivers.

5.2.3. Networking and usability

Figure **34** shows the networking structure of the MEEP system, and the interaction among its different elements. The figure below represents a schematic of the envisioned final system; in which users connect to the BSC LAN and access to any of the resources of the machine via Slurm. The elements shown in the figure correspond to the ones described in Table **12**.

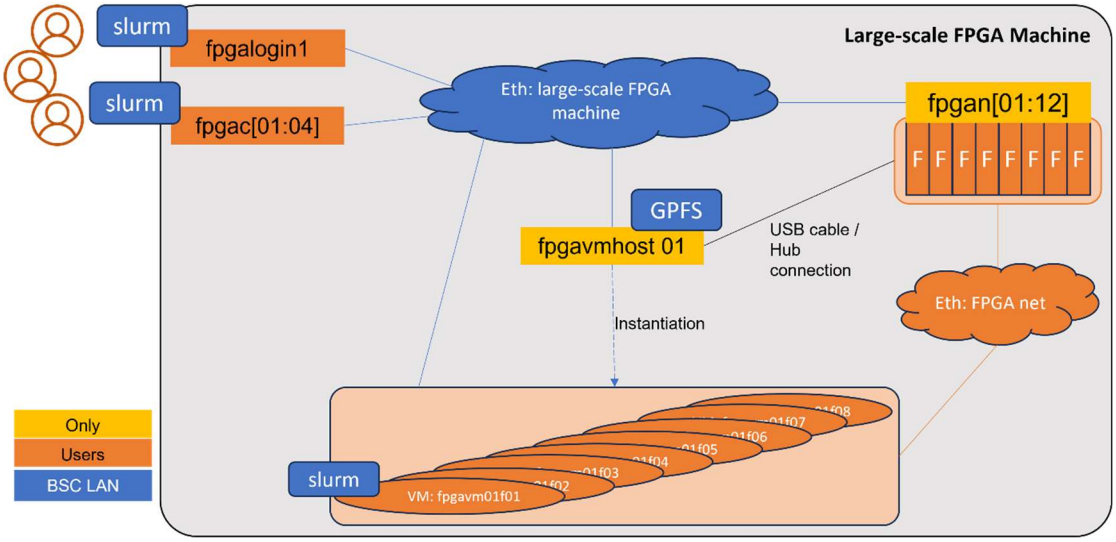


Figure 34. General schematic of the network designs and interactions

There are specific elements only accessible to Operations (yellow boxes{*fpgan[01:12], fpgavmhost01*}), and others to final users (orange boxes {*fpgalogin, fpgac[01:04], fpgavm[01:12]f[01:08]*}).

The flow is as follows:

- The large-scale FPGA machine is connected to the BSC LAN network (blue cloud: *Eth:large-scale FPGA machine*).
- As part of the system, Operations team has:
  - A host node (*fpgavmhost*) to control the status of each of the FPGA cards of the system, and it also allows sysadmin operations through the network.
  - 12 FPGA nodes to control each of the nodes to which users can operate.

A user connects to the machine through the *fpgalogin* node via `ssh`, allocating resources for his operation and configuring his requirements via Slurm. The system will assign him the requested resources, if there are any available, after configuring the resources. This process implies creating a VM per each of the resources requested (*fpgavm<node>f<fpga>*).

Sysadmin responsibilities: Operations team

Operations team manages the infrastructure (management, maintenance, support, and software installation). The team is responsible for:

- Creating a generic *operator* in the *fpgan[01:12]* for allowing users to access.
  - Operation manages the *sudo* requirements.
  - Operation develops scripts to manage FPGA tasks.
- Developing the prologue and epilog of Slurm; trying to include all common tasks and needs as part of the root definition.
- Enabling basic functionalities for users. In this sense, for those operatives that need to be done by the user, the team will develop:
  - Stub script with `setgid` to some special group that has access to `ssh` private key of *operator.*

- The stub script will `ssh` to the associated fpgan[01:12] node and perform a specific action only to its FPGA.

Machine operative: User level

Figure **35** represents the first scenario that was enabled to allow to the FPGA team accessing to the system. In the first stage, only one node (*fpgan01*) was configured, and direct access to each of the FPGAs was permitted via `ssh`; with no slum intervention. This facilitated that several users could execute small experiments simultaneously on specific FPGAs within the same node.

To facilitate users the possibility of transferring files from/to external repositories to/from the machine, two different NFS servers were configured: 1) One for the hosts (*fpgalogin* & *fpgan[01:12]*), and 2) one for the FPGAs. The relationship among them is shown in Figure **34**.

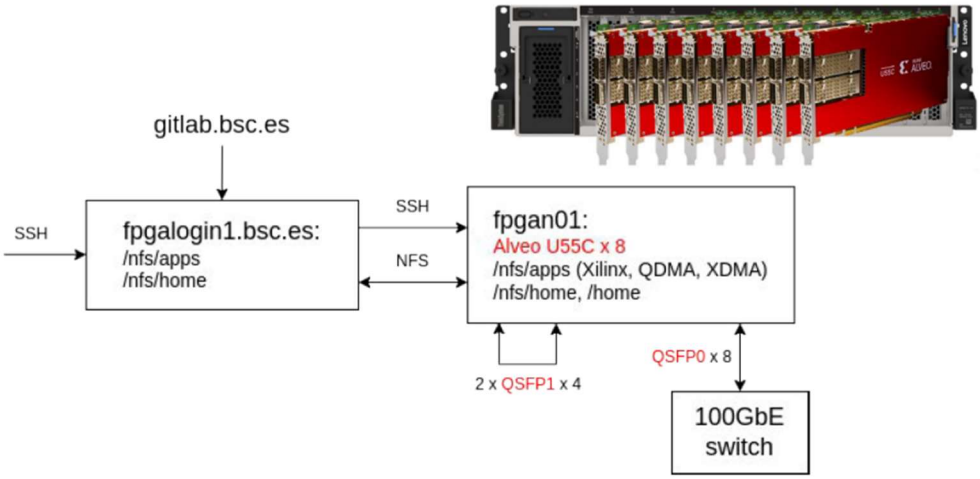


Figure 35. User interactions and features in Rack1, FPGA node 01

For security reasons there is not any bridge between NFS systems (hosts, and FPGAs). That means that users must do a secure copy (`scp`) of files into the FPGA to operate with them.

A welcome message is shown in the terminal when a user login in the machine (Figure **36**). Then, to do automation, the mapping of different FPGA interfaces (PCIe slot, USB UART, USB JTAG, Ethernet IP) to each other is required. The mapping per node is collected in a special file found in the path: */etc/motd* that might be checked after login. Figure **37** shows an example of the FPGA interconnection mapping, for the FPGA node 03 (*fpgan03*).

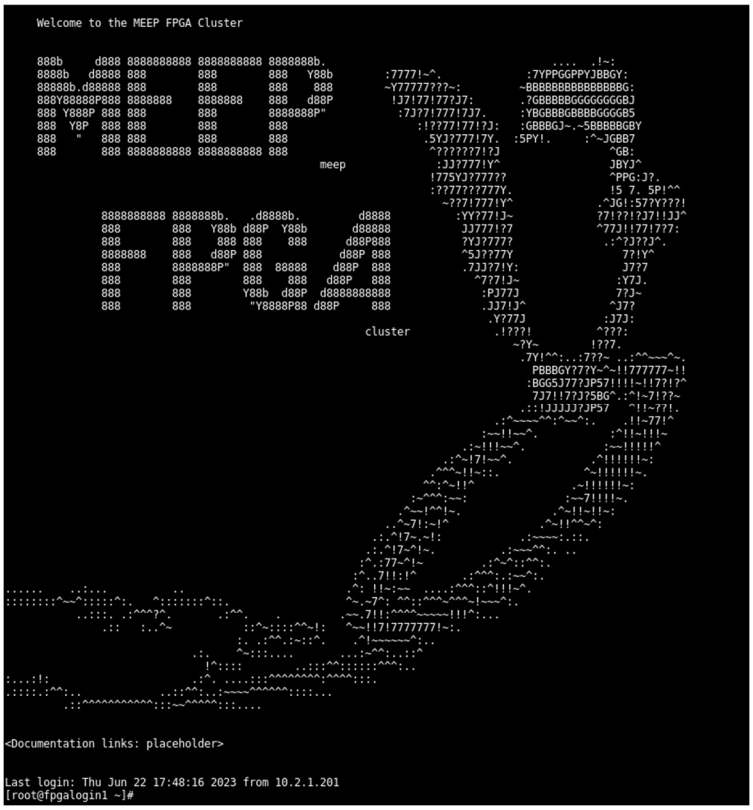


Figure 36. Welcome message when login in MEEP system

This table below includes the information for each of the FPGAs of the node. First column indicates each of the cards (*fpgan03f[01:08]*), the chassis PCIe port (*Chassis*), the serial number that identifies univocally each card (*FPGA serial*), the MAC address for the PCIe bus (*PCIe Bus*), the port assigned to the in the USB Hub (*USBPort*), link to use the UART (*ttyUSBx*), and the rest of the columns are related to networking information. All FPGA cards are connected to the 100GbE switch by using the QSPF0 (*QSFP0*), directly connected to its immediate neighboring FPGA using the QSFP1 (this is only true in Rack1) (*QSFP1*), and last two columns are related to the PCIe using ONIC driver. *QDMA onic* column identifies univocally the PCIe to allow communication between the FPGA and its corresponding host; whereas the last column (*onic IP*) refers to the MAC address for enabling Ethernet over PCIe.

```
akropotov@KROP-BSC-PC:~/Projects$
akropotov@KROP-BSC-PC:~/Projects$ ssh fn03
+------------+---------+--------------+----------+---------+----------------------+--------+------------+------------+------------+
| FPGA Card  | Chassis | FPGA Serial  | PCIe Bus | USBPort | ttyUSBx              | QSFP0  | QSFP1      | QDMA onic  | onic IP    |
+------------+---------+--------------+----------+---------+----------------------+--------+------------+------------+------------+
| fpgan03f01 | 3       | XFL10FQTD3UO | 34:00.0  | 1       | USB-UART-XFL10FQTD3UO | Switch | fpgan03f02 | onic52s0f0 | 10.0.1.1/24 |
+------------+---------+--------------+----------+---------+----------------------+--------+------------+------------+------------+
| fpgan03f02 | 4       | XFL1LIPHT51Z | 33:00.0  | 2       | USB-UART-XFL1LIPHT51Z | Switch | fpgan03f01 | onic51s0f0 | 10.0.2.1/24 |
+------------+---------+--------------+----------+---------+----------------------+--------+------------+------------+------------+
| fpgan03f03 | 5       | XFL104MXE44V | 19:00.0  | 3       | USB-UART-XFL104MXE44V | Switch | fpgan03f04 | onic25s0f0 | 10.0.3.1/24 |
+------------+---------+--------------+----------+---------+----------------------+--------+------------+------------+------------+
| fpgan03f04 | 6       | XFL1A1AHHGFS | 1a:00.0  | 4       | USB-UART-XFL1A1AHHGFS | Switch | fpgan03f03 | onic26s0f0 | 10.0.4.1/24 |
+------------+---------+--------------+----------+---------+----------------------+--------+------------+------------+------------+
| fpgan03f05 | 7       | XFL1EU4UUV2G | cd:00.0  | 5       | USB-UART-XFL1EU4UUV2G | Switch | fpgan03f06 | onic205s0f0 | 10.0.5.1/24 |
+------------+---------+--------------+----------+---------+----------------------+--------+------------+------------+------------+
| fpgan03f06 | 8       | XFL15N5LI3OL | cc:00.0  | 6       | USB-UART-XFL15N5LI3OL | Switch | fpgan03f05 | onic204s0f0 | 10.0.6.1/24 |
+------------+---------+--------------+----------+---------+----------------------+--------+------------+------------+------------+
| fpgan03f07 | 9       | XFL1A2WO1XO3 | b3:00.0  | 7       | USB-UART-XFL1A2WO1XO3 | Switch | fpgan03f08 | onic179s0f0 | 10.0.7.1/24 |
+------------+---------+--------------+----------+---------+----------------------+--------+------------+------------+------------+
| fpgan03f08 | 10      | XFL1RRBE2CTN | b4:00.0  | 8       | USB-UART-XFL1RRBE2CTN | Switch | fpgan03f07 | onic180s0f0 | 10.0.8.1/24 |
+------------+---------+--------------+----------+---------+----------------------+--------+------------+------------+------------+
Last login: Thu Jun  8 19:24:56 2023 from 10.2.1.211
[bsc00497@fpgan03 ~]$
```

Figure 37. FPGA card interconnection mapping

5.2.4. SLURM configuration

Once the first working nodes were stable, in terms of configuration, Operations team configured those to be accessed via SLURM. That process required the configuration of the *prolog* and *epilog* files to define what kind of actions need to be done when one user request resources, as it is depicted in Figure **38**.

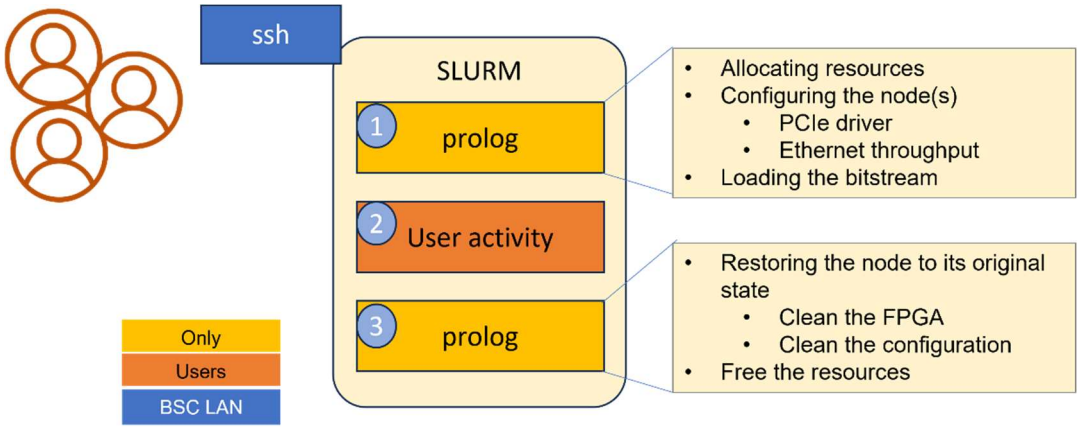


Figure 38. Generic SLURM workflow

SLURM constraints

Until now, and based on our experience, there are two main constraints that need to be considered when a user wants to use the large-scale FPGA system:

- Ethernet link speed (`eth:{auto, 10g, 100g}`)
  - Configures all switch ports of the FPGAs in the node to the given speed.
  - Please note that ALL cards have to be configured to the given Ethernet speed.

- DMA driver: (`dma{none, xdma, qdma}`)
  - Loads the given kernel module.

These constraints might be used separately or together, depending on users' needs. An example of this is shown in Figure **39**. Constraints are given as a comma separated list with the --constraint flag at the beginning of the jobscript. Examples below:

```
# One constraint: Ethernet at 10GbE
--constraint=eth:10g

# One constraint: qdma driver
--constraint=dmaqdma

# Two constraints: Ethernet at 100GbE and qdma driver
--constraint=eth:100g,dma=dmaqdma
```

Figure 39. Constraints setup

Each combination of key-value pair of constraints must be included in the *slurm.conf* file:

```
NodeName=fpgan-[01-12]
Features=eth:10g,eth:100g,dma:none,dma:xdma,dma:qdma
```

Users can only include valid job constraints, otherwise the system will show an error message as a response:

```
$ salloc -N 1 -t 1-00:00 --constraint=dma:custom_driver
salloc: error: Job submit/allocate failed: Invalid feature specification
```

If there are conflicting constraints in the same job submission only the latest applies. For example, `--constraint=dma:none,dma:dmaxdma` will set up the XDMA driver.
There is still one open ticket support with Lenovo, related to the reset of the FPGA to move the FPGA to an idle state without forcing a node reboot every time a new user is going to use it. This action will be included as part of the SLURM epilog.

Use case 1: One developer per node

Developers can reprogram the FPGAs with whatever bitstream they want and talk to the boards either via PCIe and/or UART. This use case is the same as the environment that was used for running the bring-up, but with two major differences:

1) Only one user is in the node at the same time and accesses via SLURM.
2) Users cannot load/unload kernel modules, they request the DMA driver as a job constraint. This removes the need for *sudo* on certain scripts.

5.2.5. Infrastructure as a service (usability)

As shown in Figure **38**, the MEEP system has been configured to allow a flexible use from the users' perspective. With the current configuration, a user might use the infrastructure to implement one of the three following scenarios:

- Single FPGA design (regular accelerators/SDV/silicon prototyping)
- Multiple concurrent FPGA designs (several hardware kernels synthesized and managed by OmpSs/AIT)
- Multi-FPGA designs (scaled-up above cases)

From the design perspective, the MEEP infrastructure envisions supporting any of the scenarios depicted in Figure **40**. Two of the three have been tested as part of the bring-up process. More specifically, designs with one FPGA and multiple FPGAs, with one design per FPGA, have been tested. The third scenario has not been tested because we do not have any design with that capability or need yet.

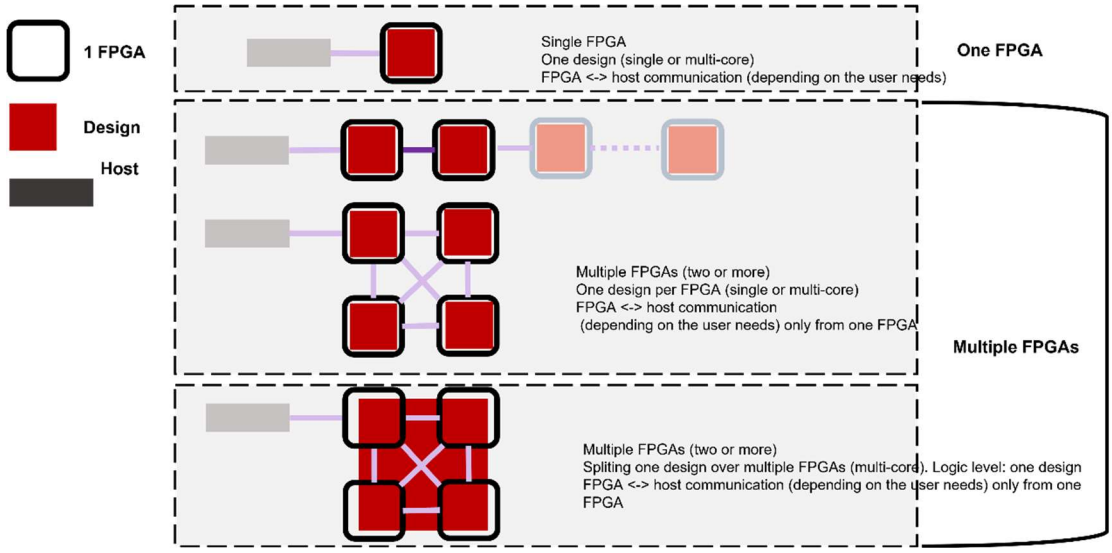


Figure 40. Design configurations allowed in MEEP system

It is important to mention that there are not direct cables connected among all the FPGAs to support all topologies. However, those could be implemented though the ethernet switch.

5.2.6. Configuration progress and status

Regarding the infrastructure, as it is shown in Figure **41** all nodes are accessible via SLURM. The image depicts an specific moment in which the first FPGA nodes (`fpgan[01-02]`) are not available (rebooting process), the next two FPGA nodes are being used (`fpgan[03-04]`), the rest of them are idle (`fpgan[05-12]`).

```
[root@fpgahead1 postscripts]# sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
main         up   infinite      2  down* fpgan[01-02]
main         up   infinite      2  alloc fpgan[03-04]
main         up   infinite      8   idle fpgan[05-12]
```

Figure 41. Slurm partitions (one per node)

A more detailed description of the user access hierarchy is represented in Figure **42**, although the image only shows Rack1. However, this is the same for Rack2, except for the point-to-point communication of pairs of FPGAs using QSFP1; which is only available in Rack1 for the moment. A user accesses the compute node with 8 FPGA cards through two level `ssh`: 1) access to the login node, and 2) access to the FPGA node. The access is enabled and controlled by SLURM. All nodes share a common NFS. Gitlab access and SLURM control are available at the first-stage FPGA login node. In addition, Table **14** summarizes the current features enabled for a user when using SLURM.

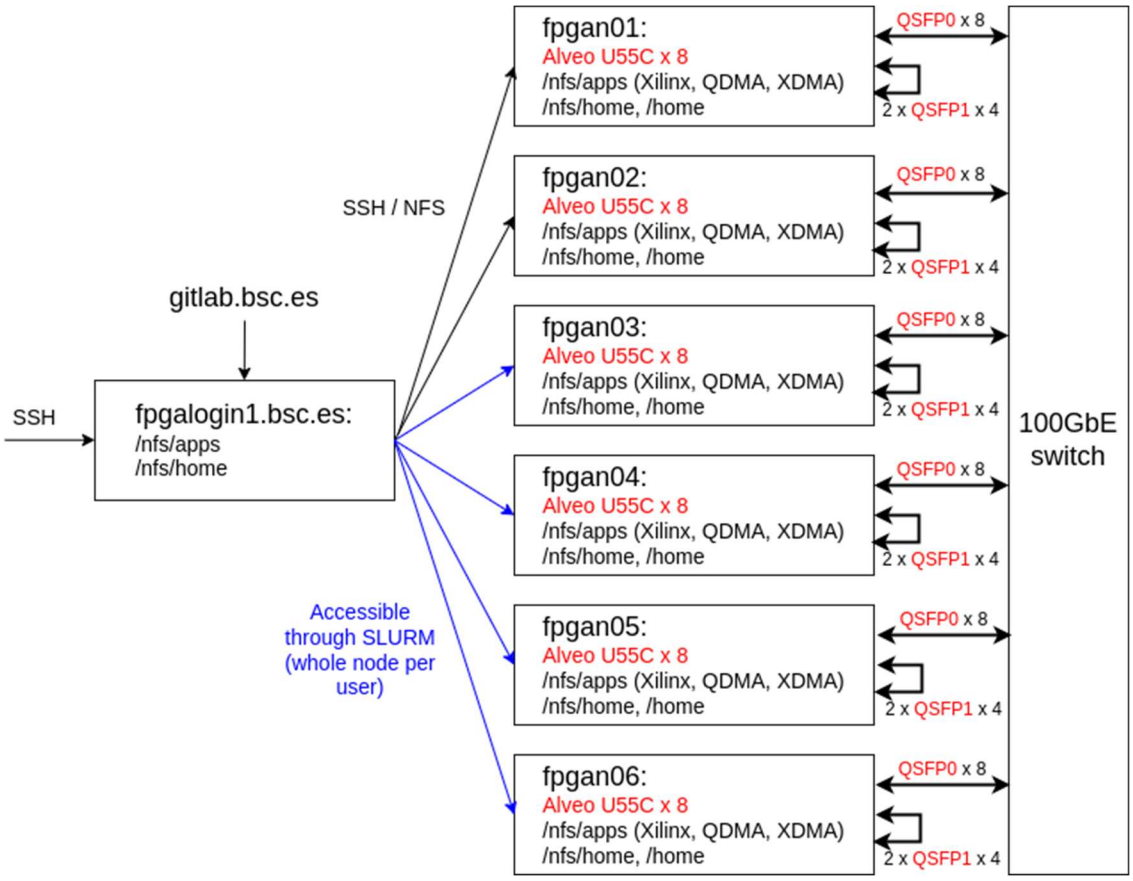


Figure 42. Hierarchy of user access to FPGA nodes in Rack1

Table 14. Summary of user features

	Feature	Comment
✗	Load/Unload kernel modules	Done in the SLURM prolog
☑	R/W permissions to ttyUSB devices	Already implemented with `udev` rules
☑	R/W permissions to PCIe remove and rescan files	Already implemented with `udev` rules
☑	R/W permissions to /dev/xdma*	Already implemented with `udev` rules
☑	Any bitstream can be programmed	Same as today The wrong bitstream could break the node

5.3 Capabilities and bring-up

The bring-up step requires validating the correct behavior of the whole infrastructure, and all its individual components (nodes, switches, processors and FPGAs). This step must guarantee access to the infrastructure for its future exploitation by the users and check all the necessary tools are available to ensure the appropriate usability of the resources. An acceptance process has been prepared for this purpose. More details on the status of the acceptance are in Table **15**.

Table 15. Checklist with experiments to be included as part of the bring-up

Acceptance tests	Status
Checking components	
All nodes (hosts) are alive and reachable: Rack1: [fpganode01:fpganode06] Rack2: [fpganode07:fpganode12]	✓ ✓
All switches are alive and reachable Rack1: fpgaibsw1 Rack2: fpgaibsw2	✓ ✓
All FPGAs are alive and visible (*xbtest / xbutils*) fpganode01|f01:f08 – fpganode12|f01:f08	✓
Checking FPGA programmability	
All FPGAs programmability using JTAG fpganode01|f01:f08 – fpganode12|f01:f08	✓
All FPGAs serial output using UART fpganode01|f01:f08 – fpganode12|f01:f08	✓
Checking FPGA connectivity (Ethernet and FPGA-to-FPGA)	
~~Point-to-point communication using direct QSFP links (cabling) (*ibert*)~~	-
Point-to-point communication using Ethernet over QSFP1 between FPGAs in the same node (Only Rack1) Rack1: fpganode01:06|f01-f02 f03-f04 f05-f06 f07-f08	✓ ✓ ✓ ✓
FPGA-to-FPGA communication through the switch using Ethernet over QSFP0 between FPGAs in the same or different nodes, but same rack	✓
FPGA-to-FPGA communication through the switch using Ethernet over QSFP0 between FPGAs in different racks	✓
FPGA-to-FPGA communication using Aurora over QSFP1 between FPGAs in the same node (Only Rack1)	✓
Host-to-FPGA and FPGA-to-host communication using PCIe (no Ethernet)	✓
Host-to-FPGA and FPGA-to-host communication using Ethernet over PCIe	✓
Checking toolchain, software and EDA tools	
Access to EDA tools	✓
Loading a bitstream with a functional design and booting Linux (OpenSBI)	✓
Configuring several FPGAs with the same bitstream	✓
SLURM installation, configuration and exploitation at user level	✓
CICD flow from Gitlab repo to large-scale FPGA machine (Optional)*	✓

~~RISC-V toolchain compatibility (Optional)~~	-
~~SW toolset installation, compatibility, and correctness (OpenMP, MPI) (Optional)~~	-

The list of tests has been organized into four main categories, where the three first are the most relevant for ensuring the correctness of all hardware components and the possibility of using those for MEEP and future projects:

1) Checking components: The goal of this set of tests is to verify the correctness of each of the hardware components of the system: cabling, nodes, power supply, cooling, switches, and FPGA cards.
2) Checking FPGA programmability: These tests aim for ensuring the possibility of using the FPGA cards by being able to: a) load a bitstream via JTAG, and b) reading serial outputs of the tasks executed on the FPGA using the UART. In the end, all these tests guarantee the correct behavior and configuration of the USB Hubs.
3) Checking FPGA connectivity: Ensuring the networking capabilities of the system is necessary to offer the targeted services to be offered by the large-scale FPGA system. Thus, validating the networking inter and intra nodes (from/to the FPGAs) is mandatory. This includes: PCIe ←→FPGA, FPGA←→FPGA via QSFP1, and through the switch (no matter if the FPGAs are in the same node, in the same Rack but different node, or in different Racks.
4) Checking toolchain, software, and EDA tools: Most of these tests are optional for the bring-up process. However, they have been considered to start testing, and evolving our tools to deal with the complexity of a large-scale system as this.

Some tests that have been removed from the list for the following reasons:

- *Point-to-point communication using direct QSFP links (cabling)* (*ibert*): The networking tests were much more exhaustive than this one, and consequently we decided to avoid redundancies and simplify the experiments to the most sophisticated ones.
- *RISC-V toolchain compatibility & SW toolset installation, compatibility, and correctness (OpenMP, MPI)*: both tests were set as *Optional* since they are not mandatory for the bring-up. These are the kind of checks that are nice to test to demonstrate some of the functionalities of the system as software development vehicle. In this sense, these tests will be run soon.

In the case of the *CICD flow from Gitlab repo to large-scale FPGA machine* test, BSC team developed initial tests to check the possibility of pointing to the large-scale FPGA machine as target where to deploy a bitstream generated through the FPGA flow. The mechanism was validated, although not put in production yet.

5.3.1. Xilinx tests

The hardware validation of all FPGAs cards has been done using some of the tools provided by the vendors (*AMD/Xilinx*) for this purpose; more specifically: *xbutils* and *xbtest*. These consist of running a set of tests in all FPGA cards to check their correct status.

1) *xbutils* provides basic validation for all the U55C FPGA cards connected to one PCIe node.

2) *xbtest:* provides more advanced tests and validates the host server environment under a variety of stress conditions. The application monitors the system and validates the functionality of essential hardware and software components of the platform.

Once both racks (Rack1 and Rack2) were fully connected (cabling and networking) and configured, it was required to do a firmware (FW) upgrade. The FW implements the logic to interface the Satellite Controller (SC), and the SC itself. A mechanism that permits the measurements. Figure **43** shows the output of all FPGAs of the large-scale machine (96 FPGAs) after running a firmware upgrade validation testing using *xbutil* (FW version 2023.1).

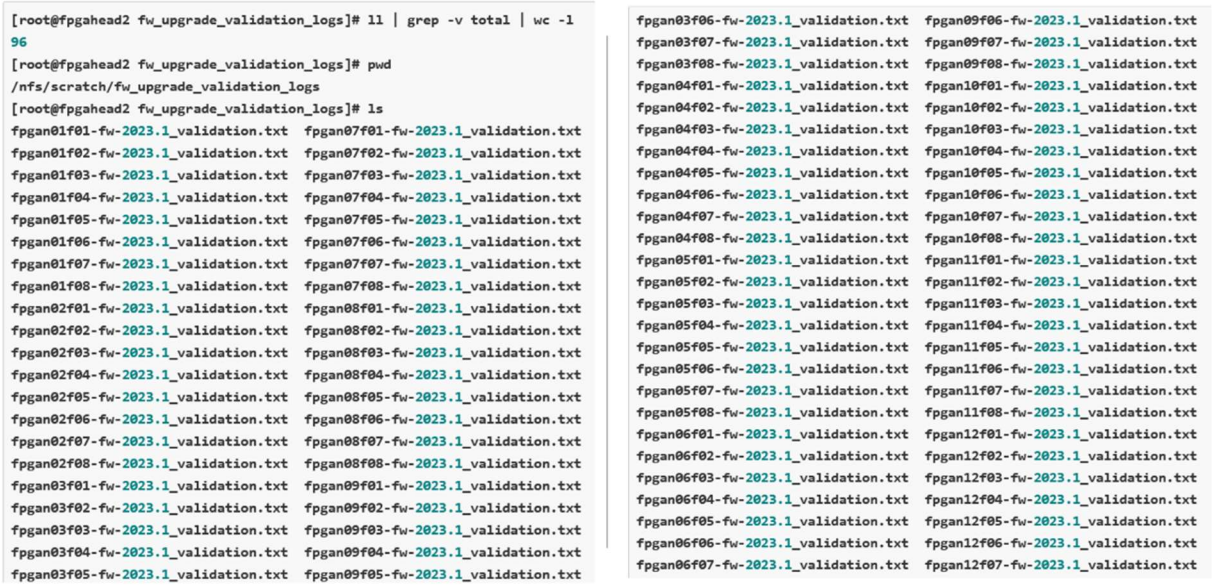


Figure 43. Firmware upgrade and validation testing, in all FPGAs, using xbutil

The output shows each of the cards of the system per node, starting from FPGA node 01 (*fpgan01*) (Figure **43**, left column), going through each of the eight FPGA cards, to the last FPGA card of FPGA node 12 (*fpgan12*) (Figure **43**, right column).

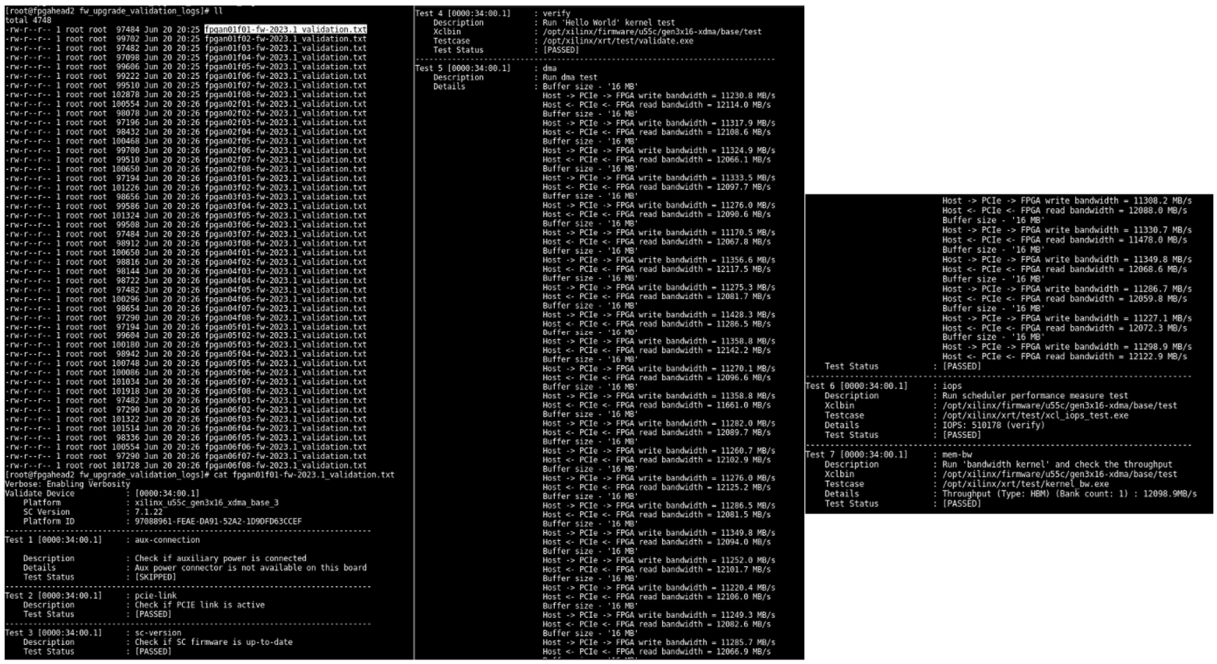


Figure 44. Firmware upgrade and validation testing detail on fpgan01

Figure **44** depicts a detailed output of the validation tests executed (`Test 1` to `Test 7`) in one particular node, in this case *fpgan01*.

After a FW update, a cold reboot is needed to guarantee the new changes have been applied to the system. Then, the following commands are useful to validate the process: 1) `xbmgmt examine`, and 2) `xbutils examine`. Having Xilinx Platform programmed to FPGAs after cold reboot `xbmgmt` and `xbutil` utilities report all 8 FPGA cards as 2 PCIe devices per each (Figures 45 and 46). These results validate a successful FW update.

```
[bsc00497@fpgan09 ~]$
[bsc00497@fpgan09 ~]$ /opt/xilinx/xrt/bin/xbmgmt examine
System Configuration
  OS Name              : Linux
  Release              : 4.18.0-305.25.1.el8_4.x86_64
  Version              : #1 SMP Mon Oct 18 14:34:11 EDT 2021
  Machine              : x86_64
  CPU Cores            : 112
  Memory               : 257318 MB
  Distribution         : Red Hat Enterprise Linux 8.4 (Ootpa)
  GLIBC                : 2.28
  Model                : ThinkSystem SR670 V2

XRT
  Version              : 2.15.225
  Branch               : 2023.1
  Hash                 : adf27adb3cfadc6e4c41d6db814159f1329b24f3
  Hash Date            : 2023-05-03 17:15:51
  XOCL                 : 2.15.225, adf27adb3cfadc6e4c41d6db814159f1329b24f3
  XCLMGMT              : 2.15.225, adf27adb3cfadc6e4c41d6db814159f1329b24f3

Devices present
BDF             :  Shell                          Platform UUID                        Device ID         Device Ready*
------------------------------------------------------------------------------------------------------------------------
[0000:19:00.0]  :  xilinx_u55c_gen3x16_xdma_base_3  97088961-FEAE-DA91-52A2-1D9DFD63CCEF  mgmt(inst=6400)   Yes
[0000:1a:00.0]  :  xilinx_u55c_gen3x16_xdma_base_3  97088961-FEAE-DA91-52A2-1D9DFD63CCEF  mgmt(inst=6656)   Yes
[0000:33:00.0]  :  xilinx_u55c_gen3x16_xdma_base_3  97088961-FEAE-DA91-52A2-1D9DFD63CCEF  mgmt(inst=13056)  Yes
[0000:34:00.0]  :  xilinx_u55c_gen3x16_xdma_base_3  97088961-FEAE-DA91-52A2-1D9DFD63CCEF  mgmt(inst=13312)  Yes
[0000:b3:00.0]  :  xilinx_u55c_gen3x16_xdma_base_3  97088961-FEAE-DA91-52A2-1D9DFD63CCEF  mgmt(inst=45824)  Yes
[0000:b4:00.0]  :  xilinx_u55c_gen3x16_xdma_base_3  97088961-FEAE-DA91-52A2-1D9DFD63CCEF  mgmt(inst=46080)  Yes
[0000:cc:00.0]  :  xilinx_u55c_gen3x16_xdma_base_3  97088961-FEAE-DA91-52A2-1D9DFD63CCEF  mgmt(inst=52224)  Yes
[0000:cd:00.0]  :  xilinx_u55c_gen3x16_xdma_base_3  97088961-FEAE-DA91-52A2-1D9DFD63CCEF  mgmt(inst=52480)  Yes


* Devices that are not ready will have reduced functionality when using XRT tools
[bsc00497@fpgan09 ~]$
```

Figure **45**. Output of `xbmgmt examine` command on fpganode09.

```
[bsc00497@fpgan09 ~]$
[bsc00497@fpgan09 ~]$ /opt/xilinx/xrt/bin/xbutil examine
System Configuration
  OS Name              : Linux
  Release              : 4.18.0-305.25.1.el8_4.x86_64
  Version              : #1 SMP Mon Oct 18 14:34:11 EDT 2021
  Machine              : x86_64
  CPU Cores            : 112
  Memory               : 257318 MB
  Distribution         : Red Hat Enterprise Linux 8.4 (Ootpa)
  GLIBC                : 2.28
  Model                : ThinkSystem SR670 V2

XRT
  Version              : 2.15.225
  Branch               : 2023.1
  Hash                 : adf27adb3cfadc6e4c41d6db814159f1329b24f3
  Hash Date            : 2023-05-03 17:15:51
  XOCL                 : 2.15.225, adf27adb3cfadc6e4c41d6db814159f1329b24f3
  XCLMGMT              : 2.15.225, adf27adb3cfadc6e4c41d6db814159f1329b24f3

Devices present
BDF             :  Shell                          Platform UUID                        Device ID         Device Ready*
------------------------------------------------------------------------------------------------------------------------
[0000:19:00.1]  :  xilinx_u55c_gen3x16_xdma_base_3  97088961-FEAE-DA91-52A2-1D9DFD63CCEF  user(inst=128)  Yes
[0000:1a:00.1]  :  xilinx_u55c_gen3x16_xdma_base_3  97088961-FEAE-DA91-52A2-1D9DFD63CCEF  user(inst=129)  Yes
[0000:33:00.1]  :  xilinx_u55c_gen3x16_xdma_base_3  97088961-FEAE-DA91-52A2-1D9DFD63CCEF  user(inst=130)  Yes
[0000:34:00.1]  :  xilinx_u55c_gen3x16_xdma_base_3  97088961-FEAE-DA91-52A2-1D9DFD63CCEF  user(inst=131)  Yes
[0000:b3:00.1]  :  xilinx_u55c_gen3x16_xdma_base_3  97088961-FEAE-DA91-52A2-1D9DFD63CCEF  user(inst=132)  Yes
[0000:b4:00.1]  :  xilinx_u55c_gen3x16_xdma_base_3  97088961-FEAE-DA91-52A2-1D9DFD63CCEF  user(inst=133)  Yes
[0000:cc:00.1]  :  xilinx_u55c_gen3x16_xdma_base_3  97088961-FEAE-DA91-52A2-1D9DFD63CCEF  user(inst=134)  Yes
[0000:cd:00.1]  :  xilinx_u55c_gen3x16_xdma_base_3  97088961-FEAE-DA91-52A2-1D9DFD63CCEF  user(inst=135)  Yes


* Devices that are not ready will have reduced functionality when using XRT tools
[bsc00497@fpgan09 ~]$
```

Figure **46**. Output of `xbutil examine` command on fpganode09.

Some more details about Xilinx Platform and about the last loaded validation kernel (*bandwidth.xclbin*) for a particular card at PCIe slot 1 are provided by `xbutil examine -d 000:34:00.1` in Figure **47**.

```
[bsc00497@fpgan09 ~]$
[bsc00497@fpgan09 ~]$ /opt/xilinx/xrt/bin/xbutil examine -d 0000:34:00.1

-------------------------------------------------
[0000:34:00.1] : xilinx_u55c_gen3x16_xdma_base_3
-------------------------------------------------
Platform
  XSA Name                : xilinx_u55c_gen3x16_xdma_base_3
  Platform UUID           : 97088961-FEAE-DA91-52A2-1D9DFD63CCEF
  FPGA Name               :
  JTAG ID Code            : 0x14b7d093
  DDR Size                : 0 Bytes
  DDR Count               : 0
  Mig Calibrated          : true
  P2P Status              : disabled
  P2P IO space required   : 32 GB

Clocks
  DATA_CLK (Data)         : 300 MHz
  KERNEL_CLK (Kernel)     : 500 MHz
  hbm_aclk (System)       : 450 MHz

Mac Addresses             : 00:0A:35:0E:07:E0
                          : 00:0A:35:0E:07:E1
                          : 00:0A:35:0E:07:E2
                          : 00:0A:35:0E:07:E3
                          : 00:0A:35:0E:07:E4
                          : 00:0A:35:0E:07:E5
                          : 00:0A:35:0E:07:E6
                          : 00:0A:35:0E:07:E7

  Hardware Context ID: 0
    Xclbin UUID: 11F03FB7-84C7-D53B-6673-A65092E5A0BB
    PL Compute Units
      Index  Name                    Base Address  Usage  Status
      ------------------------------------------------------------
      0      bandwidth:bandwidth_1   0x800000      13     (IDLE)

[bsc00497@fpgan09 ~]$
```

Figure 47. Output of `xbutil examine` for a particular FPGA card at PCIe slot 1 on fpganode09

Xilinx Platform allows users to run validation tests for a particular FPGA card by `xbutil validate` command (Figure **48**). This is performed through loading to the FPGA dedicated hardware kernels, which run a set of basic health tests. Such validation procedure has been successfully run for all 96 FPGA cards of the FPGA cluster (*Test 1 to Test 12*).

```
[bsc00497@fpgan09 ~]$
[bsc00497@fpgan09 ~]$ /opt/xilinx/xrt/bin/xbutil validate -d 0000:34:00.1 --verbose
Verbose: Enabling Verbosity
Validate Device           : [0000:34:00.1]
    Platform              : xilinx_u55c_gen3x16_xdma_base_3
    SC Version            : 7.1.22
    Platform ID           : 97088961-FEAE-DA91-52A2-1D9DFD63CCEF
-----------------------------------------------------------------------------
Test 1 [0000:34:00.1]     : aux-connection

    Description           : Check if auxiliary power is connected
    Details               : Aux power connector is not available on this board
    Test Status           : [SKIPPED]
-----------------------------------------------------------------------------
Test 2 [0000:34:00.1]     : pcie-link
    Description           : Check if PCIE link is active
    Test Status           : [PASSED]
-----------------------------------------------------------------------------
Test 3 [0000:34:00.1]     : sc-version
    Description           : Check if SC firmware is up-to-date
    Test Status           : [PASSED]
-----------------------------------------------------------------------------
Test 4 [0000:34:00.1]     : verify
    Description           : Run 'Hello World' kernel test
    Xclbin                : /opt/xilinx/firmware/u55c/gen3x16-xdma/base/test
    Testcase              : /opt/xilinx/xrt/test/validate.exe
    Test Status           : [PASSED]
-----------------------------------------------------------------------------
Test 5 [0000:34:00.1]     : dma
    Description           : Run dma test
    Details               : Buffer size - '16 MB'
                            Host -> PCIe -> FPGA write bandwidth = 11190.3 MB/s
                            Host <- PCIe <- FPGA read bandwidth = 11477.0 MB/s
                            Buffer size - '16 MB'
                            Host -> PCIe -> FPGA write bandwidth = 11284.0 MB/s
                            Host <- PCIe <- FPGA read bandwidth = 11482.7 MB/s
```

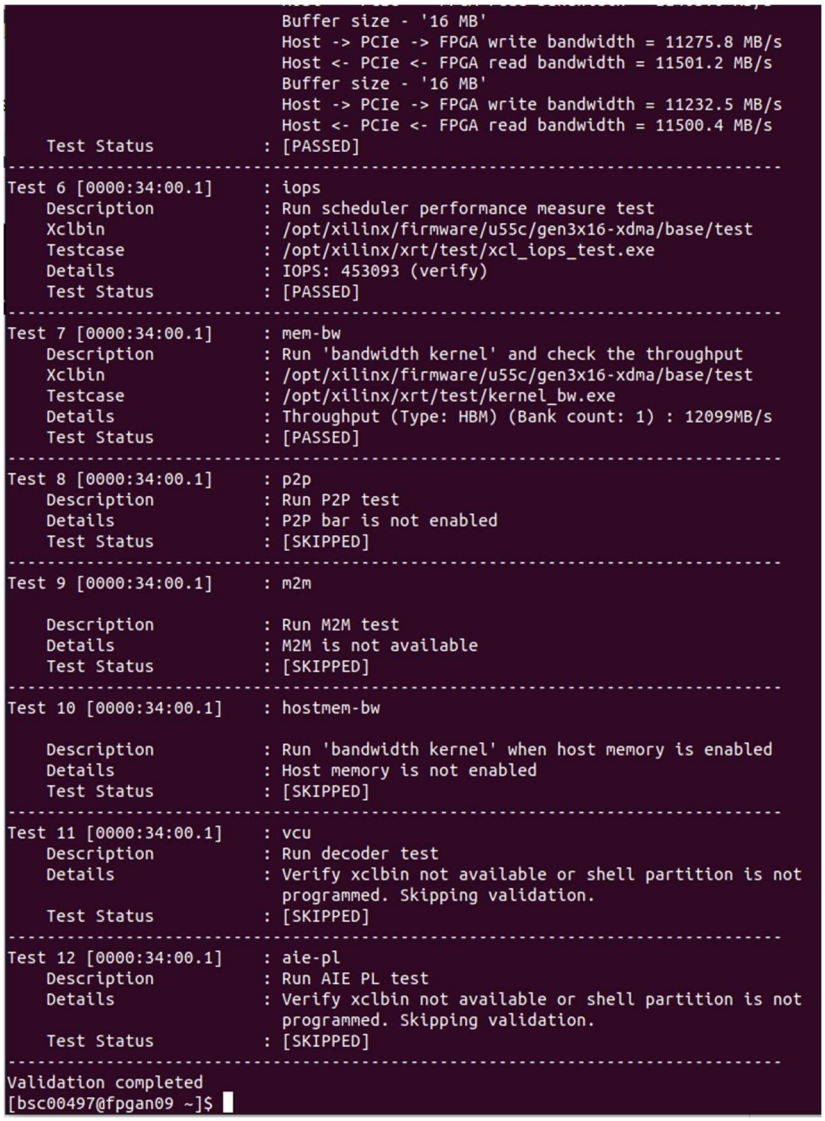


Figure 48. `xbutils validate` of FPGA card in PCIe slot 1 in fpganode09.

In Figure **49** default (post cold reboot) configurations of PCIe for cards in slots 3 and 4 are presented with activated two types of PCIe drivers coming with default XRT distribution: *xocl* (OpenCL for Vitis hardware kernels) and *xclmgmt* (FPGA card management).

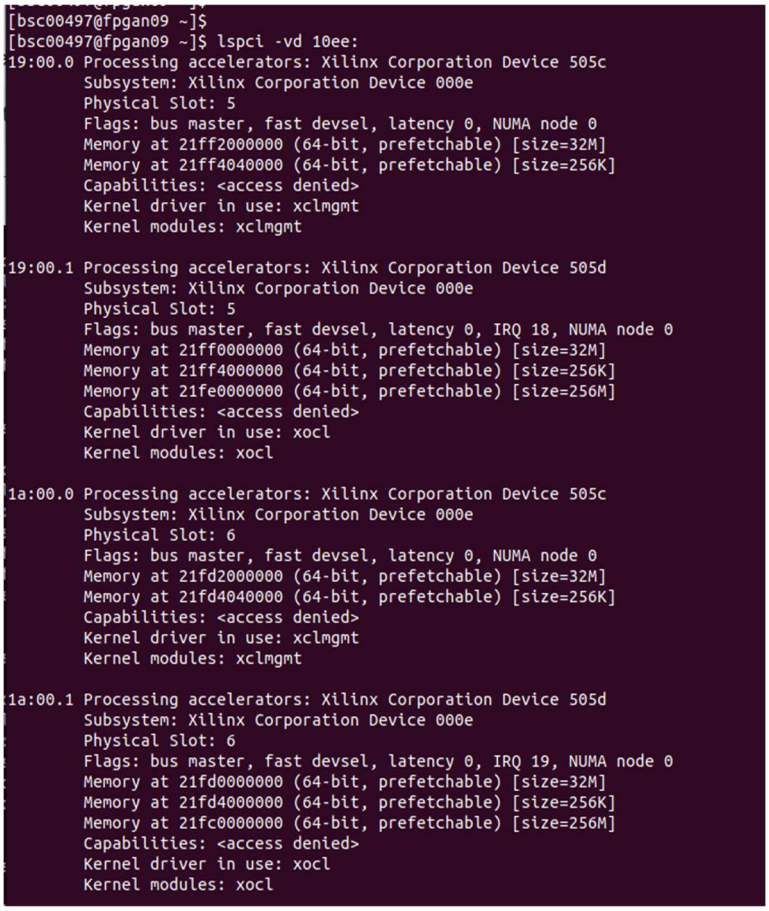


Figure **49**. Post cold reboot PCIe configurations for FPGA cards at PCIe slots 3 and 4.

5.3.2. Custom tests

Vendor tests guarantee the correctness of the FPGA cards of the system. But it is still pending to validate the basic functionality of the system, which is critical for ensuring the possibility of using the machine for the targeted functionalities: 1) being used as a pre-silicon validation platform for hardware developers, and 2) being used as a software development vehicle for software developers. The correctness of the basic features for succeeding on these functionalities have been validated by executing the BSC custom tests summarized in Table **16**.

Table 16. Summary of BSC custom tests

List of BSC custom tests
Basic FPGA card peripherals

• Access to local BSC repositories (in Gitlab)
• USB Hub connection
  ○ JTAG programming,
  ○ UART access,
• PCIe
  ○ QDMA tests,
  ○ XDMA tests,
• Bitstream deployment

Node Behavior

• Programming several times, the same FPGA, with different bitstreams
  ○ RISC-V accelerators mapping (SpAcc prototype, EPI@SDV prototype)
  ○ Networking tests
• Booting Linux (OpenSBI)
• RISC-V based (FPGA card – FPGA node (Host) communication through PCIe)
  ○ FPGA – host (no Ethernet)
  ○ FPGA – host (Ethernet over PCie)

Networking

• Point to point communication (cabling using QSFP1)
  ○ Ethernet tests with 100GbE point-to-point
  ○ Ethernet tests with 100GbE via switch

- The first set of tests, *Basic FPGA card peripherals*, are intended to check the functionality of basic I/O peripherals required for programming, establishing basic communication between the FPGA and host node, and checking output results using the UART. All these tests have required an intense development of BSC tools[4] to allow mapping automation for the following tasks over specific FPGA cards (*FPGA card #*):
  - Programming (fpgajtag). Allows programming an FPGA via JTAG.
  - QDMA actions (pcienum). Used for executing actions such as: remove, qmax, add, dma-ctl.
  - USB UART (/dev/ttyusb). Its function is to enable the UART communication using a specific USB port.
  - Ethernet over PCIe: Allows activating Ethernet over PCIe.
- The second set of tests, Node behavior, were intended to ensure that different designs could be deployed on the large-scale FPGA system without any difficulty. In this case two different accelerator designs were used: 1) ACME accelerator from MEEP project[5], and 2) the SDV design from EPI project[6].
  - In any of the cases, FPGA users were able to execute the following sequence of instructions, as it is shown in Figure **50**:
    - Program an FPGA in fpgan01
    - Rescan PCIe devices.
    - Check PCIe devices.
    - Set a device with the proper drivers.
    - Offload Linux kernel to the FPGA.
    - Send start signal to the RISC-V core in the FPGA.
      - Observe Linux booting via UART (*ttyUSB7*)

[4] Gitlab *fpga-tools* repository: https://gitlab.bsc.es/meep/FPGA_implementations/AlveoU280/fpga-tools/-/tree/develop/fpga_cluster
[5] MEEP Project: www.meep-project.eu
[6] EPI project: www.european-processor-initiative.eu

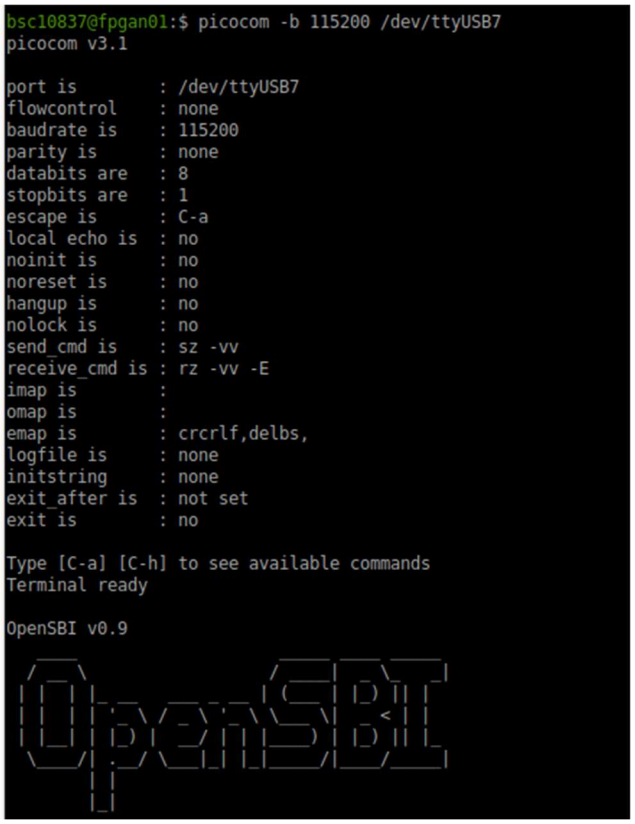


Figure 50. Linux Booting of a RISC-V based design on fpgan01

Moreover, the team has enabled several interesting features to exploit the system:

- ONIC driver: is being improved to add extra features like enabling Ethernet over PCIe from one node to multiple FPGAs, as part of the same node.
- Coexistence of QDMA and XDMA drivers in the same node.
- Coexistence of 10GbE and 100GbE Ethernet throughputs in the same node. This characteristic makes possible to have different FPGAs in the same node with the switch configured in a different way; some FPGAs at 10Gb, and others at 100Gb.

In case when custom bitstreams are programmed, accordingly a custom QDMA or XDMA host driver is activated to support PCIe functioning. In Figure 51 the changed PCIe configuration is presented for card at slot 3 after programming QDMA based custom bitstream with activation of custom *qdma-pf.ko* kernel module.

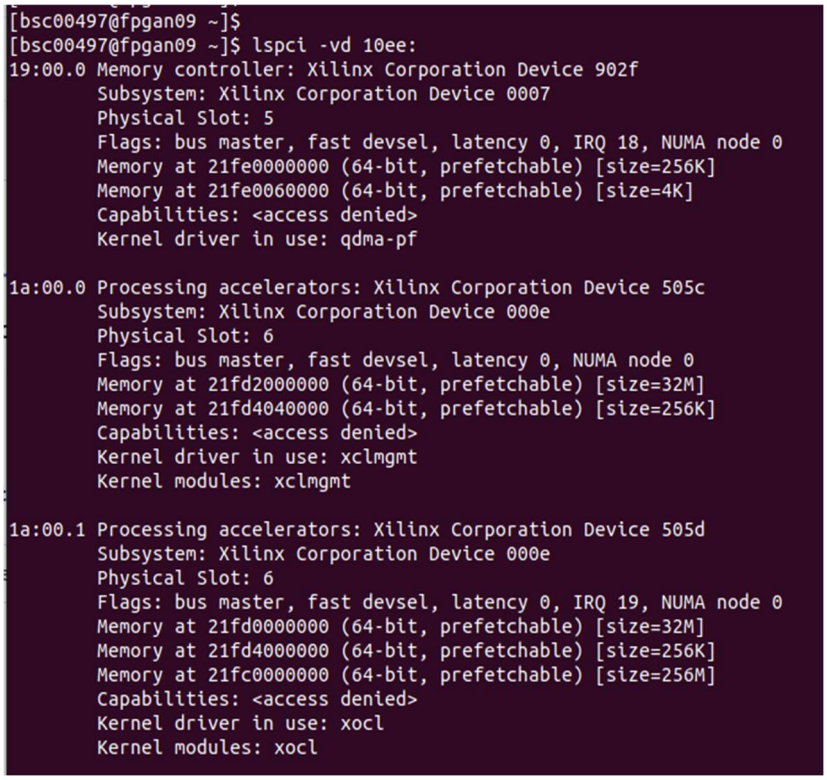


Figure **51**. Changed PCIe configurations for FPGA cards at PCIe slot 3 after its programming with custom bitstream and activation of custom PCIe QDMA driver.

Experimentally it has been checked that different types of PCIe drivers may coexist simultaneously with each other, and a user is able to reprogram different FPGA cards with bitstreams having arbitrary PCIe configurations. But generally, after such reprogramming a user is required to do a hot reboot of the host (FPGA state is kept untouched in that case) in order the OS can do PCIe bus enumeration and device address assignment. During such a procedure the OS extracts corresponding information from the Base Address Register (BAR) per each PCIe device. Configuration of BAR(s) is a part of PCIe IP configuration while implementing a bitstream.

An option to avoid a host reboot is to make the PCIe device `remove/rescan` operations[7]. This option is chosen as the basic one for FPGA cluster usage. But it assumes previously done preallocation of resources on a specific PCIe port during PCIe enumeration. Even more, this preallocation should satisfy the rule that it is done with BAR address space not less than BAR(s) in further reprogrammed custom bitstreams. In most cases Xilinx Platform bitstreams initialized after cold reboot satisfy this rule, but in order to have less restrictions for custom bitstreams, a special bitstream having big enough BAR configuration 512 MB is created and is planned to be programmed to FPGAs before rare cases of FPGA cluster node reboot. In order to do automation the mapping of different FPGA interfaces (PCIe slot, USB UART, USB JTAG, Ethernet IP) to each other is required. The mapping is collected in special file /etc/motd per a node showed at initial login, as depicted in Figure **52**:

[7] https://stackoverflow.com/questions/32334870/how-to-do-a-true-rescan-of-pcie-bus

```
akropotov@KROP-BSC-PC:~/Projects$
akropotov@KROP-BSC-PC:~/Projects$ ssh fn03
+------------+---------+-------------+---------+---------+----------------------+--------+-----------+------------+------------+
| FPGA Card  | Chassis | FPGA Serial | PCIe Bus | USBPort | ttyUSBx              | QSFP0  | QSFP1     | QDMA onic  | onic IP    |
+------------+---------+-------------+---------+---------+----------------------+--------+-----------+------------+------------+
| fpgan03f01 | 3       | XFL10FQTD3UO | 34:00.0  | 1       | USB-UART-XFL10FQTD3UO | Switch | fpgan03f02 | onic52s0f0 | 10.0.1.1/24 |
+------------+---------+-------------+---------+---------+----------------------+--------+-----------+------------+------------+
| fpgan03f02 | 4       | XFL1LIPHT51Z | 33:00.0  | 2       | USB-UART-XFL1LIPHT51Z | Switch | fpgan03f01 | onic51s0f0 | 10.0.2.1/24 |
+------------+---------+-------------+---------+---------+----------------------+--------+-----------+------------+------------+
| fpgan03f03 | 5       | XFL104MXE44V | 19:00.0  | 3       | USB-UART-XFL104MXE44V | Switch | fpgan03f04 | onic25s0f0 | 10.0.3.1/24 |
+------------+---------+-------------+---------+---------+----------------------+--------+-----------+------------+------------+
| fpgan03f04 | 6       | XFL1A1AHHGFS | 1a:00.0  | 4       | USB-UART-XFL1A1AHHGFS | Switch | fpgan03f03 | onic26s0f0 | 10.0.4.1/24 |
+------------+---------+-------------+---------+---------+----------------------+--------+-----------+------------+------------+
| fpgan03f05 | 7       | XFL1EU4UUV2G | cd:00.0  | 5       | USB-UART-XFL1EU4UUV2G | Switch | fpgan03f06 | onic205s0f0 | 10.0.5.1/24 |
+------------+---------+-------------+---------+---------+----------------------+--------+-----------+------------+------------+
| fpgan03f06 | 8       | XFL15N5LI3OL | cc:00.0  | 6       | USB-UART-XFL15N5LI3OL | Switch | fpgan03f05 | onic204s0f0 | 10.0.6.1/24 |
+------------+---------+-------------+---------+---------+----------------------+--------+-----------+------------+------------+
| fpgan03f07 | 9       | XFL1A2WO1XO3 | b3:00.0  | 7       | USB-UART-XFL1A2WO1XO3 | Switch | fpgan03f08 | onic179s0f0 | 10.0.7.1/24 |
+------------+---------+-------------+---------+---------+----------------------+--------+-----------+------------+------------+
| fpgan03f08 | 10      | XFL1RRBE2CTN | b4:00.0  | 8       | USB-UART-XFL1RRBE2CTN | Switch | fpgan03f07 | onic180s0f0 | 10.0.8.1/24 |
+------------+---------+-------------+---------+---------+----------------------+--------+-----------+------------+------------+
Last login: Thu Jun  8 19:24:56 2023 from 10.2.1.211
[bsc00497@fpgan03 ~]$
```

Figure **52**. FPGA card interconnection mappings.

Further this mapping is used in automation scripts for FPGA programming, interactions with FPGA through PCIe, commands in UART terminal, Ethernet IP assignments.

In addition, an experimental python script (*env.py*) provides several functions to help on the operation and testing of the cluster. It can be executed with an interactive python console (python3 -i env.py). We list the main functions of the script below.

The **`fpga_dashboard`** function lists the details of all the FPGA accelerators connected to the node. It outputs details of the used PCIe slots, the physical and logical USB connections, the kernel drivers used to interact with the boards, the *tty* interface to interact with the accelerators' UART, the FPGA serial numbers, and details about the Ethernet-over-PCIe interfaces. Below you can see an example output of the function.

```
>>>fpga_dashboard()

| FPGA |  PCI   | Board | kernel Driver   | Phy. USB | Log. USB | tty       | Serial        | qx | onic              |
+------+--------+-------+-----------------+----------+----------+-----------+---------------+----+-------------------+
|    0 | 34:00 | u55c  | qdma-pf         | 1-6.5    | 1-10     | ttyUSB22 | XFL1VYH4S0UR | 0  | onic52s0f0 [··]   |
|    1 | 33:00 | u55c  | qdma-pf         | 1-6.6    | 1-11     | ttyUSB26 | XFL1DF0P10SS | 0  | onic51s0f0 [··]   |
|    2 | 19:00 | u55c  | qdma-pf         | 1-6.7    | 1-12     | ttyUSB30 | XFL1KEQBL4IM | 0  | onic25s0f0 [··]   |
|    3 | 1A:00 | u55c  | qdma-pf         | 1-6.4    | 1-8      | ttyUSB14 | XFL105L3VVVU | 0  | onic26s0f0 [··]   |
|    4 | CD:00 | u55c  | qdma-pf         | 1-6.3    | 1-6      | ttyUSB6  | XFL1BZIEUP0P | 0  | onic205s0f0 [··]  |
|    5 | CC:00 | u55c  | qdma-pf         | 1-6.2    | 1-5      | ttyUSB2  | XFL1L43EGBAE | 0  | onic204s0f0 [··]  |
|    6 | B3:00 | u55c  | qdma-pf         | 1-6.1.4  | 1-9      | ttyUSB18 | XFL1QTI2Z0CV | 2  | onic179s0f0 [UP]  |
|    7 | B4:00 | u55c  | qdma-pf         | 1-6.1.3  | 1-7      | ttyUSB10 | XFL1RIVW202O | 2  | onic180s0f0 [UP]  |
```

Some of the information listed is statically written in the script and it is obtained in a configuration step that requires scanning the system to collect all the details. To assist this manual process, we also provide a function **`scan_new_node`** that scans and collects relevant information of the PCIe/USB/UART/and kernel information. Below you can see an example output of the function.

```
>>> scan_new_node()
Checking FPGA serials
Hostname: fpgan01
FPGA serials
['XFL1KEQBL4IM',  'XFL1DF0P10SS',  'XFL1VYH4S0UR',  'XFL105L3VVVU',  'XFL1BZIEUP0P',  'XFL1L43EGBAE',  'XFL1QTI2Z0CV',
'XFL1RIVW202O']
Add the following line to fpga_serial variable:
'fpgan01':['XFL1KEQBL4IM',   'XFL1DF0P10SS',   'XFL1VYH4S0UR',   'XFL105L3VVVU',   'XFL1BZIEUP0P',   'XFL1L43EGBAE',
'XFL1QTI2Z0CV',                                                                                          'XFL1RIVW202O']
Checking PCI slots
...by now PCI slots are hardcoded
Checking                                          USB                                                cables
{(1, 11): '1-6.6', (1, 8): '1-6.4', (1, 1): 'usb1', (1, 5): '1-6.2', (1, 9): '1-6.1.4', (1, 2): '1-1', (1, 12): '1-6.7',
(1,  10):  '1-6.5',  (2,  1):  'usb2',  (1,  6):  '1-6.3',  (1,  3):  '1-6',  (1,  4):  '1-6.1',  (1,  7):  '1-6.1.3'}
Add the following line to usb_ports
'fpgan01':['1-6.7', '1-6.6', '1-6.5', '1-6.4', '1-6.3', '1-6.2', '1-6.1.4', '1-6.1.3']
```

After identifying the hardware, the user typically executes a sequence of operations to bring up a running SDV system in any of the FPGA accelerators. First, a bitstream with the hardware design must be programmed into the FPGA. This can be done with the **`vivado_download`** function.

Once the SDV is configured in the FPGA accelerator, the user needs to set up the QDMA channels to be able to communicate with the SDV. This is done by the **`create_qdma_queues`** function.

Depending on the operating system to run on the SDV (either Buildroot or Fedora), different steps are needed. Buildroot is simpler, it only requires placing the binary image that combines OpenSBI, Linux Kernel, and Buildroot distribution in the main memory of the system before issuing a reset of the processor to start the booting process. This can be done with the **`boot_buildroot`** function. On the other hand, Fedora is slightly more complex as it requires using an additional persistent memory file-system that is placed into an reserved area of the HBM memory. The file-system can be downloaded with the **`download_fedora`** function. Then, the download of the image containing the OpenSBI and the Linux kernel, and a reset to the processor to start the booting process can be performed with the **`boot_fedora`** function.

Once the SDV processor is started it is interesting to analyze the boot log messages that are outputted through the UART interface. A serial terminal connection with the SDV can be established with the **`picocom`** function, which calls the `picocom` Linux application.

Some additional functions are provided to help with the configuration of the SDV parameters. For instance, the **`init_eop_by_uart`** function is used to configure the details (MAC and IP address) of the Ethernet-over-PCI device of the SDV. Similarly, the **`init_qsfp_by_uart`** function is used to configure the details (MAC and IP address and routing) of the Gigabit QSFP Ethernet interface.

5.4. Use Cases

5.4.1. b8c SpMV accelerator

Sparse-matrix dense-vector multiplication (SpMV), computing $y = A \times x$ where y and x are vectors and A is a sparse matrix, is a key kernel in many scientific applications. However, when using the de-facto sparse matrix representation (CSR), and for reasonably large problems, the random access pattern on x accesses penalizes performance due to the high number of cache misses generated by this sparse behavior. Under these circumstances, FPGAs and their ability to generate ad-hoc memory hierarchies for specific problem types arise as an interesting alternative to accelerate SpMV computations.

This section describes a real use-case of the MEEP FPGA-Cluster in order to achieve that goal. The starting point is a single FPGA design including multiple hardware kernels that implements an fpga-optimized version of SpMV. These kernels have been synthesized using AIT and, at run-time, are managed by OmpSs@FPGA.

This SpMV implementation, developed under the MEEP project and named b8c (block-8-compress), is specially tailored for FPGAs. It makes use of different techniques to implement parallel memory access in all the memory data involved in the computation: *x, y* and *A*.

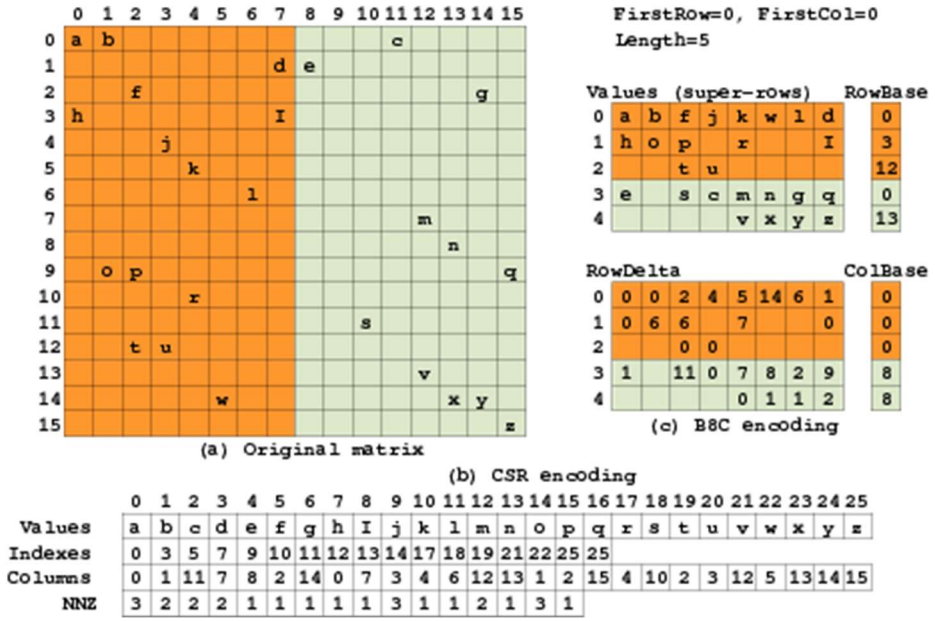


Figure 53. Example of a matrix representation in both CSR and *b8c*.

To be able to do so, b8c transforms the original matrix and replaces its CSR representation by its own, b8c representation. This b8c representation relies on the "compression" of the original matrix in order to generate a more dense representation. The "compression" is performed on a block basis: the original matrix is divided in blocks (or tiles) of $N \times M$ (rows/columns) elements, where N and M are determined by the capacity of on-chip memory assigned to each kernel instance to store x and y segments. Each block is further subdivided into 8-column slices. Then, each slice is processed to "compress" it. The compression algorithm tries to merge as many slice rows as possible, creating a new structure named *super-row*. A *super-row*, then, is a set of 8 contiguous elements in the same slice. There are three conditions to be able to merge a slice row into an already generated *super-row*:

1. They must be at a distance less or equal than a parameter "d" that is determined by the number of bits configured to be able to store this information.
2. They must "belong" to y positions that do not clash modulo "a", being "a", again, a configuration parameter that indicates how many row-accumulations can be performed in parallel. The only exception to this is that both (the slice row and the super-row update the same y position).
3. They must not have any common x index.

For a slice row, it can be merged with any already generated *super-row* given that the three previous conditions are met. If they are not met, the slice row constitutes its own super-row, which will be available to try to merge additional slice rows into it.

Figure 53 depicts a simple example of a matrix transformed from its CSR representation into a b8c representation. As can be observed, the b8c structure includes additional meta-data needed to identify the original row/column of each element in the new super-row structure.

Once transformed into this b8c format, the matrix can be processed using the b8c accelerators leveraging all of the representation characteristics that, fruit of a co-design process, are tailored for its own hardware architecture. Figure 54 shows a simplified scheme of a b8c hardware accelerator. Given a (sub)matrix (or block) represented in b8c format, this accelerator can process in the following sequence:

Preload x (source vector) values into local memory (on-chip memory)

Preload y (destination vector) values into local memory

Stream-process A (matrix) values (in its super-*row* form) directly from off-chip memory (DDR/HBM)

i) For each *super-row*:
   1) Read the corresponding x values
   2) Perform partial products ($A \times x$)
   3) Group/accumulate partial products that update the same row
   4) Update y with the result of the previous step

ii) Update y values on off-chip memory

Steps a, b and c can be skipped if the same accelerator already has the information needed (a,b) or will use the same information to process the next block (c).

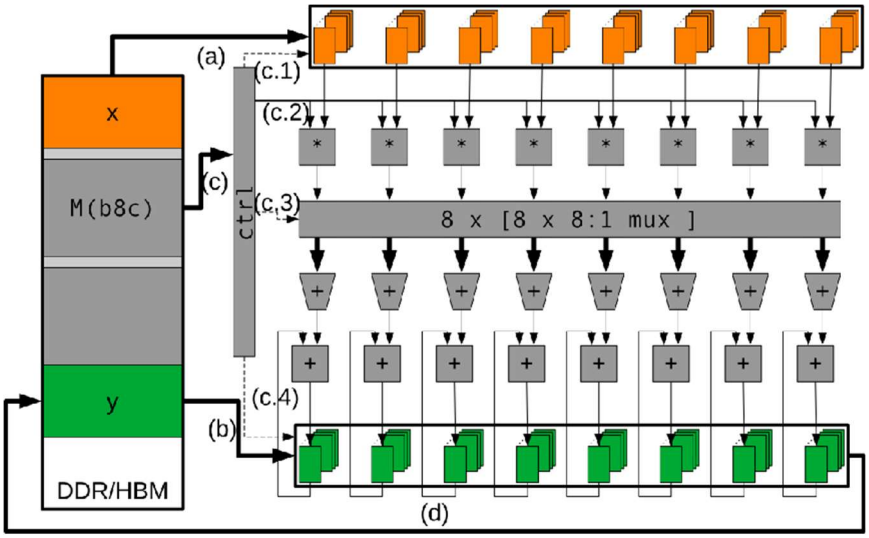


Figure **54**. *b8c* accelerator hardware scheme.

The on-chip memory that stores x and y in the accelerator is partitioned in a cyclic fashion using the same parameters as the b8c encoding of the matrix. In this way, up to 8 x values can be read in parallel without collision and up to "a" (usually 4 or 8) values of y can be read/updated in parallel. The accelerator is fully pipelined and thus, it is able to process one super-row per cycle without stalling. This translates to a peak performance, according to SpMV arithmetic, of 16 FLOPS per cycle.

The FPGA design of this accelerator, including 16 b8c SpMV kernels, can be deployed in the MEEP cluster leveraging the mechanisms described previously. In single-accelerator mode, a single FPGA is used to process any given matrix. In multiple-accelerator mode, a set of FPGAs can be allocated and loaded with the same bitstream containing the SpMV b8c kernels. This set of FPGAs, then, can be used to process each one, a sub-matrix of the original matrix, allowing multi-FPGA b8c SpMV computations.

5.4.2. FPGA and ACME designs: Proof of concept

In addition to the bring-up tests, where different MEEP designs were used (Table **17**), this section describes how MEEP system has been used to put together all MEEP project developments from WP4, WP5 and WP6.

To demonstrate the achievements of the project different tests have been executed, using as a baseline the execution environment shown in Figure **55**.

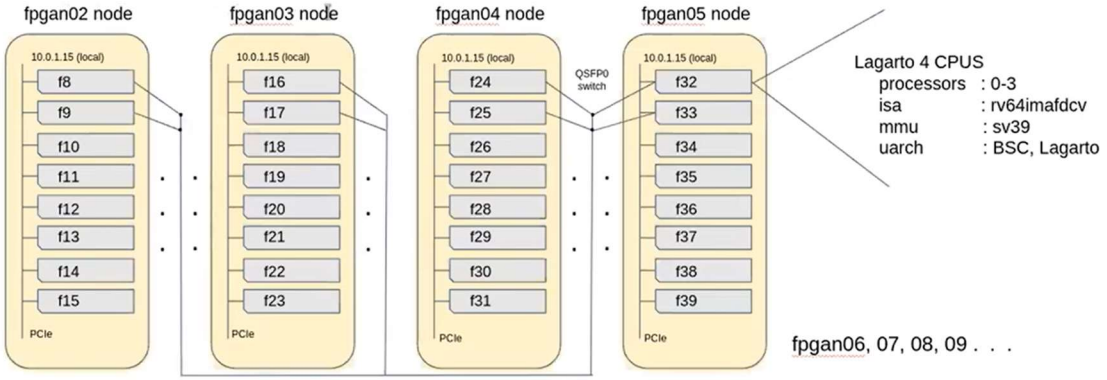


Figure **55** Execution environment in the MEEP system

All the tests have been executed programming all the FPGAs with one of the released bitstreams: ACME EA 4H2V. This bitstream includes the ACME accelerator (Lagarto Hun+VPU with 2 lanes) design and the FPGA Shell with HBM, PCIe and Ethernet. The main characteristics of the MEEP system are included in Table 17.

Table 17. Execution environment setup in MEEP system

Item	Details
Execution environment	8 nodes (fpganode02: fpganode09)
MEEP system nodes	Login node: fpgalogin1 Compute nodes: fpganode02: fpganode09
Total number of FPGAs	8 FPGAs/node 64 FPGAs in total (fpganode02/f8 to fpganode09/f71)
MAC Range using QSFP0	10.5.1.159 : 10.5.1.222 (*fpganode02 : fpganode09*)

Execution examples:

- Ping to all nodes and FPGAs. Connected to fpganode03 and pinging to all FPGAs (from fpganode02/f8 to fpganode09/f71)

- Get the Ethernet/MAC addresses IPs for several nodes. User connected to the FPGA login node (fpgalogin1) and get the MAC addresses of all the FPGAs from node fpganode02 to fpganode09.
- Run an MPI Hello world on 64 nodes.
- Booting Fedora on ACME EA 4H2V.

6. Conclusions

WP6 was developed with three goals:

- Preparing a platform definition document, including the chosen emulation system, HW specifications for the FPGA platform, RTL, FPGA Shell and other required IPs.
- Implementing the FPGA Shell, including PCIe interface drivers and IP, HBM interface, and other IP to enable inter-FPGA communication.
- Implementing FPGA core elements (scalar core, vector core, caches, ring) and mapping, placing and routing toon the targeted FPGA platform, yielding the final acceleration emulator prototype.

The first goal was fully covered in deliverable *D6.1 Emulation platform specification*. The remaining two goals have been developed in parallel. On the FPGA Shell, the project delivers a flexible, scalable, configurable, and extensible shell, including communication IPs such as HBM, PCIe, and Ethernet. Moreover, a set of tools, including the FPGA flow, have been developed with the aim of simplifying the exploitation and utilization of FPGAs by hardware and software developers.

The MEEP FPGA Shell, and the toolbox have been used to achieve the third of the proposed goals. Different configurations of the ACME accelerator have been implemented and run on both MEEP infrastructures, MEEP servers (*Phase 1*) and MEEP system (*Phase 2*).

A very interesting path has been created among the activities developed in different work-packages, where similar configurations of ACME accelerator have been tested running AXPY benchmark. On one hand, Coyote simulator executed a version of ACME design based on ACME specification (deliverable D4.1). RTL run simulations with the latest ACME design, a simplified approach to the envisioned ACME accelerator. The software team used several configurations that design as SDV, and finally FPGA synthesized and implemented the final released version of the RTL ACME together with the FPGA Shell (ACME EA). A collection of experiments is summarized in Table 18.

Table 18. Collection of experiments related to ACME accelerator across WP4, WP5, WP6

WP /Task	ACME design	ACME details	Experiment	Results
WP4. Coyote	4V-1C-1M	Configuration based on ACME specs	**Benchmark**: AXPY **Exec. modes**: epi-mode, acme-classic, acme-mode	D4.4 Section 3.4.2
WP4. RTL	ACME 1H4G1M	RTL design: Proof of concept (simplified version of ACME specs)	**Benchmarks**: AXPY MATMUL Bolt65 **Exec. modes:** classic-mode, acme-mode	D4.3 Section 2.3.3
WP6. FPGA	ACME EA 4H2V		CICD and synthesis	D6.4
WP5. SW	ACME EA 4H2V		**Benchmarks:** AXPY GEMM SPMV SOMIER **Exec. mode**: classic-mode	D5.4 Section 7

6.1. Key Performance Indicators (KPIs)

Table **19** contains the Key Performance Indicators (KPI) set out for the MEEP infrastructure in Deliverable D4.1. The KPIs cover WP6 components, and functionalities developments, but also support for the RISC-V community, including the ACME accelerator.

Table 19. General KPIs of the MEEP infrastructure

Goal	KPI	Objective

FPGA-based Platform Infrastructure	FPGA-based infrastructure with many FPGAs per server and 100GbEthernet communication.	Enabling remote connection to the infrastructure, and capability for using its resources according to the user needs:
		- Accelerator topology,
		- Resources allocation:
		  - one node & one FPGA
		  - one node multiple FPGAs
		  - multiple nodes and multiple FPGAs/node

	Support to emulate a **single core** accelerator design in **one FPGA** using the MEEP Shell for host communication.	. Bitstream generation using the FPGA Shell + one instance of the VAS Tile core.
		. Accelerate the execution of one MEEP targeted HPC application in one FPGA.
	Support to emulate a **many-core** accelerator in **one FPGA** using the MEEP Shell for the host communication.	. Bitstream generation using the FPGA Shell + one instance of the VAS Tile.
		. Accelerate the execution of one MEEP targeted HPC application in one FPGA.
	Support to emulate a **many-core** accelerator in **multiple FPGAs, physically connected to the same node**, using the MEEP shell for the host and FPGA2FPGA communication.	. Bitstream generation using the FPGA Shell + one instance of the VAS Tile core.
		. Accelerate the execution of one MEEP targeted HPC application in, at least, two FPGAs
	Support to emulate a **many-core accelerator in multiple FPGAs**, physically connected to **different nodes**, using the MEEP shell for the host and FPGA2FPGA communication.	. Bitstream generation using the FPGA Shell + one instance of the VAS Tile core.
		. Accelerate the execution of one MEEP targeted HPC application in, at least, two servers with up to two FPGAs per node.
FPGA Emulator	Support to integrate different designs in MEEP infrastructure using the MEEP Shell.	4 (ACME EA, Ethernet design, EPI/SGA2, OmSs@FPGA)
RISC-V Ecosystem	Extend the RISC-V vector extension	Define new RISC-V ISA instructions, compliant with the standard, for performing vector operations with the specific accelerators (SAs)
	Contribute to the Open Hardware ecosystem with new open source MEEP IPs	3 MEEP FPGA Shell, Coyote, Aurora, 100GbE, ACME*, Lagarto Hun*

Goal	KPI	Objective
Computing IPs (accelerators)	Support for different specialized accelerators as part of the VAS Tile core (4.3.2.2.2)	Demonstrate SAs functionality by reusing the same SA-Shell and using custom instructions: - Executing a NN app on SA-NN. - Executing Bolt65 app on SA-HEVC.
	VPU – increase of peak performance of the accelerator when executing memory-bound workloads using the Memory Tile (ACME_mode) vs without using it (EPI_mode) (4.3.2.2.1)	10%
MEEP Shell	Number of Communication IPs integrated in FPGA-Shell	5 (HBM, PCIe, Aurora, Ethernet, UART)
	Increase of peak performance when using smart reordering memory access of HBM	10%
	Increase of peak performance when using parallel accesses to the HBM with multiple MCs	2% per extra MC This performance will depend on the application and the data mapping in memory
	100 Gb Ethernet (HW/SW development)	Enabling OS control ever the 100Gb Ethernet
NoC	Aggregate bisection BW	
	Max Latency	
	Max Queue Size	
EPI Extension	VPU extension	Less number of elements/register in the VRF Dual port VRF 16 lanes Inter-lane communication variation Vector extension upgrade to v0.10 (at least 10 instructions) Support for processing short and long vectors
HPC applications	Execution of well-known HPC representative Microkernels: Stream, FFT, Saxpy, SpmV, DGEM and matmul	6
*IPs not public yet.		

Most of the KPIs have been achieved totally or partially, as the color code reflects in Table **18**. Green color means the goal has been completely achieved, Orange that it has been partially achieved and Red that there is not enough information. More in detail:

- FPGA-based Platform Infrastructure. Performance improvements are not fully achieved, since even though the MEEP FPGA Shell has the potential to execute host-accelerator communication through Ethernet (over PCIe and/or QSFP port) at peak performance, ACME design has several limitations. This is the reason why we have

preferred to color orange "*Accelerate the execution of one MEEP targeted HPC application in one FPGA*".

- FPGA Emulator. Four different designs have been emulated on both MEEP infrastructures, MEEP servers (Phase 1) and MEEP system (Phase 2): ACME accelerator, MEEP custom designs for testing the communication IPs, EPI/SGA2 SDV, and designs from OmSs@FPGA and AIT flow.
- RISC-V Ecosystem. As reported in Deliverable D5.3, RVV has been extended to support custom SA instructions. These have been tested through RTL simulations, reported in deliverable D4.3. Moreover, several IPs have been developed during the MEEP project, and are available in GitHub. Others, like ACME, cannot be open sourced yet because of their dependencies with other modules. That is the case of Lagarto Hun and the VPU. Although MEEP project has extended their features, the baseline IPs are still in the process of being open-source.
- Computing IPs. Regarding SA-Shell, the same wrapper has been used for both SAs, SA-HEVC, and SA-NN. Although they couldn't be tested on the FPGA, their functionality was tested under RTL simulation, using the Memory Tile behavioral model. About the performance increase of the VPU, this goal was not achieved. The reason is due to the inherent limitations of the developed ACME: low performance scalar core, small bus width for the NoC bus, and no-HPC memory hierarchy. However, the ACME design is a perfect framework for continuing working on these well-identified limited points to become an HPC-accelerator.
- MEEP Shell. Regarding the HBM and MCs, the experiments run with the Memory Sandbox tool demonstrate that these numbers are reachable. However, we couldn't run experiments with realistic designs, in this case ACME accelerator to validate these numbers. That is the reason why we prefer using orange color, instead of green.
- NoC. Not an extensive work has been done on the NoC development, due to the fact that OpenPiton project offers a stable framework. We preferred to start using that with the aim of making progress on the ACME accelerator and ensuring correctness, before starting to do modifications on it.
- EPI Extension. After using EPI.VPU v1.0 as a baseline for MEEP project, several modifications have been developed. However, it was not possible to upgrade to RVV v1.0 or improving the inter-lane communication module.
- HPC Applications. Deliverable D5.4 executes several benchmarks on released versions of the ACME EA, however there was no time to run those benchmarks in acme-mode. That is the reason why we prefer to mark the goal as partially achieved.

6.2. MEEP FPGA contributions to other projects

Apart from the ones mentioned in the previous deliverable, we have established and keep collaborating in multiple external groups in different directions, as listed below:

- **Princeton University (USA)**:

Since ACME is based on OpenPiton (developed in the Princeton Parallel Group), we have applied the following contributions to the original OpenPiton framework (https://github.com/PrincetonUniversity/openpiton/tree/openpiton-dev):

· Support of Alveo family Xilinx FPGA boards (protosyn switch –board alveou280), including PCIe interface in QDMA mode.

- Extension of Ethernet support to Ultrascale+ 100Gb CMAC hard-macro with DMA and Alveo board level QSFP connectors (protosyn switch –eth), including updates in Device Tree script for support of Linux driver.
- Support of HBM as an option for system memory (protosyn switch –hbm).
- Support of multiple Memory Controllers for HBM usage (protosyn switch –multimc).
- Support of non-cached access to system memory (protosyn switch –ncmem, required to work with DMA).
- Modifications of NoC-AXI4 bridge (fix of bug causing a stuck working with HBM; Big-End/Little-End data conversion, required to work with PCIe/DMA; parameterizable AXI data width conversion; option to reorder HBM replies according to conveyed in HBM request AXI ID mapped from core X/Y coordinate or MSHR ID NoC fields).
- Support of alternate 2D-mesh NoC (protosyn switch –pronoc).
- Support of Lagarto scalar core (protosyn switch –core lagarto).
- Support of building under FPGA Shell (protosyn switch –meep)

- **University of Paderborn (Germany):**

The collaboration with the University of Paderborn is ongoing and consists of several topics: the FPGA cluster, Aurora designs, and the FPGA Shell.

- FPGA Cluster. Technical experiences have been exchanged with them, since Uni Paderborn has an FPGA-based cluster preparing for HPC simulations.
- Aurora Designs. The Paderborn group also showed interest in the Aurora designs developed by the FPGA team. They want to implement Aurora in their cluster. We have been sharing information about the Aurora designs we have developed and the specs these designs have. We have given access to our resources to test the projects.
- FPGA Shell. It was also discussed as a final element that includes all the main IPs developed by the team. They are using it to test the features, e.g., Aurora DMA provides the Cluster.

- **Technical University of Crete (Greece):**

We are technically supporting a Ph.D. candidate at the Technical University of Crete to use our FPGA Shell:

- The aim of this collaboration is to use our FPGA Shell to map an Arianne processor on the U55C (using our flow for generating bitstream and booting Linux) while equipping it with an IDS (Intrusion Detection System) accelerator. We update and synchronize all the main repos to be accessible on GitHub. https://github.com/MEEPproject

- **Polytechnique Montréal (Canada):**

We are technically supporting a student from Polytechnique Montréal.

- The aim of this collaboration is to use the FPGA Shell for experimenting with our OpenPiton repo using Ariane. Her research project is about enabling the vectorized version of Ariane, Ara with OpenPiton.

Source code repositories

Lagarto-sdk: https://gitlab.bsc.es/meep/meep-os/lagarto-openpiton-sdk

FPGA Shell

We follow the git submodule strategy to store our code, which means that each module has its own repository.

FPGA Shell: https://gitlab.bsc.es/meep/FPGA_implementations/AlveoU280/fpga_shell (Stable branch: production)

IPs:

Ethernet 100Gb:
https://gitlab.bsc.es/meep/FPGA_implementations/AlveoU280/fpga-tools

Ethernet 10Gb:
https://gitlab.bsc.es/meep/FPGA_implementations/AlveoU280/10gb_ethernet

Aurora:
https://gitlab.bsc.es/meep/FPGA_implementations/AlveoU280/aurora_user_interface

https://gitlab.bsc.es/meep/FPGA_implementations/AlveoU280/aurora_raw

AXI-BROM:
https://gitlab.bsc.es/meep/FPGA_implementations/AlveoU280/axi_brom

UART: https://gitlab.bsc.es/meep/FPGA_implementations/AlveoU280/ariane_uart

FPGA tools

FPGA tools : https://gitlab.bsc.es/meep/FPGA_implementations/AlveoU280/fpga-tools

ACME emulator accelerator (ACME_EA)

MEEP_openpiton:
https://gitlab.bsc.es/meep/FPGA_implementations/AlveoU280/meep_openpiton

Appendix A

Baseline throughput and address mapping policies impact

The presented experiments were performed on a Xilinx Alveo U280 since we are interested in HPC applications and is the only Xilinx HPC-oriented board that has DDR and HBM, ensuring fair comparisons [7], [49]. The Alveo U55C, which is the next and latest version, has a common topology to the U280.

In these experiments, the throughput of a pseudo-channel is not multiplied to obtain the overall throughput. In fact, in each case every pseudo-channel or bank is accessed simultaneously, and their throughput is added to obtain the overall performance. From experiments in Figure8 and Figure 9 we formulate the following conjectures:

- For HBM, by obtaining the same result as performing a multiplication, we demonstrate that HBM micro-switches are fully implemented 4x4 crossbars and that the memory controller handles its two pseudo-channels without losing performance.
- The best throughput for the sequential configuration is always achieved by the default policy ensuring that both memories are configured for this type of access.
- Read and write transactions follow the same trend regardless of the policy, but read performance is slightly better than write performance.
- The best achievable throughput of DDR4 is 6.18% lower than the bandwidth, while that of HBM is only 0.01% lower.
- The worst achievable performance of DDR4 is 92.02% lower than bandwidth, while that of HBM is only 56.39%.
- At maximum performance, each bank of DDR4 delivers up to 19.2 GB/s due to its wider port width. HBM, on the other hand, is able to deliver 14.4GB/s because, although the port is half the width of DDR, it operates at higher frequencies.
- HBM needs to enable 4 pseudo channels in parallel to outperform DDR4 and, by adding more channels, this difference increases.

Micro-switches cross domain

Most modern computer applications including HPC occupy large memory regions. Therefore, we were interested in how the HBM behaves when accessing different memory regions across pseudo-channels and micro-switches.

Figure 10 indicates the performance of a single channel as the baseline. Figure 11.A shows the results of accessing the different pseudo-channels of the HBM emulating a single-threaded processing element connected to the AXI0 port. These experiments are performed with a sequential access pattern (RST), a burst size of 16 which is the maximum for AXI3 and RBC (true) as address mapping policy which offers the best performance for this type of access pattern according to our experiments. Two main conclusions can be drawn from these experiments:

- The first is that the performance of the pseudo-channels on the same micro-switch is the same regardless of the AXI port accessing them.
- The second, and perhaps more important, is that if the processing element leaves the micro-switch to which it is connected, performance is reduced by about 50%. It is therefore important that accelerator or processor accesses when using HBM are kept

at least within the same micro-switch. If more memory is needed, the data should be split between the micro-switches and accessed in parallel from another AXI port. Otherwise, no matter whether the access comes from an adjacent or the farthest micro-switch, performance will be severely affected equally.

Burst impact on performance

For HPC is not enough to analyze sequential accesses, therefore we intend to also analyze processing threads performing sparse accesses. We emulate this by performing pseudo-random accesses with our AXI Traffic Generators, as this data is not in consecutive addresses the burst concept does not apply directly. To have a fair sequential baseline comparison we performed sequential accesses with a burst size of 1 element = 1 beat = 256 bits = 32 Bytes.

By changing the burst between 2 and 8 no difference has been noticed in the performance but removing at all causes a throughput decrease of around 30% when the AXI Port is accessing its vertical micro-switch. Is interesting to note that, as previously, the rest of the micro-switches in the same stack behave alike and with an overall lower throughput but they only get affected by around 15% compared to the experiments with burst size. It is even more significant that, for the first time, we notice a difference between the two stacks in throughput because the pseudo-channels of the further stack (only in the write transactions) get affected by 60% compared to the experiments with burst size (Figure 11.B).

Randomizing inside a pseudo-channel

Our first approach when emulating processors or accelerators with sparse patterns was to maintain the address bits related to the pseudo channel and just randomize the application address bits which in the case of HBM are [27:5]. Our Memory Sandbox eases doing this by just changing one parameter (*Rand_Whole_Addr*). With this configuration we analyze the impact produced by pseudo-random accesses when opening and closing bank groups and banks when accessing different columns and rows of the pseudo-channel.

In this experiment (Figure 11.C) we notice for the first time almost no differences between the micro-switch to which is connected the AXI port and the rest of the AXI ports in different micro-switches within the same stack. In the case of the read transaction every micro-switch behaves almost the same. On the other hand, in the write transactions in the first stack all the AXI ports get affected by around 44% but the vertically attached one reduces its throughput by around 65%. The ones in further stack on the other hand behave like the previous experiment.

Randomizing across different pseudo-channels

In this section we wanted to emulate processors or accelerators with sparse patterns by randomizing the address inside the pseudo-channel as previous, but also randomizing which pseudo-channels are accessed. This is done by setting the parameters *Rand_Whole_Addr* and *Rand_PSCH*.

With this configuration we analyze the huge impact of the pseudo-channel change in consecutive pseudo-random accesses where everything changes in the address. The idea was to emulate a single thread processor which performs accesses which start in its vertical micro-switch and in each new experiment add micros-switches to randomize e.g., the first experiment

randomizes among 4 pseudo-channels, the second one among 8 and up to the 32 pseudo-channels.

The performance in this case (Figure 11.D) is the worst measured one. It decreases to 0.61% of the best-case scenario measured for the write transactions and 0.17% for the read ones. We are aware that some of these experiments are quite extensive and that there is only slight probability that, for example, an accelerator always accesses to different pseudo-channels. Nevertheless, the goal is to generate baselines for future comparison and for other developers to have a starting point to compare with. In fact, our Memory Sandbox is provided as part of the FPGA Shell with the purpose of providing a software development and experimentation platform to enable software readiness for new hardware as one of the main MEEP project goals.

Simultaneous accesses to the same pseudo-channel

In all the previous experiments the emulated architecture is based on a single thread processing element. Nevertheless, common accelerator multi-core architectures are heterogeneous ones where several processing elements are accessing the memory in parallel. All the information presented up to now applies to any of the cores or accelerators in this heterogeneous system. Nevertheless, we still must explore what happens when several of these processing elements target the same memory region. With this purpose a set of experiments were designed with multiple AXI Traffic Generators emulating the different processing elements. The impact on throughput is analyzed while adding more AXI Traffic Generators always accessing the same pseudo-channel 0. This was performed with sequential access to the same address to increase the probabilities of creating the desired collisions that we wanted in this experiment to measure their impact.

In Figure 12 is shown the huge impact that simultaneous access can have on the HBM. It is important to highlight that once again all the processing elements connected to the same micro-switch have similar behavior. Nonetheless, when going from one processing element to two, the throughput is reduced by around 50% in each of those two (Figure 12.B). Then when going from two to three, all of them get a throughput reduction of 32% again (Figure 12.C) and from 3 to 4, the 4 of them get a throughput reduction of 25% once again (Figure 12.D). According to our measurements, the aggregate throughput of each experiment is the same as that of only one processing element.

The 3 rightmost sets of bars are for 8 (Figure 12.E), 16 (Figure 12.F) and 32 (Figure 12.G) processing elements in different micro-switches. Across the 3 experiments it can be observed that regardless of the number of processing elements those in the first micro-switch have a similar throughput, and the same happens for the rest. This is explained by understanding that the micro-switch has 4 inputs for the AXI Ports that have priority to the one connected to the other micro-switch. According to our measurements, by adding more processing elements the throughput decreases every time and, as before, the aggregate throughput of each experiment is the same as that of only one processing element.

The powerful capabilities of FPGAs to address challenging HPC workloads with a Heterogeneous computing paradigm are currently underexplored because leveraging these devices is quite burdensome. When FPGAs are used to address HPC applications, but their resources are not properly configured, the resulting implementations can underperform quite significantly. This is especially important regarding memories, as FPGAs do not have a

preconfigured and tested cache hierarchy like microprocessors or GPUs. HBM appears as a solution being integrated into FPGAs to face the memory wall issue and large companies are already committed to its wide use.

Appendix I

Table A shows the MEEP VPU characteristics used in each of the FPGA releases.

MEEP VPU features		
	MEEP VPU v1.1 (FPGA 1st release)	**MEEP VPU v2.2.1 (FPGA 2nd release)**
Number of vector lanes	Up to 16 (grouped in vector-lane pairs for smaller configurations (2-4-8))	
Maximum vector length	128 elements x 64 bits	*ACME-classic* mode 128 elements x 64 bits *ACME* mode 512 elements x 64 bits
Number of FMAs	1 Fused Multiply Accumulate (FMA) unit per lane (2 DP FLOP/cycle)	
FP operation support	Support for 64- and 32-bit FP operation	
Integer operation support	Support for 64-, 32-, 16-, and 8-bit integer operations, signed and unsigned	
Vector Register File (VRF)	VRF number of banks: 5. N of physical vector registers: 40. Single-Port limited access.	VRF number of banks: 4. N of physical vector registers: 32 Dual-Port access. Redesign of lane control logic to leverage VRF concurrent read/write accesses.
RISC-V vector version	RVV v0.7.1	
Core's Interface	OVI 1.0 [OVI]	
Memory's interface	OVI 1.0	OVI 1.0 & Direct Memory Access
Direct access to L2	Through OVI	Through OVI & Long Vector Register File
Execution modes support	*ACME-classic* mode vector lanes config: 2,4,8,16	*ACME-classic* mode vector lanes config: 2,4,8,16 *ACME* mode vector lanes config: 2, 16

Table A. MEEP VPU characteristics for each of the FPGA releases

Appendix II

Test resources:

1. 20230407_84559:

   acme_ea_1h16v_u280

Name	CLB LUTs (1303680)	CLB Registers (2607360)	CARRY8 (162960)	F7 Muxes (651840)	F8 Muxes (325920)	CLB (162960)	LUT as Logic (1303680)	LUT as Memory (600960)	Block RAM Tile (2016)	URAM (960)	DSPs (9024)	Bonded IOB (624)	HPIOB_M (288)	HPIOB_S (288)
meep_shell_inst (meep_shell)	92635	121792	1815	3016	599	24771	84069	8566	113	7	0	0	0	0
openpiton_wrapper_inst (openpiton_wrapper)	874561	736351	18636	78209	16089	141445	864218	10343	184.5	2	201	0	0	0
ACME_OP (system)	874561	736351	18636	78209	16089	141445	864218	10343	184.5	2	201	0	0	0
chip (chip)	858095	724951	18592	77977	16089	139248	849158	8937	169.5	2	201	0	0	0
rst_sync (synchronizer)	2	0	0	0	0	1	1	1	0	0	0	0	0	0
tile0 (tile)	858093	724951	18592	77977	16089	139248	849157	8936	169.5	2	201	0	0	0
cgni_blk1 (credit_to_valrdy_176)	50	8	0	0	0	11	10	40	0	0	0	0	0	0
cgni_blk2 (credit_to_valrdy_177)	47	8	0	0	0	10	7	40	0	0	0	0	0	0
cgni_blk3 (credit_to_valrdy_178)	52	8	0	0	0	12	12	40	0	0	0	0	0	0
cgno_blk1 (valrdy_to_credit)	10	8	0	0	0	3	10	0	0	0	0	0	0	0
cgno_blk2 (valrdy_to_credit_179)	10	8	0	0	0	3	10	0	0	0	0	0	0	0
cgno_blk3 (valrdy_to_credit_180)	10	8	0	0	0	5	10	0	0	0	0	0	0	0
g_lagarto_m20_core.core (vas_tile_co	835783	710480	18441	76368	15593	135472	827335	8448	160	0	201	0	0	0
genblk1[0].i_irq_sync (synchronizer	0	2	0	0	0	2	0	0	0	0	0	0	0	0
genblk1[1].i_irq_sync (synchronizer	0	2	0	0	0	2	0	0	0	0	0	0	0	0
i_csr_regfile (csr_regfile)	7984	5172	1096	361	38	2104	7984	0	0	0	0	0	0	0
i_ipi_sync (synchronizer_297)	0	2	0	0	0	2	0	0	0	0	0	0	0	0
i_sync (synchronizer_298)	2	3	0	0	0	4	2	0	0	0	0	0	0	0
i_timer_sync (synchronizer_299)	0	2	0	0	0	2	0	0	0	0	0	0	0	0
lagarto_m20 (top_drac)	827802	705281	17343	76007	15555	134310	819354	8448	160	0	201	0	0	0
datapath_inst (datapath)	30636	10554	551	1285	520	6803	30636	0	0	0	25	0	0	0
i_cache_subsystem (wt_cache_s	7424	2584	96	1652	609	1705	7424	0	32	0	0	0	0	0
i_mmu (mmu)	1835	2746	5	0	0	583	1835	0	0	0	0	0	0	0
icache_interface_inst (icache_int	17	66	2	0	0	23	17	0	0	0	0	0	0	0
lagarto_dcache_interface_inst (l	10495	142	0	1088	544	3085	10495	0	0	0	0	0	0	0
SP_inst (meep_lvf_top)	18840	320	16	0	0	4485	18840	0	128	0	0	0	0	0
vpu_inst (multi_lane_wrapper)	749401	679269	16671	71734	13786	127245	740953	8448	0	0	176	0	0	0

2. 20230503_86660:

   acme_ea_16h_u280

Name	CLB LUTs (1303680)	CLB Registers (2607360)	CARRY8 (162960)	F7 Muxes (651840)	F8 Muxes (325920)	CLB (162960)	LUT as Logic (1303680)	LUT as Memory (600960)	Block RAM Tile (2016)	URAM (960)	DSPs (9024)	Bonded IOB (624)	HPIOB_M (288)	HPIOB_S (288)
system_top	1055666	737857	28991	67016	20683	162882	1032093	23573	795	39	400	9	4	4
meep_shell_inst (meep_shell)	96292	121849	1815	3016	599	23965	87725	8567	113	7	0	0	0	0
openpiton_wrapper_inst (openpiton_wrapper)	957822	615928	27176	64000	20084	147497	942816	15006	682	32	400	0	0	0
ACME_OP (system)	957822	615928	27176	64000	20084	147497	942816	15006	682	32	400	0	0	0
chip (chip)	939395	603147	27072	63728	20064	145645	925795	13600	664	32	400	0	0	0
rst_sync (synchronizer)	1	2	0	0	0	1	1	0	0	0	0	0	0	0
tile0 (tile)	58400	37732	1692	3983	1254	11859	57595	805	41.5	2	25	0	0	0
tile1 (tile_206)	58662	37692	1692	3983	1254	12392	57817	845	41.5	2	25	0	0	0
tile2 (tile_213)	58720	37693	1692	3983	1254	11240	57875	845	41.5	2	25	0	0	0
tile3 (tile_214)	58016	37679	1692	3983	1254	12401	57291	725	41.5	2	25	0	0	0
tile4 (tile_215)	58659	37693	1692	3983	1254	9791	57814	845	41.5	2	25	0	0	0
tile5 (tile_216)	59378	37708	1692	3983	1254	10352	58413	965	41.5	2	25	0	0	0
tile6 (tile_217)	59330	37708	1692	3983	1254	11264	58365	965	41.5	2	25	0	0	0
tile7 (tile_218)	58697	37694	1692	3983	1254	11821	57852	845	41.5	2	25	0	0	0
tile8 (tile_219)	58688	37693	1692	3983	1254	11846	57843	845	41.5	2	25	0	0	0
tile9 (tile_220)	59358	37709	1692	3983	1254	12542	58393	965	41.5	2	25	0	0	0
tile10 (tile_207)	59245	37708	1692	3983	1254	10412	58280	965	41.5	2	25	0	0	0
tile11 (tile_208)	58729	37694	1692	3983	1254	11753	57884	845	41.5	2	25	0	0	0
tile12 (tile_209)	58159	37677	1692	3983	1254	11570	57434	725	41.5	2	25	0	0	0
tile13 (tile_210)	58658	37692	1692	3983	1254	10468	57813	845	41.5	2	25	0	0	0
tile14 (tile_211)	58637	37694	1692	3983	1254	10865	57792	845	41.5	2	25	0	0	0
tile15 (tile_212)	58078	37679	1692	3983	1254	10757	57353	725	41.5	2	25	0	0	0
chipset (chipset)	18421	12774	104	272	20	3249	17015	1406	18	0	0	0	0	0

3. 20230607_90803:

   acme_ea_16h_u280

Name	CLB LUTs (1303680)	CLB Registers (2607360)	CARRY8 (162960)	F7 Muxes (651840)	F8 Muxes (325920)	CLB (162960)	LUT as Logic (1303680)	LUT as Memory (600960)	Block RAM Tile (2016)	URAM (960)	DSPs (9024)	Bonded IOB (624)	HPIOB_M (288)	HPIOB_S (288)
N system_top	1062918	737729	30015	67016	20683	162725	1039346	23572	795	39	400	9	4	4
meep_shell_inst (meep_shell)	96330	121678	1815	3016	599	22537	87764	8566	113	7	0	0	0	0
openpiton_wrapper_inst (openpiton_wrapper)	965038	615971	28200	64000	20084	147438	950032	15006	682	32	400	0	0	0
ACME_OP (system)	965038	615971	28200	64000	20084	147438	950032	15006	682	32	400	0	0	0
chip (chip)	946643	603192	28096	63728	20064	145838	933043	13600	664	32	400	0	0	0
rst_sync (synchronizer)	1	2	0	0	0	1	1	0	0	0	0	0	0	0
tile0 (tile)	58873	37737	1756	3983	1254	11972	58068	805	41.5	2	25	0	0	0
tile1 (tile_206)	59228	37696	1756	3983	1254	11485	58383	845	41.5	2	25	0	0	0
tile2 (tile_213)	59158	37696	1756	3983	1254	10672	58313	845	41.5	2	25	0	0	0
tile3 (tile_214)	58495	37682	1756	3983	1254	11468	57770	725	41.5	2	25	0	0	0
tile4 (tile_215)	59160	37694	1756	3983	1254	11285	58315	845	41.5	2	25	0	0	0
tile5 (tile_216)	59717	37709	1756	3983	1254	12188	58752	965	41.5	2	25	0	0	0
tile6 (tile_217)	59732	37710	1756	3983	1254	11346	58767	965	41.5	2	25	0	0	0
tile7 (tile_218)	59155	37698	1756	3983	1254	11321	58310	845	41.5	2	25	0	0	0
tile8 (tile_219)	59151	37698	1756	3983	1254	11106	58306	845	41.5	2	25	0	0	0
tile9 (tile_220)	59730	37713	1756	3983	1254	9967	58765	965	41.5	2	25	0	0	0
tile10 (tile_207)	59708	37710	1756	3983	1254	10126	58743	965	41.5	2	25	0	0	0
tile11 (tile_208)	59136	37697	1756	3983	1254	10970	58291	845	41.5	2	25	0	0	0
tile12 (tile_209)	58463	37680	1756	3983	1254	11922	57738	725	41.5	2	25	0	0	0
tile13 (tile_210)	59189	37694	1756	3983	1254	10921	58344	845	41.5	2	25	0	0	0
tile14 (tile_211)	59201	37695	1756	3983	1254	10700	58356	845	41.5	2	25	0	0	0
tile15 (tile_212)	58566	37681	1756	3983	1254	10662	57841	725	41.5	2	25	0	0	0
chipset (chipset)	18390	12772	104	272	20	3419	16984	1406	18	0	0	0	0	0

acme_ea_16h_u55c

Name	CLB LUTs (1303680)	CLB Registers (2607360)	CARRY8 (162960)	F7 Muxes (651840)	F8 Muxes (325920)	CLB (162960)	LUT as Logic (1303680)	LUT as Memory (600960)	Block RAM Tile (2016)	URAM (960)	DSPs (9024)	Bonded IOB (624)	HPIOB_M (288)	HPIOB_S (288)
N system_top	1062955	737572	30019	67016	20683	162744	1039376	23579	795	39	400	9	4	4
meep_shell_inst (meep_shell)	95903	121580	1819	3016	599	22873	87330	8573	113	7	0	0	0	0
openpiton_wrapper_inst (openpiton_wrapper)	965493	615912	28200	64000	20084	148423	950487	15006	682	32	400	0	0	0
ACME_OP (system)	965493	615912	28200	64000	20084	148423	950487	15006	682	32	400	0	0	0
chip (chip)	947018	603193	28096	63728	20064	146471	933418	13600	664	32	400	0	0	0
rst_sync (synchronizer)	1	2	0	0	0	2	1	0	0	0	0	0	0	0
tile0 (tile)	58909	37730	1756	3983	1254	11007	58104	805	41.5	2	25	0	0	0
tile1 (tile_206)	59162	37694	1756	3983	1254	9537	58317	845	41.5	2	25	0	0	0
tile2 (tile_213)	59217	37696	1756	3983	1254	10259	58372	845	41.5	2	25	0	0	0
tile3 (tile_214)	58544	37681	1756	3983	1254	9956	57819	725	41.5	2	25	0	0	0
tile4 (tile_215)	59136	37695	1756	3983	1254	10277	58291	845	41.5	2	25	0	0	0
tile5 (tile_216)	59832	37711	1756	3983	1254	10831	58867	965	41.5	2	25	0	0	0
tile6 (tile_217)	59720	37711	1756	3983	1254	11023	58755	965	41.5	2	25	0	0	0
tile7 (tile_218)	59203	37698	1756	3983	1254	9975	58358	845	41.5	2	25	0	0	0
tile8 (tile_219)	59069	37698	1756	3983	1254	10518	58224	845	41.5	2	25	0	0	0
tile9 (tile_220)	59736	37713	1756	3983	1254	10412	58771	965	41.5	2	25	0	0	0
tile10 (tile_207)	59756	37712	1756	3983	1254	10603	58791	965	41.5	2	25	0	0	0
tile11 (tile_208)	59161	37698	1756	3983	1254	10640	58316	845	41.5	2	25	0	0	0
tile12 (tile_209)	58587	37680	1756	3983	1254	11720	57862	725	41.5	2	25	0	0	0
tile13 (tile_210)	59188	37695	1756	3983	1254	10360	58343	845	41.5	2	25	0	0	0
tile14 (tile_211)	59246	37696	1756	3983	1254	10695	58401	845	41.5	2	25	0	0	0
tile15 (tile_212)	58574	37683	1756	3983	1254	10419	57849	725	41.5	2	25	0	0	0
chipset (chipset)	18489	12712	104	272	20	3071	17083	1406	18	0	0	0	0	0

acme_ea_1h16v_u280

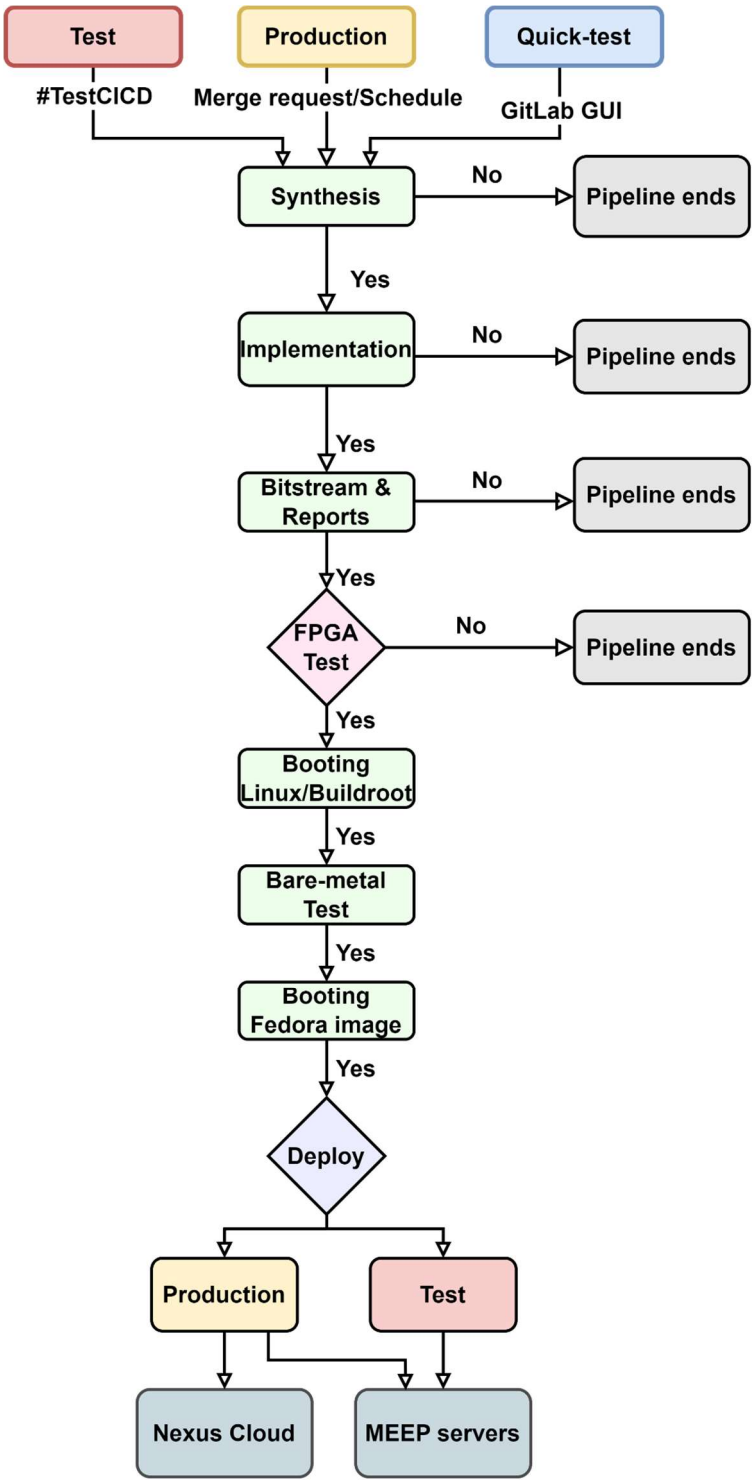
Name	CLB LUTs (1303680)	CLB Registers (2607360)	CARRY8 (162960)	F7 Muxes (651840)	F8 Muxes (325920)	CLB (162960)	LUT as Logic (1303680)	LUT as Memory (600960)	Block RAM Tile (2016)	URAM (960)	DSPs (9024)	Bonded IOB (624)	HPIOB_M (288)	HPIOB_S (288)
ACME_OP (system)	878746	738856	18855	78360	16114	141113	868326	10420	184.5	2	201	0	0	0
chip (chip)	862122	727453	18811	78128	16114	138755	853108	9014	169.5	2	201	0	0	0
rst_sync (synchronizer)	2	0	0	0	0	1	1	1	0	0	0	0	0	0
tile0 (tile)	862120	727453	18811	78128	16114	138755	853107	9013	169.5	2	201	0	0	0
cgni_blk1 (credit_to_valrdy_176)	50	8	0	0	0	11	10	40	0	0	0	0	0	0
cgni_blk2 (credit_to_valrdy_177)	47	8	0	0	0	10	7	40	0	0	0	0	0	0
cgni_blk3 (credit_to_valrdy_178)	52	8	0	0	0	13	12	40	0	0	0	0	0	0
cgno_blk1 (valrdy_to_credit)	10	8	0	0	0	3	10	0	0	0	0	0	0	0
cgno_blk2 (valrdy_to_credit_179)	10	8	0	0	0	5	10	0	0	0	0	0	0	0
cgno_blk3 (valrdy_to_credit_180)	10	8	0	0	0	3	10	0	0	0	0	0	0	0
g_lagarto_m20_core.core (vas_tile_co	839820	712981	18660	76519	15618	136169	831295	8525	160	0	201	0	0	0
genblk1[0].i_irq_sync (synchronizer	0	2	0	0	0	2	0	0	0	0	0	0	0	0
genblk1[1].i_irq_sync (synchronizer	0	2	0	0	0	2	0	0	0	0	0	0	0	0
i_csr_regfile (csr_regfile)	9121	5172	1246	361	38	2140	9121	0	0	0	0	0	0	0
i_ipi_sync (synchronizer_297)	0	2	0	0	0	2	0	0	0	0	0	0	0	0
i_sync (synchronizer_298)	2	3	0	0	0	4	2	0	0	0	0	0	0	0
i_timer_sync (synchronizer_299)	1	2	0	0	0	3	1	0	0	0	0	0	0	0
lagarto_m20 (top_drac)	830700	707782	17412	76158	15580	134843	822175	8525	160	0	201	0	0	0
datapath_inst (datapath)	32001	13052	548	1559	656	6835	31924	77	0	0	25	0	0	0
i_cache_subsystem (wt_cache_s	7479	2584	96	1652	609	1723	7479	0	32	0	0	0	0	0
i_mmu (mmu)	2328	2746	70	0	0	645	2328	0	0	0	0	0	0	0
i_xbar_dut (axi_xbar)	17106	2752	422	4556	0	5494	17106	0	0	0	0	0	0	0
icache_interface_inst (icache_int	17	66	2	0	0	21	17	0	0	0	0	0	0	0
lagarto_dcache_interface_inst (l	10517	142	8	1088	544	3307	10517	0	0	0	0	0	0	0
SP_inst (meep_lvrf_top)	18766	320	16	6	0	4737	18766	0	128	0	0	0	0	0
vpu_inst (multi_lane_wrapper)	733207	676501	16249	67129	13715	125726	724759	8448	0	0	176	0	0	0
l2 (l2)	14593	7409	75	632	64	2664	14593	0	5.5	2	0	0	0	0

acme_ea_1h16v_u55c

Name	CLB LUTs (1303680)	CLB Registers (2607360)	CARRY8 (162960)	F7 Muxes (651840)	F8 Muxes (325920)	CLB (162960)	LUT as Logic (1303680)	LUT as Memory (600960)	Block RAM Tile (2016)	URAM (960)	DSPs (9024)	Bonded IOB (624)	HPIOB_M (288)	HPIOB_S (288)
tile0 (tile)	863820	727404	18811	78128	16114	137517	854807	9013	169.5	2	201	0	0	0
cgni_blk1 (credit_to_valrdy_176)	50	8	0	0	0	11	10	40	0	0	0	0	0	0
cgni_blk2 (credit_to_valrdy_177)	47	8	0	0	0	9	7	40	0	0	0	0	0	0
cgni_blk3 (credit_to_valrdy_178)	53	8	0	0	0	12	13	40	0	0	0	0	0	0
cgno_blk1 (valrdy_to_credit)	10	8	0	0	0	3	10	0	0	0	0	0	0	0
cgno_blk2 (valrdy_to_credit_179)	10	8	0	0	0	5	10	0	0	0	0	0	0	0
cgno_blk3 (valrdy_to_credit_180)	10	8	0	0	0	3	10	0	0	0	0	0	0	0
g_lagarto_m20_core.core (vas_tile_co	841517	712932	18660	76519	15618	133294	832992	8525	160	0	201	0	0	0
genblk1[0].i_irq_sync (synchronizer	0	2	0	0	0	2	0	0	0	0	0	0	0	0
genblk1[1].i_irq_sync (synchronizer	0	2	0	0	0	2	0	0	0	0	0	0	0	0
i_csr_regfile (csr_regfile)	9121	5172	1246	361	38	2330	9121	0	0	0	0	0	0	0
i_ipi_sync (synchronizer_297)	0	2	0	0	0	2	0	0	0	0	0	0	0	0
i_sync (synchronizer_298)	2	3	0	0	0	4	2	0	0	0	0	0	0	0
i_timer_sync (synchronizer_299)	0	2	0	0	0	2	0	0	0	0	0	0	0	0
lagarto_m20 (top_drac)	832397	707733	17412	76158	15580	131847	823872	8525	160	0	201	0	0	0
datapath_inst (datapath)	31992	13055	548	1559	656	8057	31915	77	0	0	25	0	0	0
i_cache_subsystem (wt_cache_s	7477	2584	96	1652	609	1668	7477	0	32	0	0	0	0	0
i_mmu (mmu)	2329	2746	70	0	0	650	2329	0	0	0	0	0	0	0
i_xbar_dut (axi_xbar)	17146	2752	422	4556	0	6317	17146	0	0	0	0	0	0	0
icache_interface_inst (icache_int	17	66	2	0	0	27	17	0	0	0	0	0	0	0
lagarto_dcache_interface_inst (l	10517	142	8	1088	544	3244	10517	0	0	0	0	0	0	0
SP_inst (meep_lvrf_top)	18769	320	16	6	0	4768	18769	0	128	0	0	0	0	0
vpu_inst (multi_lane_wrapper)	734863	676449	16249	67129	13715	121240	726415	8448	0	0	176	0	0	0
I2 (I2)	14589	7409	75	632	64	2753	14589	0	5.5	2	0	0	0	0

Appendix III

Flow chart of CICD FPGA Shell Flow



Appendix IV

1. Production resources results using U55C Alveo card.

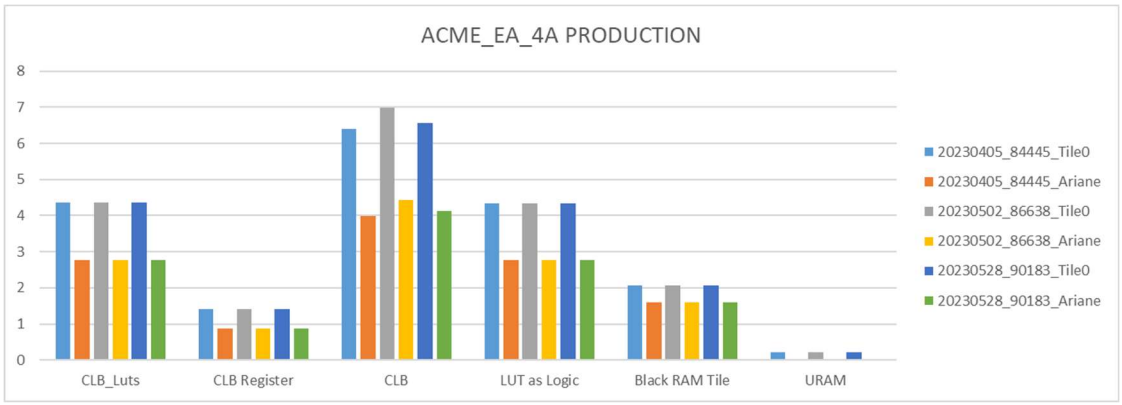
ID	Module	CLB_Luts	CLB Register	CLB	LUT as Logic	Block RAM Tile	URAM
20230405_84445	**ACME EA 4A**						
	TILE0	56831	36847	10441	56543	41.5	2
	ARIANE	36120	22798	6505	36120	32	0
	ACME EA 1H						
	TILE0	52960	34657	9464	52672	41.5	2
	LAGARTO_M20	26672	15449	5016	26672	32	0
	ACME EA 4H2V						
	TILE0	160779	126538	32806	159287	161.5	2
	LAGARTO_M20	132019	107317	27343	130815	152	0
	VPU_INST	99788	89843	20780	98584	0	0
20230502_86638	**ACME EA 4A**						
	TILE0	56827	36848	11376	56539	41.5	2
	ARIANE	36119	22799	7228	36119	32	0
	ACME EA 1H						
	TILE0	55257	37159	10445	54892	41.5	2
	LAGARTO_M20	27844	17951	5443	27767	32	0
	ACME EA 4H2V						
	TILE0	163668	129080	29495	162099	161.5	2

	LAGARTO_M20	133769	109859	23986	132488	152	0
	VPU_INST	99855	89694	18360	98651	0	0
	ACME EA 1H2G						
	TILE0	432534	237321	84648	429379	161.5	2
	LAGARTO_M20	402964	218106	78850	400097	152	0
	VPU_INST	100264	89714	21087	99061	0	0
	SA-HECV	117965	63406	24534	117007	0	0
	SA-NN	146995	43800	26502	146366	0	0
20230528_90183	**ACME EA 4A**						
	TILE0	56898	36848	10680	56610	41.5	2
	ARIANE	36162	22799	6726	36162	32	0
	ACME EA 1H						
	TILE0	55731	37161	10363	55366	41.5	2
	LAGARTO_M20	28321	17954	5519	28244	32	0
	ACME EA 4H2V						
	TILE0	164088	129076	33948	162519	161.5	2
	LAGARTO_M20	134206	109855	28258	132925	152	0
	VPU_INST	99653	89692	20824	98449	0	0
	ACME EA 1H2G						
	TILE0	432632	237322	83958	429477	161.5	2
	LAGARTO_M20	403058	218107	78801	400191	152	0

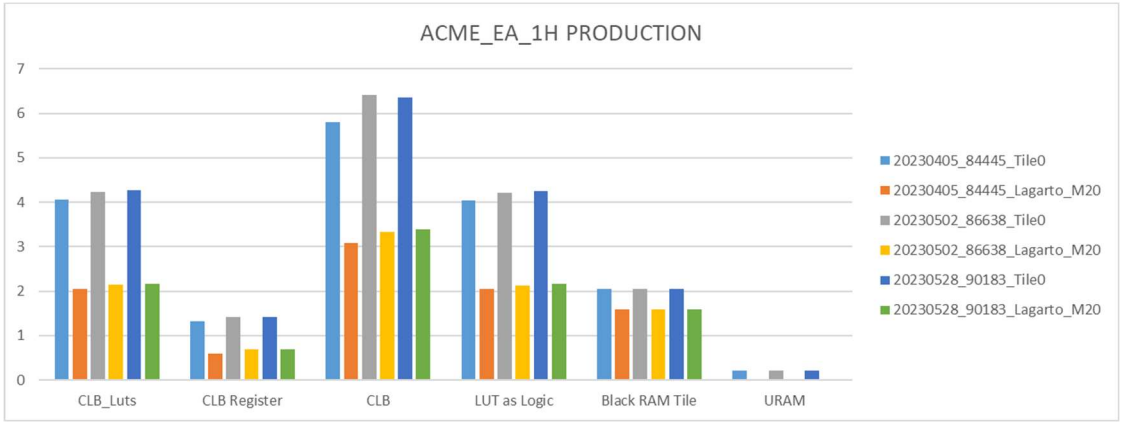
	VPU_INST	100235	89714	20611	99032	0	0
	SA-HECV	117951	63407	25562	116993	0	0
	SA-NN	147011	43801	26402	146382	0	0

2. Dashboard of each ACME flavor

a . ACME_EA_4A Production release



b. ACME_EA_1H Production release



c. ACME_EA_4H2V Production release