Accelerating AI Workshop 2023 – Challenges and Opportunities in Cloud and Edge Computing

Polara: a RISCV multicore vector processor

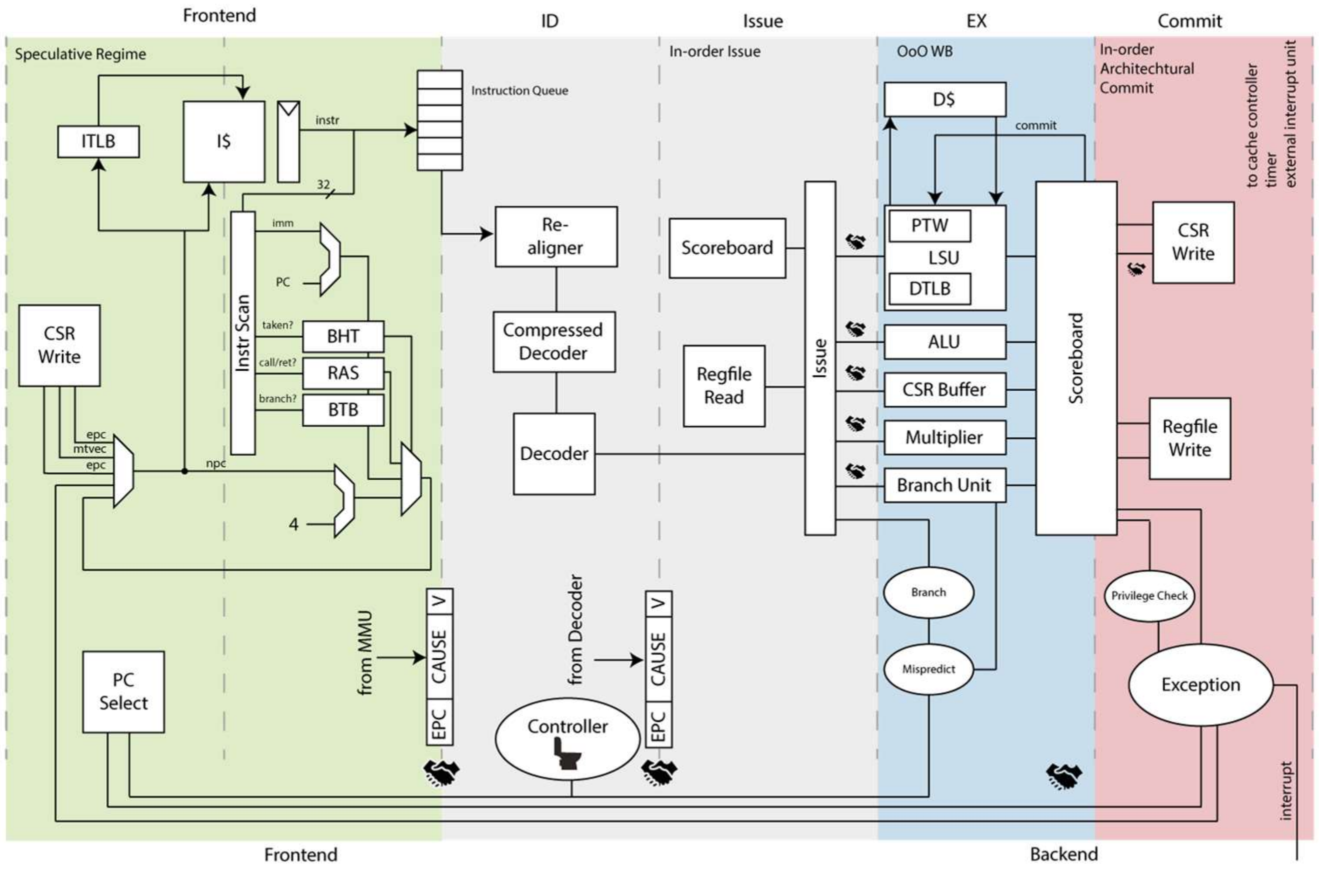
Nizar El Zarif, Mohammad Hossein Askari Hemmat, Theo Dupuis,Yoan Fournier
Elisabeth Humblet, Francois Leduc-Primeau, Jean-Pierre David, Yvon Savaria

May -4 -2023

outline

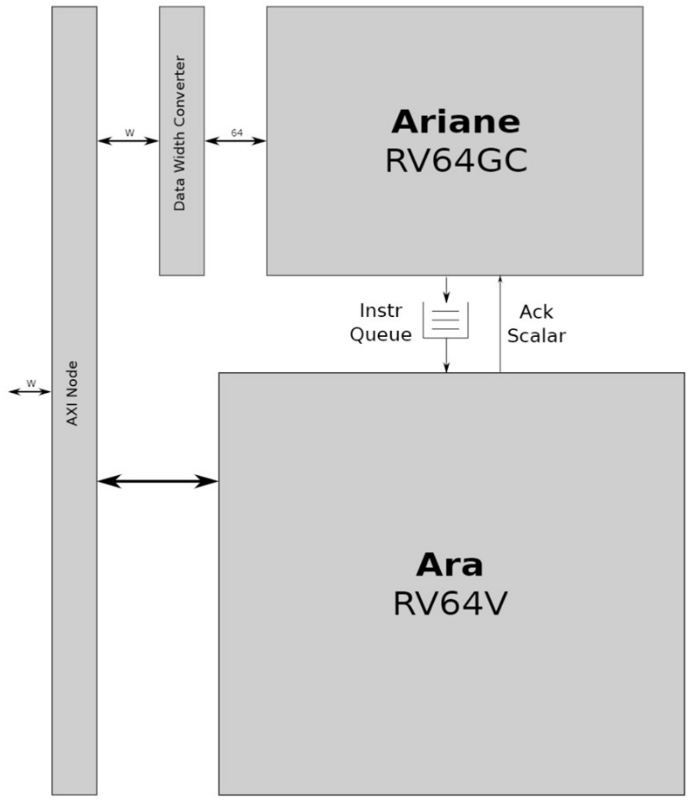
CVA6 processor (Ariane)

- Ariane is the codename for the risc V processor using RV64 instruction set.
- CVA6 is a 6 stage in order, single issue wide processor with support to M,A and C extension.
- https://github.com/openhwgroup/cva6

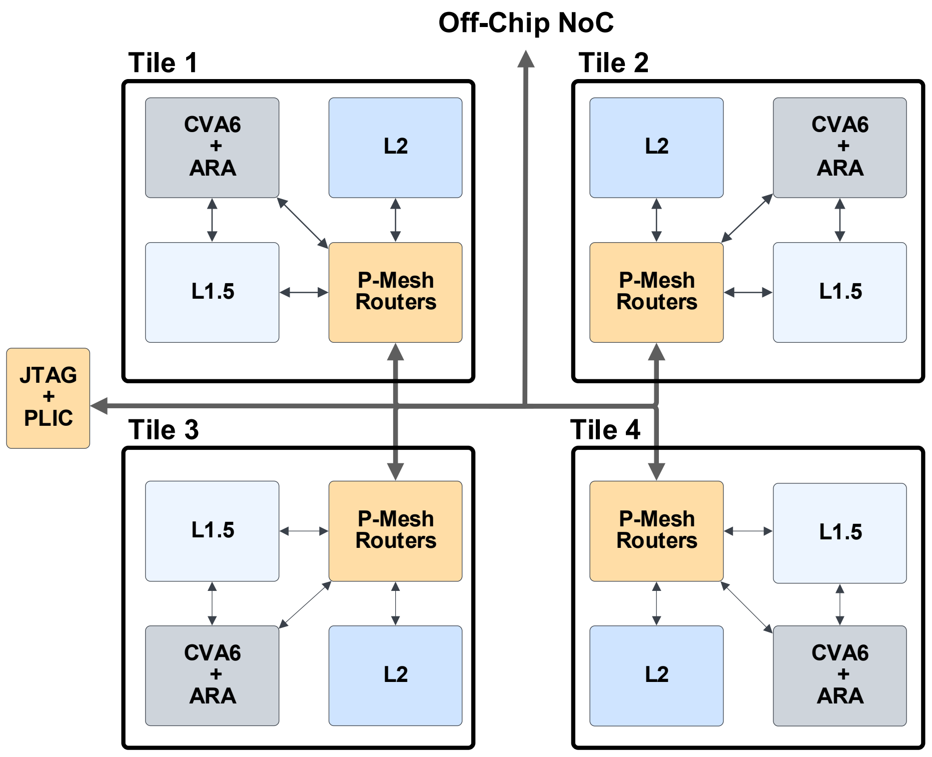


Ara

- Ara (CV-VEC) is the codename for the vector processor.
- Vectors and array processor can use SIMD programming model to improve computation throughput.
- Example of Array processors is x86 that supports AVX instructions, or ARM with neon.
- The vector in an array processor are fixed in size , as opposed to in vector processors which can use dynamic length up to 4096-bit operations.
- Ara can be configured with different numbers of lanes allowing more parallelism based on how wide the design is.



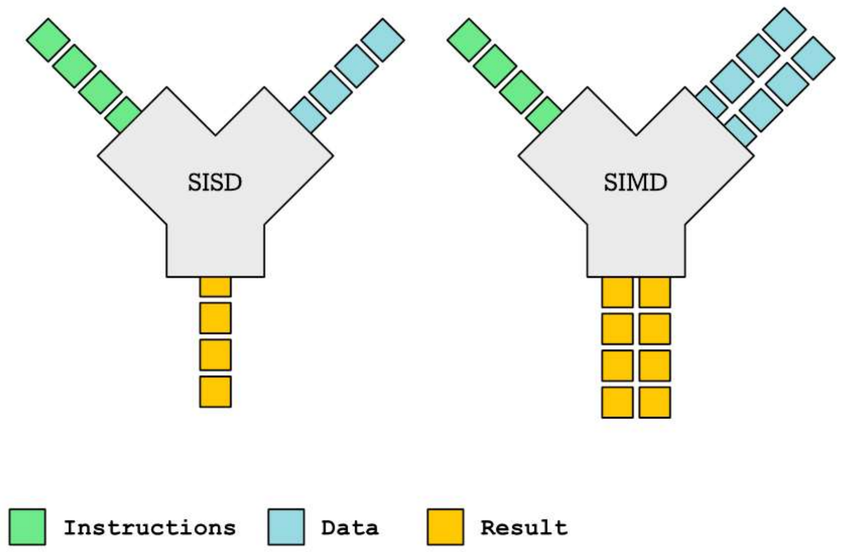
What is Polara ?



- Polara is aiming to create a **multi-core version of the ARA** vectorial processor using the **OpenPiton** framework
- The ASIC will posses **4** instances or **ARA/CVVEC** with **4 lanes** each

SIMD recap

- SIMD allows multiple operation with a single instruction.
- This type of parallelism is called data-level parallelism (DLP).
- DLP allows for better hardware utilization
- Vector processor and array processor are both different implementation of SIMD computation model



Deep learning use

- AI workload can be divided into training and inference.
- Deep learning model usually use BLAS (basic linear algebra subprograms) libraries
- BLAS are set of programming routines that are commonly used in learn algebra operations
- Neural networks commonly use AXPY

$$y = \propto x + y$$

where x and y are vectors

- MATMUL is a second level BLAS and defined as

$$C \ := AB + C$$

where C, A, and B are matrices

Research Objective

1. Extending Ara to support low precision instruction and sub-byte computations. This would help with low precision Quantized neural networks and Binary neural networks
2. Build an FPGA model of the chip for verification
3. Build a custom chip from Ara + Ariane + extension design on GLOBALFOUNDERIES 22FDX 22 nm technology with the help of CMC
4. Build software stack and runtime to run on neural networks on the fabricated chip

SUB-BYTE COMPUTATIONS METHODS

Introduces software/hardware techniques to **compute sub-byte dot product** more efficiently

BIT-SERIAL [1]

Computation done between corresponding bits of operands in a serial manner

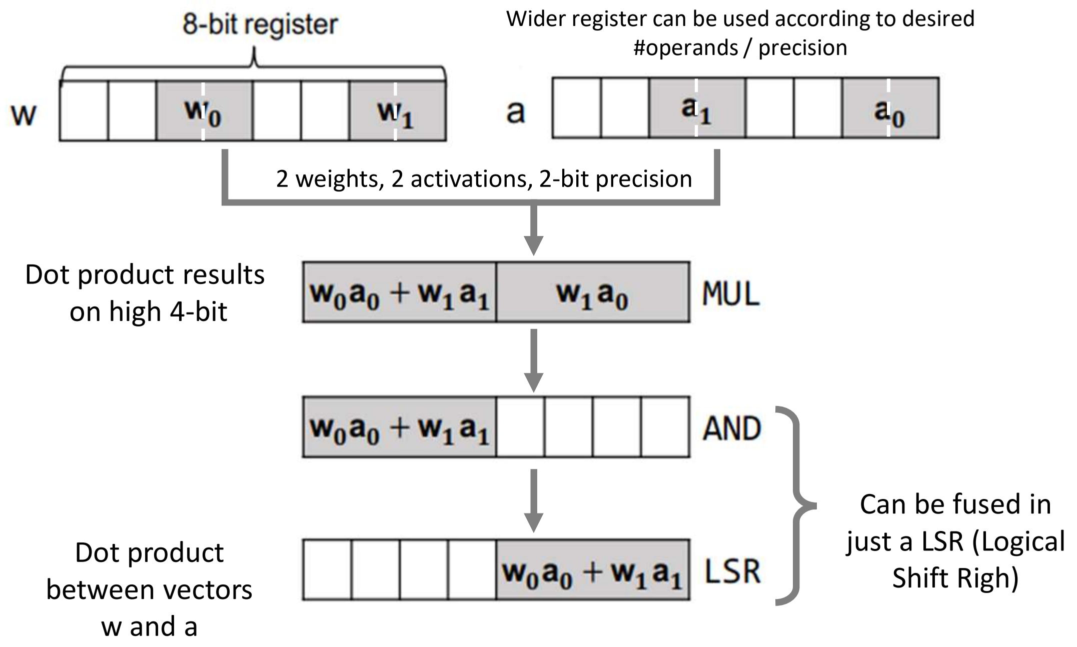
$$w \cdot a = \sum_{n=0}^{N} \sum_{m=0}^{M} 2^{n+m} popcnt(w_m \& a_n)$$

Where N, M are respectively activation precision and weight precision

At minimum, requires the implementation (SW or HW) of $popcnt$ instruction (non-existing in RISC-V « V » ISA)

ULPPACK [2]

Vectorizes the computation using scalar instructions



8-bit register
Wider register can be used according to desired #operands / precision
w
w_0
w_1
a
a_1
a_0
2 weights, 2 activations, 2-bit precision
Dot product results on high 4-bit
$w_0a_0 + w_1a_1$
w_1a_0
MUL
$w_0a_0 + w_1a_1$
AND
Can be fused in just a LSR (Logical Shift Righ)
Dot product between vectors w and a
$w_0a_0 + w_1a_1$
LSR

[1] M. Cowan, T. Moreau, T. Chen, J. Bornholt, and L. Ceze, 'Automatic generation of high-performance quantized machine learning kernels', 02 2020, pp. 305–316.

[2] J. Won, J. Si, S. Son, T. J. Ham, and J. W. Lee, 'ULPPACK: Fast Sub-8-bit Matrix Multiply on Commodity SIMD Hardware', in *Proceedings of Machine Learning and Systems*, 2022, vol. 4, pp. 52–63.

SUB-BYTE COMPUTATIONS METHODS

Introduces software/hardware techniques to **compute sub-byte dot product** more efficiently

BIT-SERIAL [1]

Computation done between corresponding bits of operands in a serial manner

Bit-serial complexity is $O(N \times M)$
Suitable for very small precision (typ. 1-bit to 2-bit)

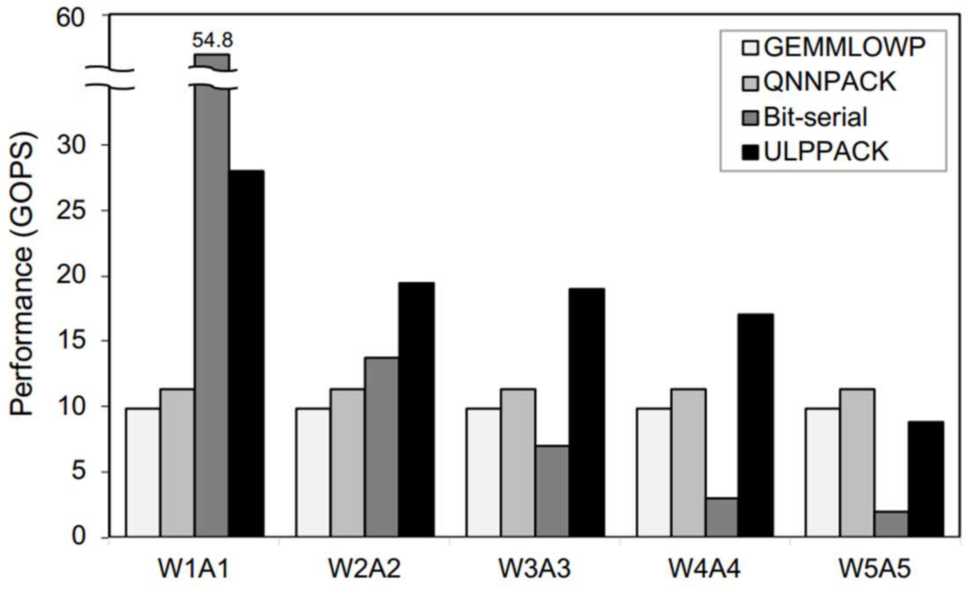
Requires modification on the HW

Exemple from [2] comparing both methods
Bit-serial performance decreases rapidely whereas ULPPACK yields great performance from W2A2 to W4A4

ULPPACK [2]

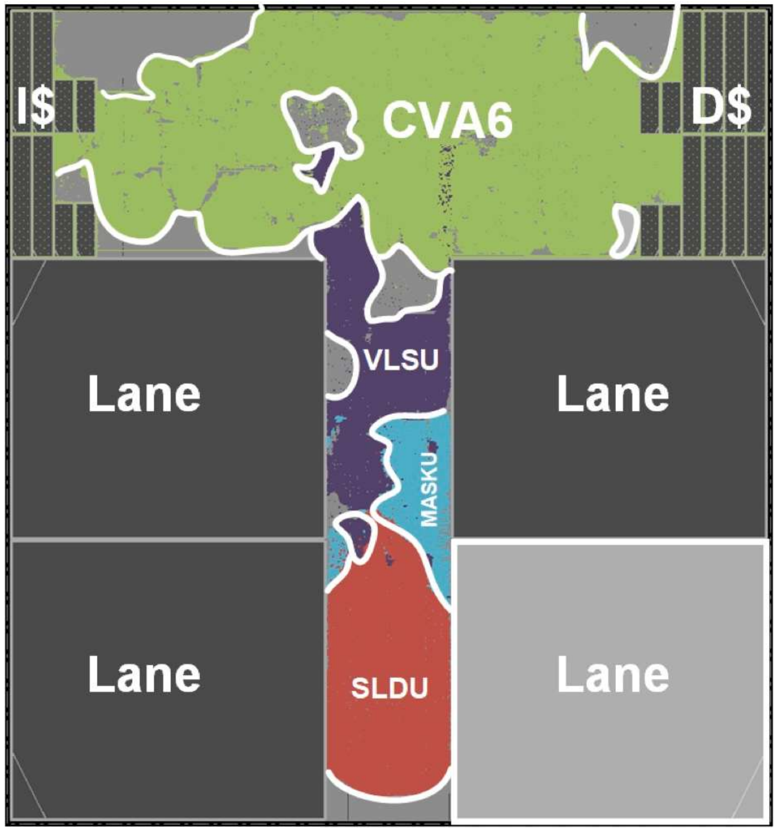
Vectorizes the computation using scalar instructions

More versatile (up to 4-bit precision)
Efficient in SW on small precision (no HW added)



[1] M. Cowan, T. Moreau, T. Chen, J. Bornholt, and L. Ceze, 'Automatic generation of high-performance quantized machine learning kernels', 02 2020, pp. 305–316.
[2] J. Won, J. Si, S. Son, T. J. Ham, and J. W. Lee, 'ULPPACK: Fast Sub-8-bit Matrix Multiply on Commodity SIMD Hardware', in *Proceedings of Machine Learning and Systems*, 2022, vol. 4, pp. 52–63.

RISC-V VECTOR PROCESSOR: PARALLELIZATION OF COMPUTATIONS



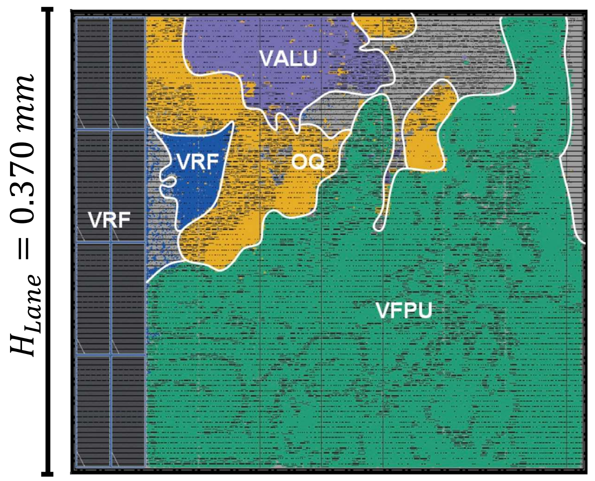
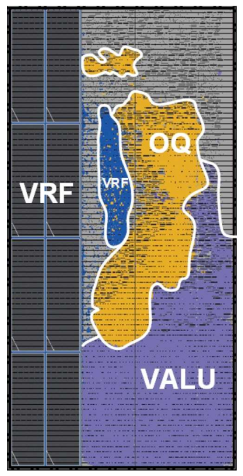
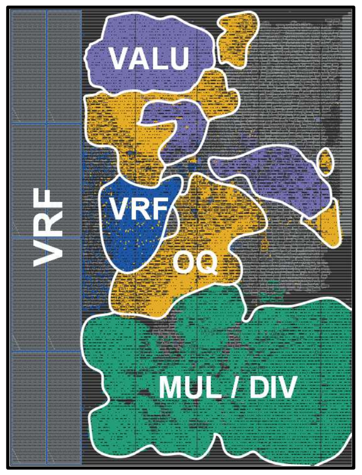
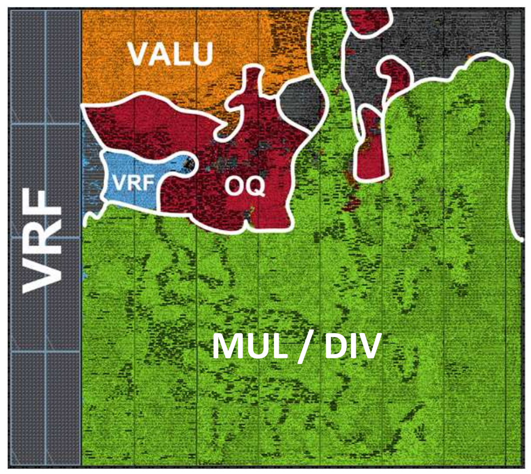
4 lane ara micro-architecture

POLYTECHNIQUE MONTRÉAL

SEVERAL ARCHITECTURES AIMING TOWARDS SUB-BYTE COMPUTATIONS

RISC-V

Work in Progress

	Ara	Quark [3]	Sparq [4]	Sparq v2
				
	$w_{Ara_{Lane}} = 0.415\ mm$	$w_{Quark_{Lane}} = 0.181\ mm$	$w_{Sparq_{Lane}} = 0.278\ mm$	$w_{Sparq^{v2}_{Lane}} = 0.415\ mm$
Area of one lane	$S_{Ara_{Lane}} = 0.154mm^2$	$S_{Quark_{Lane}} = 0.067mm^2$	$S_{Sparq_{Lane}} = 0.103mm^2$	$S_{Spark^{v2}_{Lane}} = 0.154mm^2$
Specifications	Vanilla RISC-V « V »	Bit-serial dedicated Hardware (No FPU, MUL/DIV)	ULPPACK dedicated Hardware (No FPU)	ULPPACK dedicated Hardware (barrel-shifter) (No FPU)

[1] M. AskariHemmat, T. Dupuis, Y. Fournier, N. E. Zarif, M. Cavalcante, M. Perotti, F. Gurkaynak, L. Benini, F. Leduc-Primeau, Y. Savaria, and J.-P. David, "Quark: An integer risc-v vector processor for sub-byte quantized dnn inference," 2023. [Online]. Available: https://arxiv.org/abs/2302.05996

[2] T. Dupuis, Y. Fournier, M. AskariHemmat, N. E. Zarif, F. Leduc-Primeau, J. P. David, and Y. Savaria, "Sparq: A custom risc-v vector processor for efficient sub-byte quantized inference," 2023, Unpublished.

POLYTECHNIQUE MONTRÉAL

SEVERAL ARCHITECTURES AIMING TOWARDS SUB-BYTE COMPUTATIONS

RISC-V®

Ara	Ara	Quark	Sparq	Sparq v2 (Work in Progress)
FP32 **12.33 operations/cycle**	ULPPACK (Software only)	BIT-SERIAL (HW acc.)	ULPPACK (HW acc.)	ULPPACK (HW acc.)

Performance of 2D convolution (3x3 kernel over 64x128x128 input)

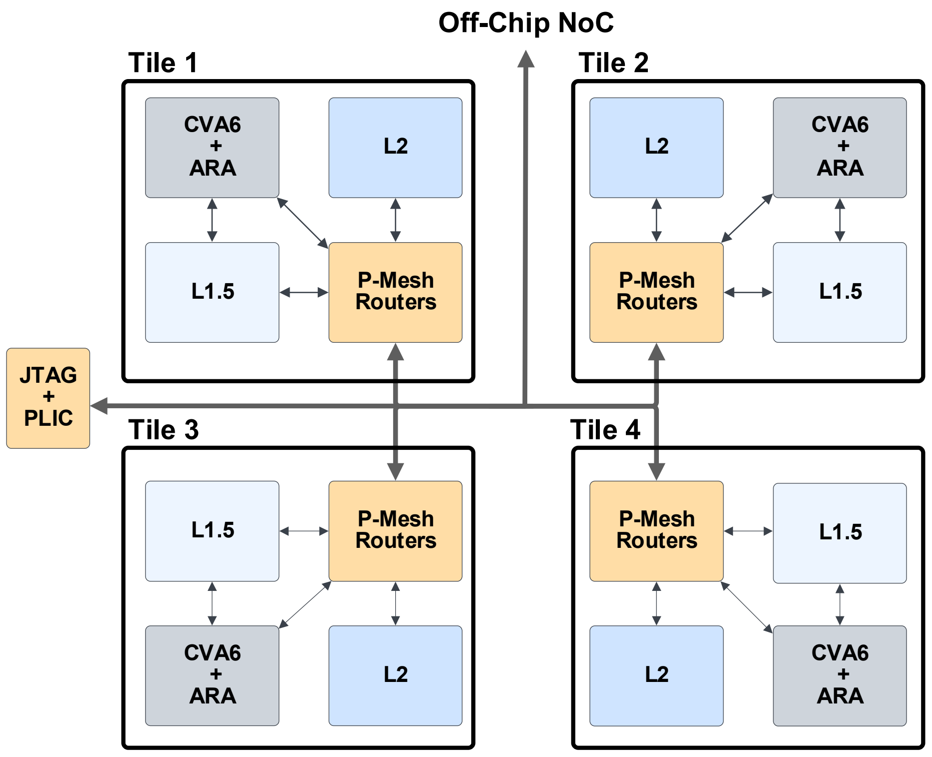
	Ara (ULPPACK, Software only)	Quark	Sparq	Sparq v2
W1A1	43.32 (×3.51)	51.67 (×4.19)	48.84 (×3.96)	Same performance as Sparq
W2A2	34.40 (×2.78)	30.80 (×2.50)	37.96 (×3.08)	
W3A3	27.57 (×2.23)	N/A	37.96 (×3.08)	

performance of W2A2 = W3A3

Results are in operations per cycle

Simulated using RTL simulation (4 lanes configuration)

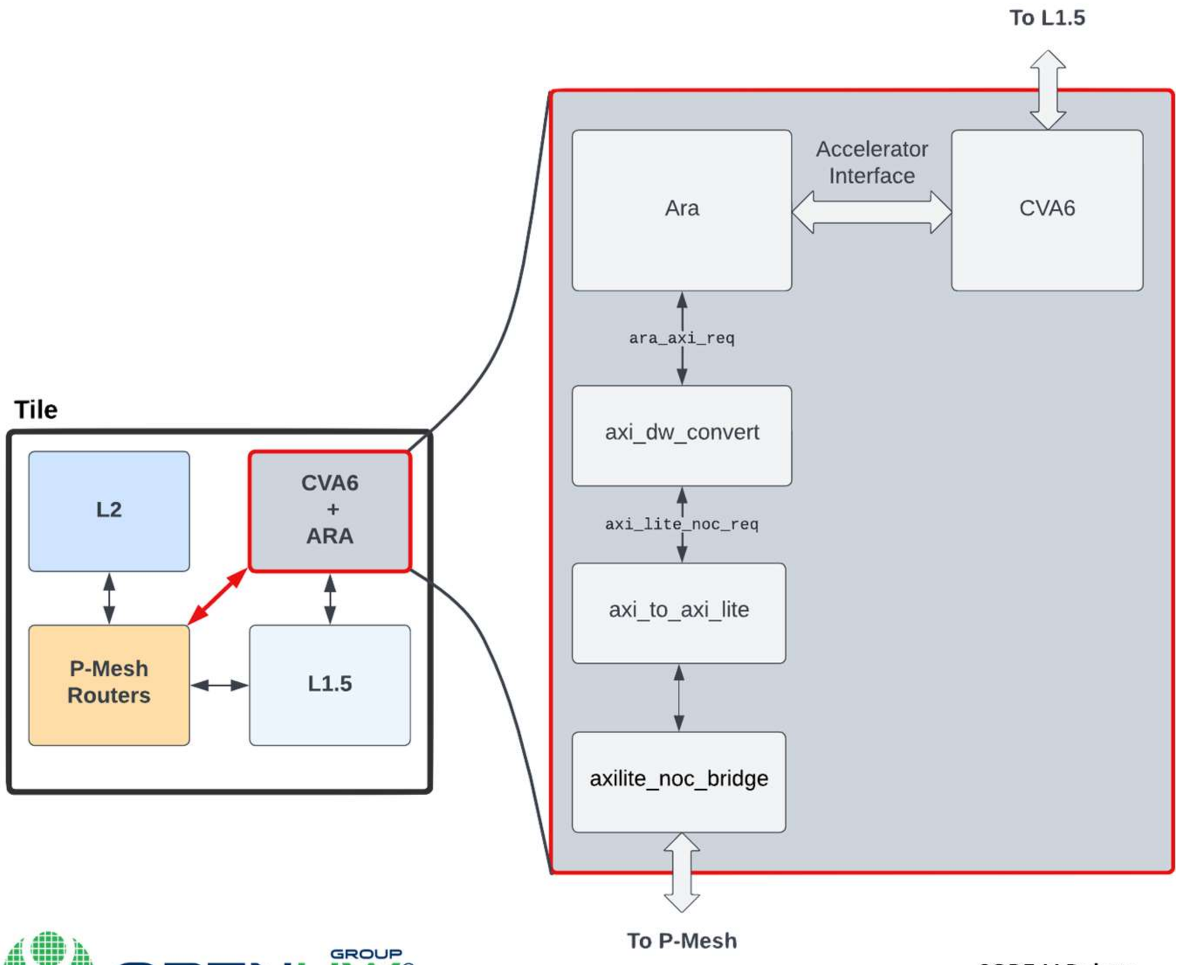
CORE-V Polara Architecture



- Polara is aiming to create a **multi-core version of the ARA** vectorial processor using the **OpenPiton** framework

- The ASIC will posses **4** instances or **ARA/CVVEC** with **4 lanes** each

OpenPiton Integration



- With help from UC Santa Barbara, a Polara tile is integrated in OpenPiton using an **AXI to NOC bridge**.
- For the first stage of Integration, initial **RISC-V vector tests** are used for **1 tile configuration**.
- Next in our plan is to run these tests in **multi tile configuration**.

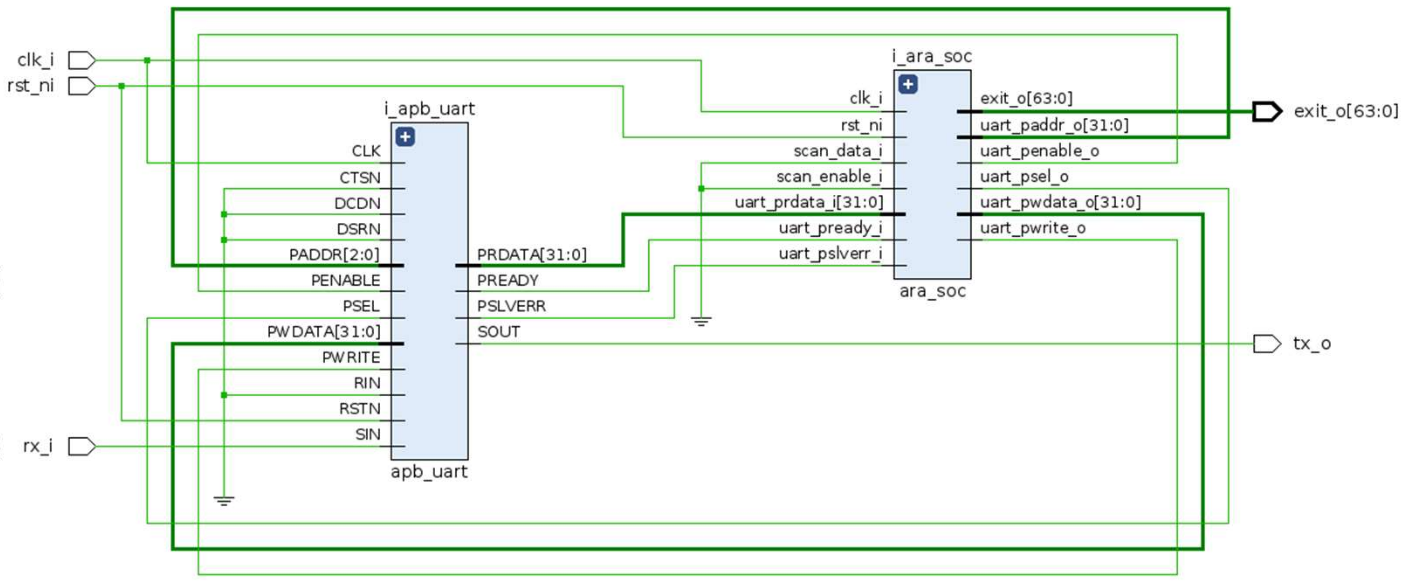
FPGA Emulation - Ara

Synthesis and implementation completed on Xilinx Alveo U280

Configuration:

- 4 lanes
- 512kB L2 cache
- Max achievable frequency: 75MH

Bitstream generation:
Adaptation of the board's constraints file to generate the bitstream



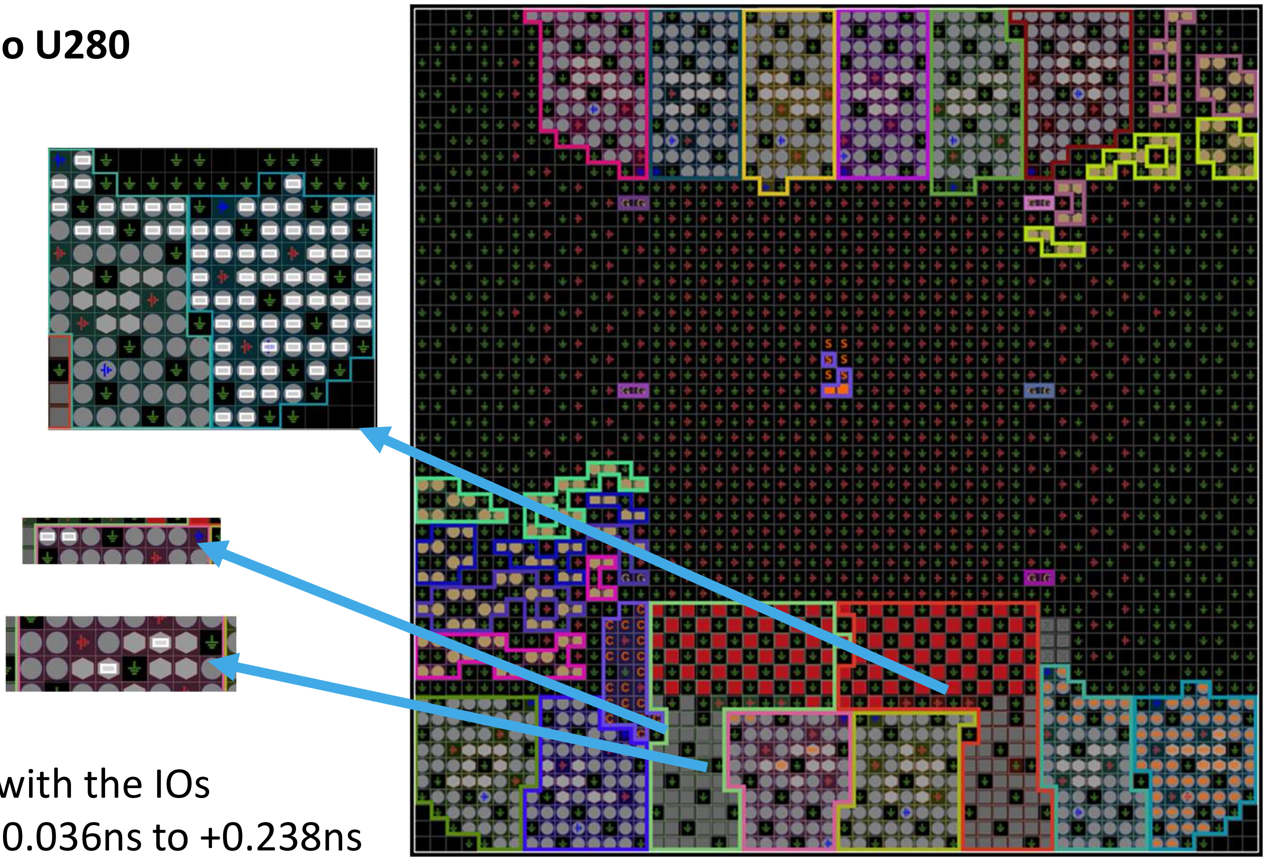
FPGA Emulation - Ara

IO planning on Xilinx Alveo U280

64 bits for the exit signal

UART: RX and TX

Clock and Reset



Adding the constraint file with the IOs increased the WNS from +0.036ns to +0.238ns at 75MHz

FPGA Emulation

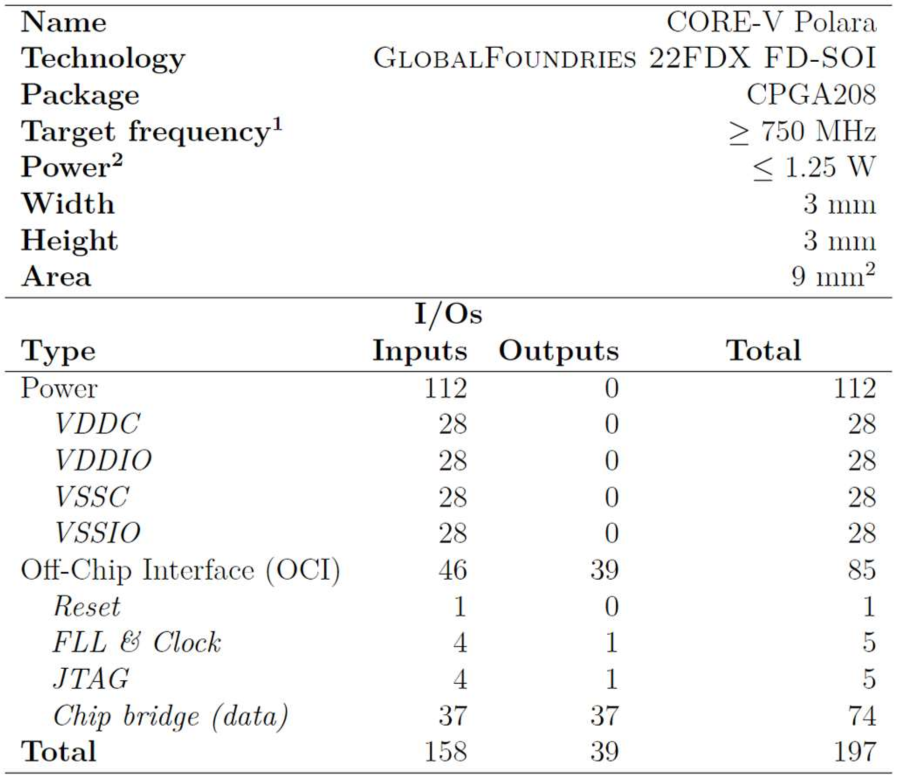
- **Next steps:**

- Ara
- Programming the FPGA board with Ara
- Run basic tests on Ara

- OpenPiton
- Write Fusesoc script for Polara emulation: synthesis, implementation, generation of bitstream
- Programming the FPGA board with 1 tile of Polara's OpenPiton
- Run tests on Polara

ASIC design - Specifications

Table 1: CORE-V Polara specifications

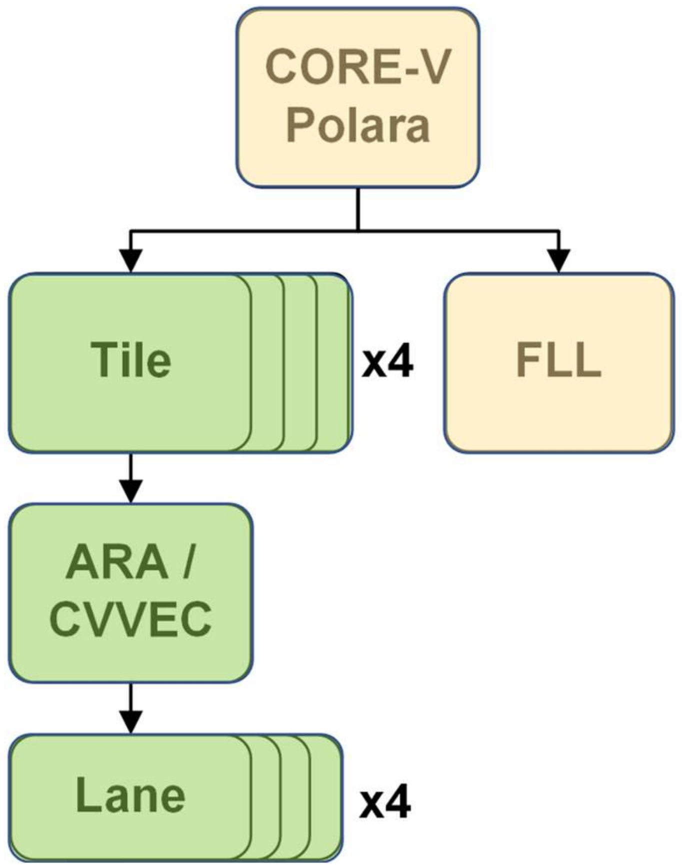


Name	CORE-V Polara
Technology	GlobalFoundries 22FDX FD-SOI
Package	CPGA208
Target frequency[1]	$\geq$ 750 MHz
Power[2]	$\leq$ 1.25 W
Width	3 mm
Height	3 mm
Area	9 mm^2

	I/Os		
Type	**Inputs**	**Outputs**	**Total**
Power	112	0	112
VDDC	28	0	28
VDDIO	28	0	28
VSSC	28	0	28
VSSIO	28	0	28
Off-Chip Interface (OCI)	46	39	85
Reset	1	0	1
FLL & Clock	4	1	5
JTAG	4	1	5
Chip bridge (data)	37	37	74
Total	158	39	197

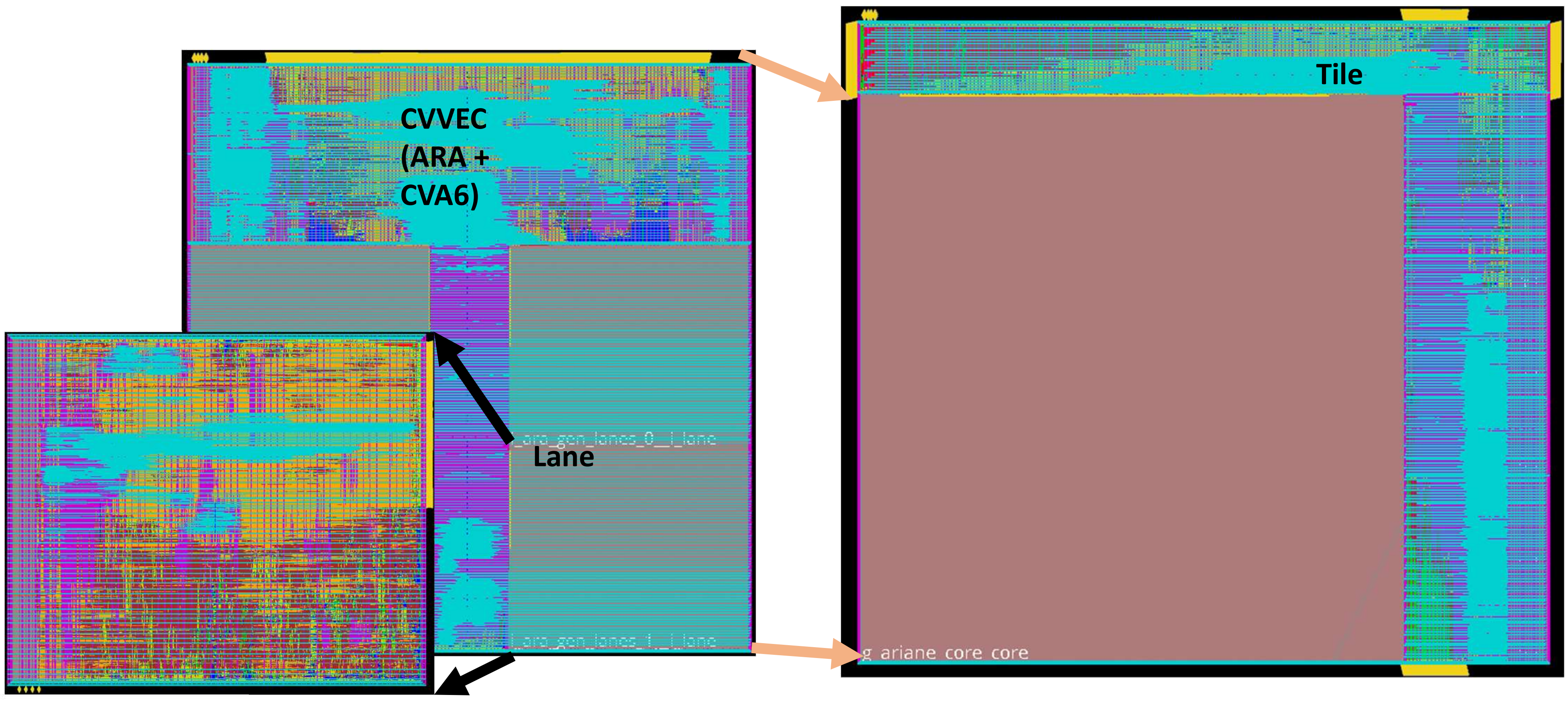
- CORE-V Polara is aimed to be **taped-out this summer** in **GF22FDX**
- We are aiming for a **3 x 3 mm die** at a **> 750 MHz clock**

ASIC design – Work left to do

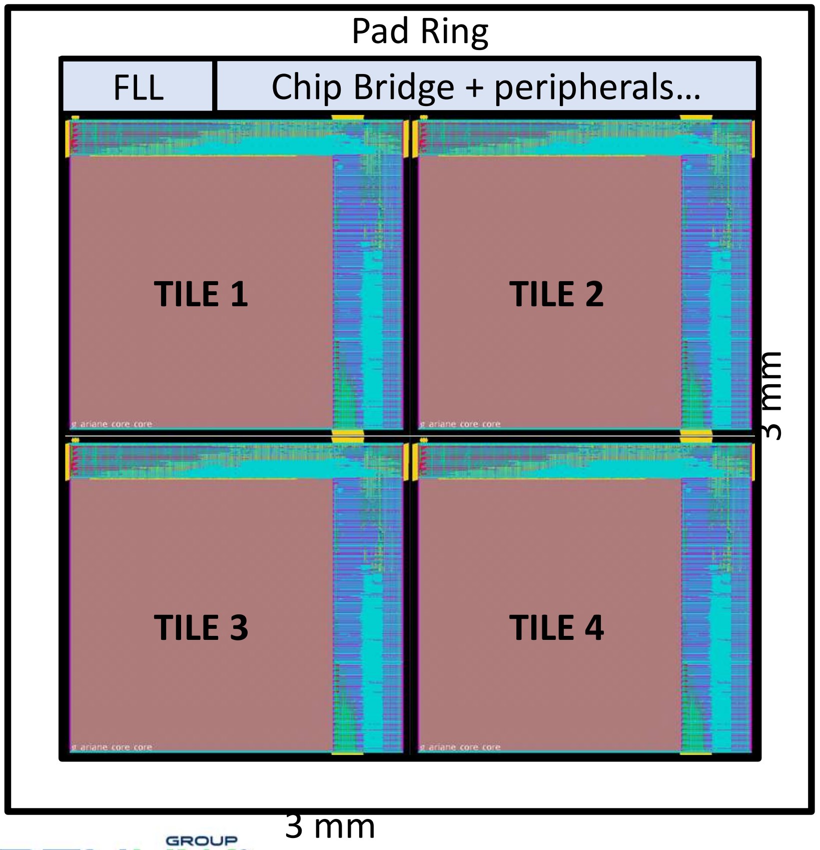


- The **Lanes and ARA macros** are tape-out ready
- Currently **finalising** the **OpenPiton Tile** macro
- Currently **starting** working on the **Top-Level** macro

ASIC design - Macros

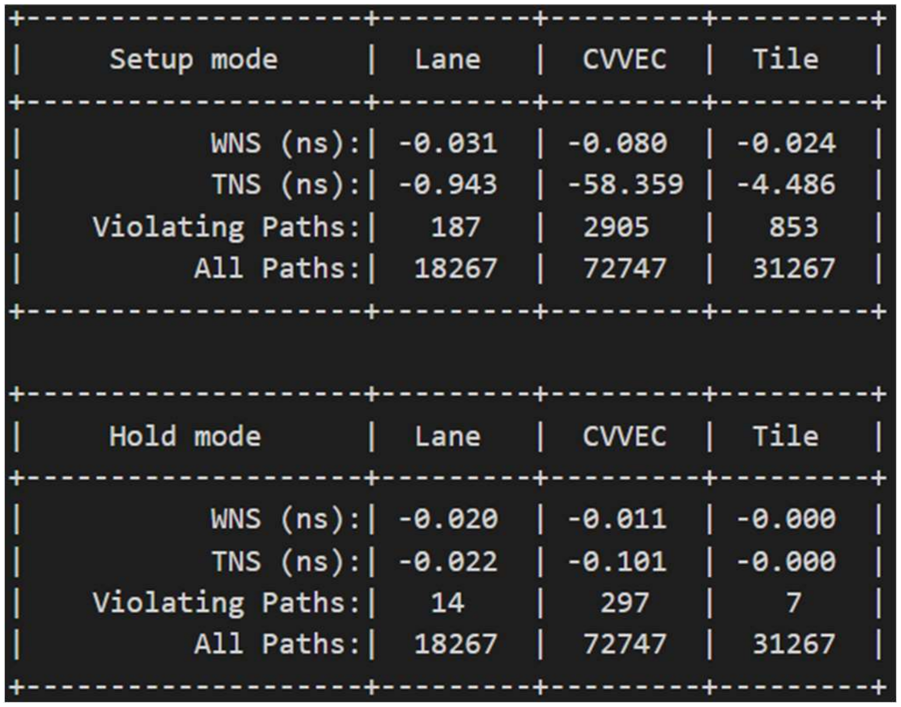


ASIC design – Top Level



- Starting to work on the **Top Level**
- Total of **4** Tiles with the chip bridge and the OpenPiton peripherals
- **FLL** integration still left to do

PnR timing results



```
+-------------------+---------+---------+---------+
|     Setup mode    |  Lane   |  CVVEC  |  Tile   |
+-------------------+---------+---------+---------+
|          WNS (ns):| -0.031  | -0.080  | -0.024  |
|          TNS (ns):| -0.943  | -58.359 | -4.486  |
|    Violating Paths:|  187    |  2905   |  853    |
|         All Paths:| 18267   | 72747   | 31267   |
+-------------------+---------+---------+---------+

+-------------------+---------+---------+---------+
|     Hold mode     |  Lane   |  CVVEC  |  Tile   |
+-------------------+---------+---------+---------+
|          WNS (ns):| -0.020  | -0.011  | -0.000  |
|          TNS (ns):| -0.022  | -0.101  | -0.000  |
|    Violating Paths:|  14     |  297    |  7      |
|         All Paths:| 18267   | 72747   | 31267   |
+-------------------+---------+---------+---------+
```

- Timing results for the current macros
  - Clock @ **750 MHz**
  - Some **small violations** left to fix
  - Violations can be **fixed** using ECO on the critical paths

Tape-out plan

MPW2254 :

- Application deadline : **2023-04-10**
- Cancelation date : **2023-07-05**
- Export Control Date : 2023-07-19
- Design Submission Deadline : 2023-07-19
- Delivery Date : 2024-01-05

AI runtime

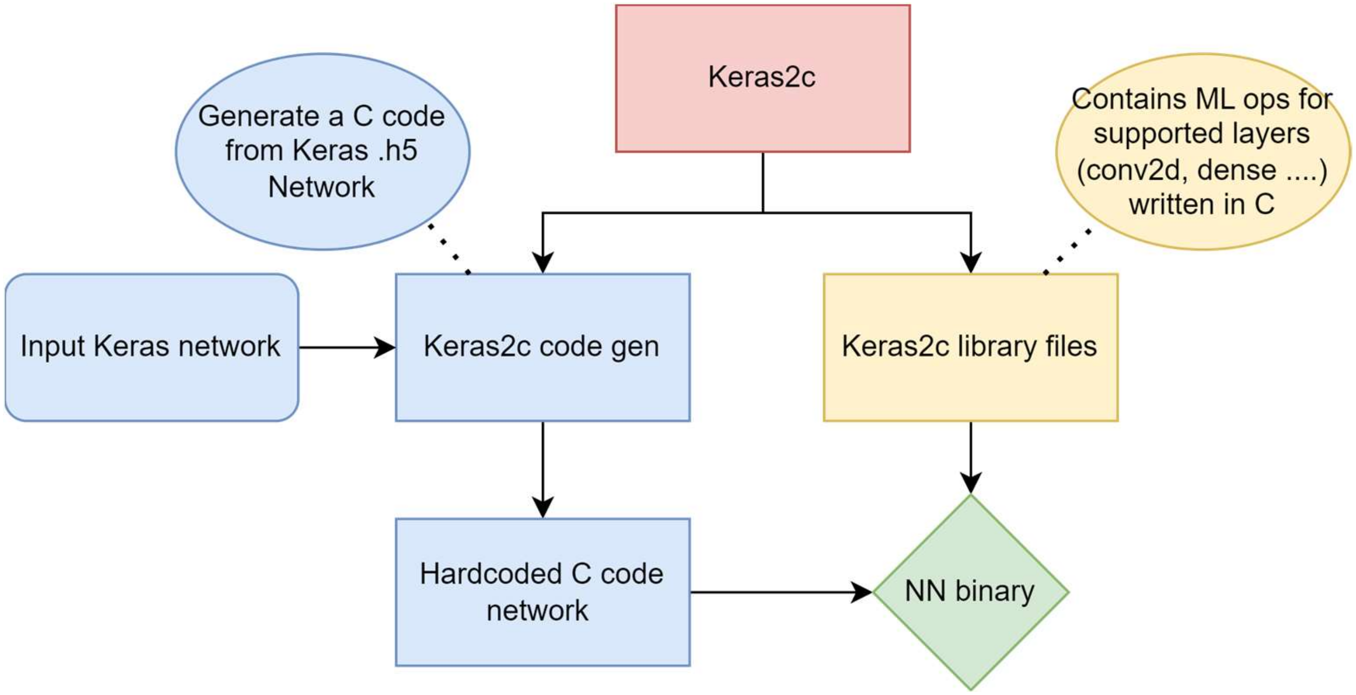
- Usually, To run AI workload on RISC V we need to have a full OS (like Linux) build with support user level vector instructions.
- Or run the AI workload directly on Metal which presents it is own challenges.
- While CVA6 does have an MMU (memory management unit), the Ara vector processor doesn’t
- An MMU is necessary for address translation and virtual memory (and running Linux)
- Thus, running RISCV vector code in an OS environment not possible.

AI runtime

- Solution:
  - Build an AI runtime that works directly on baremetal
  - Running baremetal AI not only would work without OS, but it also reduces memory address translation that can result in fewer memory operations, which reduces computation time and power
  - The generic code structure that we build should support a wide array of microcontrollers and processors
- Challenges:
  - Only the basic functionality of C is supported, which means that there is no support for standard C libraries, or even stuff like malloc and other dynamic memory units.

Keras2c

- Keras2c has two parts:
  - **Code gen**: takes input keras ".h5" network and convert it to C code. Written in python
  - **Library files:** written in C contain the supported ML ops and different data structures to run generate C code
- Capable to run baremetal environment with little modification
- https://github.com/f0uriest/keras2c


Keras2c
Generate a C code from Keras .h5 Network
Contains ML ops for supported layers (conv2d, dense ....) written in C
Input Keras network
Keras2c code gen
Keras2c library files
Hardcoded C code network
NN binary

Keras2c workflow

Codegen

- Write a script to load keras model and generate 2 c files and 1 h file
  - The three files are a model file, input generation file and header file

Compile

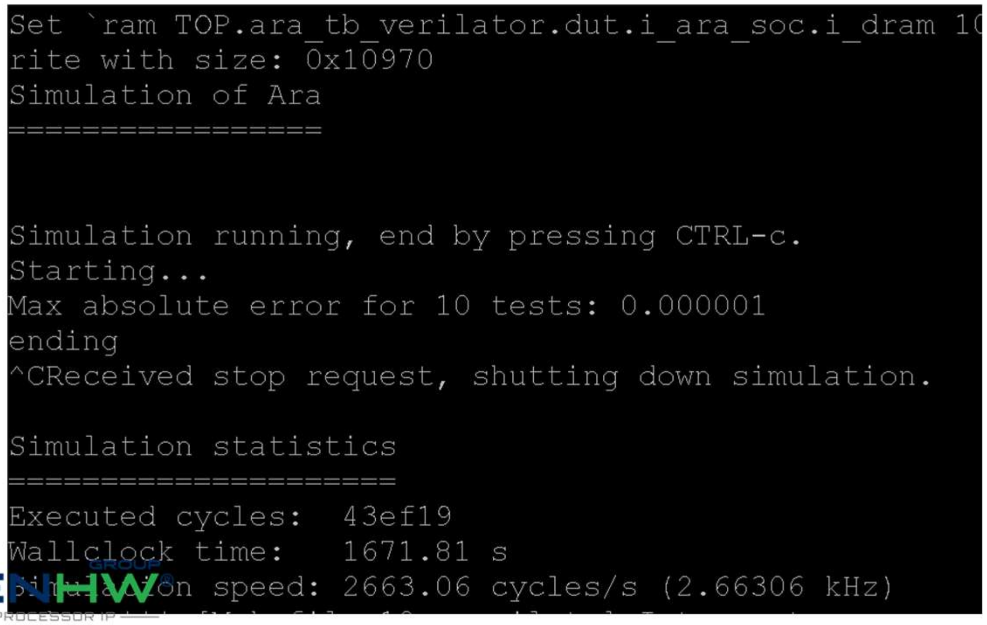
- Write a makefile to compile the three generated files with the library files
- The input file the labeled test_suite which contains generated input and calls the model files
- The model files contains the weights and biases of each layer along with the functions calls to the library (conv2d, padding, dense, add, affine matrix, ….)

run

- Generate a binary using the makefiles, in cases of polara we need to modify linker scripts to generate
- Run the code (we can use verilator to run the code on a simulated hardware)

Keras2c Riscv

- The generated binary can work in verilator which is a cycle accurate emulator,
- It is producing the expected output which is the exact output if ran natively on X86 linux, or Spike RISCV elf



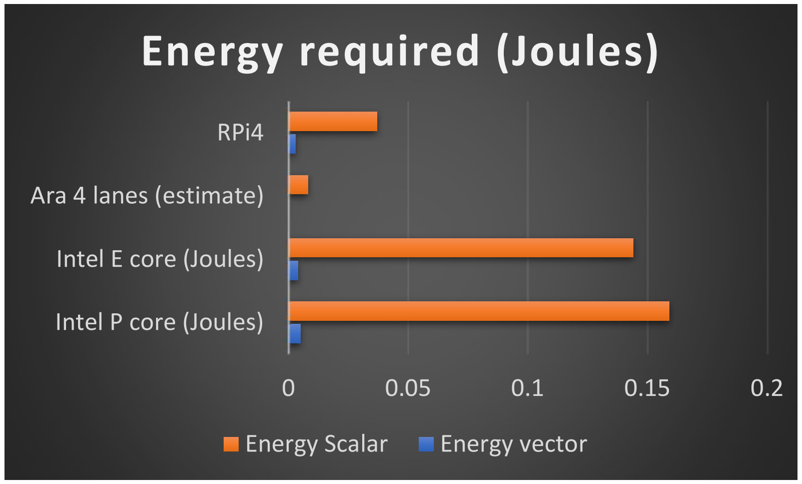
```
Set `ram TOP.ara_tb_verilator.dut.i_ara_soc.i_dram 10
rite with size: 0x10970
Simulation of Ara
=================


Simulation running, end by pressing CTRL-c.
Starting...
Max absolute error for 10 tests: 0.000001
ending
^CReceived stop request, shutting down simulation.

Simulation statistics
=====================
Executed cycles:  43ef19
Wallclock time:   1671.81 s
Simulation speed: 2663.06 cycles/s (2.66306 kHz)
```

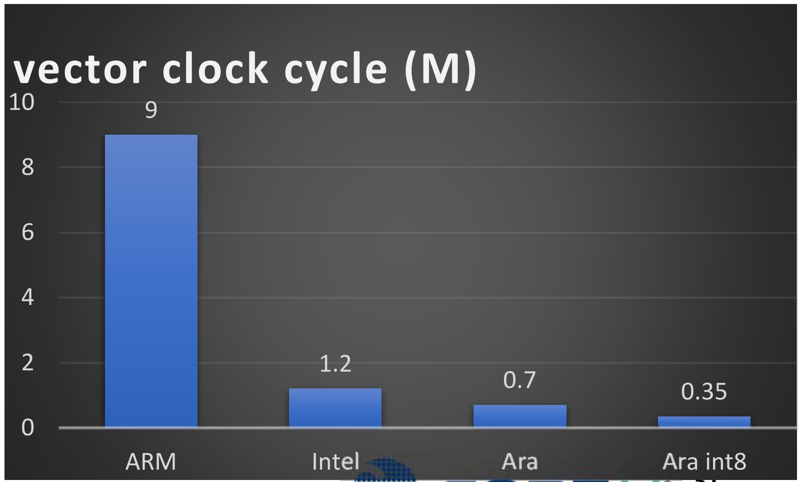
Polara-keras2c

- rewriting some of the function of Keras2c to work for baremetal with custom implementation to work for baremetal (Memcpy, memset, strlen, strnlen, strcmp, strcpy, printf…)
- Updated code generation to remove calls incompatible with Polara baremetal environment
- Added support for some operations (conv2d, relu, add, dense, padding) for safe fixed-point implementation
- The remaining layers are still implemented in floating point
- The library files can be replaced with an optimized implementation (for example RISCV vector code)

Preliminary result of inference

- 97% of cycles are spent executing conv2d
- Optimizing for conv2d
- Implemented RVV int8 vector to speedup the convolution, maxpool and Relu
- Ara vector is close to IPC performance of 12th gen intel core using AVX instructions at much lower energy use (simulation)





Future work

- Finish RTL for the sparq v2
- FPGA implementation for Polara
- Validation testing for Polara
- Fix minor timing issues with Polara
- Full stack implementation of polara-keras2c with proper scaling and shifting factors for low-precision neural network

Conclusion

- Work on Multicore Ara is progressing nicely
- Chip is expected back in 2024
- High-performance and power efficiency using RISCV vector

POLYTECHNIQUE MONTRÉAL

RISC-V®

Thank you!