Design Techniques for Multi-Core Neural Network Accelerators on Radiation-Hardened FPGAs

Andrea Portaluri, Sarah Azimi and **Luca Sterpone**

Politecnico di Torino

Motivations

- The computational power and data transfer capabilities required for modern deep space applications have drastically grown

- Recent rad-hard technology improvements blocks have unlocked a lot of potential applications, breaking through the performance limits of these devices
  - High-performance block RAMs (BRAMs)
  - Digital Signal Processing (DSP) blocks

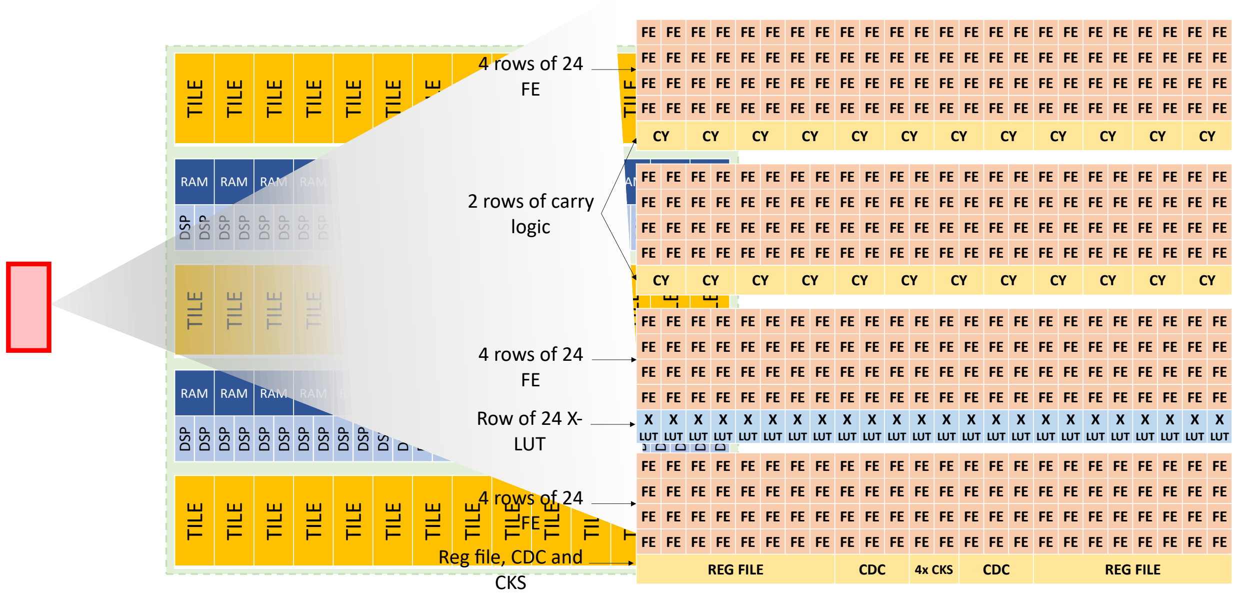
- Acceleration of Convolutional Neural Networks (CNNs) and soft processors might now be implemented in the programmable logic of most of these rad-hard FPGAs

Goal

- Evaluate the feasibility of implementation and timing performance optimization of an accelerator for convolution running on the **r-VEX** Soft Processor
- Assembly code for the **VEX** Instruction Set Architecture (ISA) has been developed in order to execute
  - **Convolutional products**
  - **Rectifier Linear Unit (ReLU)**
  - **Max Pooling**
- Single-core and Multi-core solutions for parallel computations have been implemented on NG-medium
- Evaluate the trade-off between performances and the number of cores

NanoXplore FPGAs and Machine Learning

- Very few works have focused on the feasibility and implementation of Machine Learning accelerators for rad-hard FPGAs
- CAD tools (placement and routing) and architectural data are strategic
- Cluster-oriented programmable logic with tight placement constraints.

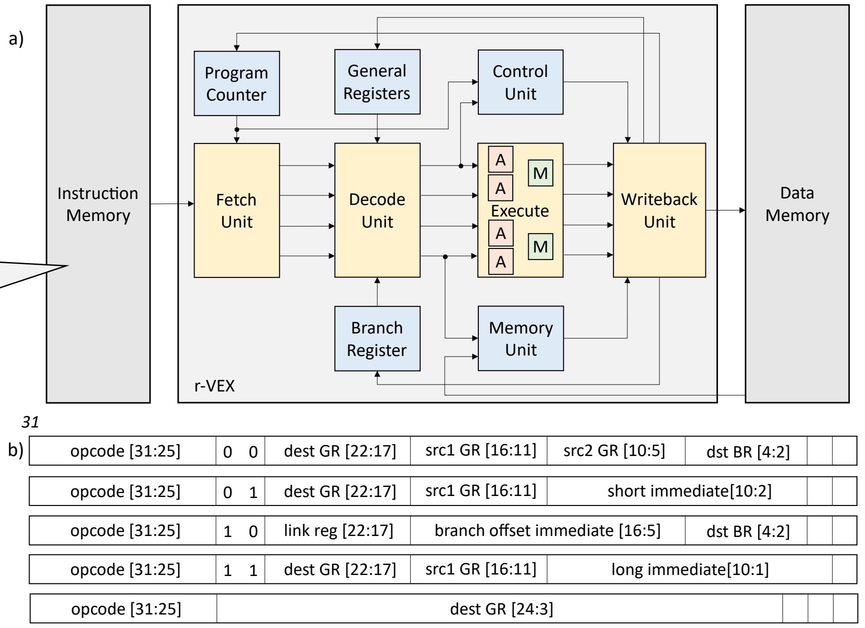


Overall Architecture of NG-MEDIUM Programmable Logic

Convolution in VEX Assembly

- **R**econfigurable **V**LIW **Ex**ample (r-VEX) is a Very-Long Instruction Word reconfigurable processor developed by the University of Delft
- Number of registers, ALUs, memory sizes and multipliers are **parametric**
- The VEX architecture works as **AI Engine** control unit based on VLIW processors

Instructions are directly coded into the VHDL through Assembly



opcode [31:25]	0	0	dest GR [22:17]	src1 GR [16:11]	src2 GR [10:5]	dst BR [4:2]
opcode [31:25]	0	1	dest GR [22:17]	src1 GR [16:11]	short immediate[10:2]	
opcode [31:25]	1	0	link reg [22:17]	branch offset immediate [16:5]		dst BR [4:2]
opcode [31:25]	1	1	dest GR [22:17]	src1 GR [16:11]	long immediate[10:1]	
opcode [31:25]	dest GR [24:3]					

Scheme of the r-VEX and its instruction set

5 types of instruction packed in syllables of 4 (parametrizable)

Convolution in VEX Assembly

- **AlexNet** is a very efficient **CNN** for handwritten character recognition containing layers of
  - 2D convolution
  - Max Pooling
  - ReLU activation
- A squared input image is fed into a series of convolution products, Max Pooling, and activation in order to return a probability of that image being a class

Convolution in VEX Assembly

- Assembly equivalent of the convolutional steps has been written and coded into the instruction memory of the r-VEX

```
Input: image[N][N]
Load image from Data Memory:
 1: while branch_col_register do
 2:     while branch_row_register do
 3:         mov image[i][j], gen_register j+N*i
 4:         add 1, row_register
 5:         cmpeq 1, branch_row_register
 6:     add 1, col_register
 7:     cmpeq 1, branch_col_register
 8: goto Convolution
Convolution:
 8: while branch_col_register do
 9:     while branch_row_register do
10:         execute convolution in row_register, col_register
11:         add 1, row_register
12:         cmpeq 1, branch_row_register
13:     add 1, col_register
14:     cmpeq 1, branch_col_register
15: goto Max Pool
Max Pool:
16: while branch_col_register do
17:     while branch_row_register do
18:         max gen_register i, gen_register i+1
19:         add 1, row_register
20:         cmpeq 1, branch_row_register
21:     add 1, col_register
22:     cmpeq 1, branch_col_register
23: goto ReLU
ReLu:
24: while branch_col_register do
25:     while branch_row_register do
26:         max gen_register i, 0
27:         add 1, row_register
28:         cmpeq 1, branch_row_register
29:     add 1, col_register
30:     cmpeq 1, branch_col_register
```

Politecnico di Torino

Convolution in VEX Assembly

- Assembly equivalent of the convolutional steps has been written and coded into the instruction memory of the r-VEX

```
Input: image[N][N]
Load image from Data Memory:
while branch_col_register do
    while branch_row_register do
        mov image[i][j], gen_register j+N*i
        add 1, row_register
        cmpeq 1, branch_row_register
    add 1, col_register
    cmpeq 1, branch_col_register
goto Convolution
Convolution:
while branch_col_register do
    while branch_row_register do
        execute convolution in row_register, col_register
        add 1, row_register
        cmpeq 1, branch_row_register
    add 1, col_register
    cmpeq 1, branch_col_register
goto Max Pool
```

```
Max Pool:
while branch_col_register do
    while branch_row_register do
        max gen_register i, gen_register i+1
        add 1, row_register
        cmpeq 1, branch_row_register
    add 1, col_register
    cmpeq 1, branch_col_register
goto ReLU
ReLu:
while branch_col_register do
    while branch_row_register do
        max gen_register i, 0
        add 1, row_register
        cmpeq 1, branch_row_register
    add 1, col_register
    cmpeq 1, branch_col_register
```

Politecnico di Torino

Convolution in VEX Assembly

- Assembly equivalent of the convolutional steps has been written and coded into the instruction memory of the r-VEX

```
Input: image[N][N]
Load image from Data Memory:
 1: while branch_col_register do
 2:   while branch_row_register do
 3:     mov image[i][j], gen_register j+N*i
 4:     add 1, row_register
 5:     cmpeq 1, branch_row_register
 6:   add 1, col_register
 7:   cmpeq 1, branch_col_register
 8: goto Convolution
Convolution:
 8: while branch_col_register do
 9:   while branch_row_register do
10:     execute convolution in row_register, col_register
11:     add 1, row_register
12:     cmpeq 1, branch_row_register
13:   add 1, col_register
14:   cmpeq 1, branch_col_register
15: goto Max Pool
Max Pool:
16: while branch_col_register do
17:   while branch_row_register do
18:     max gen_register i, gen_register i+1
19:     add 1, row_register
20:     cmpeq 1, branch_row_register
21:   add 1, col_register
22:   cmpeq 1, branch_col_register
23: goto ReLU
ReLu:
24: while branch_col_register do
25:   while branch_row_register do
26:     max gen_register i, 0
27:     add 1, row_register
28:     cmpeq 1, branch_row_register
29:   add 1, col_register
30:   cmpeq 1, branch_col_register
```

Convolution in VEX Assembly

- Assembly equivalent of the convolutional steps has been written and coded into the instruction memory of the r-VEX

```
Input: image[N][N]
Load image from Data Memory:
 1: while branch_col_register do
 2:   while branch_row_register do
 3:     mov image[i][j], gen_register j+N*i
 4:     add 1, row_register
 5:     cmpeq 1, branch_row_register
 6:   add 1, col_register
 7:   cmpeq 1, branch_col_register
 8: goto Convolution
Convolution:
 8: while branch_col_register do
 9:   while branch_row_register do
10:     execute convolution in row_register, col_register
11:     add 1, row_register
12:     cmpeq 1, branch_row_register
13:   add 1, col_register
14:   cmpeq 1, branch_col_register
15: goto Max Pool
Max Pool:
16: while branch_col_register do
17:   while branch_row_register do
18:     max gen_register i, gen_register i+1
19:     add 1, row_register
20:     cmpeq 1, branch_row_register
21:   add 1, col_register
22:   cmpeq 1, branch_col_register
23: goto ReLU
ReLu:
24: while branch_col_register do
25:   while branch_row_register do
26:     max gen_register i, 0
27:     add 1, row_register
28:     cmpeq 1, branch_row_register
29:   add 1, col_register
30:   cmpeq 1, branch_col_register
```

Convolution in VEX Assembly

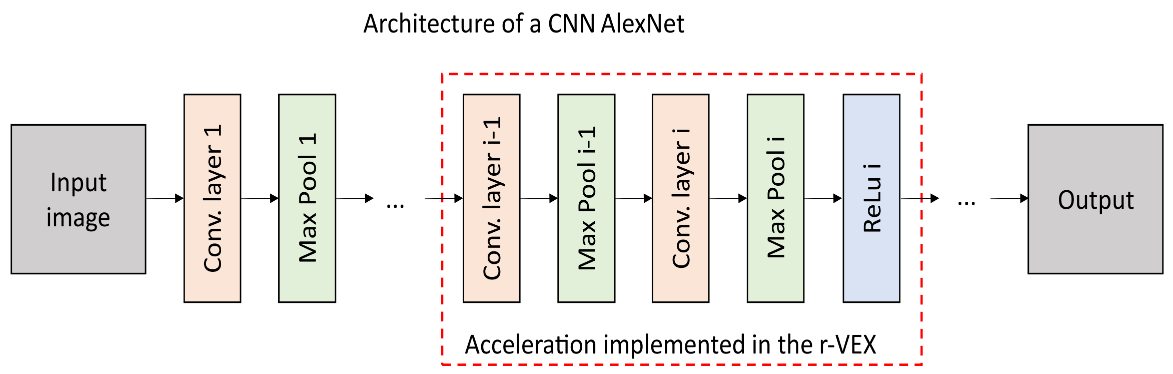
- Assembly equivalent of the convolutional steps has been written and coded into the instruction memory of the r-VEX

```
Input: image[N][N]
Load image from Data Memory:
1:  while branch_col_register do
2:      while branch_row_register do
3:          mov image[i][j], gen_register j+N*i
4:          add 1, row_register
5:          cmpeq 1, branch_row_register
6:      add 1, col_register
7:      cmpeq 1, branch_col_register
8:  goto Convolution
Convolution:
8:  while branch_col_register do
9:      while branch_row_register do
10:         execute convolution in row_register, col_register
11:         add 1, row_register
12:         cmpeq 1, branch_row_register
13:     add 1, col_register
14:     cmpeq 1, branch_col_register
15: goto Max Pool
Max Pool:
16: while branch_col_register do
17:     while branch_row_register do
18:         max gen_register i, gen_register i+1
19:         add 1, row_register
20:         cmpeq 1, branch_row_register
21:     add 1, col_register
22:     cmpeq 1, branch_col_register
23: goto ReLU
ReLu:
24: while branch_col_register do
25:     while branch_row_register do
26:         max gen_register i, 0
27:         add 1, row_register
28:         cmpeq 1, branch_row_register
29:     add 1, col_register
30:     cmpeq 1, branch_col_register
```

Politecnico di Torino

Convolution in VEX Assembly

- Assembly equivalent of the convolutional steps has been written and coded into the instruction memory of the r-VEX



Implemented part of AlexNet CNN

Layer	Input	Stride	Padding	Filter	Output
Conv. Layer 1	12 x 12	1	1	3 x 3	12 x 12
Max Pool 1	12 x 12	0	1	2 x 2	11 x 11
Conv. Layer 2	11 x 11	0	1	3 x 3	9 x 9
Max Pool 2	9 x 9	0	1	2 x 2	8 x 8

Parameters of the implemented net

NG-MEDIUM implementation

- **NanoXplore NG-MEDIUM chip** has been the target of the analysis
- 3 different designs have been implemented **varying the number of r-VEX cores**
- A USB-to-UART bridge has been used to read serial data from a UART VHDL module implemented along the design

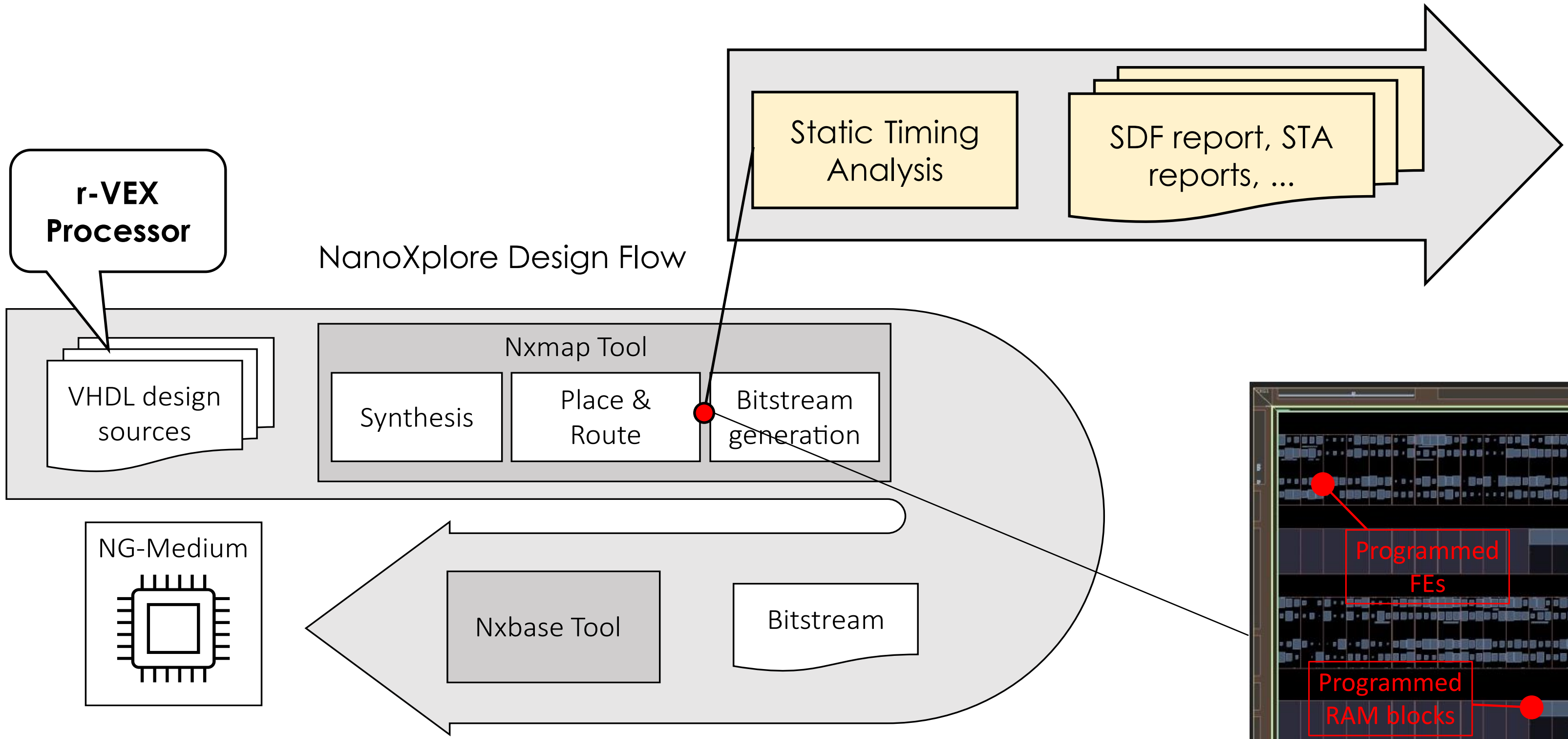
General Registers [#]	64
Branch Registers [#]	8
ALUs [#]	4
Multipliers [#]	2
Syllable-Issues [#]	4
Data Memory [kB]	32

Table III. Parameters of the r-VEX

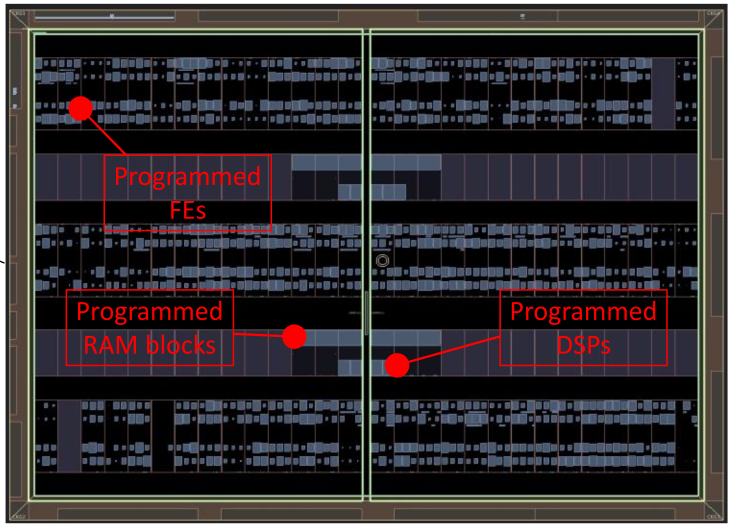
Design	**4-LUT**	**DFF**	**Carry Logic**	**DSP**	**BRAM**
1-core	6,098 (19%)	1,901 (6%)	298 (4%)	3 (3%)	4 (8%)
2-core	11,714 (37%)	3,801 (12%)	596 (8%)	6 (6%)	8 (15%)
3-core	17,946 (56%)	5,703 (18%)	894 (12%)	9 (9%)	12 (22%)

Table IV. Resources utilization on NG-MEDIUM for the designs

Experimental Analysis Flow



- Transition Delay Analysis
- Data Delay Analysis
- Placement Constrained Data Delay



Experimental Results

- The Standard Delay Format (SDF) analysis allows to **estimate transition timing delays**
  - Internal Routing: routing resources into the same tile
  - General Routing: routing resources for inter-tile communication
  - Low-Skew Routing: routing resources for distribution of global signals

Delays seem to increase as the placement spreads in the programmable logic

The delay differences are less than 1% of the clock period (40 ns)

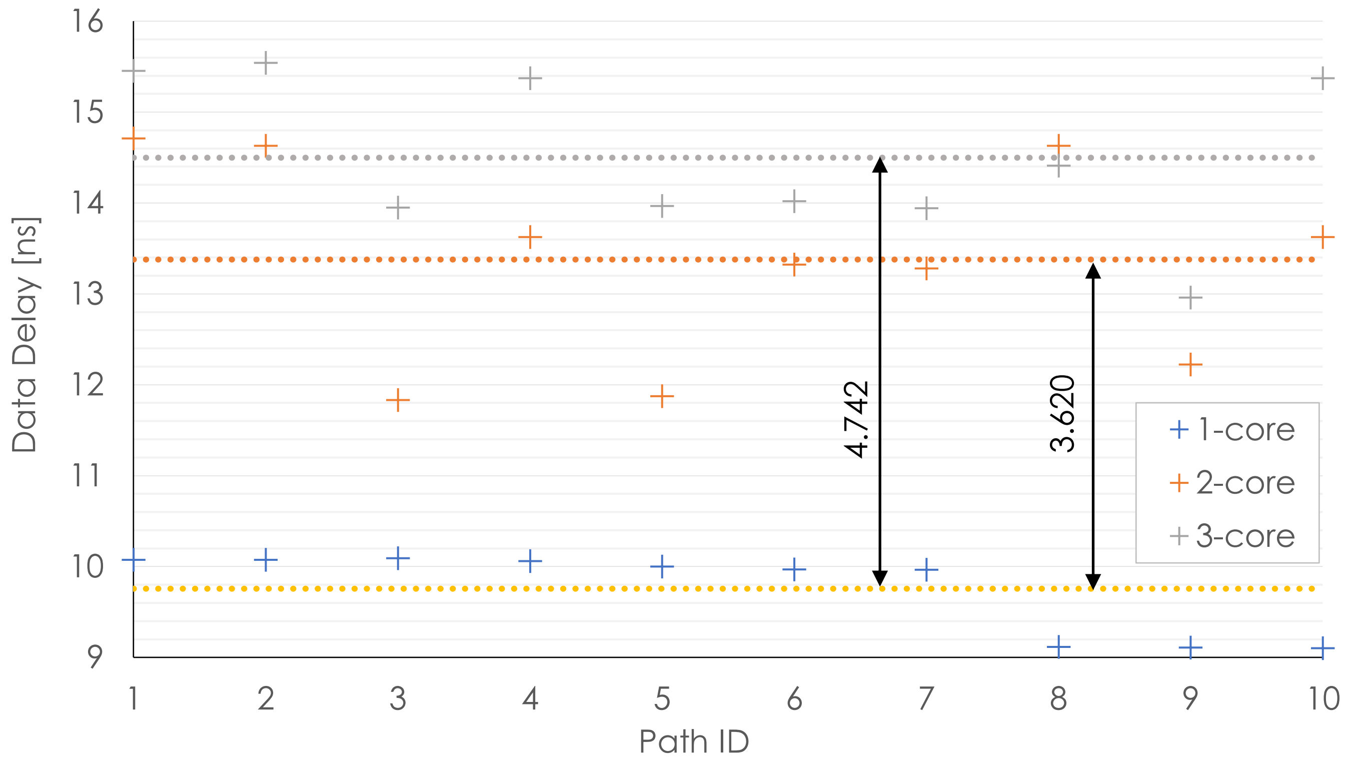
Design	$0 \rightarrow 1_{min}$ [ns]	$0 \rightarrow 1_{max}$ [ns]	$1 \rightarrow 0_{min}$ [ns]	$1 \rightarrow 0_{max}$ [ns]
1-core	2.278	2.378	2.278	2.378
2-core	2.497	2.601	2.497	2.601
3-core	2.537	2.643	2.537	2.643

Design	$0 \rightarrow 1_{min}$ [ns]	$0 \rightarrow 1_{max}$ [ns]	$1 \rightarrow 0_{min}$ [ns]	$1 \rightarrow 0_{max}$ [ns]
1-core	2.742	2.827	2.742	2.827
2-core	2.801	2.922	2.801	2.922
3-core	2.905	2.985	2.905	2.985

Design	$0 \rightarrow 1_{min}$ [ns]	$0 \rightarrow 1_{max}$ [ns]	$1 \rightarrow 0_{min}$ [ns]	$1 \rightarrow 0_{max}$ [ns]
1-core	6.605	6.667	6.605	6.667
2-core	6.587	6.646	6.587	6.646
3-core	6.683	6.743	6.683	6.743

Experimental Results

- Through the **data delay analysis**, the 10 worst routing paths in terms of timing have been evaluated



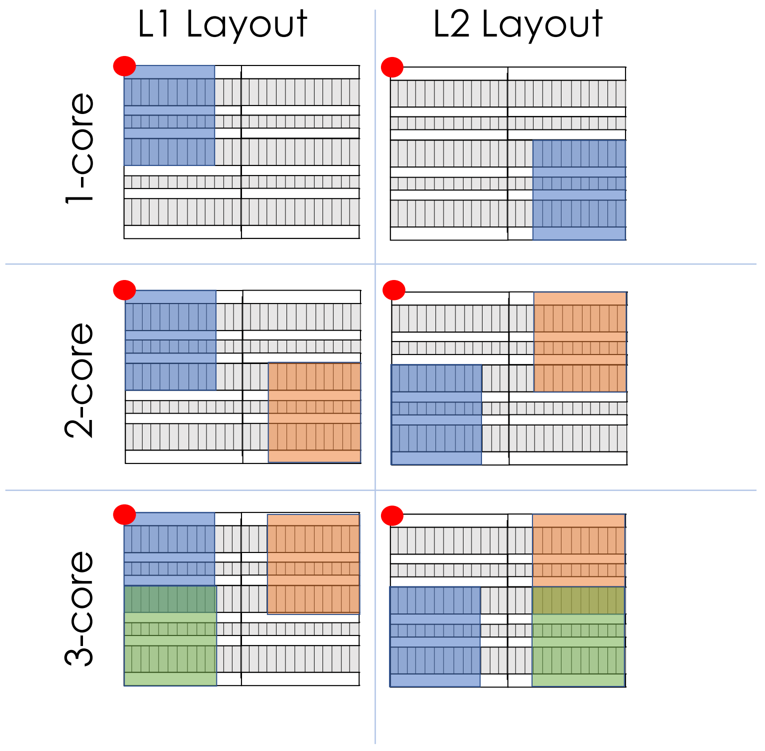
Again, resource utilization (and number of cores) affects negatively delays of nets

2-core and 3-core designs delays are comparable with clock period

Design	Max Frequency [MHz]
1-core	32.753
2-core	27.586
3-core	23.384

Experimental Results

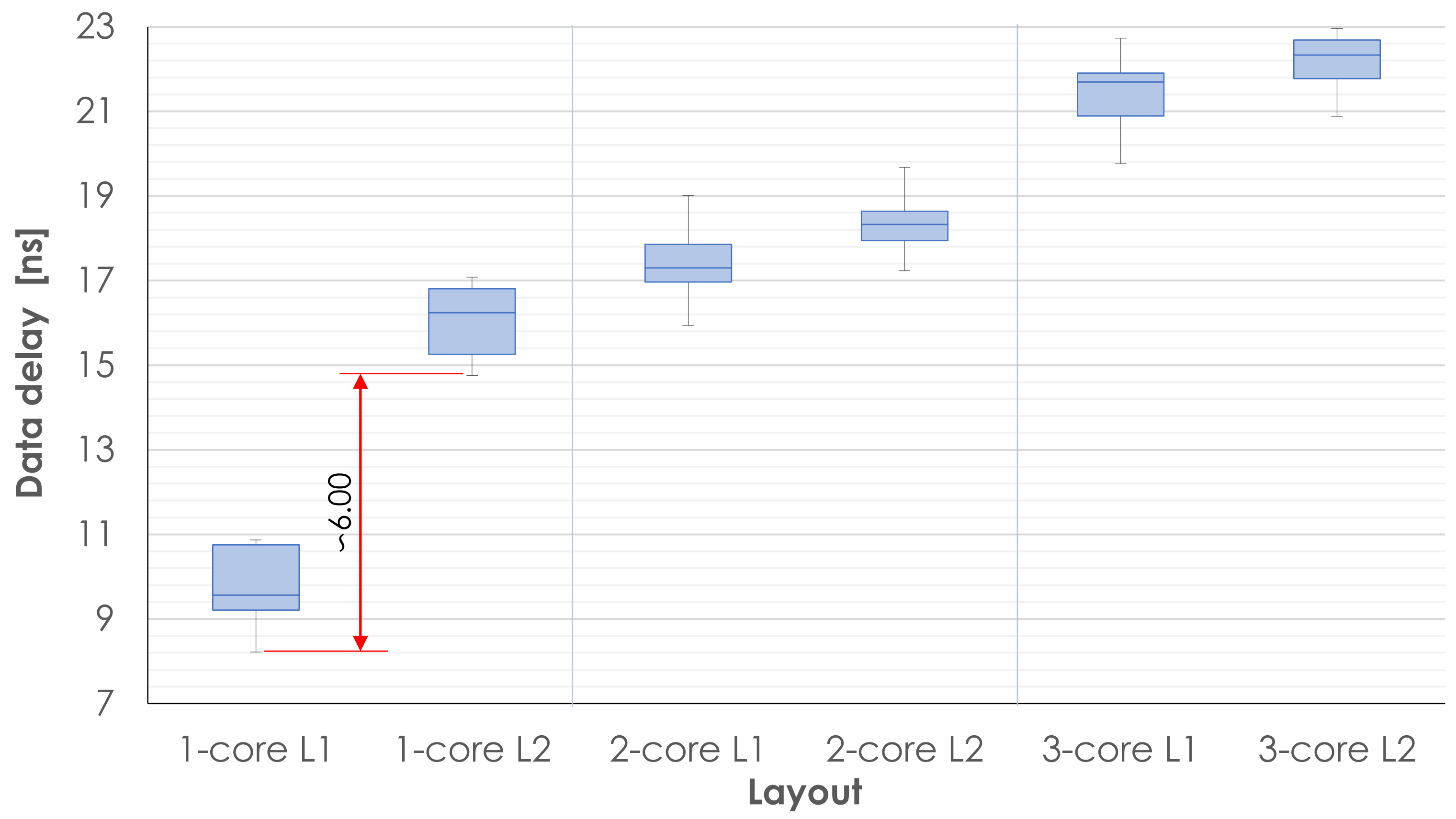
- Finally, **dependencies from cores locations** have been analyzed in terms of timing:
  - 2 different layouts have been implemented for each design
  - L1 minimizes the distance from the I/O buffers while L2 maximizes it
  - Each color represents a r-VEX core
  - The red circle represents the I/O buffer



The 3 cores overlap due to unavailability of resources

Experimental Results

- **Data Delay** analysis of the 10 worst paths with **placement influence**



Overall deterioration of the performances with higher distances to the I/O buffers

Highest difference between 1-core layouts due to the highest average distance from I/O

Degradation of other L2s appears to be less since routing paths are shorter

Politecnico di Torino

Current Work

Stark
A library for design exploration

- Allows to extract info about the implemented netlist.
- Instances, Sites and nets are instantiable objects in Python with methods associated with them

The libraries are built as Python bindings of the C++ data structures so **they are as reliable as the vendor tool itself.**

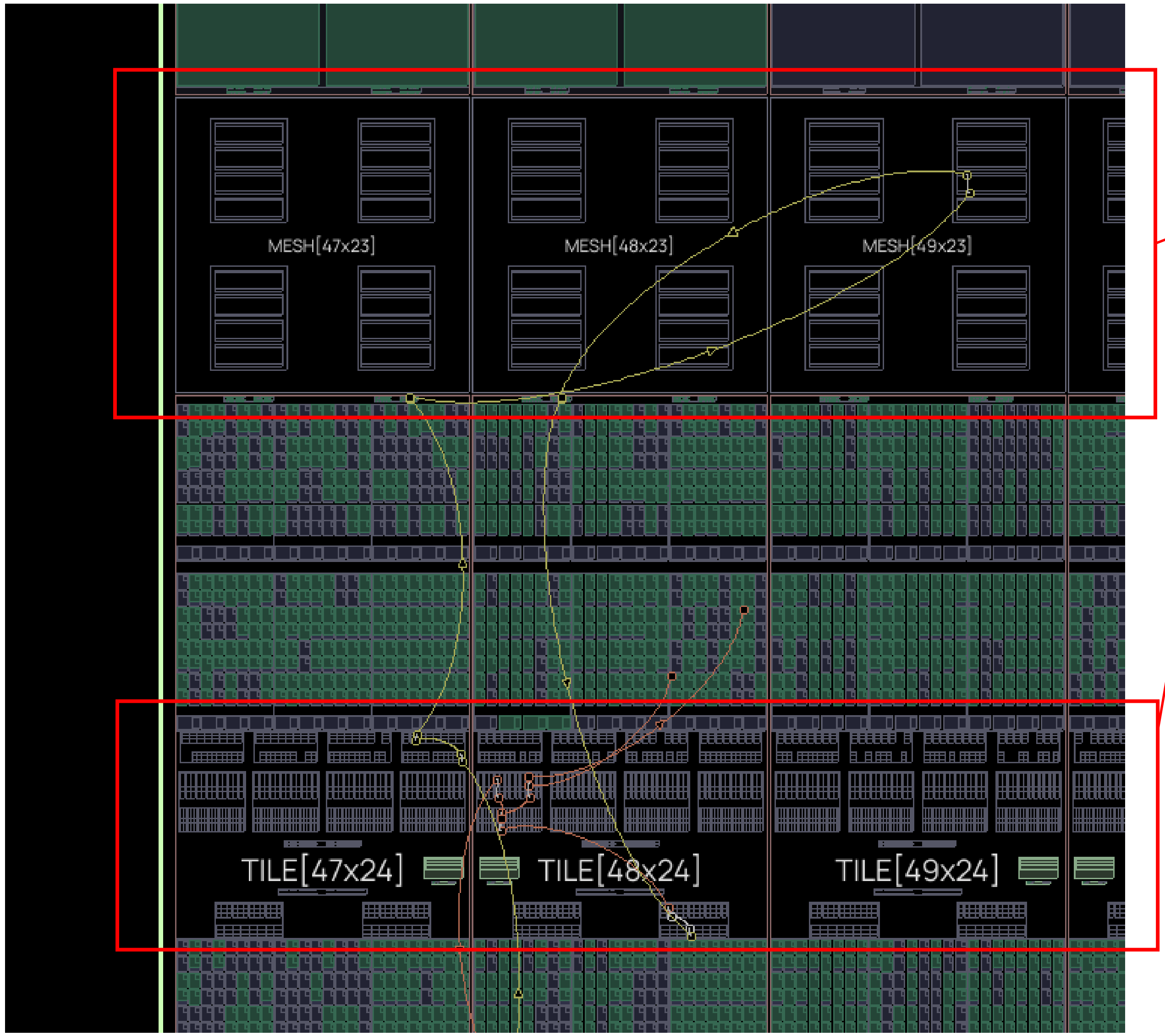
Rogers
A library for architecture exploration

- Allows to extract info about the architecture.
- Extraction of hidden modules
- Extraction of all possible routing lines with timing
- Extracting of available routing targets given a source

NG-ULTRA
Routing lines
(1.33 GB)

NG-MEDIUM
Routing lines
(84,4 MB)

Impulse Tool

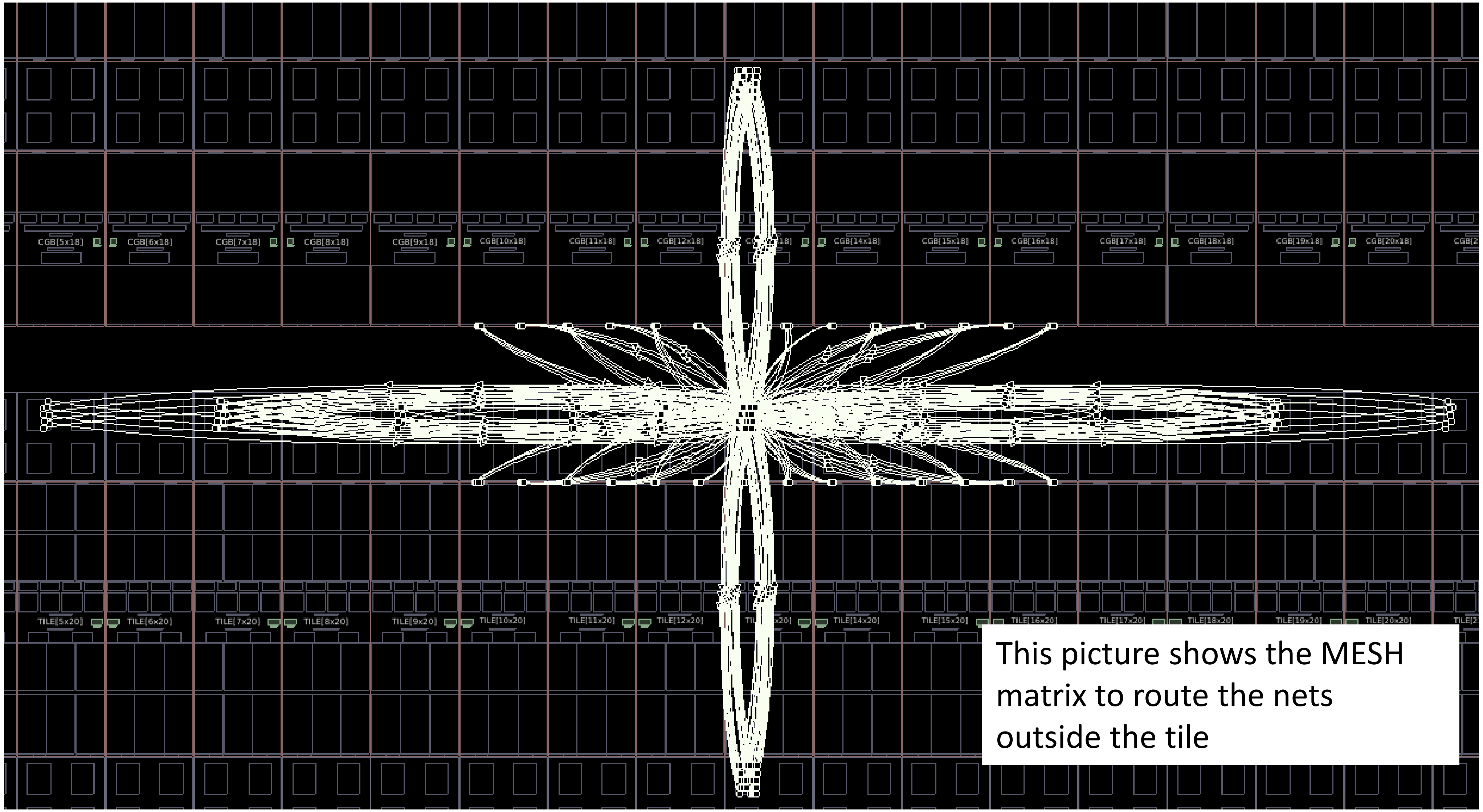


These routing resources (MESH, RI, RE, RS) are hidden in the release version of tool.

The Stark and Rogers libraries do take into account these modules when exploring the device

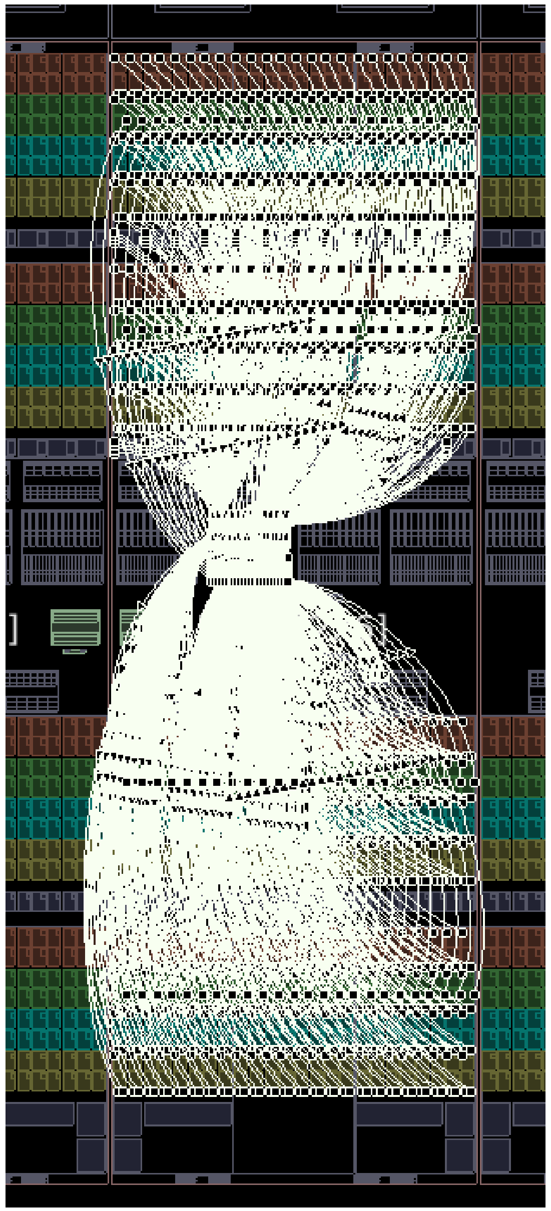
Politecnico di Torino

Current Work – Routing Topology Modeling



This picture shows the MESH matrix to route the nets outside the tile

Current Work – Detailed Routing Internal Network



This picture shows the routing lines of the internal network matrix of the tile

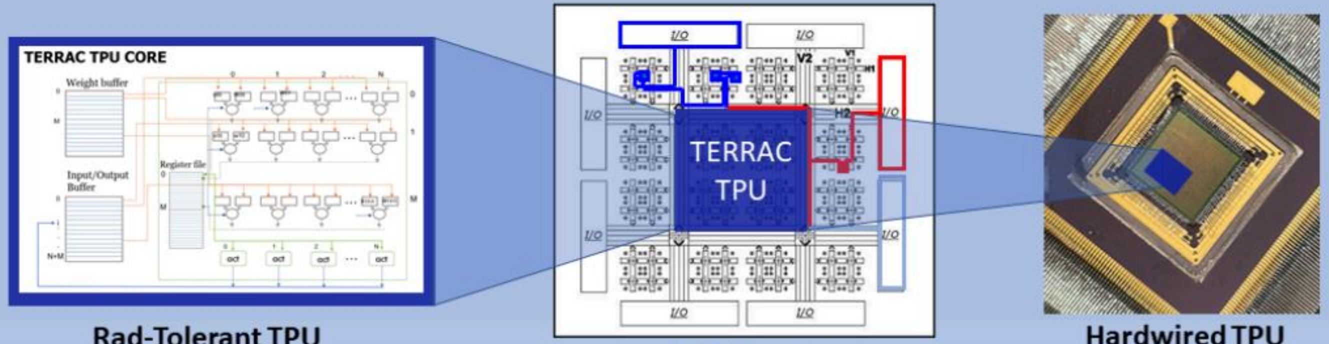
TERRAC OSIP Project

Politecnico di Torino

NanoXplore

TERRAC

Towards Exascale Reconfigurable and Rad-hard Accelerated Computing in space



Rad-Tolerant TPU Architecture

Soft-Core on BRAVE European FPGA

Hardwired TPU Feasibility Evaluation

Conclusions

- **Rad-hard technology improvements** unlocked several high-computational power applications fields such as **ML and NN accelerators,** arising the necessity to improve performances even further
- **AlexNet** has been accelerated on a **r-VEX soft processor** as benchmark on **NanoXplore NG-MEDIUM chip**
- **Timing analyses** have been performed to highlight the influence of placement in **FPGA implementations of NN accelerators**
- High criticalities have been observed while **distancing accelerator cores from the I/O buffers** of the FPGA
- Future works include the development of a place and route on NanoXplore FPGAs to improve timing performances

Thank you all for the attention

✉ Andrea andrea.portaluri@polito.it

✉ Sarah sarah.azimi@polito.it

✉ Luca luca.sterpone@polito.it

Politecnico di Torino

1859