MEEP | MareNostrum Experimental Exascale Platform

FPGA Workshop

Xavier Martorell, Daniel Jiménez-Mazure

September 12th, 2022

Outline

- FPGAs from 10000 feet...
- ... and coming down
- FPGA components
- Generating FPGA configurations

FPGAs: 10000 feet view

- The boards

Discrete (to be connected to PCIe)

source: docs.xilinx.com

Integrated (standalone)

source: www.axiom-project.eu

MEEP MareNostrum Experimental Exascale Platform

FPGAs: 10000 feet view

- The boards

Discrete (to be connected to PCIe)

AMD-Xilinx
  Versal
  Alveo
  VCU128
  ...
Intel-Altera
  Agilex
  Stratix
  Arria
  ...

source: docs.xilinx.com

Integrated (standalone)

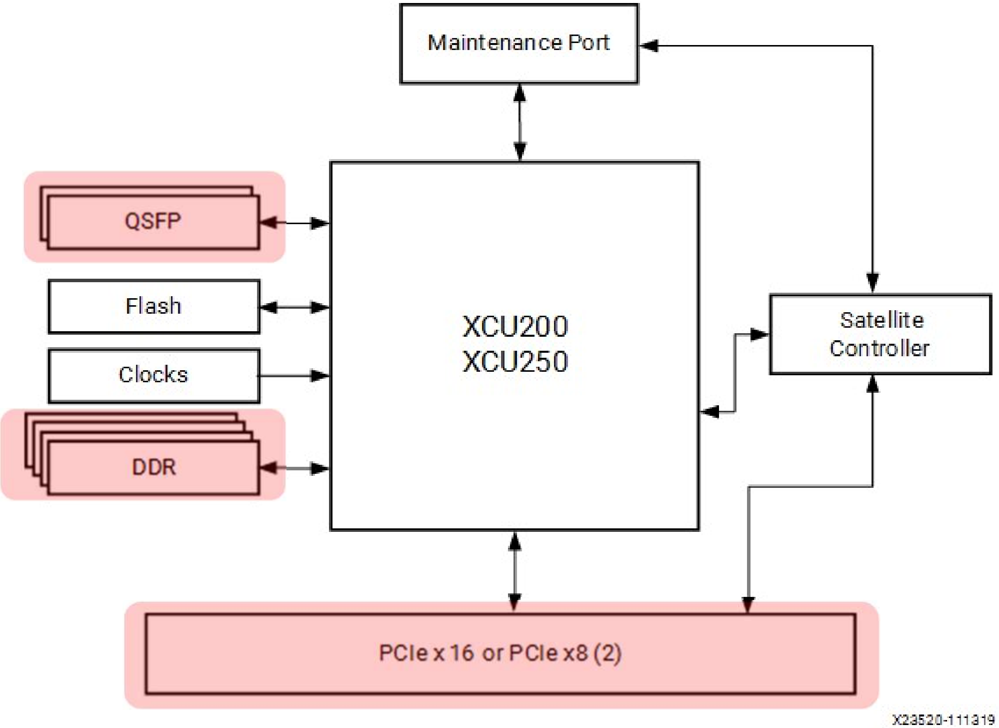
AMD-Xilinx
  Zynq 7000
  Zynq Ultrascale+
  ...

Intel-Altera
  Cyclone
  Max
  ...

source: www.axiom-project.eu

FPGAs: 5000 feet view

- Alveo U200 card basic hardware blocks

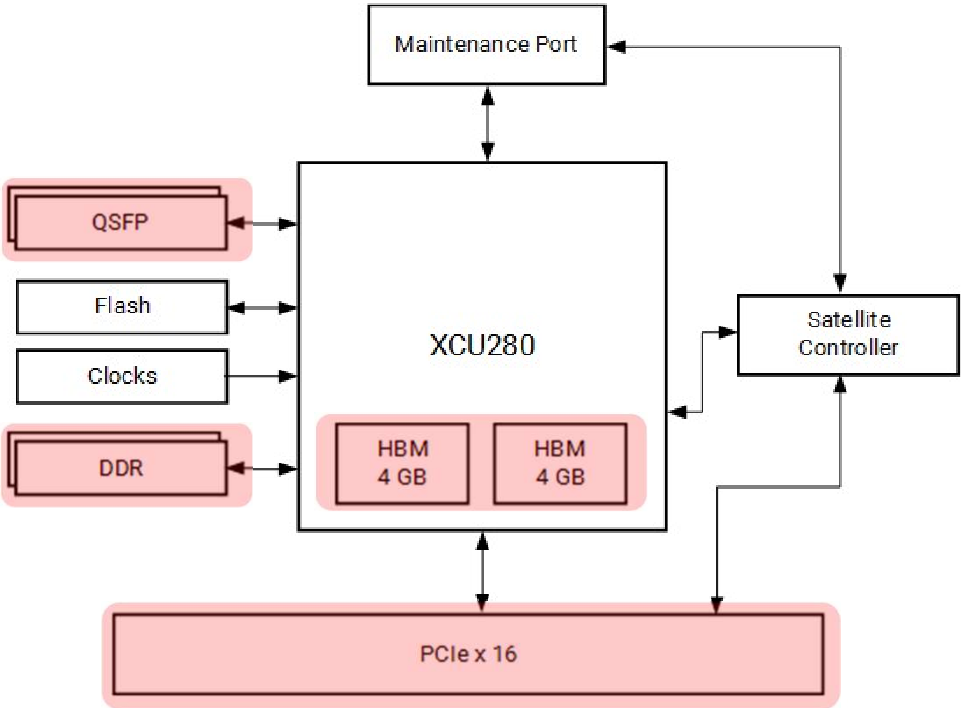


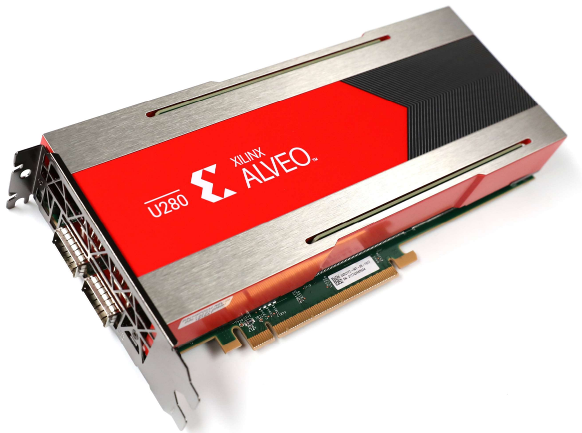
source: docs.xilinx.com

MEEP MareNostrum Experimental Exascale Platform

FPGAs: 5000 feet view

- Alveo U280 card basic hardware blocks



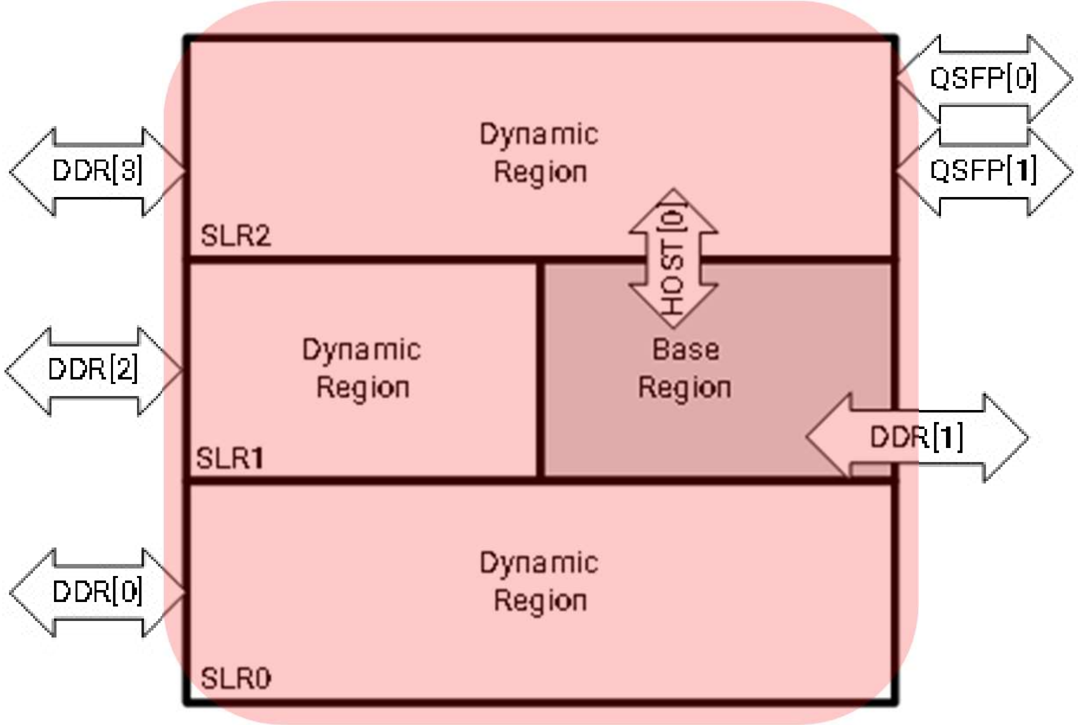


source: docs.xilinx.com

FPGAs: 10 inches view

- U200 FPGA connectivity & layout
  - DDR -> Global memory (RAM)
  - QSFP -> Ethernet 100Gb.
  - PCIe -> to host
  - USB -> serial line, jtag

  - Super Logic Regions (SLR)
    - Logic partitions of the area of the FPGA
    - Connectors attached to SLRs

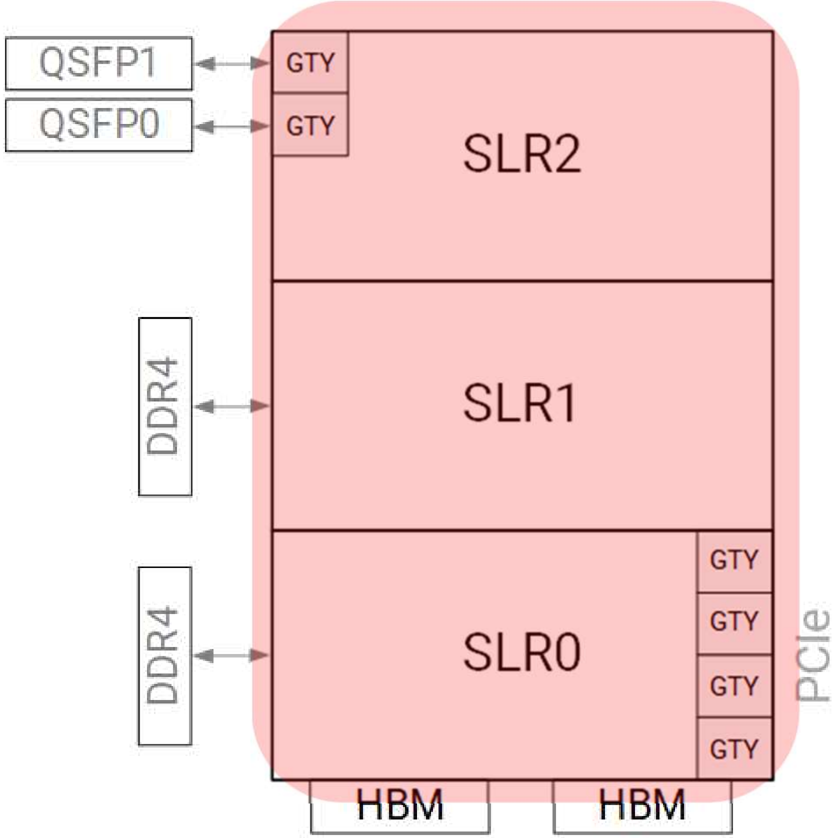


MEEP MareNostrum Experimental Exascale Platform

source: docs.xilinx.com

FPGAs: 10 inches view

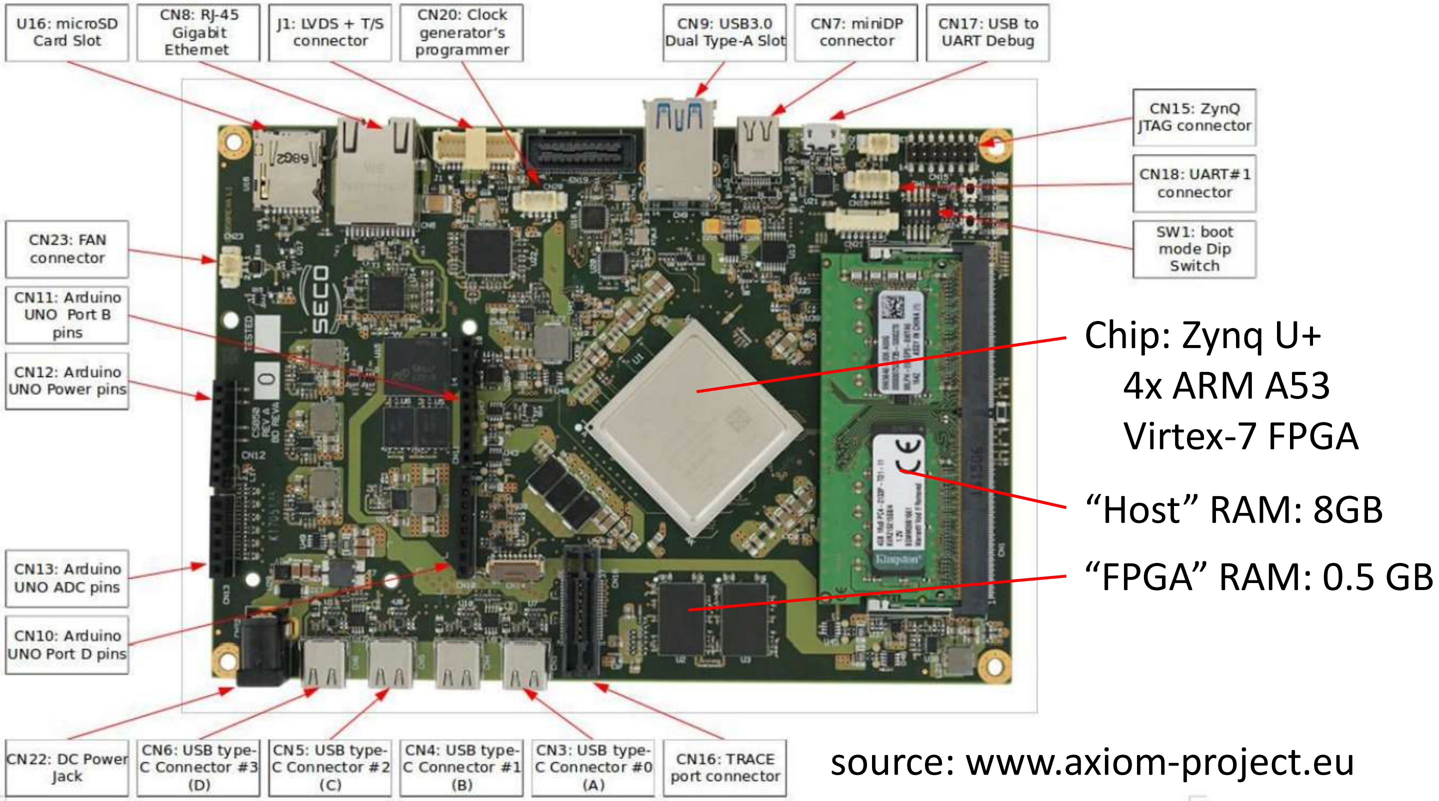
- U280 FPGA connectivity & layout
  - DDR -> Global memory (RAM)
  - HBM -> Global memory (HBM)
  - QSFP -> Ethernet 100Gb.
  - PCIe -> to host
  - USB -> serial line, jtag



source: docs.xilinx.com

FPGAs: 10 inches view

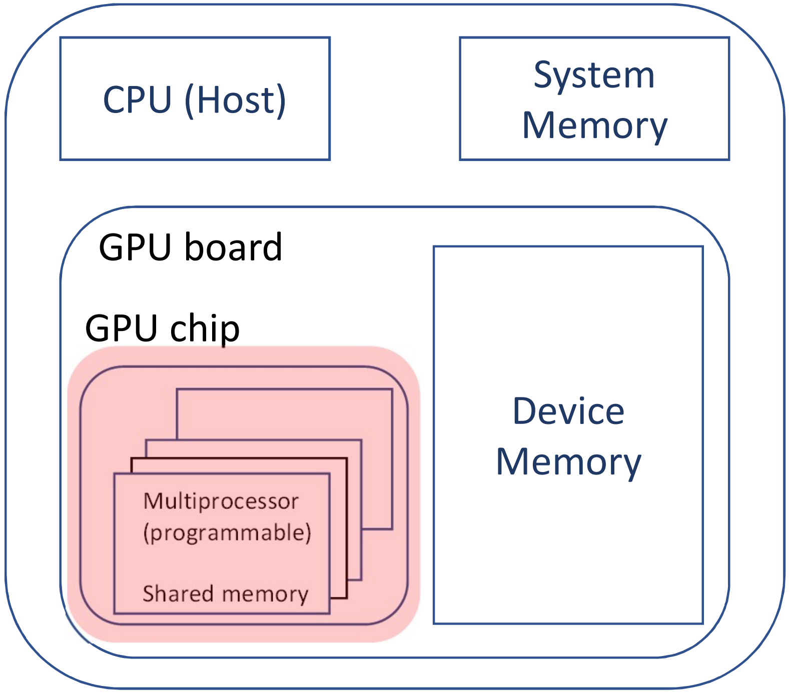
- Axiom board connectivity and layout



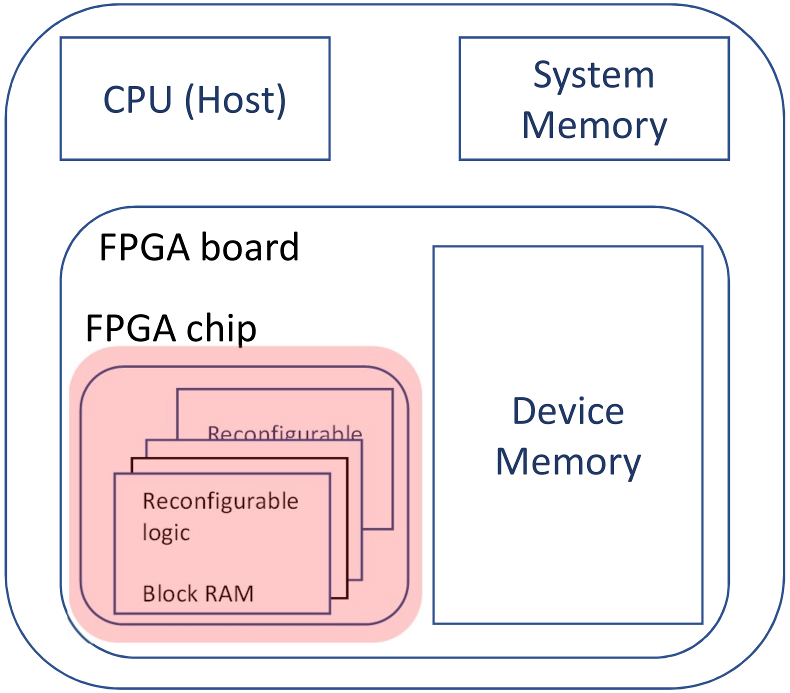
source: www.axiom-project.eu

Comparing FPGAs to GPUs

- “Basically”, replace the GPU compute device with the FPGA device



GPU System

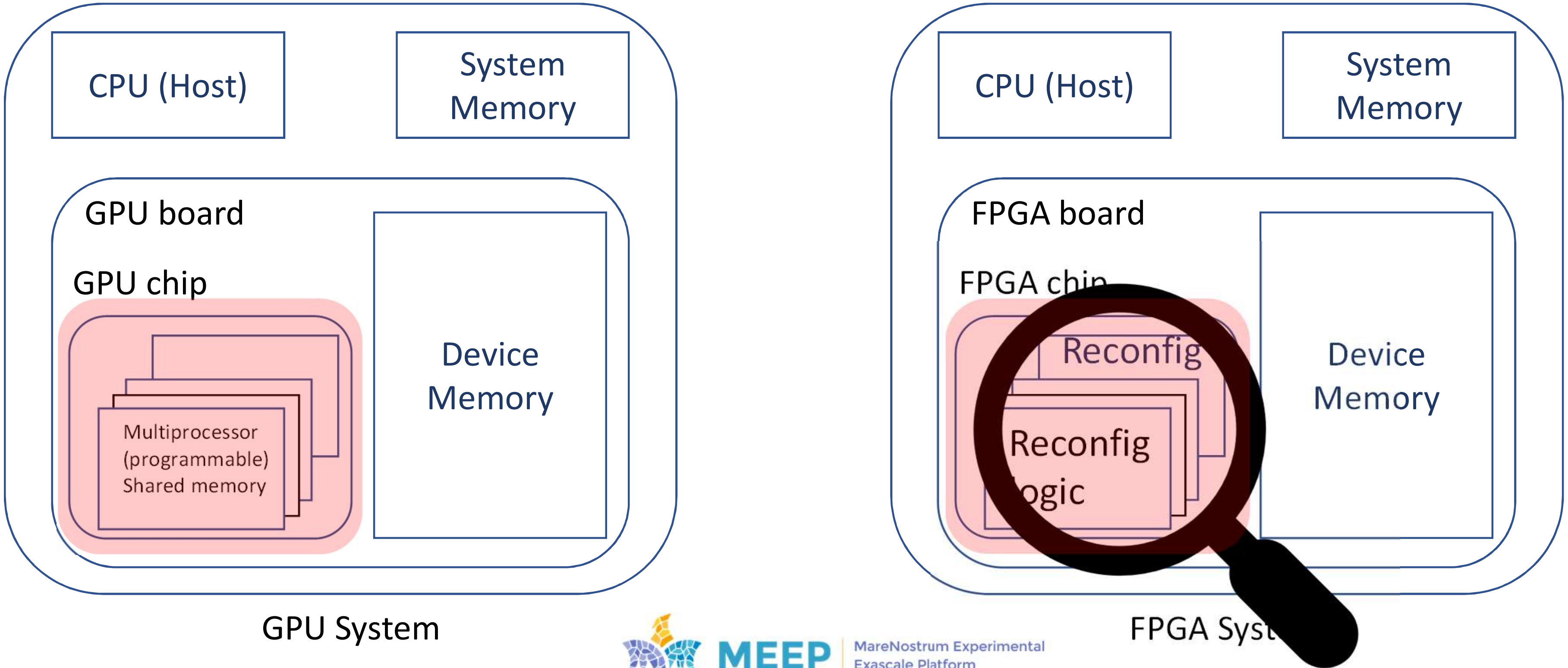


FPGA System

MEEP MareNostrum Experimental Exascale Platform

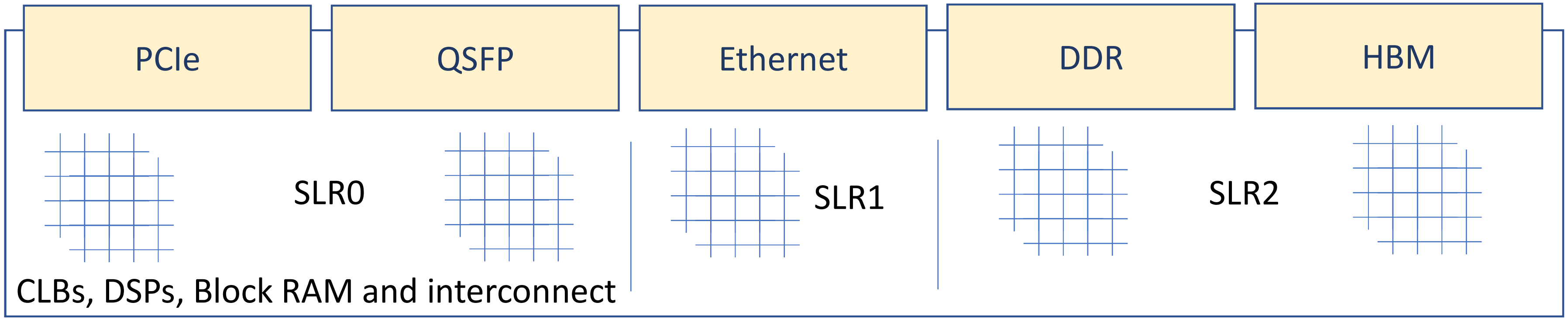
Comparing FPGAs to GPUs

- "Basically", replace the GPU compute device with the FPGA device



Inside the FPGA chip

- Interfaces
- Super Logic Regions (SLRs)
  - Only on big devices (Alveo)
- Components: Configurable Logic Blocks (CLBs)



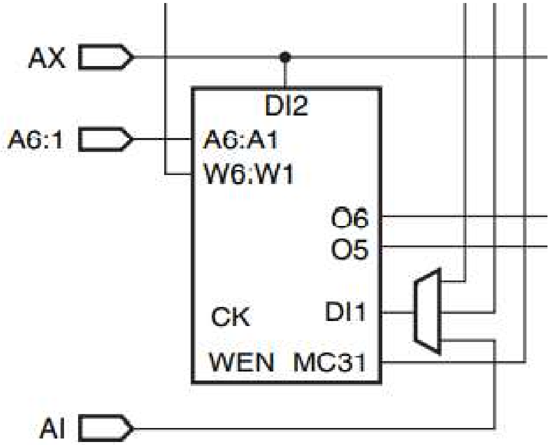
FPGA chip

SLR components

- CLB (Configurable Logic Block)
  - Carry chain (adder, counter, comparator...)
  - Multiplexer (8 / 16 to 1)
  - Shift register
  - Flip-flop
  - Look-up tables (6-input)
  - Distributed RAM
    - Recommended for 1 to 128 bits data
- Block RAM
- DSP (Digital Signal Processing)

Sample components

- Look-up table (8 on a CLB)
  - 6 inputs (A6:1)
  - 2 outputs (O6:5)
  - 128 bits total



Available resources
ZU+: 274K

U200: 388K + 205K + 385K
U250: 420K + 205K + 407K + 424K

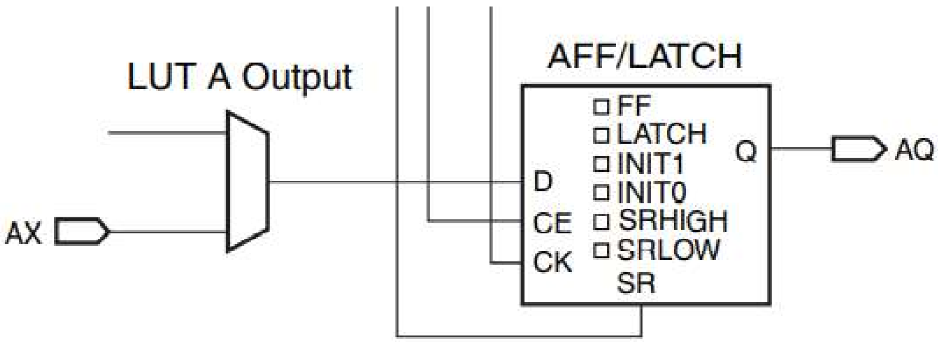
source: docs.xilinx.com

Sample components

- D flip-flop / latch (16 on a CLB)
  - 1 bit memory
  - Can take output from LUT A

Available resources
ZU+: 548K

U200: 776K + 410K + 770K
U250: 840K + 411K + 815K + 849K



MEEP MareNostrum Experimental Exascale Platform

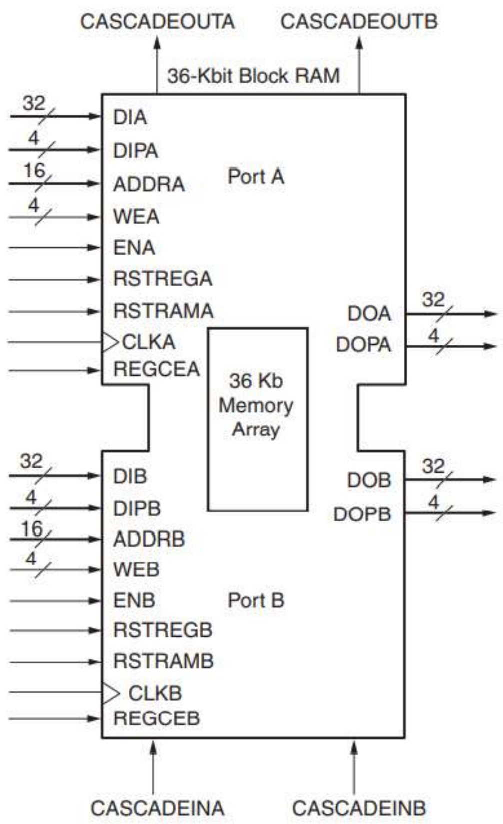
source: docs.xilinx.com

Sample components

- Block RAM
  - Dual-port 36Kb
  - Programmable FIFO
  - Configurable width
    - 32Kb x 1 ... 4Kb x 8 ... 1Kb x 36
- Advise
  - Use as many blocks as possible to achieve data parallelism

Available resources
ZU+: 32Mbit

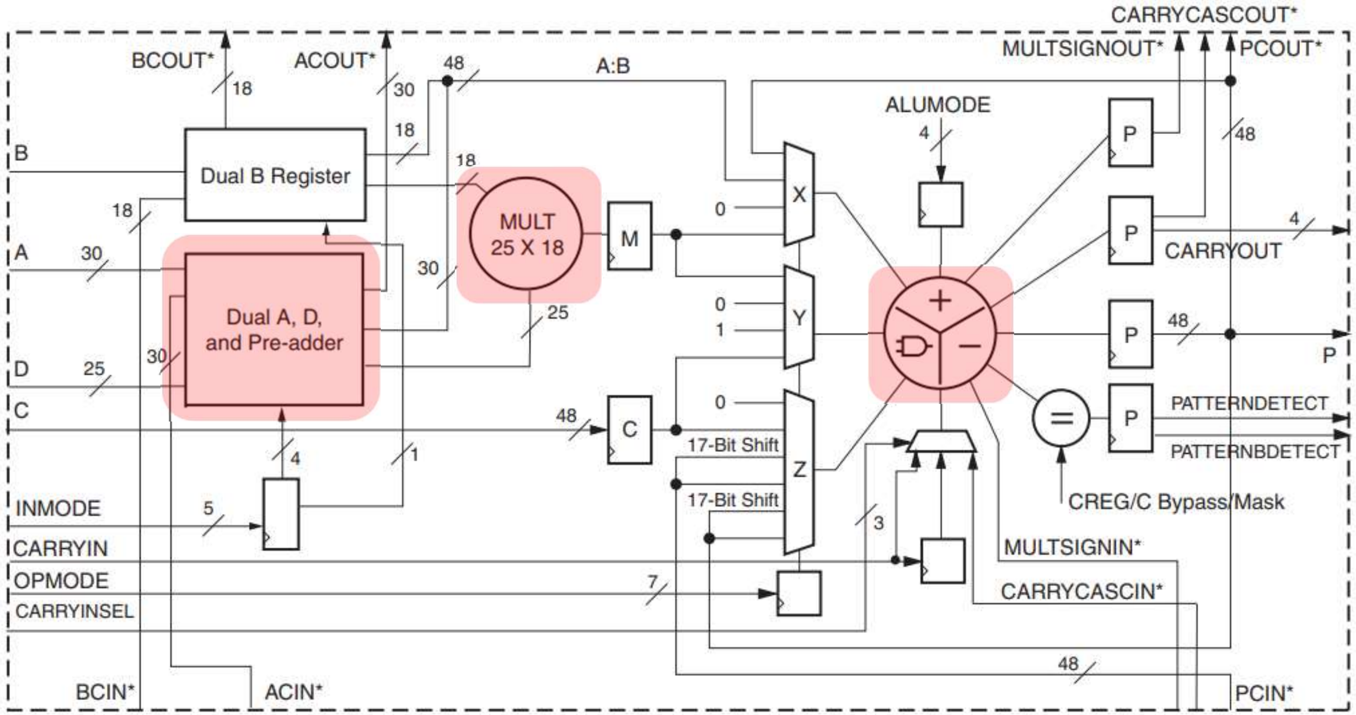
U200: 25Mb + 13Mb + 25Mb
U250: 18Mb * 4



source: docs.xilinx.com

Sample components

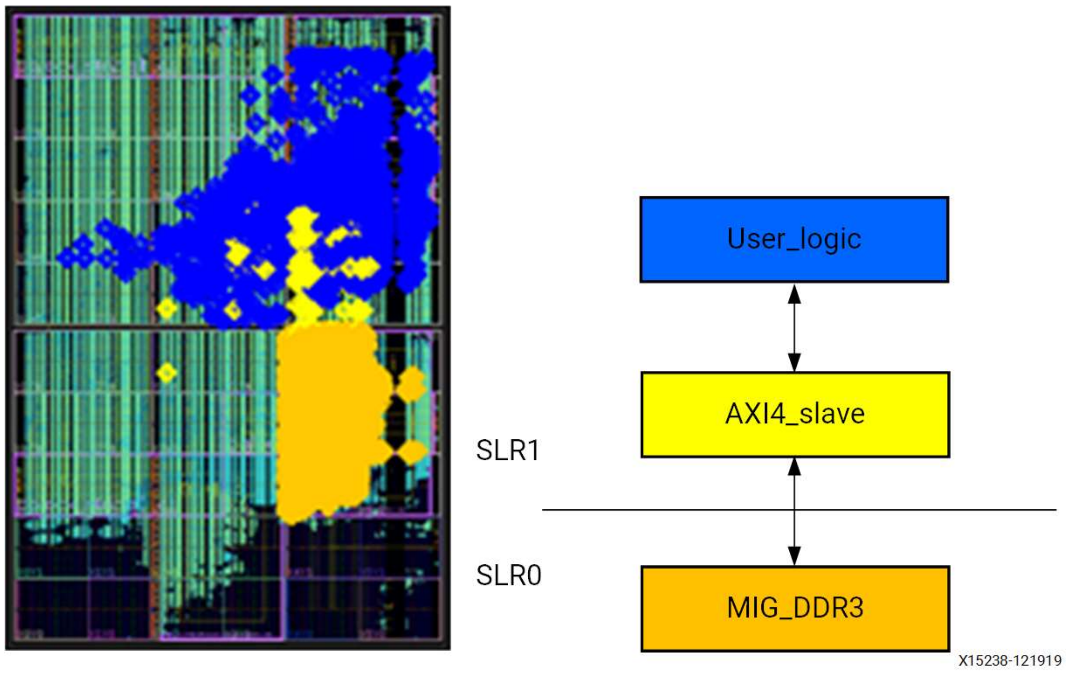
- DSPs (48-bit)
  - Multiplier
  - Accumulator
  - Pre-adder
    - C + B*(D+A) + Carryin
  - SIMD unit (+, - only)
    - 2x24bit, 3x12bit
  - Logic unit
    - And, or, xor...
  - Pattern detector
    - Detect output pattern, or
    - C match with A*B...



MEEP MareNostrum Experimental Exascale Platform

Sample design

- Memory accesses are driven by the MIG (Memory Interface Generator)
- Interconnects are implemented with the AXI ports
- User logic contains the application



source: docs.xilinx.com

Generating FPGA configurations

- High-level C/C++ programming
  - No Fortran support from vendors (as far as we know)
  - Compiled to Verilog/VHDL
- Design source files
  - Behavioral simulation
- Design synthesis, HDL to gates, generating FPGA netlists
  - Functional verification
- Design mplementation, place & route
  - Static timing analysis
- Bitstream generation
  - Actual run on the FPGA

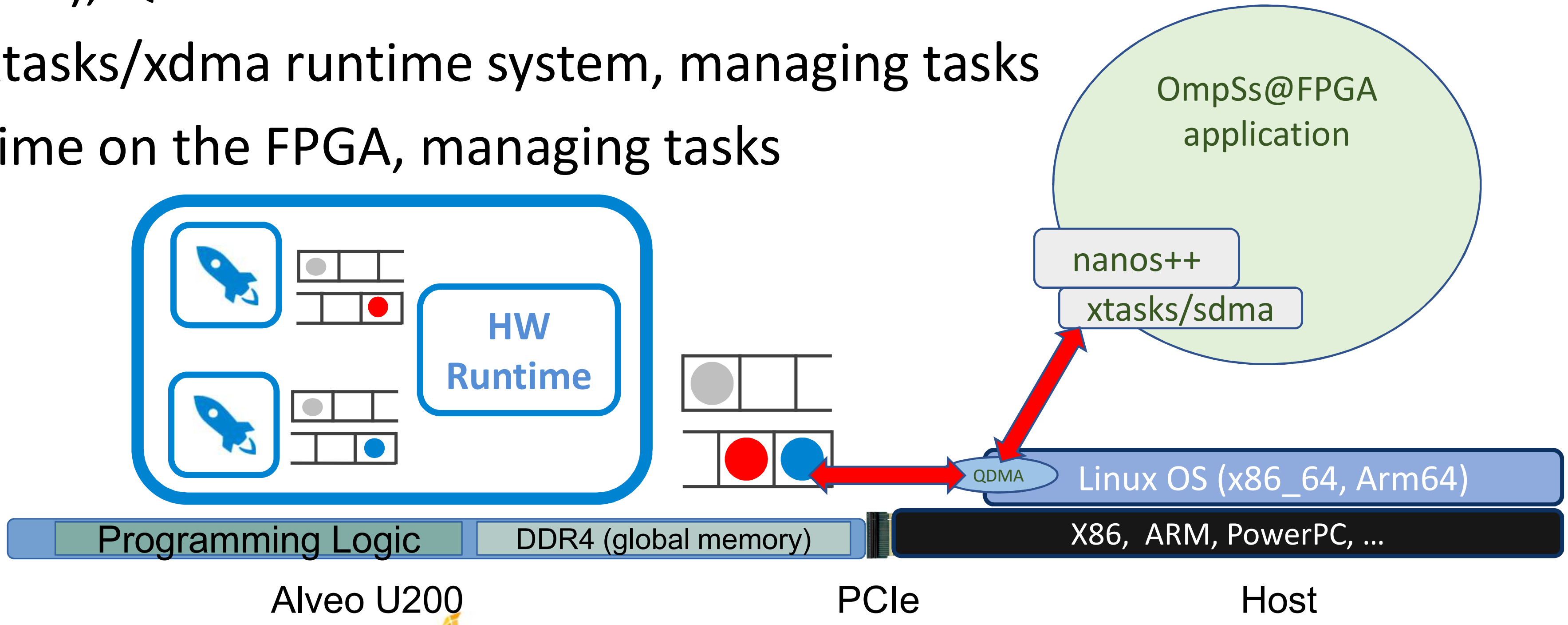
Vendor tools

- AMD - Xilinx
  - Vivado HLS / **Vitis HLS** (from 2020)
  - Vivado / **Vitis**
    - GUI and batch tools
  - Compile C/C++ code and OpenCL kernels

- Intel – Altera
  - Quartus HLS compiler
  - Quartus Prime Design Software
    - GUI and batch tools
  - Compile C/C++ code and OpenCL kernels
  - Also supports the **oneAPI** interface
    - Originated from Codeplay/Khronos SYCL
    - Kernels on C++ Lambda functions

OmpSs@FPGA execution environment

- x86_64 / Arm64 architectures
- Linux (Host), QDMA driver
- Nanos/xtasks/xdma runtime system, managing tasks
- Hw runtime on the FPGA, managing tasks



References

- References
  - DS962 – Alveo U200 and U250 Data Center Accelerator Cards Data Sheet
    - https://docs.xilinx.com/r/en-US/ds962-u200-u250
  - 7 Series FPGAs Configurable Logic Block. User Guide. Xilinx, Inc., UG474, 2016
    - https://docs.xilinx.com/v/u/en-US/ug474_7Series_CLB
  - 7 Series DSP48E1 Slice
    - https://docs.xilinx.com/v/u/en-US/ug479_7Series_DSP48E1
  - 7 Series FPGAs Memory Resources User Guide
    - https://docs.xilinx.com/v/u/en-US/ug473_7Series_Memory_Resources

References

- References
  - FPGA configuration generation steps
    - https://hardwarebee.com/fpga-design/

MEEP | MareNostrum Experimental Exascale Platform

MEEP FPGA Shell

Xavier Martorell, Daniel Jiménez-Mazure

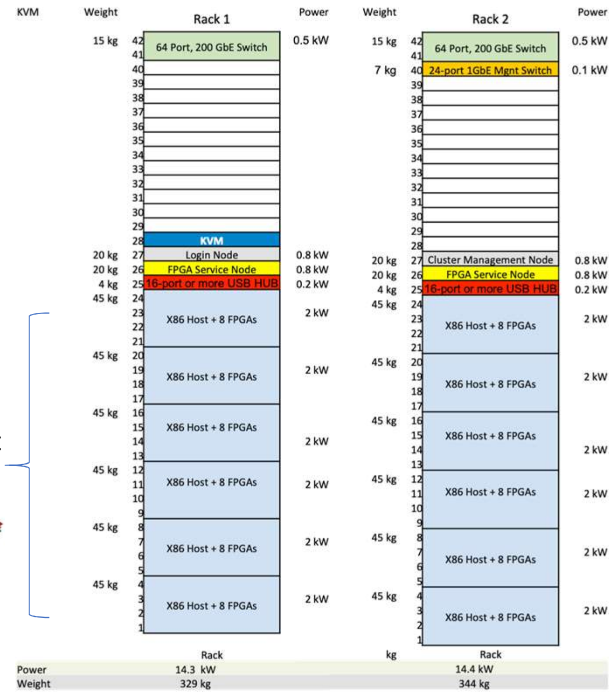
September 12th, 2022

MEEP Infrasctructure

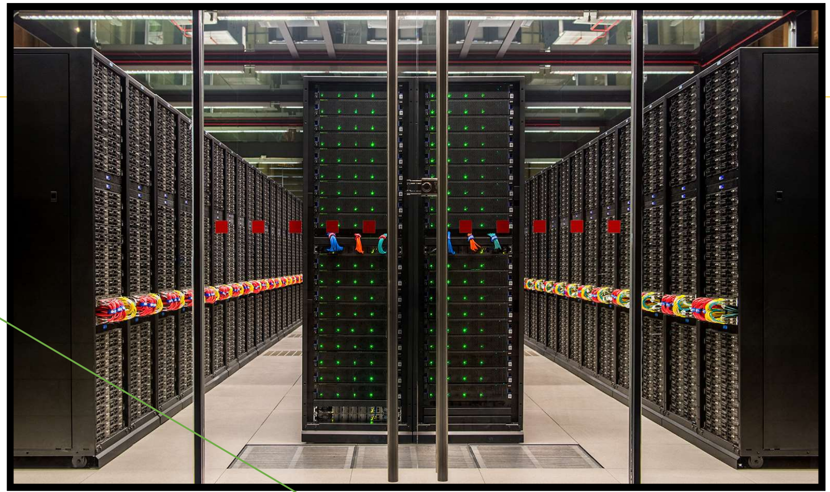
Understanding the dimension of the MEEP cluster:

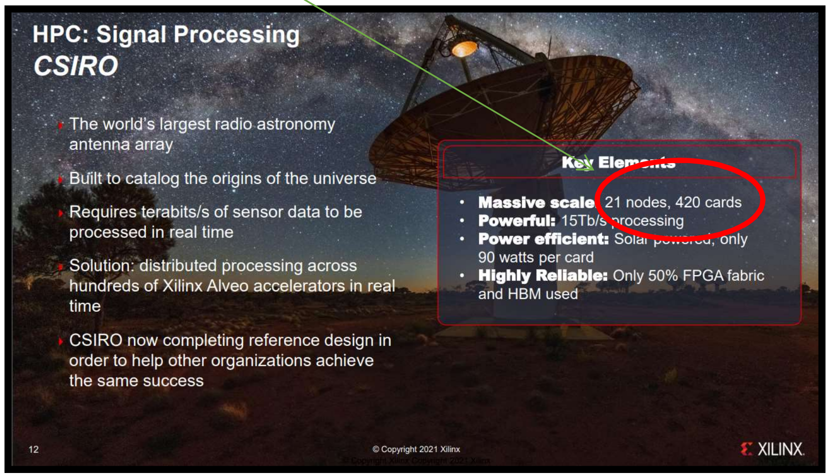
- x96 Alveo U55C boards distributed in 12 nodes
- High connectivity via 200GbE switches
- ~ 25% of the highest Antena array in the world (CSIRO)

x48 Alveo u55C



x48 Alveo u55C





https://www.csiro.au/en/about/facilities-collections/atnf/australia-telescope-compact-array

MEEP MareNostrum Experimental Exascale Platform

We are getting a lot!

Lot of resources…

X96

Display Name	Preview	LUT Elements	FlipFlops	Block RAMs	Ultra RAMs	DSPs	Gb Transceivers	MMCMs	PCIe
Alveo U280 Data Center Accelerator Card Add Daughter Card Connections		1303680	2607360	2016	960	9024	24	12	6

1 Billion LUTs!

MEEP Shell FPGA Project: Summary

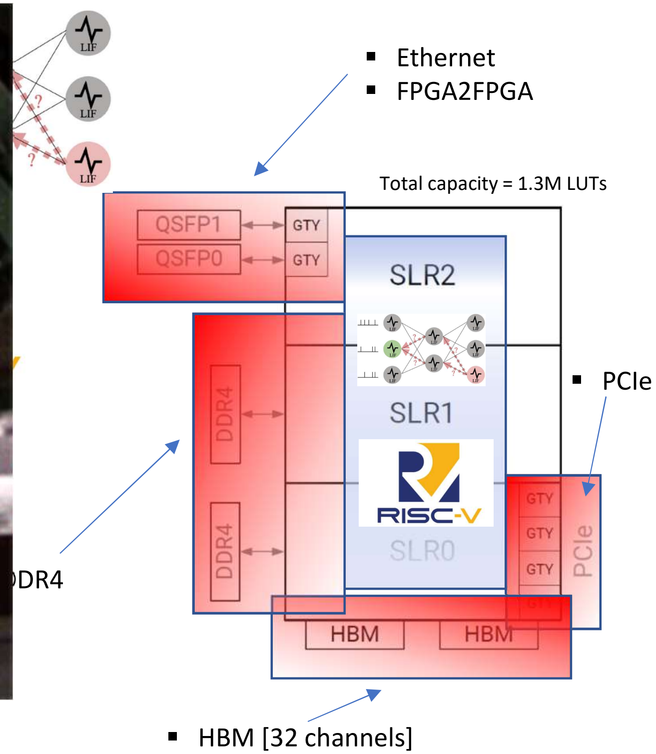
The MEEP Shell project is built on top of four main branches:

1) The SW support (Drivers, Tools)
2) The FPGA support (Shell generation process)
3) Timing closure scripts (TCL)
4) The CI/CD FPGA Flow (Gitlab CICD and Vivado Non-Project Mode)

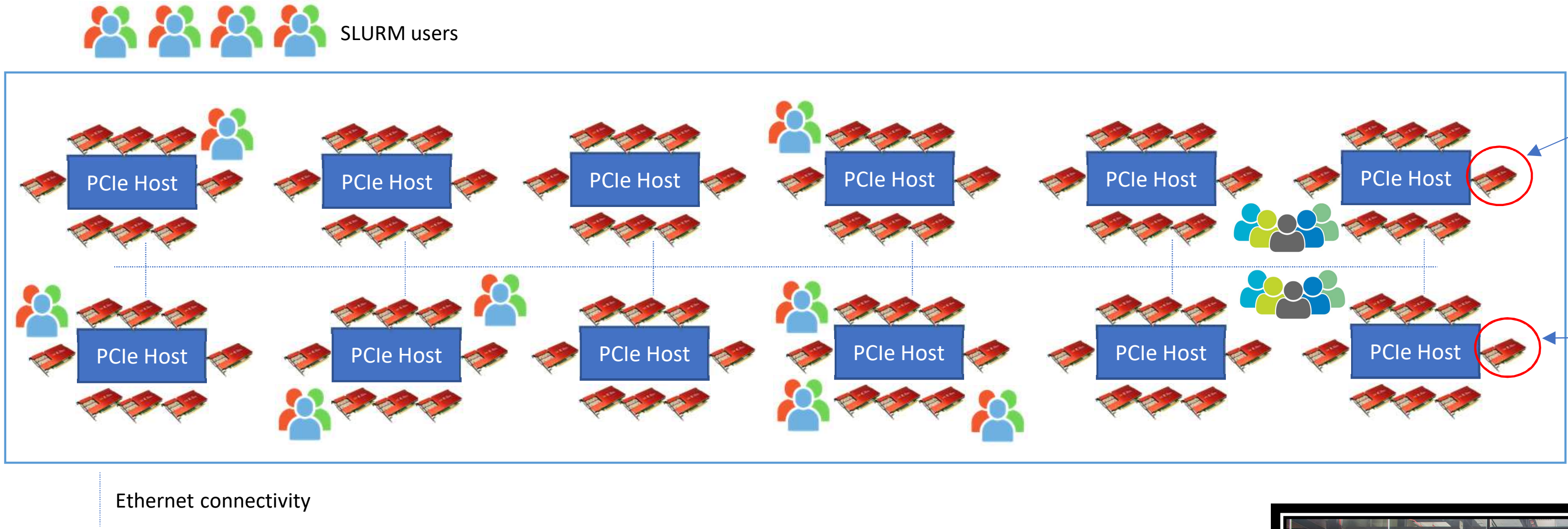
MEEP FPGA Shell (I)

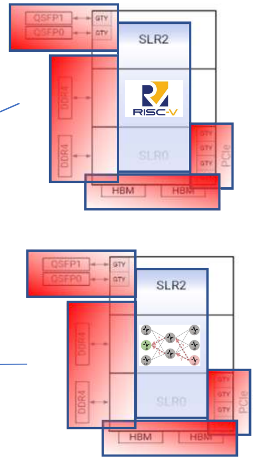
- Git friendly code for version control and project generation
- Vivado version-independent
- Compatible with Alveo family (u280, u55c, u200, u250)
- The EA becomes FPGA agnostic

Shell IP	Frequency [MHz]	User Clock	Resources [LUT]
PCIe (QDMA)	250	Fixed	70376
HBM	≤ 450	Maximum	1539
Ethernet	322.26	Fixed	7444
Aurora	≤ 402.23	Maximum	1500
Aurora (DMA Mode)			4849
DDR4	300	Fixed	18823

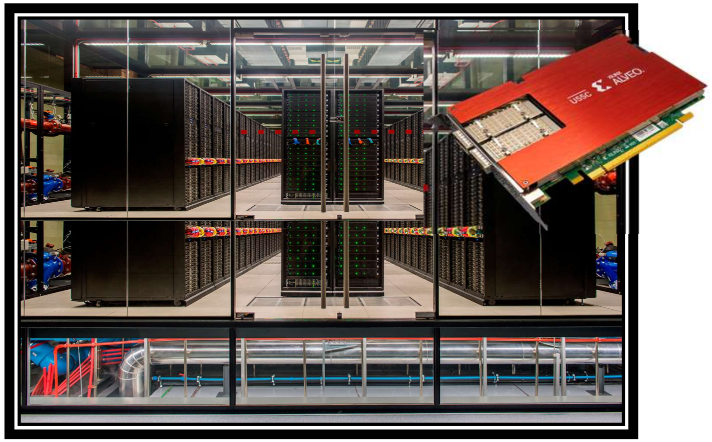


MEEP FPGA Shell (II)





x96 Alveo u55c

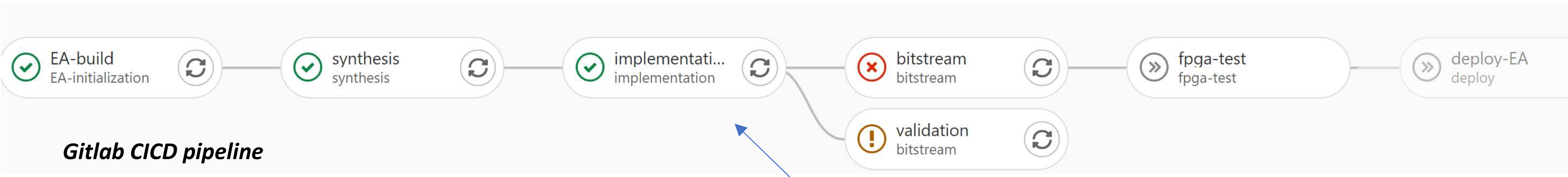


- Hundred of users potentially
- The MEEP FPGA Shell creates an homogeneus FPGA baseline
- Deals with PCIe enumeration issues
- Saves time: IP set up time x number of IPs x Number of users

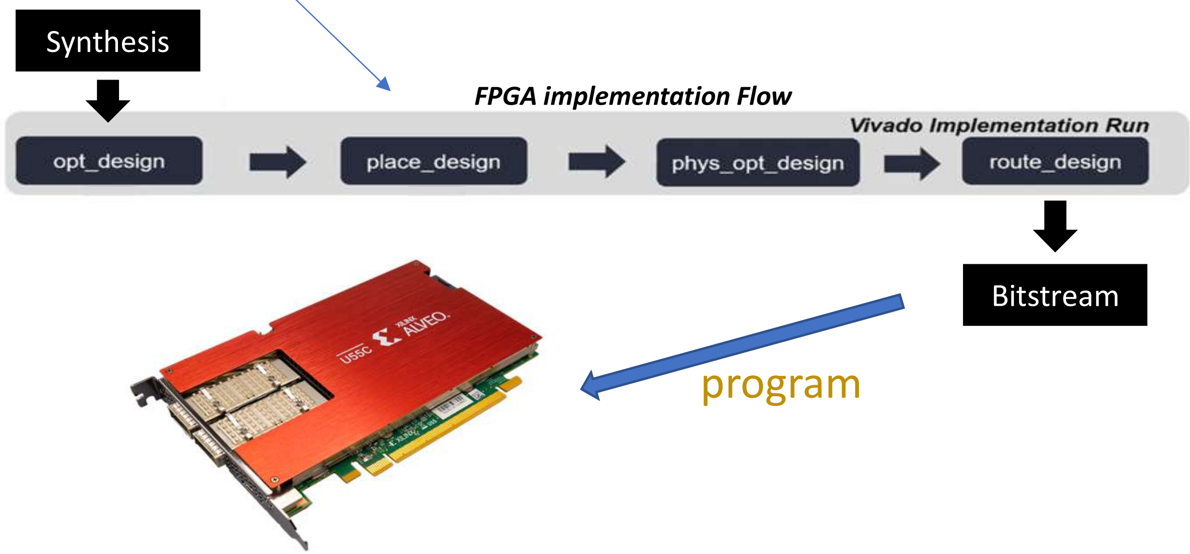
$$Time\ Saved[months] = IPsetupTime \times IPcatalog \times NumberOfUsers$$

MEEP MareNostrum Experimental Exascale Platform

MEEP FPGA Shell:CICD



- Typical staged pipeline
- In this case, *FPGA oriented*
- Saves a massive amount of debugging time



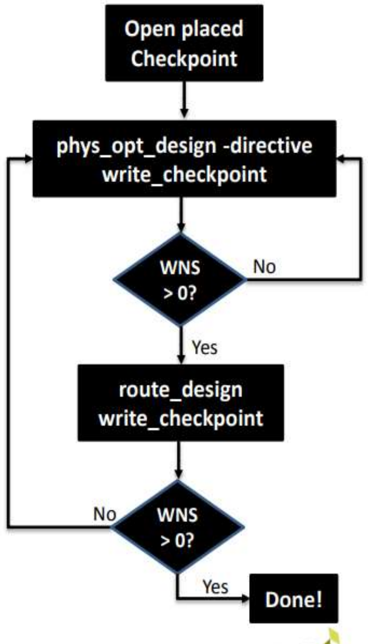
1) Generates the Vivado project
2) Synthesis
3) Implementation
4) Bitstream
5) Program & Validation
6) Deploy
7) … More (custom)

MEEP MareNostrum Experimental Exascale Platform

MEEP FPGA Shell: Timing Closure

- Custom and scripted (TCL) FPGA flow implementing advanced timing closure techniques, based on the online estimated parameters
  - WNS (Worst Negative Slack)
  - Estimated Congestion
- And other utilities
  - Reject bad timed designs
  - Reject DRC issues (pinout)
  - Unsafe CDC crossings
- Integrated in the CICD Flow

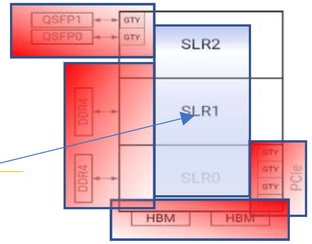
The CICD checks the WNS in order to pass!



Iterative process

MEEP MareNostrum Experimental Exascale Platform

MEEP FPGA Shell: Example

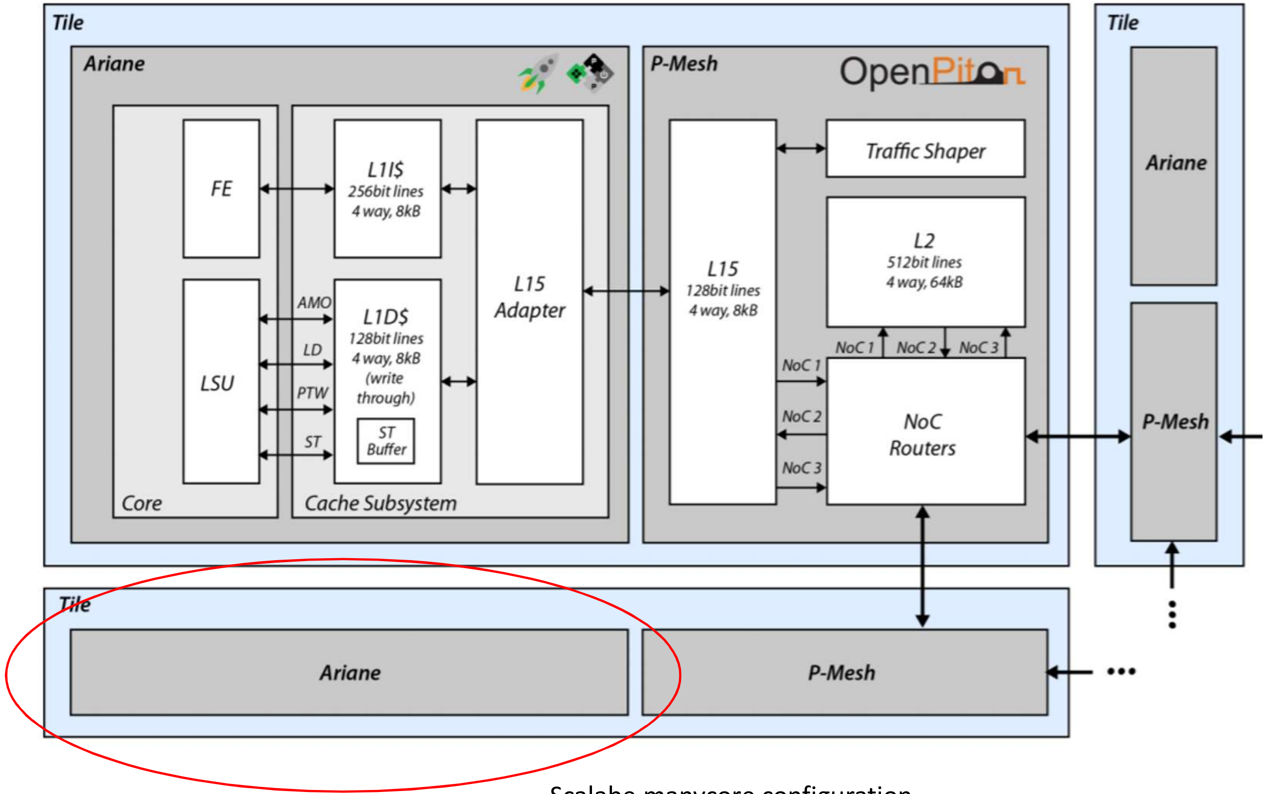


https://github.com/MEEPproject

OpenPiton:
- OpenSource manycore framework
- Based on python + Verilog + automatic code generation
- Supports RISCV
- Scalable (Up to 256x256 grid!)
- A NoC based coherence protocol
- Boots Linux (with the right CPU and memory hierarchy)

OpenPiton ETH Zürich RISC-V

FPGA Challengues
- High usage (depends on CPUs)
  - We have fit 16 Arianes (CVA6)
- Congestion (e.g, FPU)
- Floorplannig (Ultrascale + HBM)



Scalabe manycore configuration

MEEP MareNostrum Experimental Exascale Platform

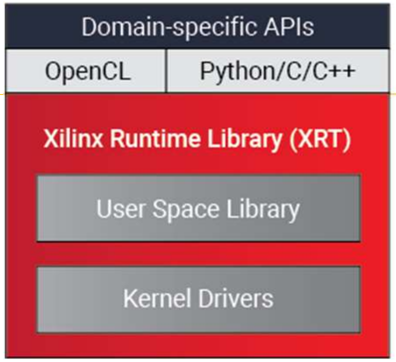
MEEP FPGA Shell: Similarities?

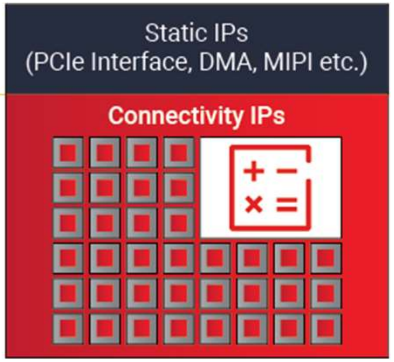
- Some analogies can be set with **Amazon F1** or **Xilinx Vitis Platform**, but the MEEP FPGA Shell is ***different***
- The FPGA Shell is currently in use as a fundational FPGA framework for different European Projects (eProcessor, MEEP) and BSC internal projects
  - Focus on RISC-V systems
- It is currently meeting the expectations as process oriented tool
- The FPGA designs are working ☺

https://github.com/MEEPproject

Vitis

- Vits offers Platforms
  - Hardcoded bitstreams, no configuration possible
  - IP can be placed inside

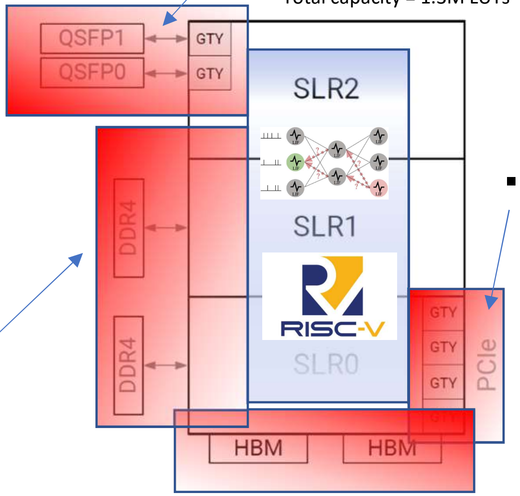




MEEP

- Ethernet
- FPGA2FPGA

Total capacity = 1.3M LUTs



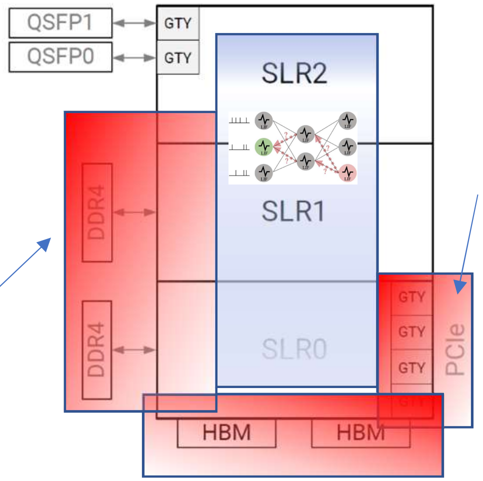
- ✓ Fully customizable
- ✓ QSFP support
- ✓ QDMA

- PCIe (QDMA)
- DDR4
- HBM [32 channels]

VS

VITIS

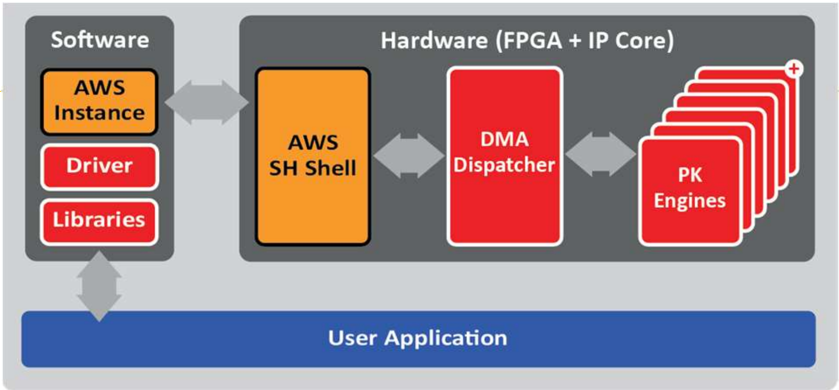
Total capacity = 1.3M LUTs



- PCIe (XDMA)
  - Fixed configurations
  - No QSFP support
  - XDMA
- DDR4
- HBM [32 channels]

Amazon F1 AWS

- F1 offers the Vitis approach:

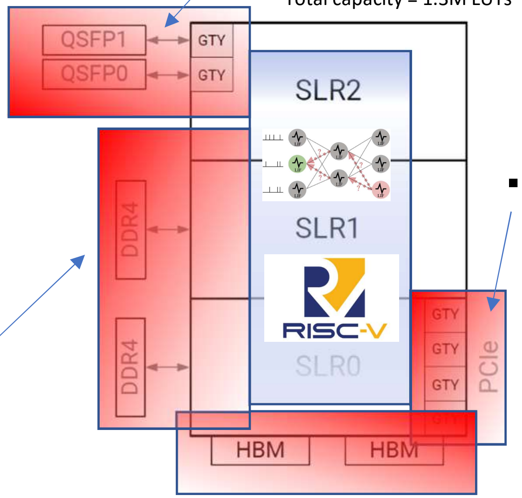


https://www.silexinsight.com/blockchain-hardware-accelerator/

MEEP

- Ethernet
- FPGA2FPGA

Total capacity = 1.3M LUTs



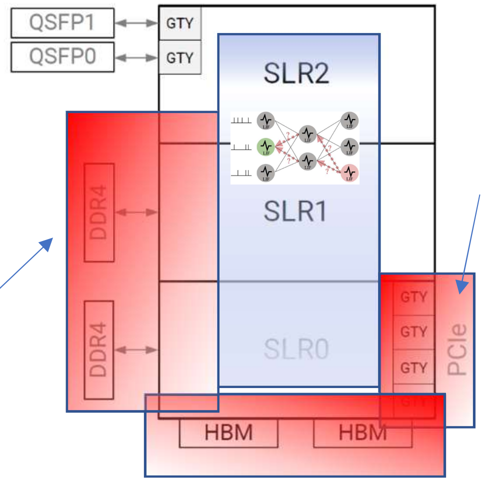
- ✓ Fully customizable
- ✓ QSFP support
- ✓ QDMA
- ✓ RISCV support

- PCIe (QDMA)
- DDR4
- HBM [32 channels]

VS

VITIS

Total capacity = 1.3M LUTs



- PCIe (XDMA)
  - Fixed configurations
  - No QSFP support
  - XDMA
  - No RISCV Support
- DDR4
- HBM [32 channels]

Vits Vs MEEP FPGA Shell

Alveo Platform vs MEEP Shell

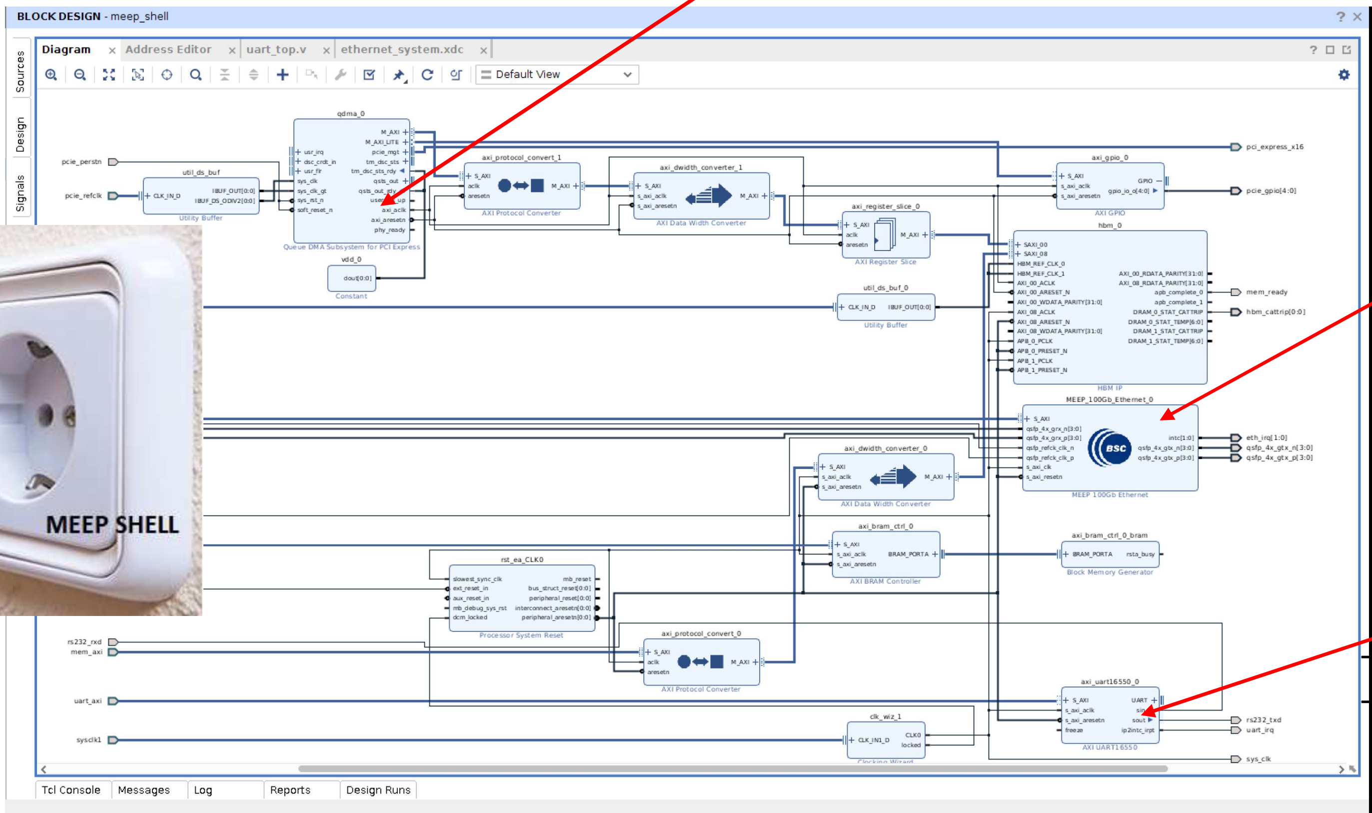
Shell \ Features	Alveo Platform	MEEP Shell
PCIe	XDMA (QDMA planned for 2021)	QDMA
QSFP+	Not Included (planned for 2021)	100Gb Ethernet and/or Aurora
HBM	Hidden inside the platform (eventually, this could be overcome)	Customizable
Portability	Attached to the board	Board independent / Supported
Flexibility	Hard Bitstream (no Vivado project associated)	Fully customizable (associated Vivado Project)
Customization	Different bitstreams (platforms) to adquire new capabilities (or Cloud Alveo Deployment)	Parametrized Vivado Design (enable/disable IPs) (AutoIT)
XCLBINS	Xilinx Support	AutoIT
Satellite Features	Temperature Contol, Clock Shutdown, Clock throttling	None (Lack of Knowledge on how handle the SC)
Partial Reconfiguration	Yes, through XDMA Tandem	No, as QDMA hasn't yet tandem capabilities. Should be through ICAP
Ramp Up	Vitis offers and Out of the Box approach, lot of features working from the start	Must be developed (QDMA features, mainly)
Host	Runtime libraries, Acceleration APIs, P2P connections, all ready to use	Must be developed (QDMA features, mainly)

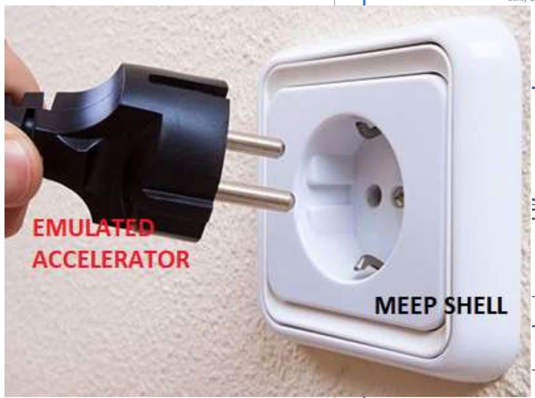
+

Tools to handle NON-PARTIAL reconfiguration !
(PCIe rescan challenge)

MEEP FPGA Shell

PCIe (QDMA, XDMA)





Ethernet IP
- 10GbE
- 100GbE

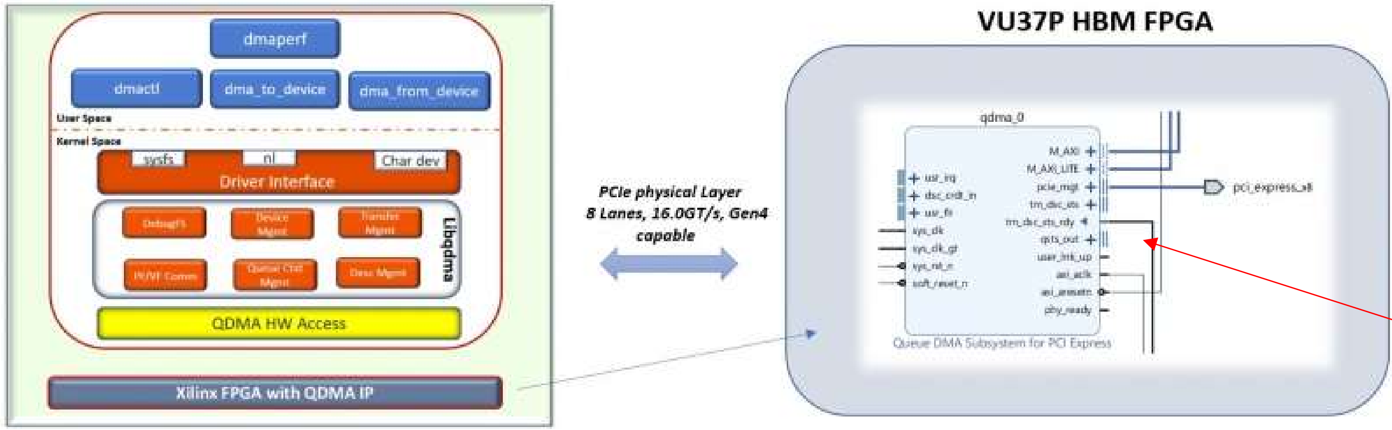
Uart
Xilinx **ns1650**
OpenSource
ns16550a

+ Aurora
- DMA, P2P
+ OpenNIC
… many more

MEEP | MareNostrum Experimental Exascale Platform

MEEP FPGA Shell Interfaces: PCIe

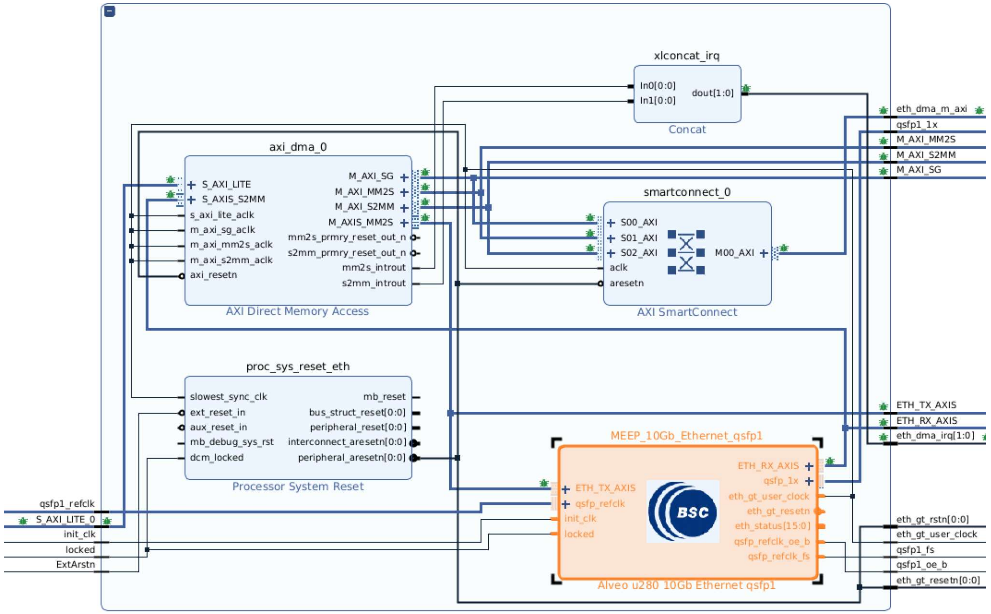
- Working out of the Box: Pcie, any speed
- FPGA logic + PCIe Host drivers
- Tm_dsc_sts_rdy needs to be ‘1’!!



Constant ‘1’

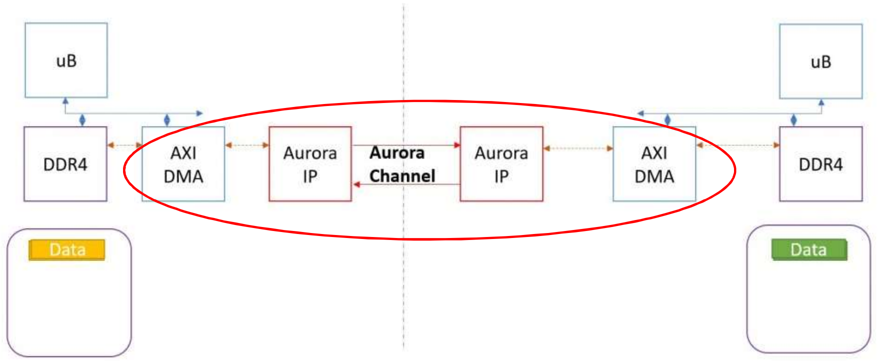
MEEP FPGA Shell Interfaces: Ethernet

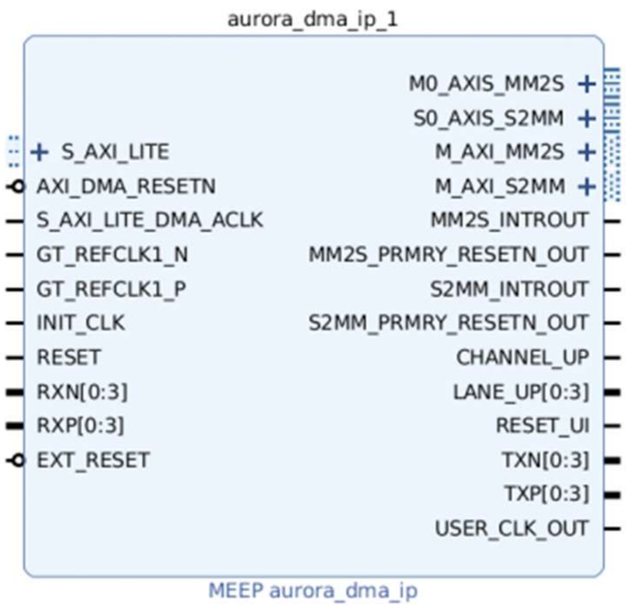
- 100GbE based on CMAC hard macro, DMA
- 10GbE based on Alex Forencich Verilog-ethernet, DMA



MEEP FPGA Shell Interfaces: Ethernet

- Chip2Chip Aurora, DMA
- RAW point 2 point connection



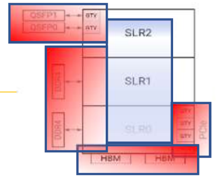


MEEP FPGA Shell

- The shell currently supports:
  - Alveo U55C
  - Alveo U280
  - Alveo U200
  - Alveo U250
- In progress:
  - VCU128

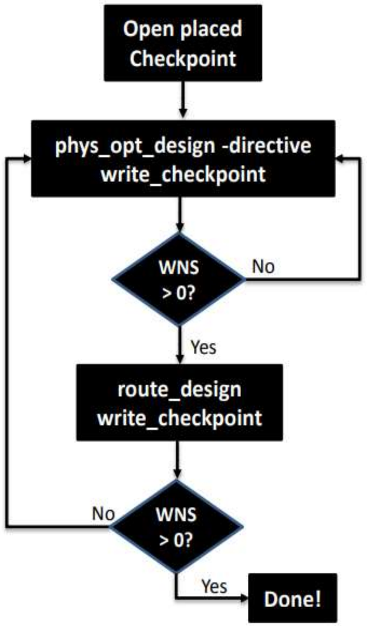
- All these platforms can be the target of a "library" of Emulated Accelerators

MEEP FPGA Shell: Timing Closure



- Custom and scripted (TCL) FPGA flow implementing advanced timing closure techniques, based on the online estimated parameters
  - WNS (Worst Negative Slack)
  - Estimated Congestion
- And other utilities
  - Reject bad timed designs
  - Reject DRC issues (pinout)
  - Unsafe CDC crossings
- Integrated in the CICD Flow

The CICD checks the WNS in order to pass!

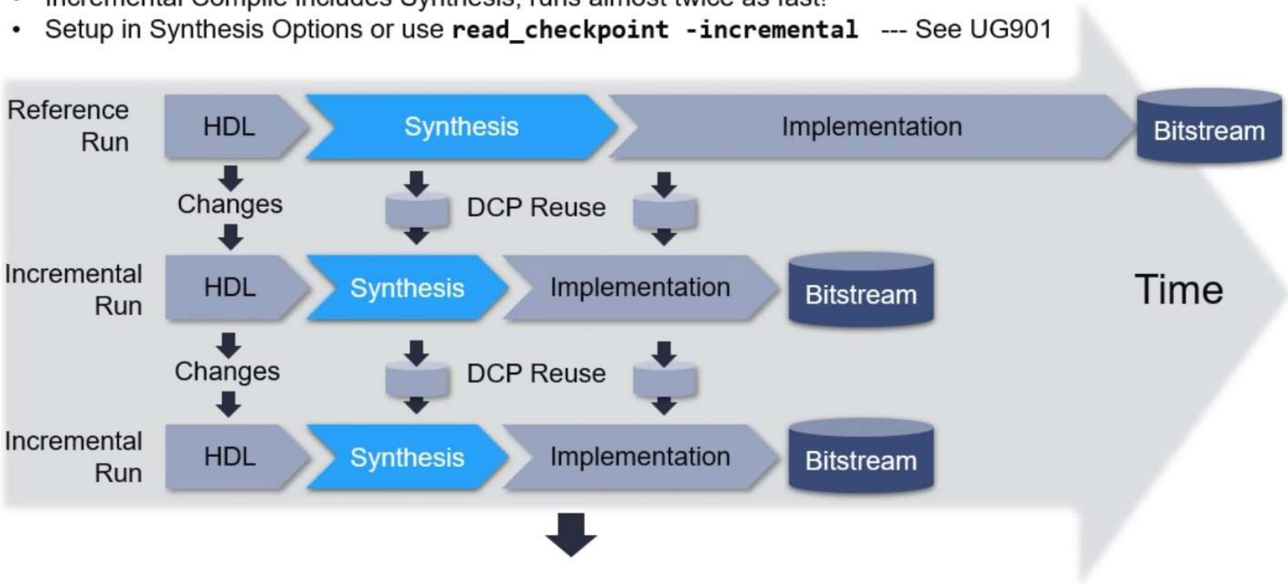


Iterative process

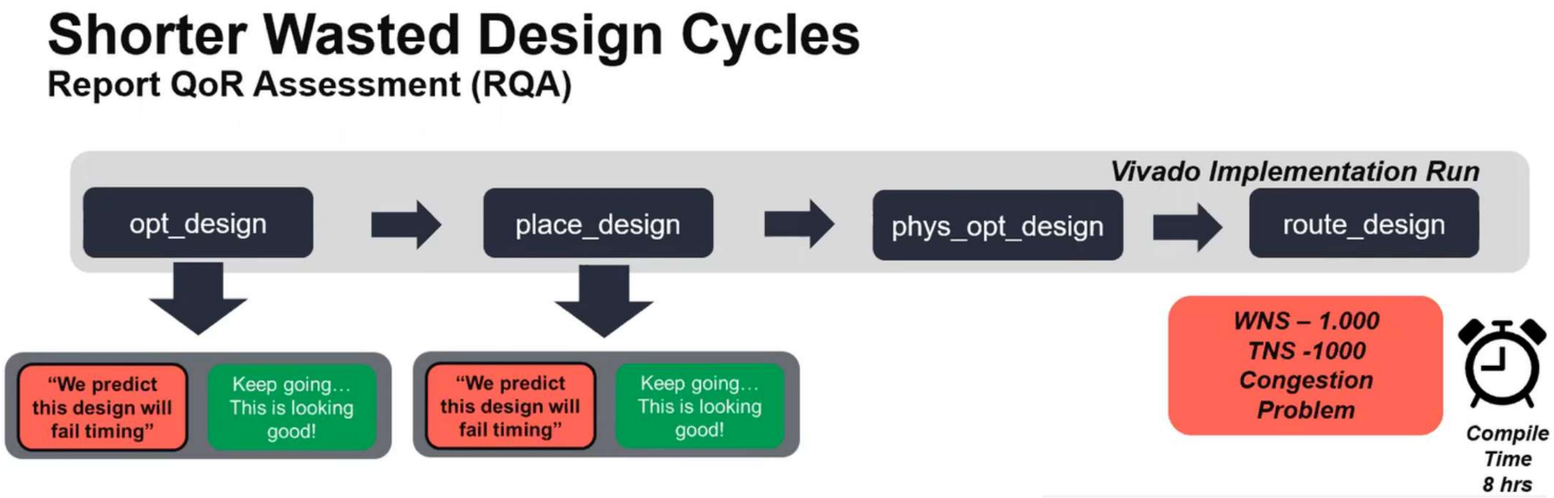
MEEP FPGA Shell: Timing closure

Incremental Synthesis

- Incremental Compile includes Synthesis, runs almost twice as fast!
- Setup in Synthesis Options or use **`read_checkpoint -incremental`** --- See UG901



MEEP FPGA Shell: Timing closure



MEEP FPGA Shell: Timing closure

QoR Assessment and Next Step Guidance

report_qor_assessment (RQA)

Assessment Scores

1. Implementation will fail
2. Timing will fail
3. Timing difficult
4. Timing fair
5. Timing easy to meet

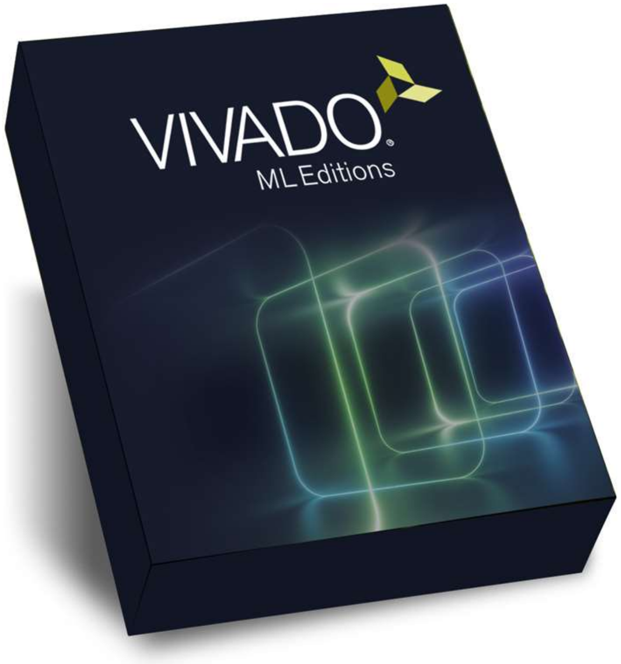
Rule of thumb:
Est. +/- 1 from final score

```
1. Overall Assessment Summary
-----------------------------

+---------------------+--------------------------------------------------------+
| QoR Assessment Score | 2 - Implementation may complete. Timing will not meet  |
+---------------------+--------------------------------------------------------+
| Flow Guidance        | Run report_methodology and fix or waive critical warnings |
+---------------------+--------------------------------------------------------+

2. QoR Assessment Details
-------------------------

+----------------------+--------+---------+--------+--------+--------+
| Name                 | Thresh |  Actual | Used   | Availa | Status |
+----------------------+--------+---------+--------+--------+--------+
| Utilization          |        |         |        |        |        |
|   SLRs - 1           |        |         |        |        |        |
|     Registers        |  50.00 |   51.79 | 447444 | 864000 | REVIEW |
|     Control Sets     |   7.50 |   10.13 |  10936 | 108000 | REVIEW |
| Clocking             |        |         |        |        |        |
|   Setup Skew         | -0.350 |  -0.720 |      - |      - | REVIEW |
|   Hold Skew          |  0.350 |   0.410 |      - |      - | REVIEW |
| Congestion           |        |         |        |        |        |
|   Number of Level 5  |        |         |        |        |        |
|     Global           |      0 |       4 |      - |      - | REVIEW |
|     Short            |      0 |       1 |      - |      - | REVIEW |
| Timing               |        |         |        |        |        |
|   WNS                | -0.100 |  -8.628 |      - |      - | REVIEW |
|   TNS                | -0.100 | -165133 |      - |      - | REVIEW |
|   WHS                | -0.500 |  -0.571 |      - |      - | REVIEW |
|   THS                | -0.500 |-2291.48 |      - |      - | REVIEW |
+----------------------+--------+---------+--------+--------+--------+
```

What's the best thing to do next?

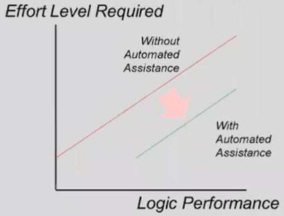
Assessment Details

MEEP FPGA Shell: Timing closure (in progress)



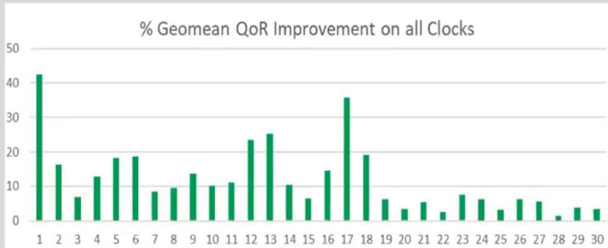
Automated Design Improvements
report_qor_suggestions (RQS)

- Automates the analysis and resolution of issues that lower QoR
  - Simplify timing closure + higher QoR



- Implement => Analyze => Rerun w/ Fix
- QoR focused
  - Focus on Internal FPGA timing
  - V. Limited XDC Constraints
  - No IO timing/HLS/IP/Power Optimization/Runtime
- 75% impl / 25% synth
  - Applies mostly properties and occasionally switches

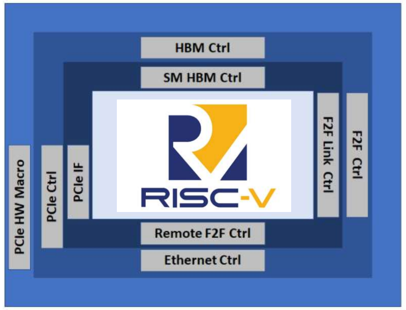
- QoR Gain

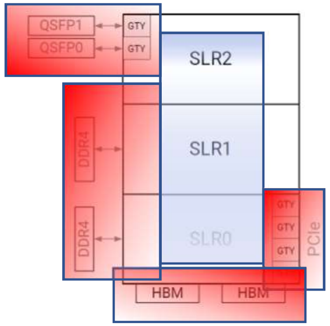


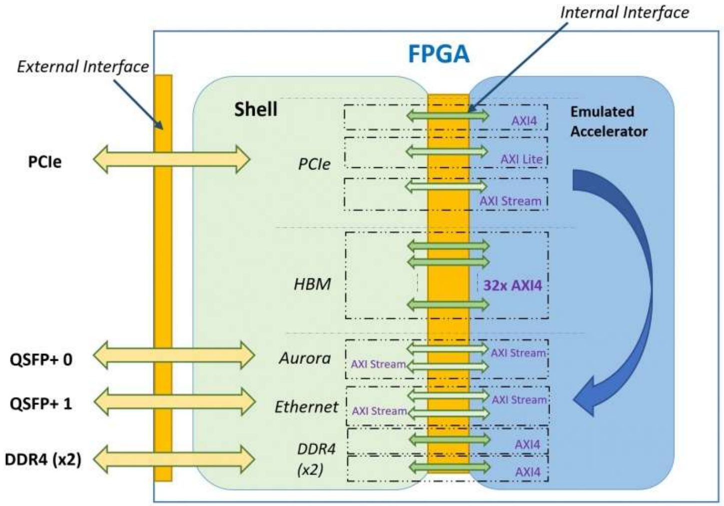
- Best case – 40% gain
  - Clocking Sync CDCs + Safe clock startup + Congestion
- Typical case – 4 - 12%
  - Smaller congestion + placement + timing path issues
- Last mile case < 2%
  - Most difficult – Timing path / placement issues

MEEP FPGA Shell

- RegExp
- TCL
- Bash
- GNU Make







```
MINGW64:/e/git_repo
dmazure@teru ~/git_repo/MEEP_Shell (master)
$ tree
.
├── cicd
│   └── ealib.yml
├── ea_url.txt
├── interfaces
│   ├── aurora.sv
│   ├── axi_intf.sv
│   ├── axilite_intf.sv
│   ├── ddr4.sv
│   ├── ethernet.sv
│   ├── pcie.sv
│   ├── README.md
│   ├── system.sv
│   └── uart.sv
├── ip
│   ├── 100GbEthernet
│   └── axi_brom
├── Makefile
├── misc
│   └── qdma_critical_paths.txt
├── README.md
├── sh
│   ├── accelerator_build.sh
│   ├── api.sh
│   ├── check_log.sh
│   ├── check_reports.sh
│   ├── define_shell.sh
│   ├── extract_part.sh
│   ├── extract_url.sh
│   ├── fpga_test.sh
│   ├── get_artifacts.sh
│   ├── gitsubmodules.sh
│   ├── init_modules.sh
│   ├── init_vivado.sh
│   ├── load_bitstream.sh
│   ├── load_module.sh
│   ├── pcie_script.sh
│   ├── run_bitstream.sh
│   ├── run_implementation.sh
│   ├── run_synthesis.sh
│   ├── run_vivado.sh
│   ├── update_sha.sh
│   └── update_variable.sh
├── shell
│   ├── shell_base.tcl
│   ├── shell_bram.tcl
│   ├── shell_brom.tcl
│   ├── shell_ddr4.tcl
│   ├── shell_ethernet.tcl
│   ├── shell_gpio.tcl
│   ├── shell_hbm.tcl
│   ├── shell_memmap.tcl
│   ├── shell_mmcm.tcl
│   ├── shell_qdma.tcl
│   └── shell_uart.tcl
├── support
│   ├── acme
│   │   ├── ea_url.txt
│   │   └── setup.yml
│   ├── dvino
│   │   ├── ea_url.txt
│   │   └── setup.yml
│   └── openpiton
│       ├── ea_url.txt
│       └── setup.yml
├── tcl
│   ├── define_shell.tcl
│   ├── environment.tcl
│   ├── gen_bitstream.tcl
│   ├── gen_implementation.tcl
│   ├── gen_meep.tcl
│   ├── gen_project.tcl
│   ├── gen_runs.tcl
│   ├── gen_shell.tcl
│   ├── gen_synthesis.tcl
│   ├── gen_top.tcl
│   ├── impl_utils.tcl
│   ├── procedures.tcl
│   ├── QoR.tcl
│   ├── report_place.tcl
│   ├── report_route.tcl
│   ├── report_synth.tcl
│   ├── run_gui.tcl
│   ├── SmartPlace.tcl
│   └── strategies.tcl
└── xdc
    ├── u280
    │   ├── aurora_u280.xdc
    │   ├── ddr4_u280.xdc
    │   ├── ethernet_u280.xdc
    │   ├── hbm_u280.xdc
    │   ├── qdma_u280.xdc
    │   ├── system_ila_u280.xdc
    │   ├── system_timing_u280.xdc
    │   ├── system_u280.xdc
    │   └── uart_u280.xdc
    └── u55c
        ├── aurora_u55c.xdc
        ├── ethernet_u55c.xdc
        ├── hbm_u55c.xdc
        ├── qdma_u55c.xdc
        ├── system_ila_u55c.xdc
        ├── system_timing_u55c.xdc
        ├── system_u55c.xdc
        └── uart_u55c.xdc

16 directories, 88 files
```

MEEP FPGA Shell: Project regeneration

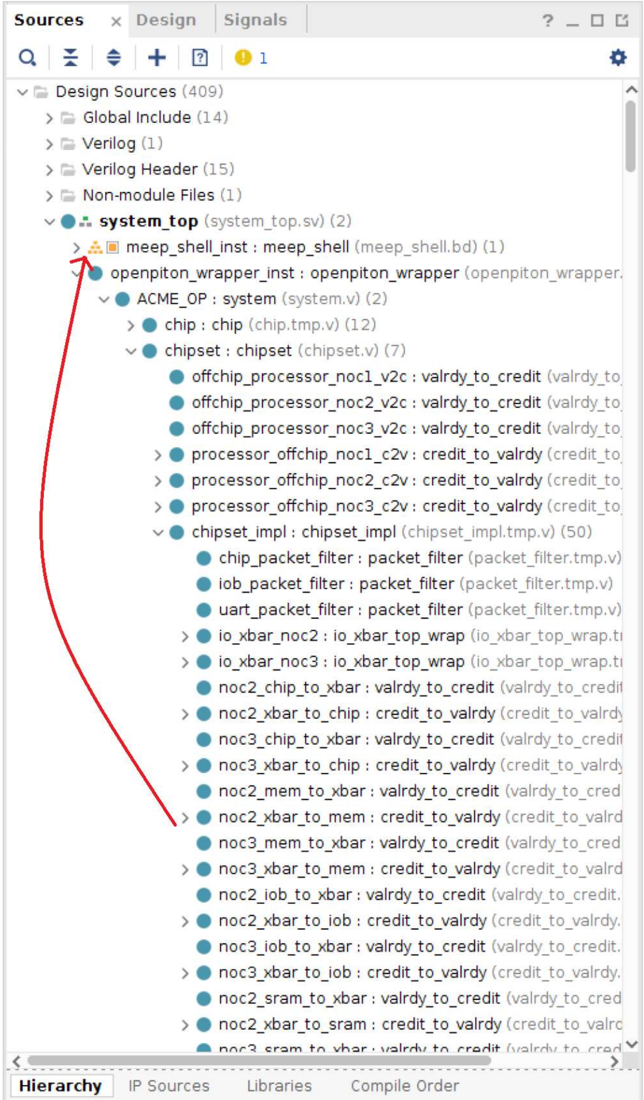
- Git based, Only commit source files (no archive, copy, zip or other bad-practices to move projects)
- Vivado version independent! It will work with future versions, as long as a little table is updated
    - What does really checks Vivado when checking versions?

✓ Board compatibility,
✓ Vivado version independency

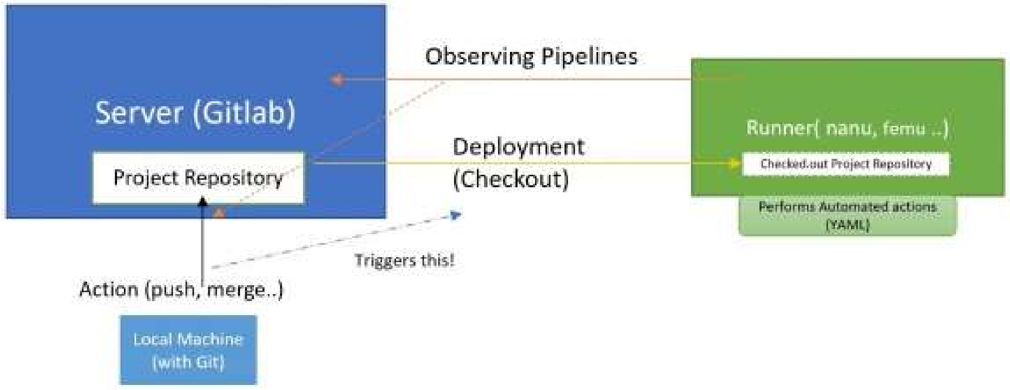

```
MINGW64:/
ache License version 2.0.
# You may obtain a copy of the License at
#
#     http://www.solderpad.org/licenses/SHL-2.1
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# Author: Daniel J.Mazure, BSC-CNS
# Date: 22.02.2022
# Description:


switch $g_vivado_version {
        2020.1 {
                #body
                set meep_util_ds_buf "xilinx.com:ip:util_ds_buf:2.1"
        }
        2021.2 {
                set meep_util_ds_buf "xilinx.com:ip:util_ds_buf:2.2"
        }
}

switch $g_board_part {
        u280 {
                set HBM_AXI_LABEL ""
                set HBMDensity "8GB"
        }
        u55c {
                set HBM_AXI_LABEL "_8HI"
                set HBMDensity "16GB"
        }
}

set MEEPUart "meep-project.eu:MEEP:MEEP_PULP_UART:1.0"
set XilinxUart "xilinx.com:ip:axi_uart16550:2.0"
```

How to do the EA compatible?

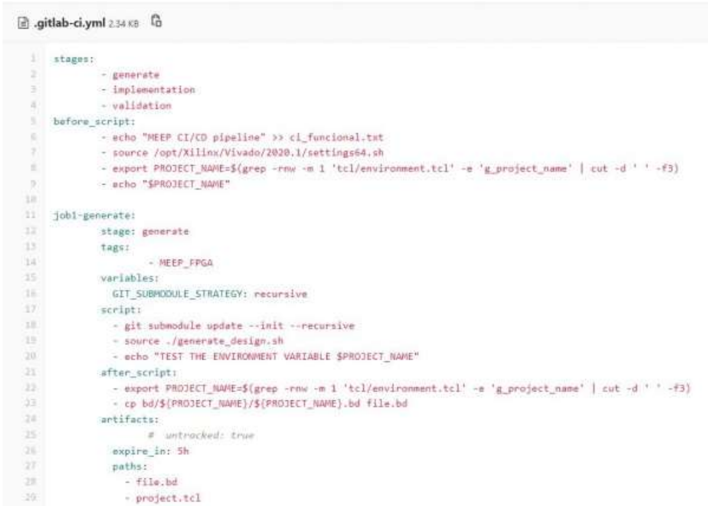
- Create a “**meep_shell**” folder in your repository
- Create tcl **file list**
- Create the “**accelerator_dev.csv**” file
- … That’s pretty much it ! ☺



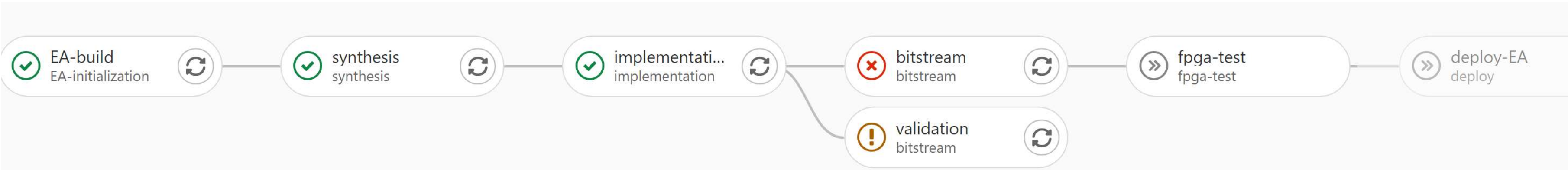
MEEP FPGA Shell: CICD



Yaml file enables the CICD


.gitlab-ci.yml 2.34 KB

```yaml
stages:
        - generate
        - implementation
        - validation
before_script:
        - echo "MEEP CI/CD pipeline" >> ci_funcional.txt
        - source /opt/Xilinx/Vivado/2020.1/settings64.sh
        - export PROJECT_NAME=$(grep -rnw -m 1 'tcl/environment.tcl' -e 'g_project_name' | cut -d ' ' -f3)
        - echo "$PROJECT_NAME"

job1-generate:
        stage: generate
        tags:
                - MEEP_FPGA
        variables:
          GIT_SUBMODULE_STRATEGY: recursive
        script:
          - git submodule update --init --recursive
          - source ./generate_design.sh
          - echo "TEST THE ENVIRONMENT VARIABLE $PROJECT_NAME"
        after_script:
          - export PROJECT_NAME=$(grep -rnw -m 1 'tcl/environment.tcl' -e 'g_project_name' | cut -d ' ' -f3)
          - cp bd/${PROJECT_NAME}/${PROJECT_NAME}.bd file.bd
        artifacts:
                #  untracked: true
          expire_in: 5h
          paths:
            - file.bd
            - project.tcl
```



MEEP MareNostrum Experimental Exascale Platform

MEEP FPGA Shell

Quick Live Demo

MEEP | MareNostrum Experimental Exascale Platform

OmpSs@FPGA programming

Xavier Martorell, Daniel Jiménez-Mazure

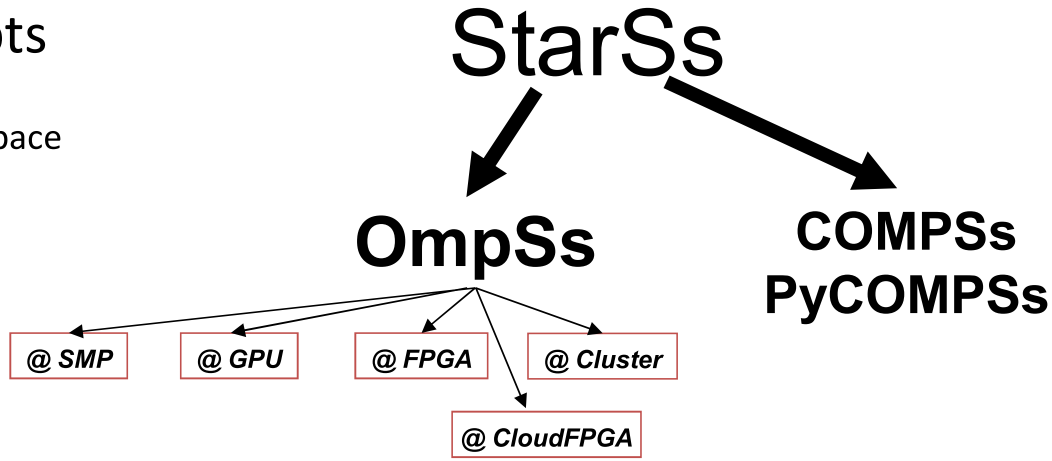
September 12th, 2022

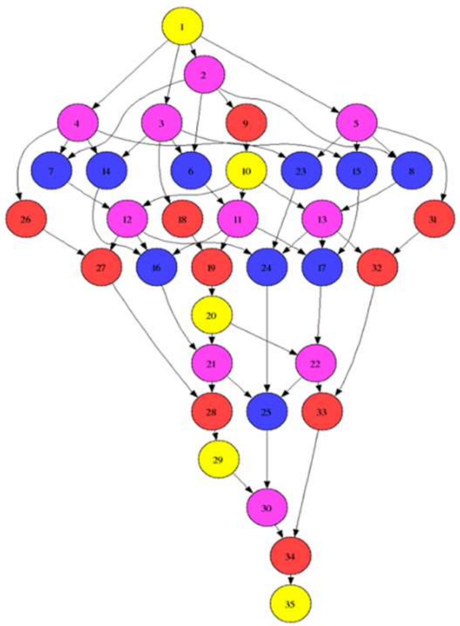
Outline

- Task-based programming
- OmpSs programming model
- OmpSs@FPGA ecosystem
- Mercurium & AIT
- Examples and evaluation
- Exploiting OpenCL kernels
- Conclusions

Task-based programming

- StarSs family key concepts
  - Sequential task-based program
  - View of a single address/name space
  - Execution in parallel: automatic
  - runtime computation of dependencies
- Productivity and portability

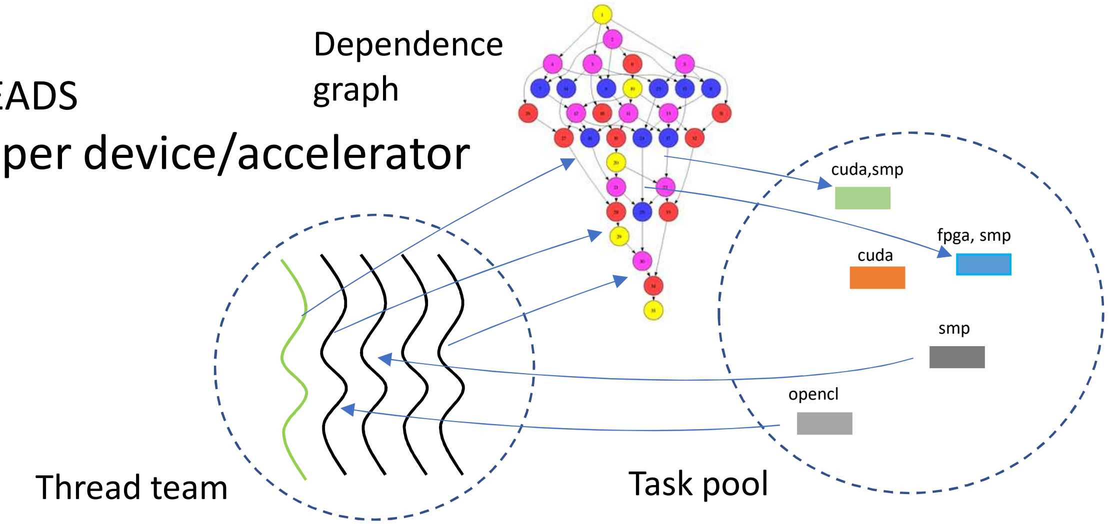




- OmpSs is used for prototyping extensions to OpenMP
  - Tasking
  - data-dependences
  - Heterogeneity
  - Multiple address spaces
  - …
- Mercurium compiler
- Nanos runtime system

Execution Model

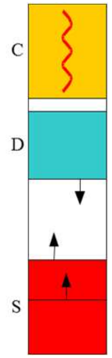
- Thread-pool model
  - OpenMP parallel not supported (it was ignored in OmpSs-1)
- All threads created on startup
  - One of them starts executing main (on SMP device)
  - P-1 workers execute SMP tasks
    - $ taskset –c 0-$((P-1))
    - Equivalent to the OpenMP OMP_NUM_THREADS
  - One representative (OpenCL/CUDA/FPGA) per device/accelerator



- All get work from a task pool
  - And can generate new work
  - Work is labeled with actual “targets”

Memory Model

- From the point of view of the programmer a single naming space exists
  - The standard sequential/shared memory address space



- From the point of view of the runtime and target platform, different possible scenarios
  - Pure SMP:
    - Single address space
  - Distributed/heterogeneous (cluster, GPUs, ...):
    - Multiple address spaces exist
      - Versions of same data may exist in multiple of these address spaces
    - Data consistency ensured by the implementation

Programming with OmpSs@FPGA

- Task offload to function
- Annotate function prototype with OmpSs
- Target the FPGA device
- Indicate data directionality hints (in, out, inout)
- Invoke task from host code
- Taskwait when needed

scale.fpga.h

```
const int SIZE=1024;
#pragma omp target device(fpga) copy_deps num_instances(1)
#pragma omp task in([SIZE]c,a) out([SIZE]b)
void scale_task(double *b, double *c, double *a);
```

main.cpp

```
#include "scale.fpga.h"

void scale_task(double *b, double *c, double *a) {
#pragma HLS ARRAY_PARTITION variable=c complete dim=1
#pragma HLS ARRAY_PARTITION variable=b complete dim=1
   double alpha=*a;
   for (int j=0; j < SIZE; j++) b[j] = alpha*c[j];
}

int main (int argc, char *argv[])
{
   for (. . . ) {
     scale_task(B, C, &alpha);
   }. . .
   #pragma omp taskwait
}
```

Programming with OmpSs@FPGA

- Directives work with “const” data, not #define
- Constant/global variable and task definitions may need to be in *.fpga.h file
- There should be at least one call the every task
- Arguments can be structured (not fully tested)

scale.fpga.h

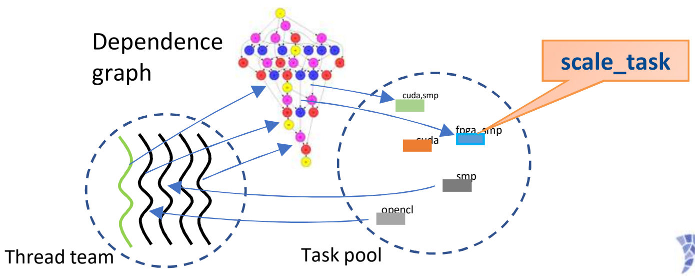
```
const int SIZE=1024;
#pragma omp target device(fpga) copy_deps num_instances(1)
#pragma omp task in([SIZE]c,a) out([SIZE]b)
void scale_task(double *b, double *c, double *a);
```

main.cpp

```
#include "scale.fpga.h"

void scale_task(double *b, double *c, double *a) {
#pragma HLS ARRAY_PARTITION variable=c complete dim=1
#pragma HLS ARRAY_PARTITION variable=b complete dim=1
   double alpha=*a;
   for (int j=0; j < SIZE; j++) b[j] = alpha*c[j];
}

int main (int argc, char *argv[])
{
   for (. . . ) {
     scale_task(B, C, &alpha);
   }. . .
   #pragma omp taskwait
}
```

Programming with OmpSs@FPGA

- Pool of threads take tasks to be executed
- Runtime decides
  - When a thread executes a task
  - Where to execute that task, based on user annotations
  - Which data copies need to be done and where



scale.fpga.h

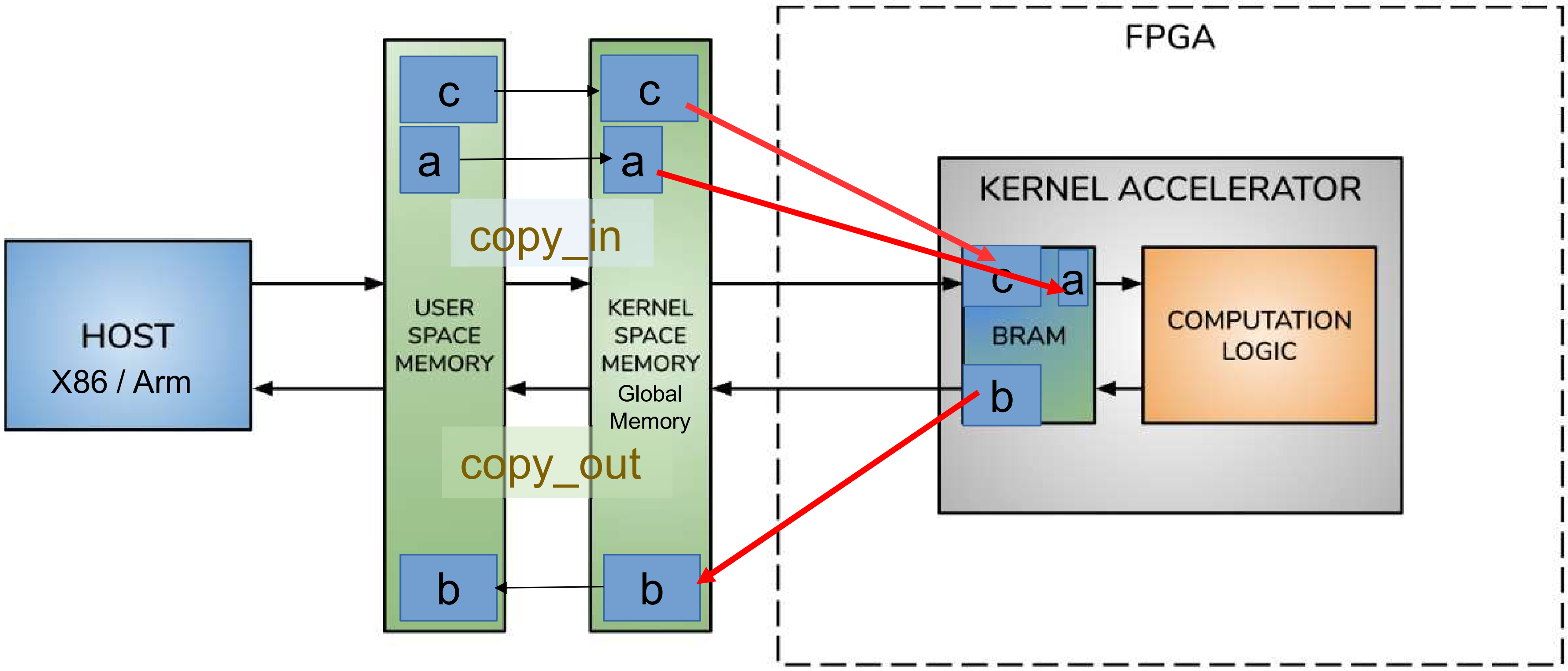
```
const int SIZE=1024;
#pragma omp target device(fpga) copy_deps num_instances(1)
#pragma omp task in([SIZE]c,a) out([SIZE]b)
void scale_task(double *b, double *c, double *a);
```

main.cpp

```
#include "scale.fpga.h"

void scale_task(double *b, double *c, double *a) {
#pragma HLS ARRAY_PARTITION variable=c complete dim=1
#pragma HLS ARRAY_PARTITION variable=b complete dim=1
   double alpha=*a;
   for (int j=0; j < SIZE; j++) b[j] = alpha*c[j];
}

int main (int argc, char *argv[])
{
   for (. . . ) {
     scale_task(B, C, &alpha);
   }. . .
   #pragma omp taskwait
}
```

MEEP MareNostrum Experimental Exascale Platform

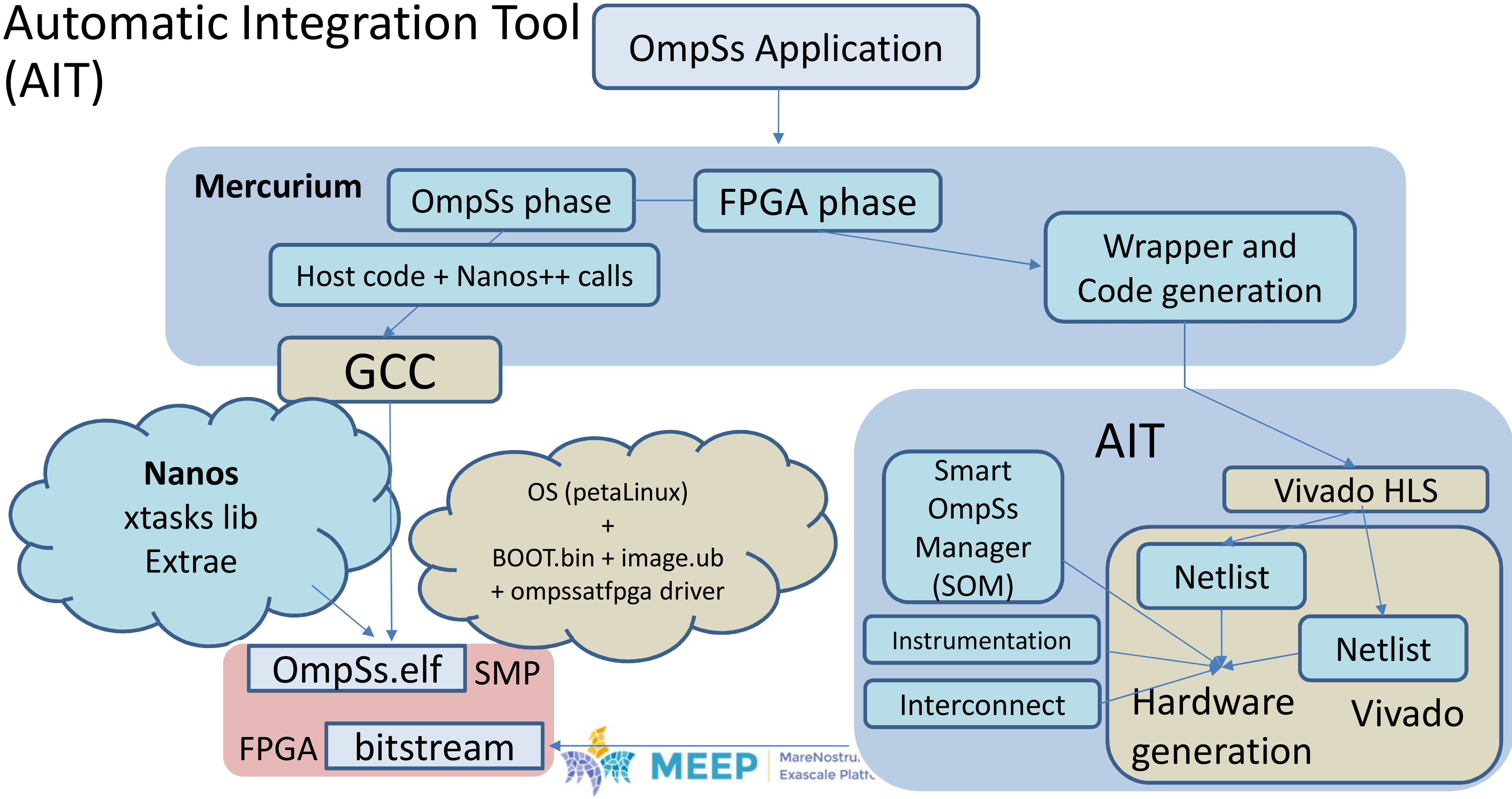
Execution and data transfers



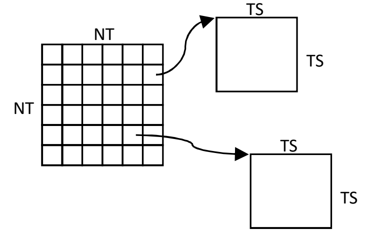
OmpSs@FPGA ecosystem

OmpSs@FPGA integration with vendor tools

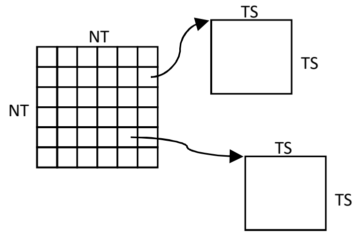
- Automatic Integration Tool (AIT)



A complete example: Matrix multiply




```
void matrix_multiply(T a[BS][BS],T b[BS][BS],T c[BS][BS]);
…
for (i_b=0; i_b<NB_I; i_b++)
  for (j_b=0; j_b<NB_J; j_b++)
    for (k_b=0; k_b<NB_K; k_b++)
       matrix_multiply(AA[i_b][k_b], BB[k_b][j_b], CC[i_b][j_b]);
…
```



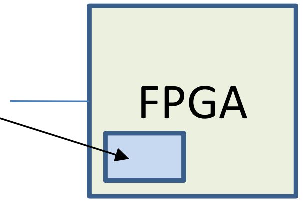

```
#pragma omp task in([BS]a,[BS]b) inout([BS]c)
void matrix_multiply(T a[BS][BS],T b[BS][BS],T c[BS][BS]);
…
for (i_b=0; i_b<NB_I; i_b++)
  for (j_b=0; j_b<NB_J; j_b++)
    for (k_b=0; k_b<NB_K; k_b++)
       matrix_multiply(AA[i_b][k_b], BB[k_b][j_b], CC[i_b][j_b]);
…
#pragma omp taskwait
// Or
// other tasks depending on input output c of matrix multiply task
```

Mercurium compiler **with AIT tool**

– Allows to express number of instances per task

– Triggers the bitstream generation automatically (stub function generated)

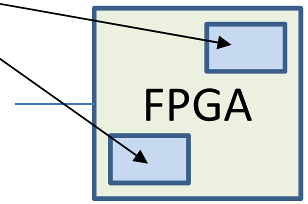
```
#pragma omp target device(fpga) copy_deps num_instances(1)
#pragma omp task in([BS]a,[BS]b) inout([BS]c)
void matrix_multiply(T a[BS][BS],T b[BS][BS],T c[BS][BS]);
…
for (i_b=0; i_b<NB_I; i_b++)
  for (j_b=0; j_b<NB_J; j_b++)
    for (k_b=0; k_b<NB_K; k_b++)
       matrix_multiply(AA[i_b][k_b], BB[k_b][j_b], CC[i_b][j_b]);
…
#pragma omp taskwait
// Or
// other tasks depending on input output c of matrix multiply task
```



Mercurium compiler **with AIT tool**

- Few extensions (num_instances)
- Triggers the bitstream generation automatically (Stub function generated)

```
#pragma omp target device(fpga) copy_deps num_instances(2)
#pragma omp task in([BS]a,[BS]b) inout([BS]c)
void matrix_multiply(T a[BS][BS],T b[BS][BS],T c[BS][BS]);
…
for (i_b=0; i_b<NB_I; i_b++)
  for (j_b=0; j_b<NB_J; j_b++)
    for (k_b=0; k_b<NB_K; k_b++)
       matrix_multiply(AA[i_b][k_b], BB[k_b][j_b], CC[i_b][j_b]);
…
#pragma omp taskwait
// Or
// other tasks depending on input output c of matrix multiply task
```

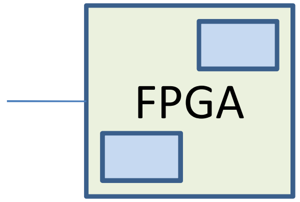


Mercurium compiler **with AIT tool**

– Few extensions (num_instances)

– Triggers the bitstream generation automatically (Stub function generated)

```
#pragma omp target device(fpga) copy_deps num_instances(2)
#pragma omp task in([BS]a,[BS]b) inout([BS]c)
void matrix_multiply(T a[BS][BS],T b[BS][BS],T c[BS][BS]);
```



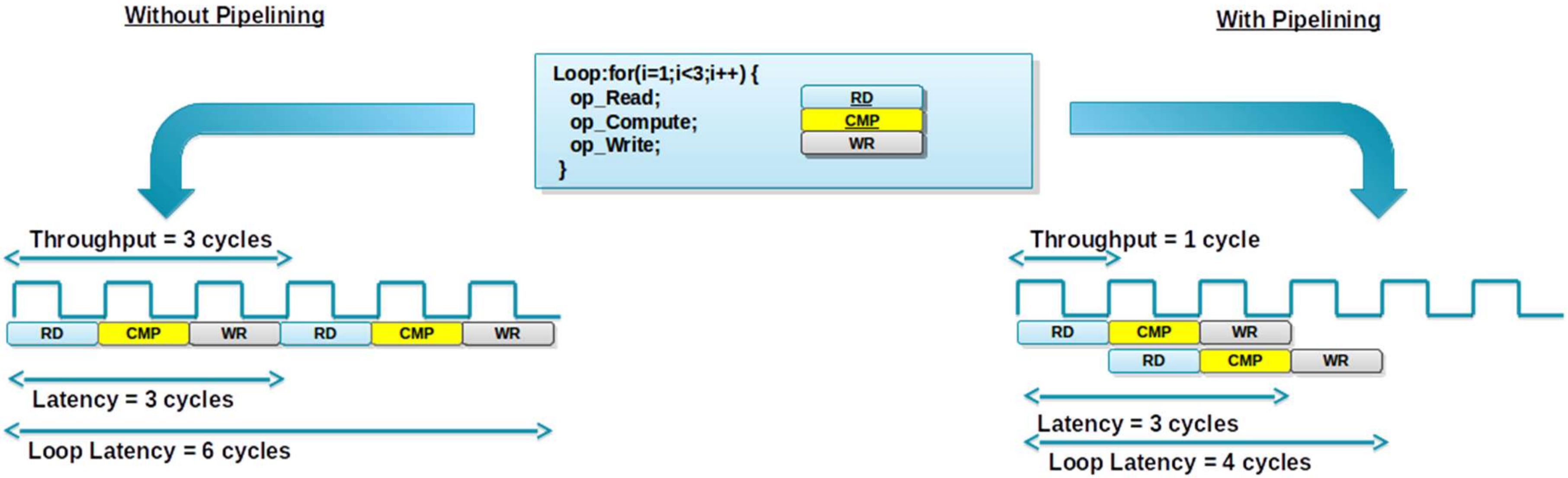

```
void matrix_multiply(float a[BS][BS], float b[BS][BS],float c[BS][BS])
{
#pragma HLS inline
  int const FACTOR = BS/2;
#pragma HLS array_partition variable=a block factor=FACTOR dim=2
#pragma HLS array_partition variable=b block factor=FACTOR dim=1
  // matrix multiplication of a A*B matrix
  for (int ia = 0; ia < BS; ++ia)
    for (int ib = 0; ib < BS; ++ib) {
#pragma HLS PIPELINE II=1
      float sum = 0;
      for (int id = 0; id < BS; ++id)
        sum += a[ia][id] * b[id][ib];
      c[ia][ib] += sum;
    }
}
```

Distribute data across Block RAMs

Pipelines the id loop with initiation interval 1 cycle, taking care of the reduction to sum

Pipelining

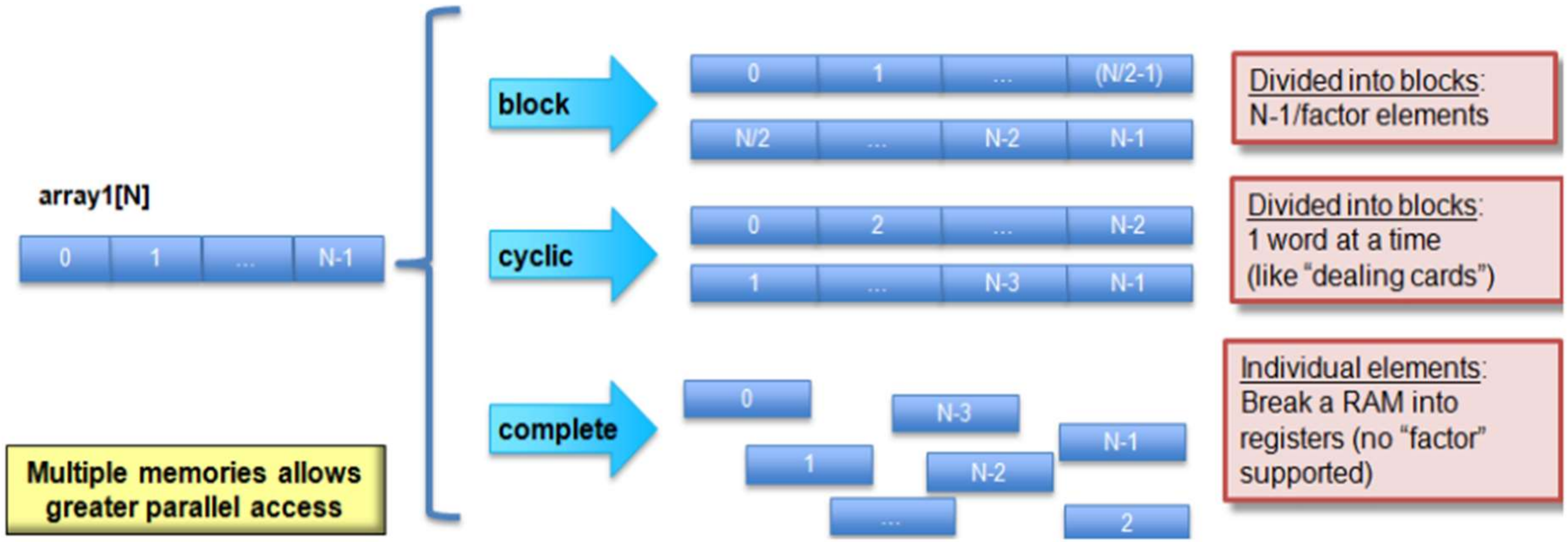
- Several instances of the hardware can operate in pipeline
- Initiation Interval: Number of cycles before you can start a new instance of the hardware



Initiation Interval (II) = 1

Partitioning arrays over Block RAMs

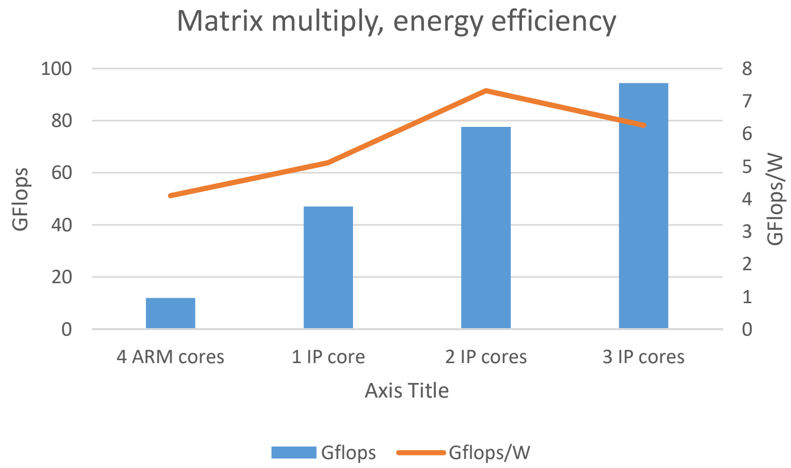
- Provides the ability to parallel compute
  - #pragma HLS ARRAY_PARTITION variable=name dim=X type factor=FACTOR

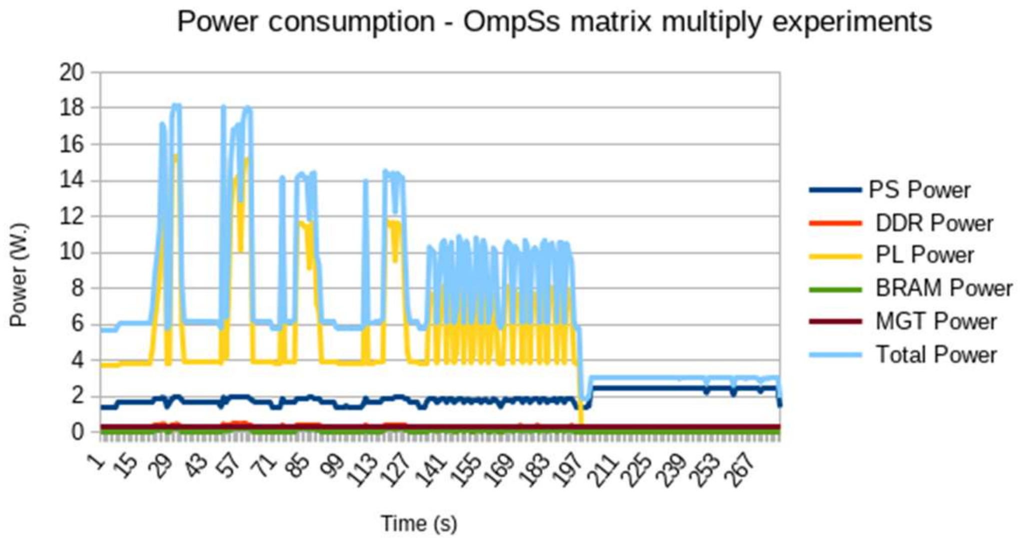


Evaluation:
Matrix multiply

Matrix multiplication

- Xilinx ZCU 102 development kit (equiv. to AXIOM board)
  - 4 Arm Cortex A-53 cores, U+ FPGA 9EG
  - Matrix size 2048x2048
  - Block size 128x128, 1 to 3 instances
  - 250 MHz



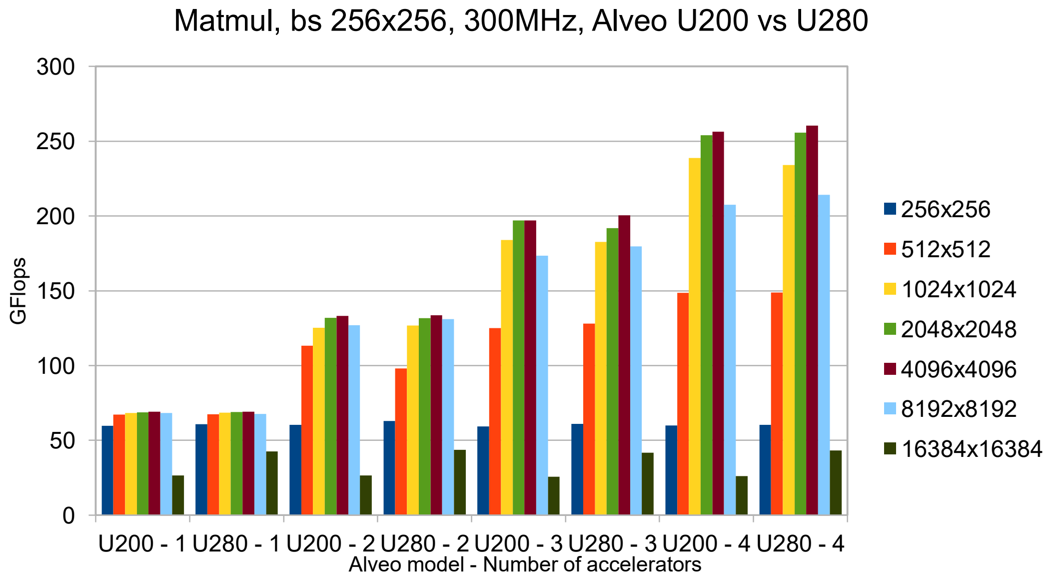


- Best performance
  - 3 IP cores
- Best energy-efficiency
  - 2 IP cores

MareNostrum Experimental Exascale Platform

Matrix multiplication

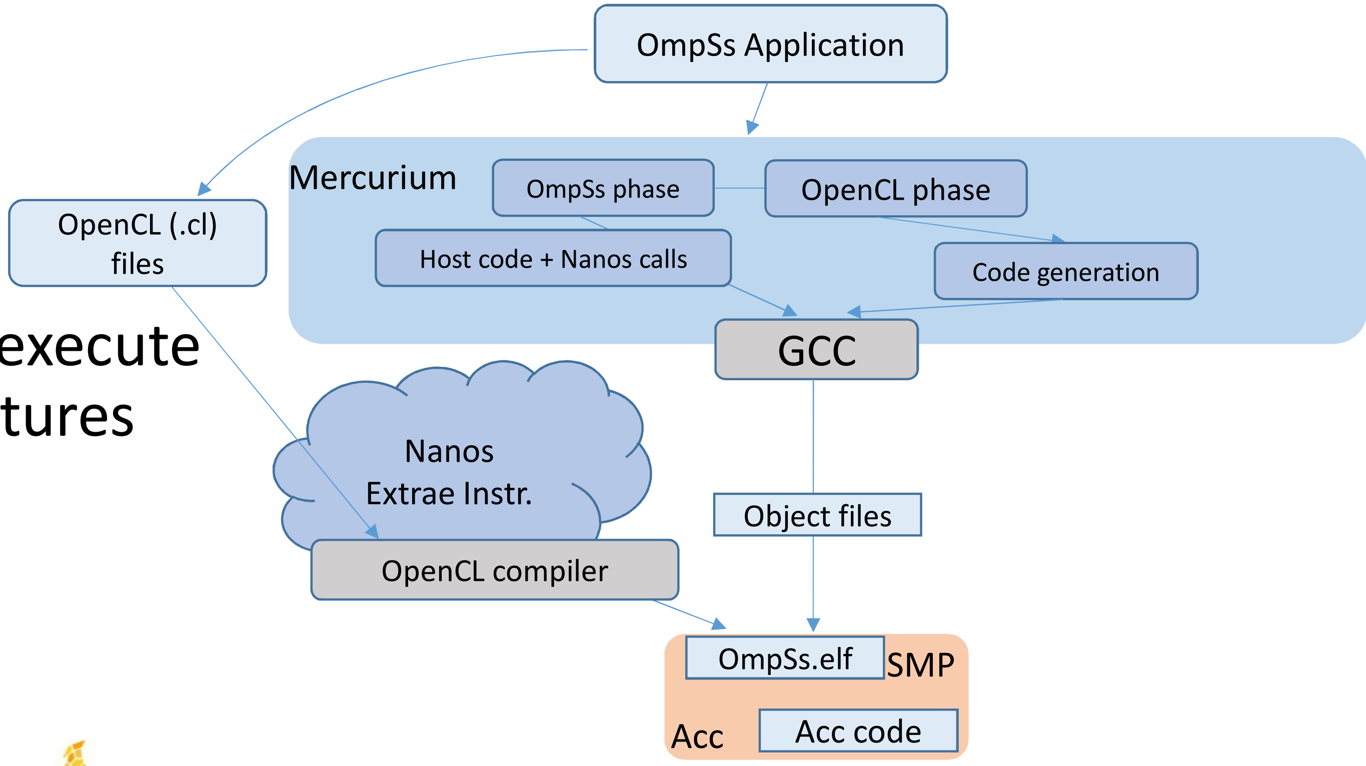
- Alveo U200 and U280 FPGA boards
- Matrix size: 2048x2048
- Block size: 256x256, 1 to 4 accelerators, 300MHz



Exploiting OpenCL kernels

Supporting OpenCL: OmpSs@OpenCL

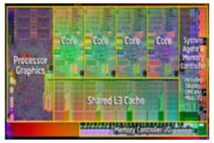
- OmpSs support for CUDA and OpenCL kernels on
  - AMD GPUs
  - **Intel FPGAs**
- Using existing kernels in CUDA/OpenCL
- **"Implements"** allows to execute kernels on the 3 architectures



Heterogeneous programming

- The "implements" approach
  - Kernel provided in OpenCL
  - Kernel compiled "offline"
  - Data transfers automatically generated by OmpSs

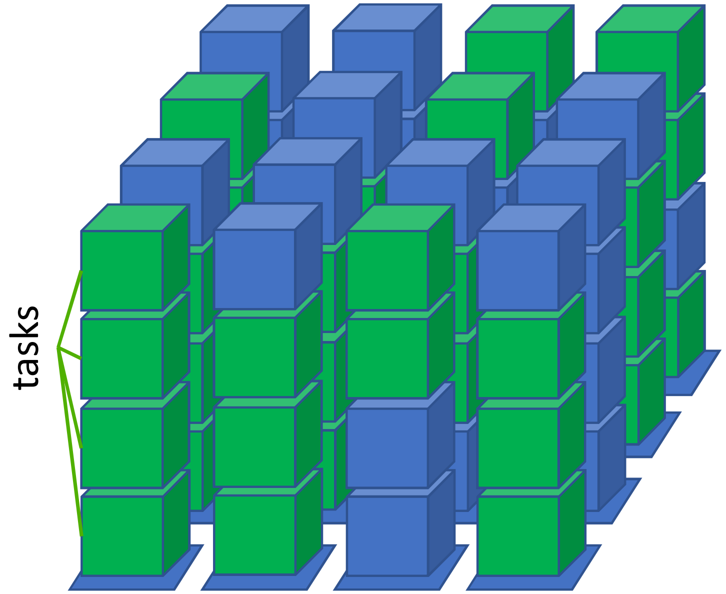
```
#pragma omp target device(opencl) ndrange(2,NB,NB,16,16) \
                implements(matrix_multiply)
#pragma omp task in(a,b) inout(c)
__kernel void matrix_multiply_opencl(float a[BS][BS],
                                     float b[BS][BS],
                                     float c[BS][BS]);
```



SMP

GPGPU
(cuda)

FPGA
(opencl)

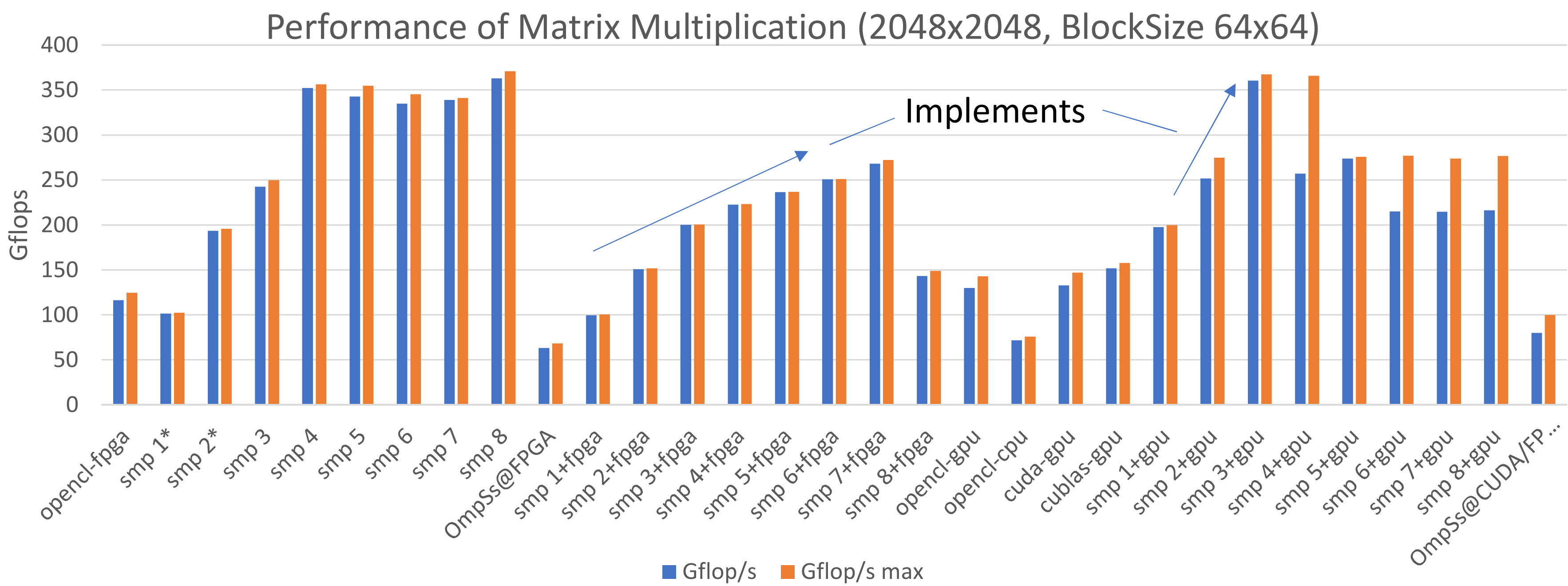


MareNostrum Experimental Exascale Platform

Runtimes– OmpSs@CUDA & OpenCL

- Evaluation on Intel CPUs, Nvidia GPU and Intel Arria 10 FPGA

Intel 4-core i7-7700 @3.6GHz with 2 ht/core
Nvidia GeForce GTX TITAN X
Intel Arria 10 de5net_a7 FPGA

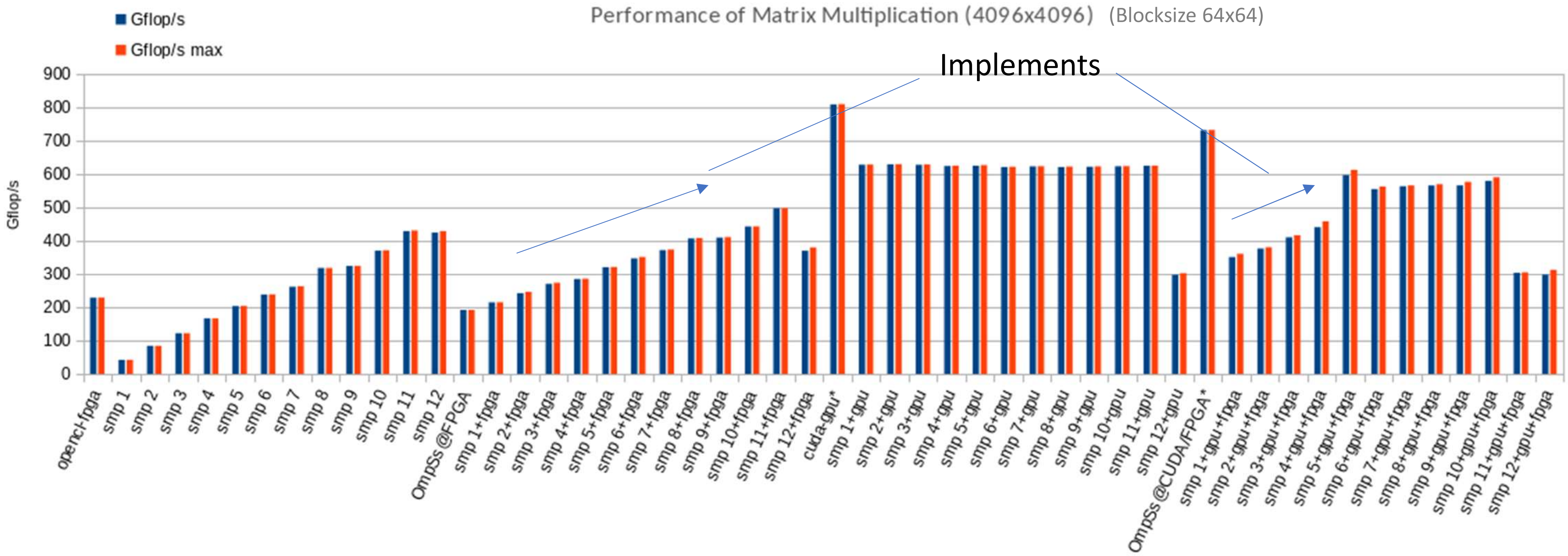


MEEP | MareNostrum Experimental Exascale Platform

Runtimes– OmpSs@CUDA & OpenCL

- Evaluation on Intel CPUs, Nvidia GPU and Intel Stratix 10 FPGA
  - Lesson learned: devices similar in performance contribute better to overall performance

6-core Intel Xeon® Bronze 3204 @1.9GHz, no ht
Nvidia GeForce RTX 2070 SUPER
Intel Stratix 10 FPGA



OmpSs@FPGA demo

Live demo

- Environment
  - Intel x86 with Alveo U200 FPGA
  - OmpSs@FPGA environment
  - N-body application

Conclusions

Concluding...

- Programming FPGAs is tricky, due to the large number of concepts that one needs to take care of...
  - Configuration of the hardware
  - Connectivity
  - Data transfers
  - Control
- Vendor tools are effective, but offer low productivity
- We are trying to get the programming perspective at a higher level
  - OmpSs@FPGA is based on programmers hints to get good performance on these devices. We expect to have OpenMP "target" onboard also ☺