



HAL
open science

Accurate Modeling of Fault Impact in Arithmetic Circuits

Pierre Guilloux, Arnaud Tisserand

► **To cite this version:**

Pierre Guilloux, Arnaud Tisserand. Accurate Modeling of Fault Impact in Arithmetic Circuits. DASIP: Conference on Design and Architectures for Signal and Image Processing (Demo Night), Oct 2016, Rennes, France. , 2016. hal-01404772

HAL Id: hal-01404772

<https://inria.hal.science/hal-01404772v1>

Submitted on 29 Nov 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

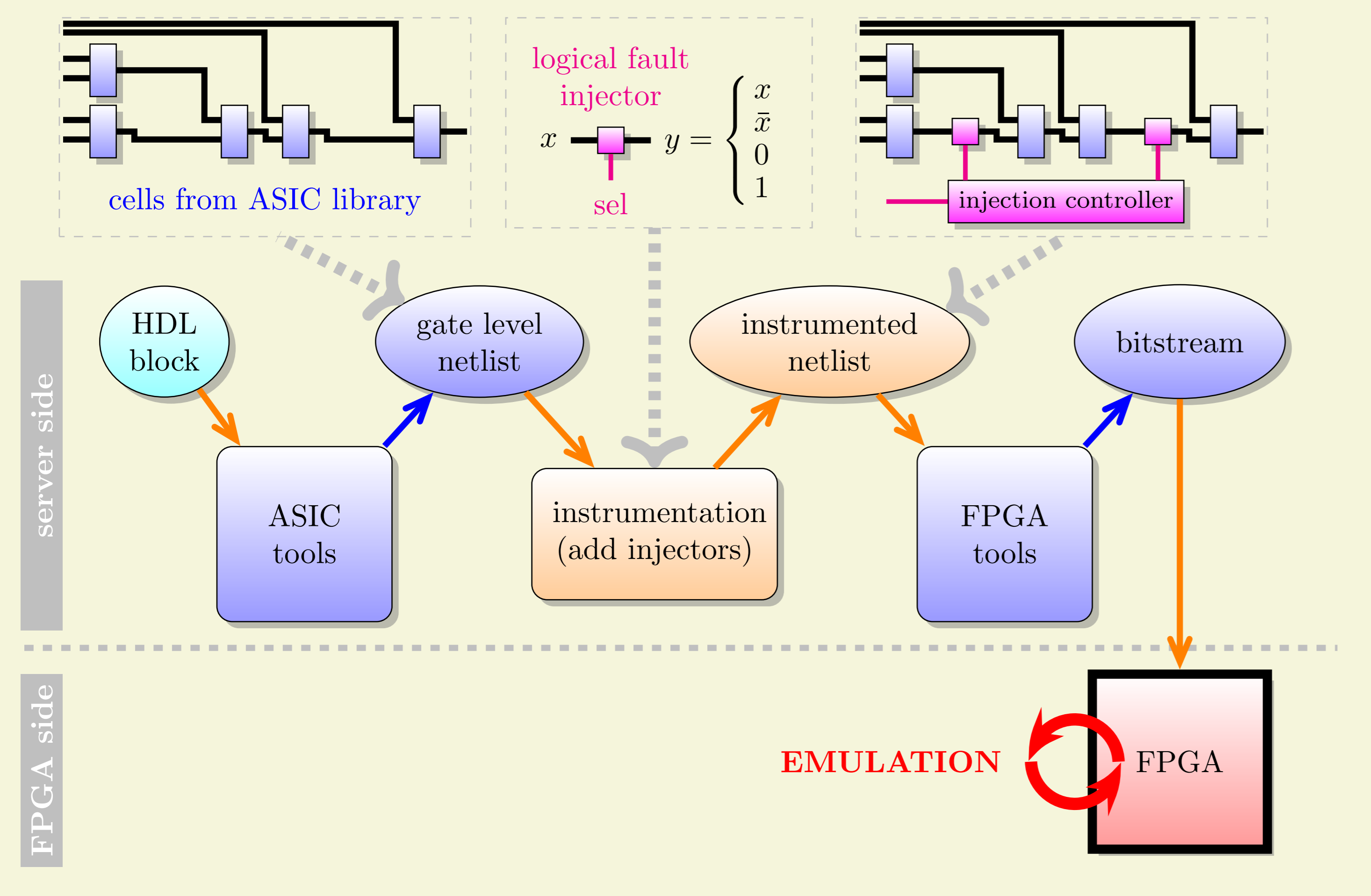
L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

1. Objective: Design of Robust Arithmetic Circuits

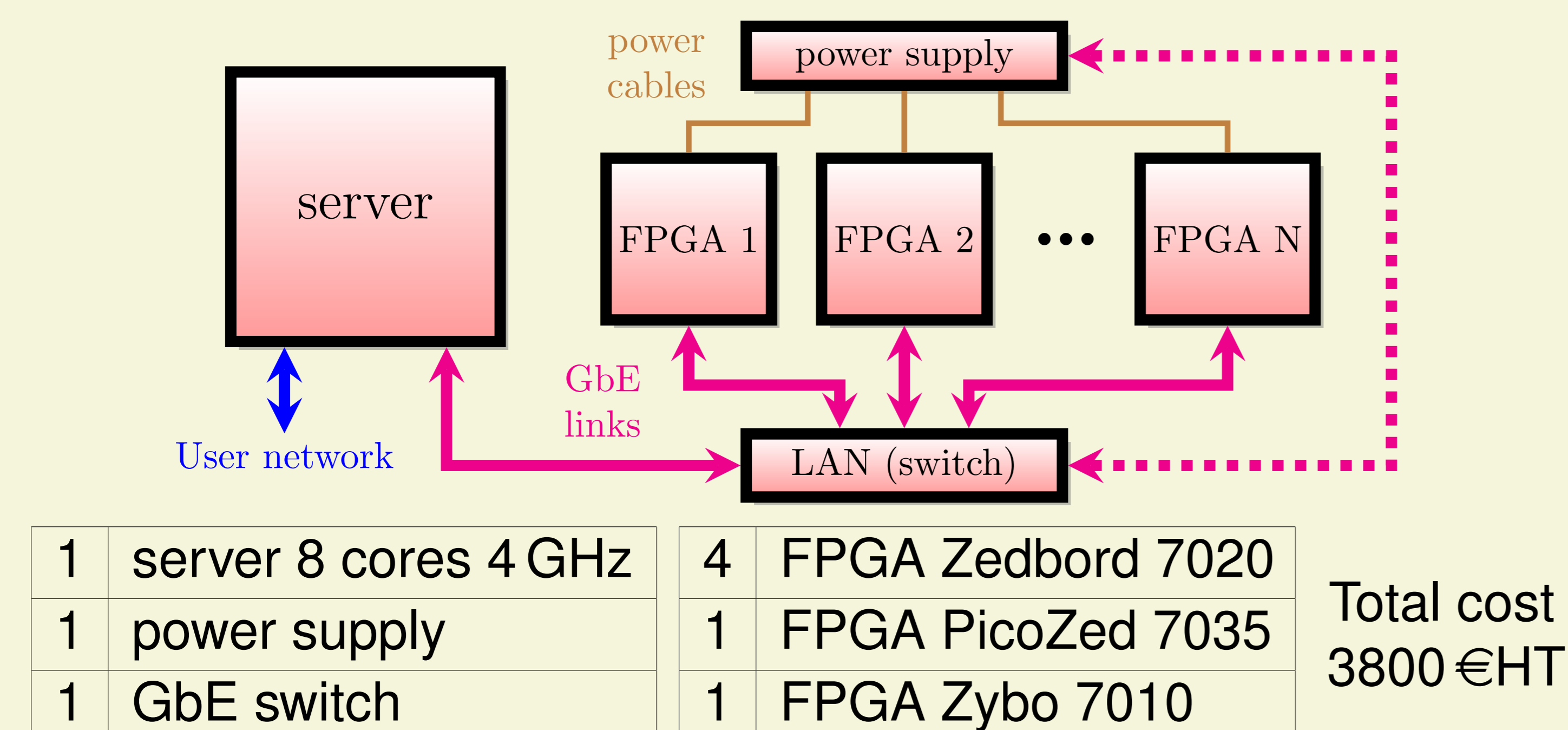
- Study links between **fault detection/tolerant** solutions and
- ▶ **types of operations** $\pm, \times, \times_{\text{est}}, \div, \sqrt{}, \sin, \cos, \log, \dots$
- ▶ **computation algorithms** and **representations** of numbers
- ▶ **optimizations** at architecture and circuit levels
- ▶ **mathematical error** due to faults and rounding modes

Applications: signal/image processing, embedded computing, protections for cryptographic devices, ...

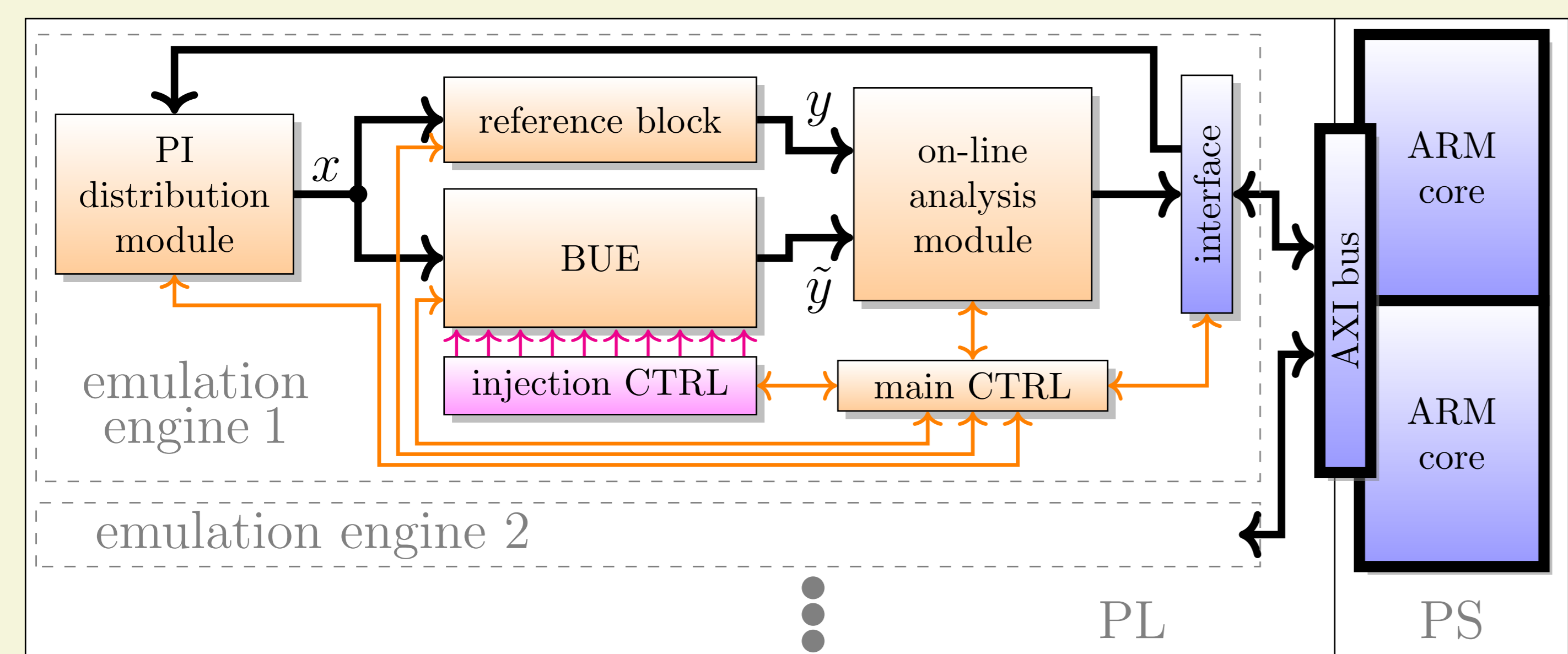
2. Intensive Logical Fault Simulations: Emulation on FPGA



3. Hardware Platform

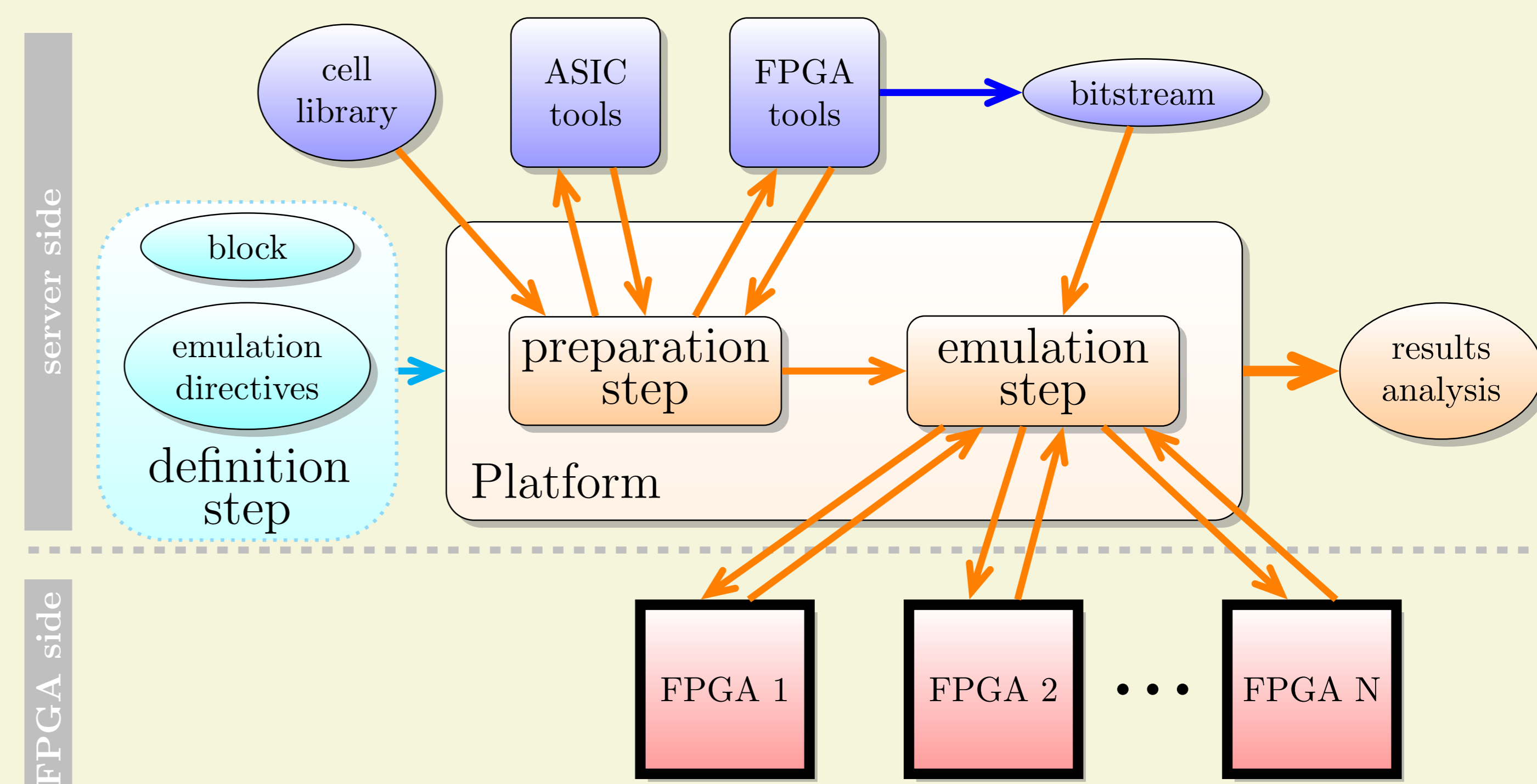


4. Emulation Engine(s) in FPGA



reference operator computes $y = f(x)$
 faulted operator computes $\tilde{y} = \tilde{f}(x)$
 mathematical error analysis $\varepsilon = |y - \tilde{y}|$

5. Tools Flow



Possible locations for logical fault injection:

- ▶ manual on selected internal nets (for debug)
- ▶ random on internal nets
- ▶ random over circuit area (based on cells area)

6. Example: Multiplier with Residue Code Detection

16×16-bit integer multiplier, moduli $m \in \{3, 7, 15\}$, with

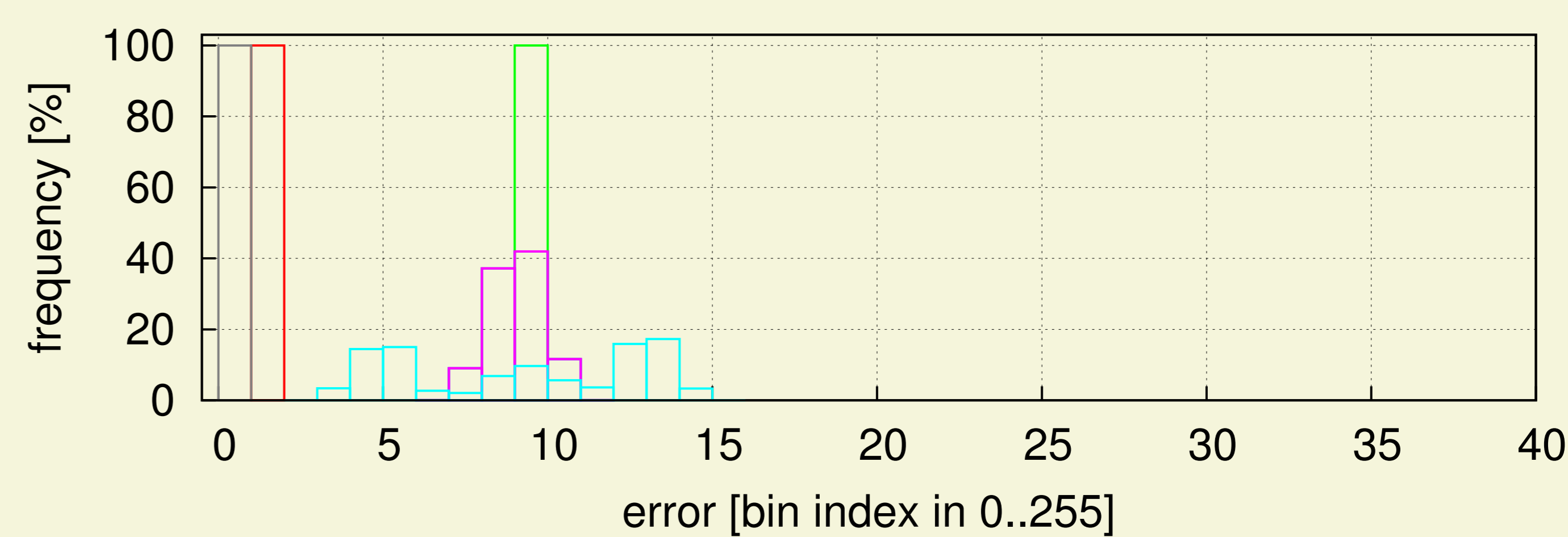
$$(a \times b) \bmod m = ((a \bmod m) \times (b \bmod m)) \bmod m$$

step	duration	area	delay
Impl. ASIC orig. mult.	25 s	1164 gates	–
Impl. FPGA orig. mult.	43 s	444 LUT	9.4 ns
Impl. FPGA instrum. mult.	48 s	453 LUT	10.1 ns
Impl. FPGA emul. engine	389 s	1714 LUT + 1 BRAM	11.0 ns
boot petalinux Zedboard	15 s	–	–
emulation 1 fault & 1 EE	94.5 s	–	–

Comparison to a GABA simulation in software (Vivado 2014.4):

×6880 speedup for 1 emulation engine

Error distribution for **exhaustive** tests (2^{32} inputs) and various single faults (1 color = 1 fault):



Residue code implementation results:

modulo m	area [LUT]	delay [ns]	measured avg. detection rate [%]
3	525	10.4	66.7
7	526	10.1	85.6
15	553	13.0	93.3

7. Future Works

100 EEs @ 100 MHz	/ second	/ hour	/ day
nb. faults or PIs	$100 \cdot 10^6 \times 100 \approx 2^{33}$	$\approx 2^{45}$	$\approx 2^{49}$

References:

Arithmetic operators library from R. Zimmermann (http://www.iis.ee.ethz.ch/~zimmi/arith_lib)
 P. Guilloux & A. Tisserand, Compas 2016, French Conf. on Parallelism, Architecture and System