December 8-10 | Virtual Event

A Tiny RISC-V Floating-Point Unit

Luca Bertaccini
PhD Student
ETH Zurich

#RISCVSUMMIT

A Tiny RISC-V Floating-Point Unit

Luca Bertaccini
PhD Student at Digital Circuits and Systems Group (ETH Zurich), part of the PULP team.

Internet of Things

- **Billions of devices** gathering and sending data to servers
- Processing **on the edge** to save bandwidth and energy
- More and more **processing** is done on the edge
- Many existing algorithms require **FP arithmetic**

Floating-Point Supports

Area-optimized cores occupy 10-20kGE

FPUs introduce **large area** overhead

- ~20kGE for single-precision
- ~50kGE for double-precision

SW emulation libraries introduce **large code size** overhead

- ~5kB for single-precision support
- ~15kB for double- and single-precision
- More system memory required

Why a Small FPU?

Large **area** overhead for full-fledged **FPU**

Large **code size** overhead for **SW emulation**

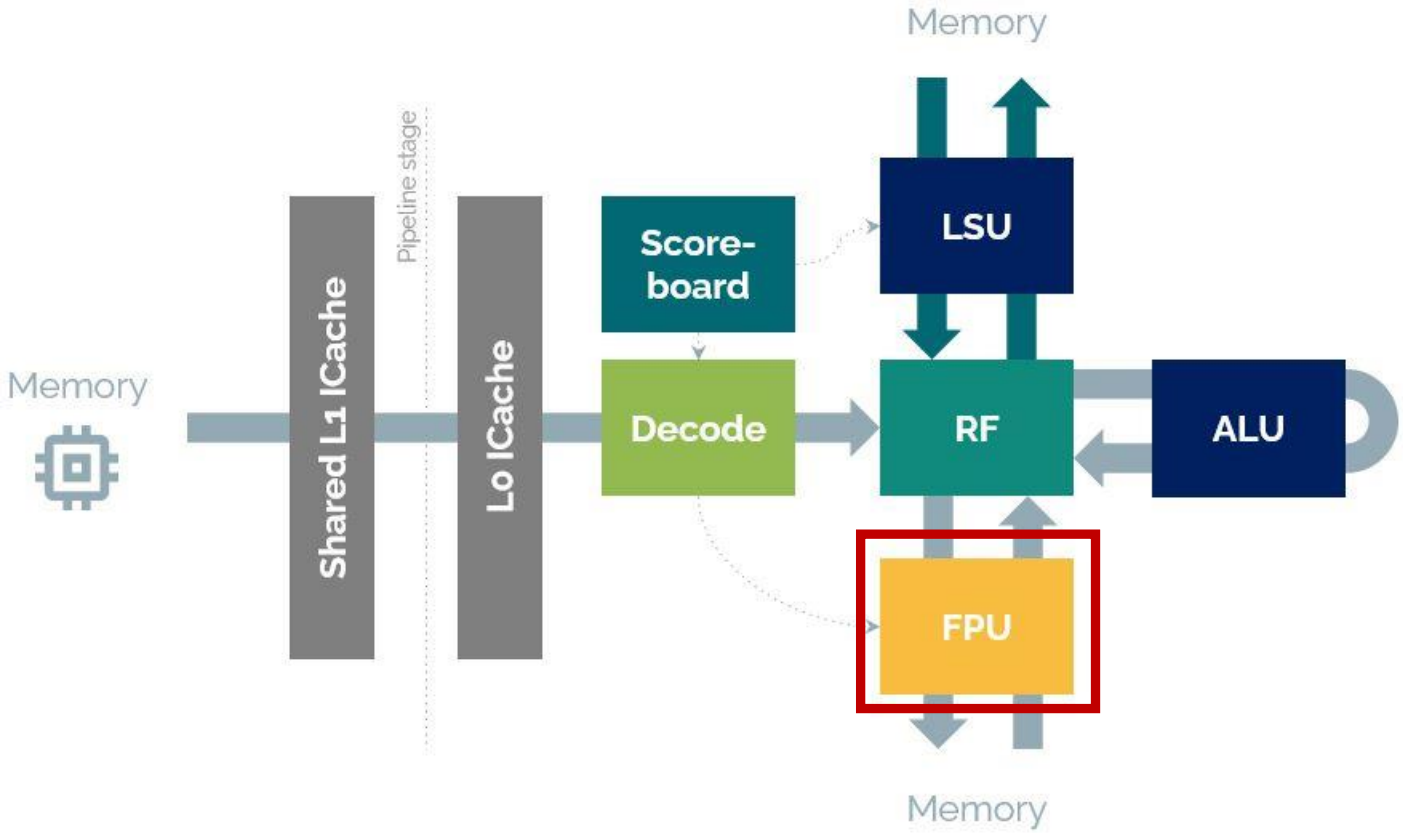
Not affordable for **low-cost MCUs**

Need for a **tiny FPU**

Snitch

- Snitch as host system
- Snitch is a **RV32IMAFD** core
- Snitch has been designed for high-performance
- Snitch includes an **open-source multi-format RISC-V FPU** optimized for high performance and energy-efficiency (FPnew*)
- The **integer** Snitch **core** is **optimized for area**

- Why not coupling a single-core Snitch with a **tiny FPU**?

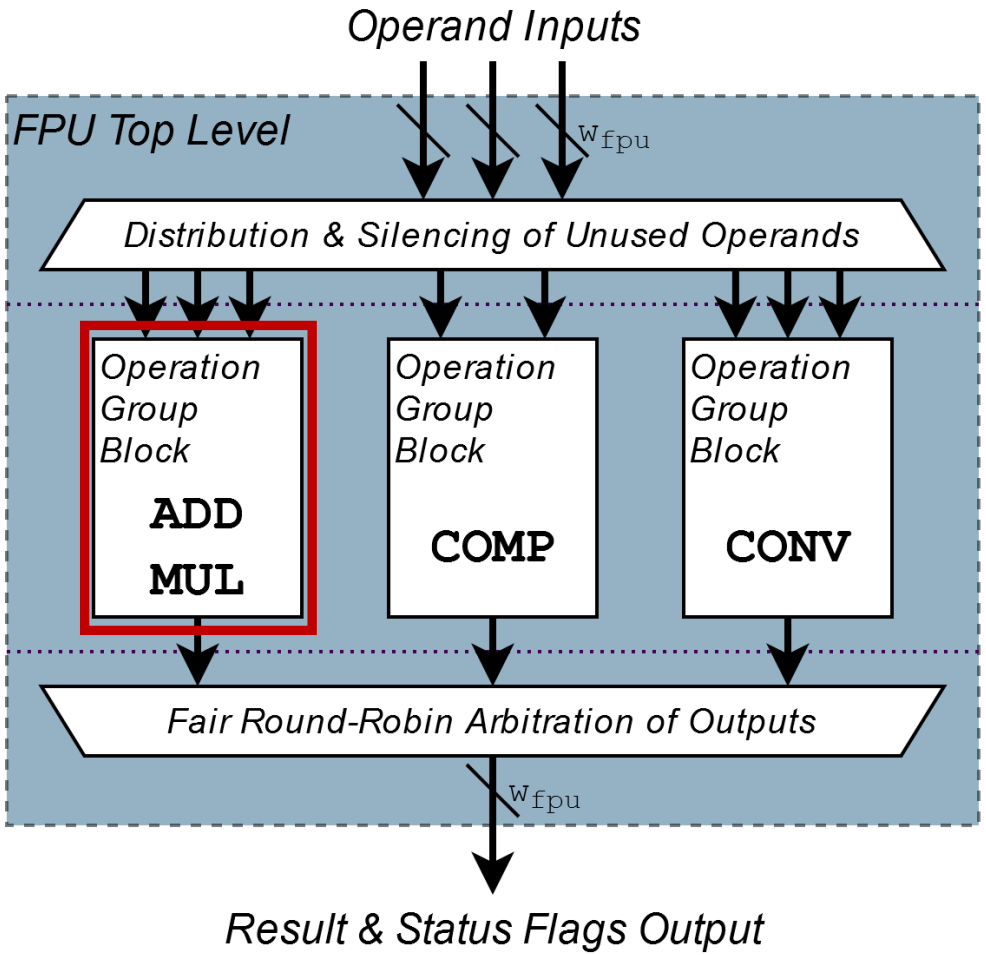


*https://github.com/pulp-platform/fpnew/

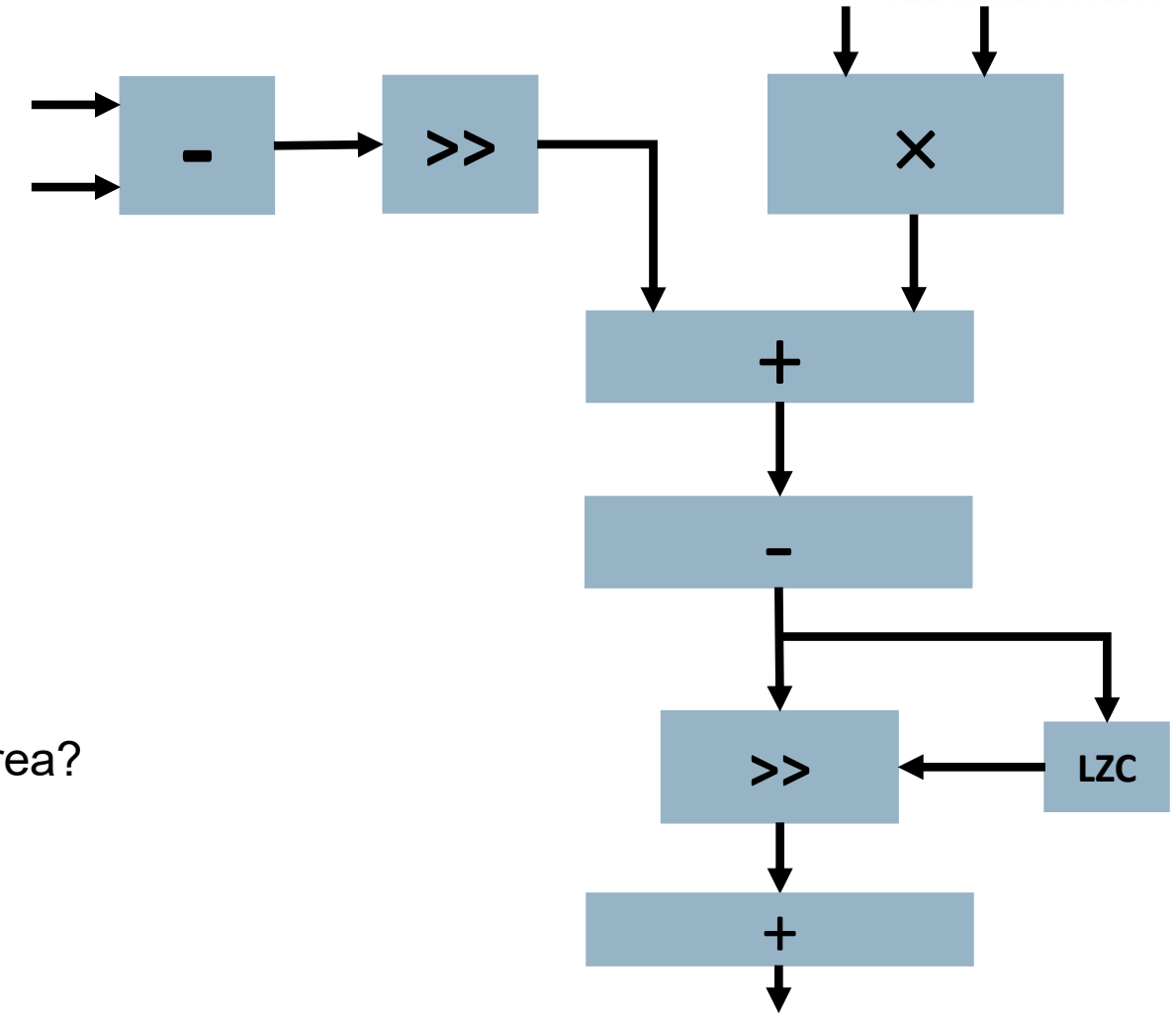
From Fast-FPU to Tiny-FPU

Snitch's FPU (Fast-FPU):

- **Modular** (ADDMUL, COMP, CONV) and multi-format
- **High-performance** and **energy-efficient** FPU
- **Large area** and fully **combinatorial**
- ADDMUL is the largest module



From Fast-FPU to Tiny-FPU



ADDMUL is the largest module:

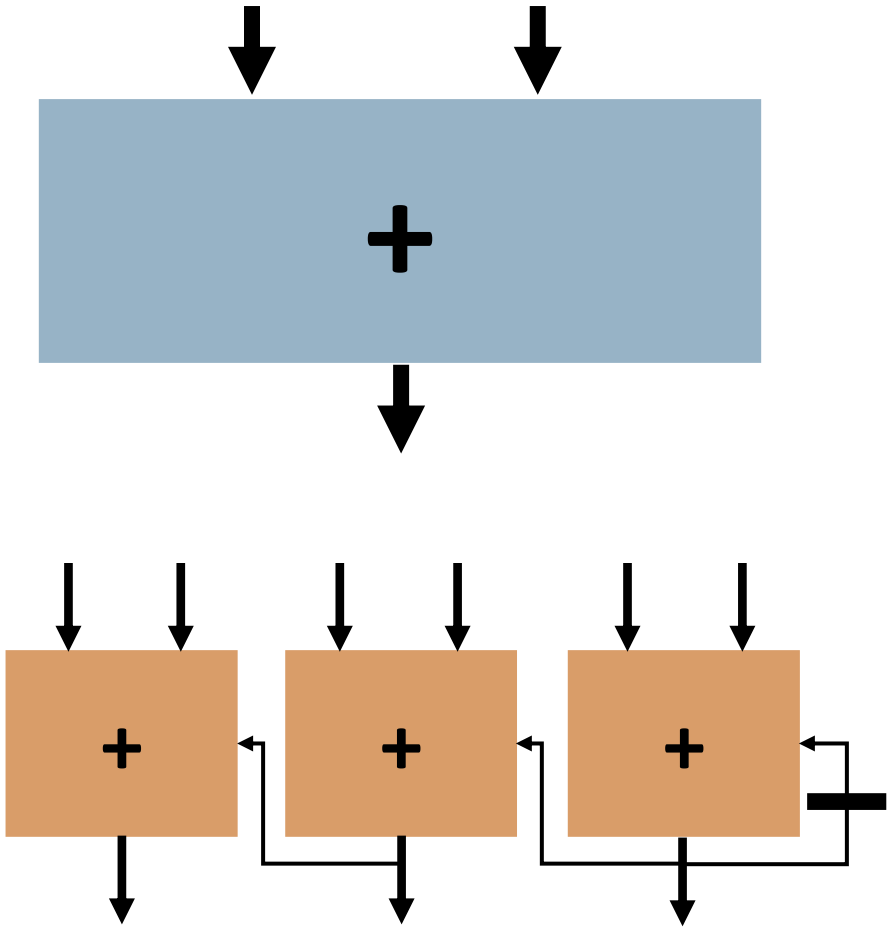
- **Multiple large adders**
- **Multiple large shifters**
- **One large multiplier**

How can we optimize Snitch's FPU for area?

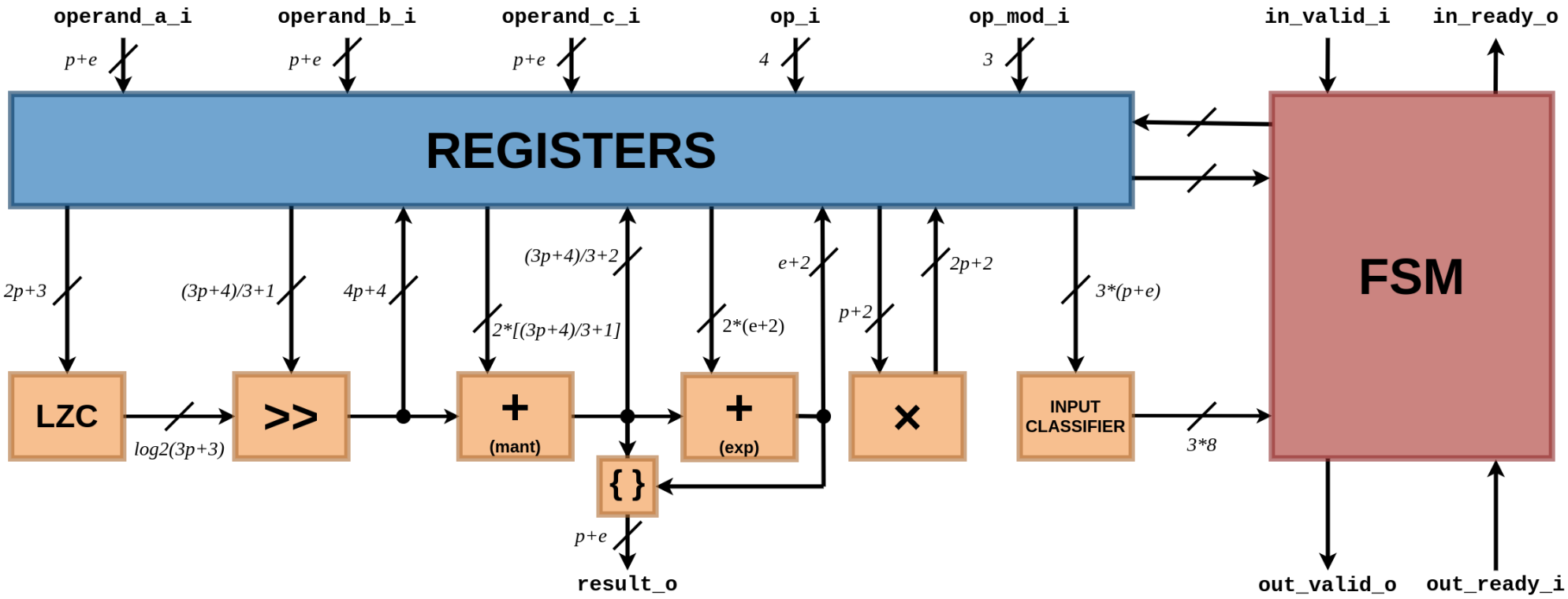
Tiny-FPU - Trading area for latency

- Two versions:
  - **Double-precision** Tiny-FPU (with support for single-precision)
  - **Single-precision** Tiny-FPU

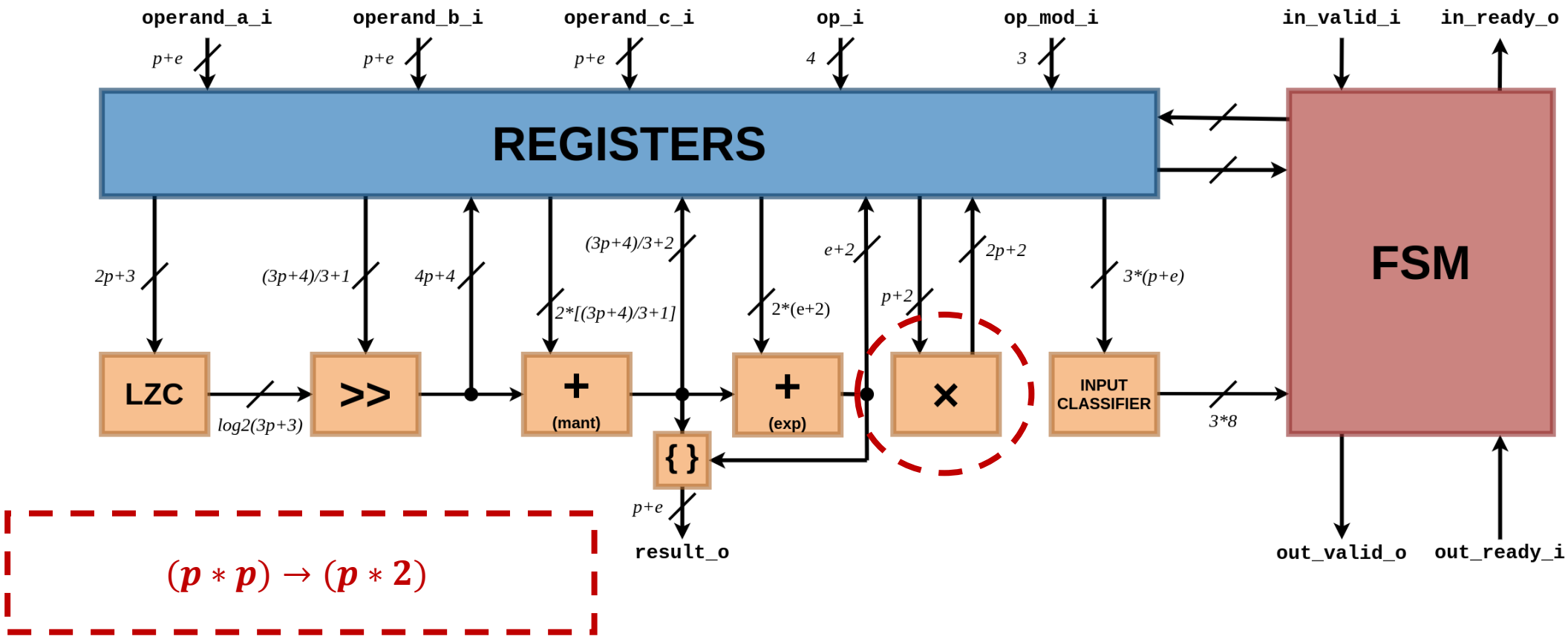
- Iterative, **multi-cycle execution**
- **Reuse** datapath **resources** in a time-multiplexed fashion
- **Maximize** internal **register utilization**



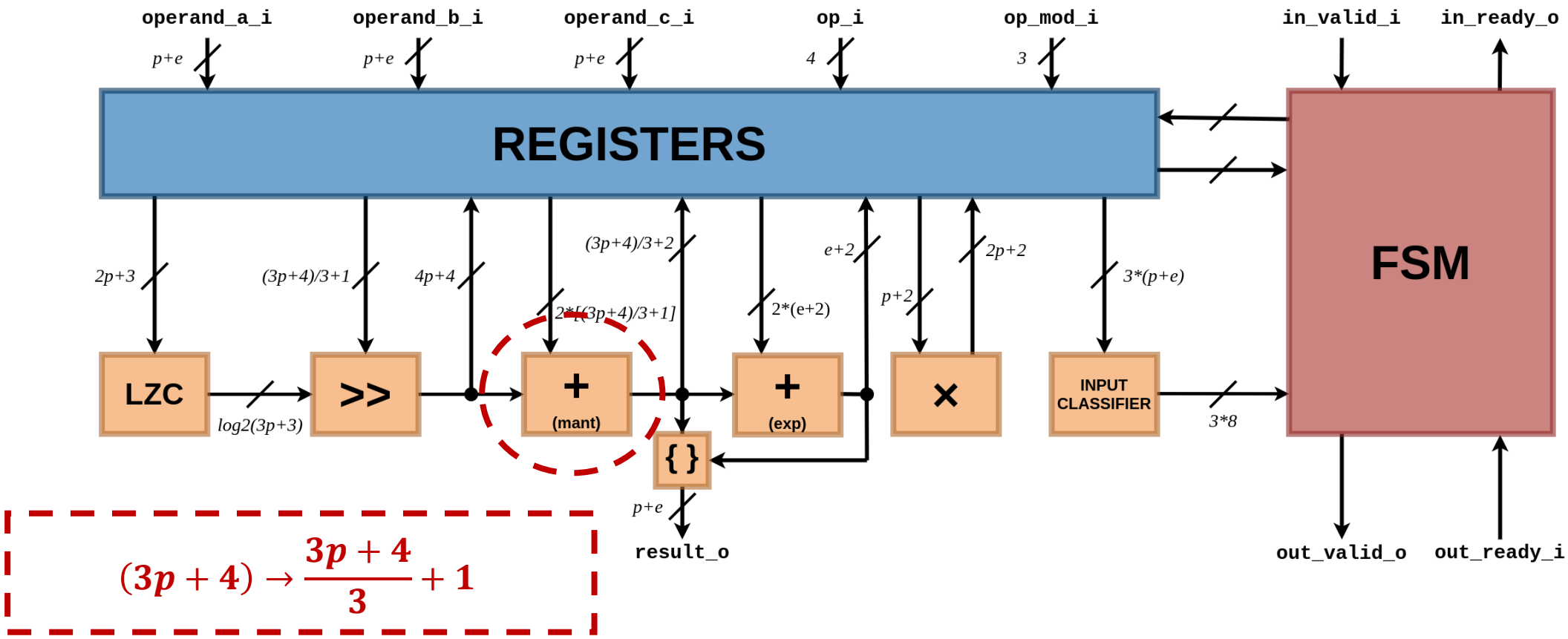
Tiny-FPU



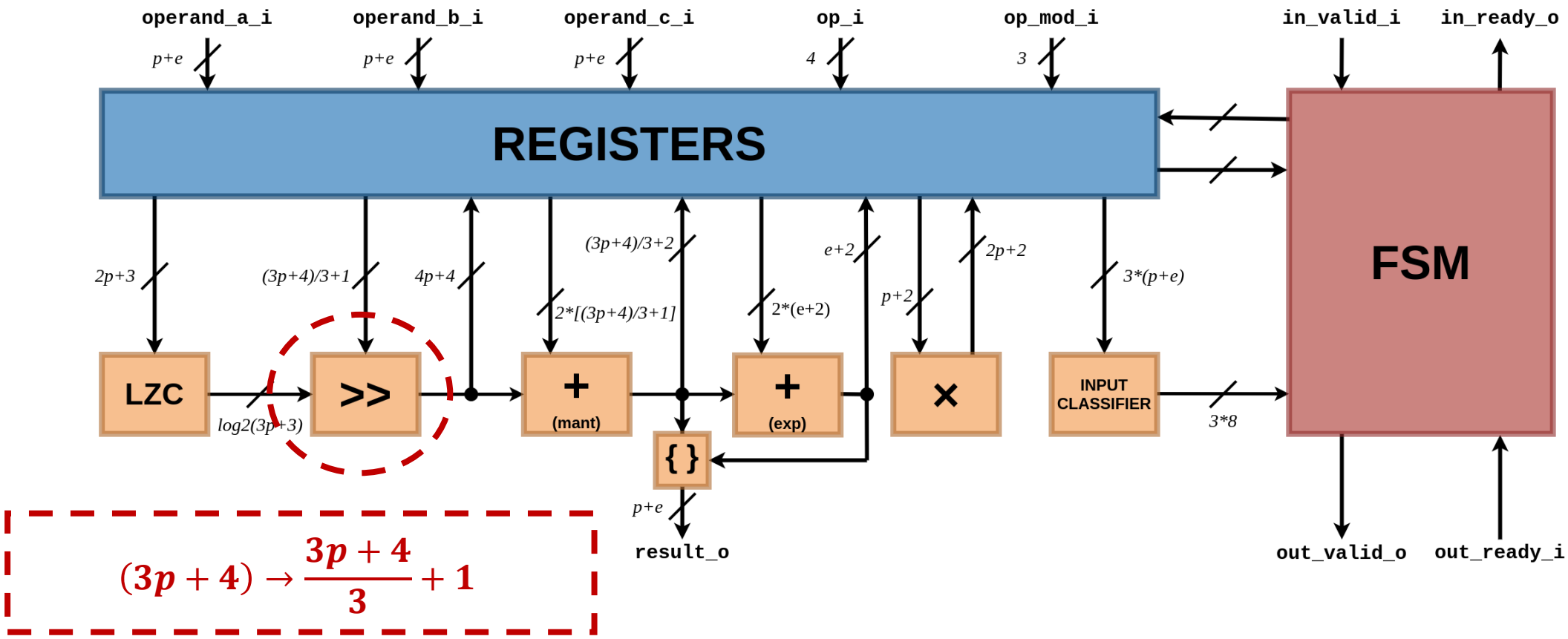
Tiny-FPU



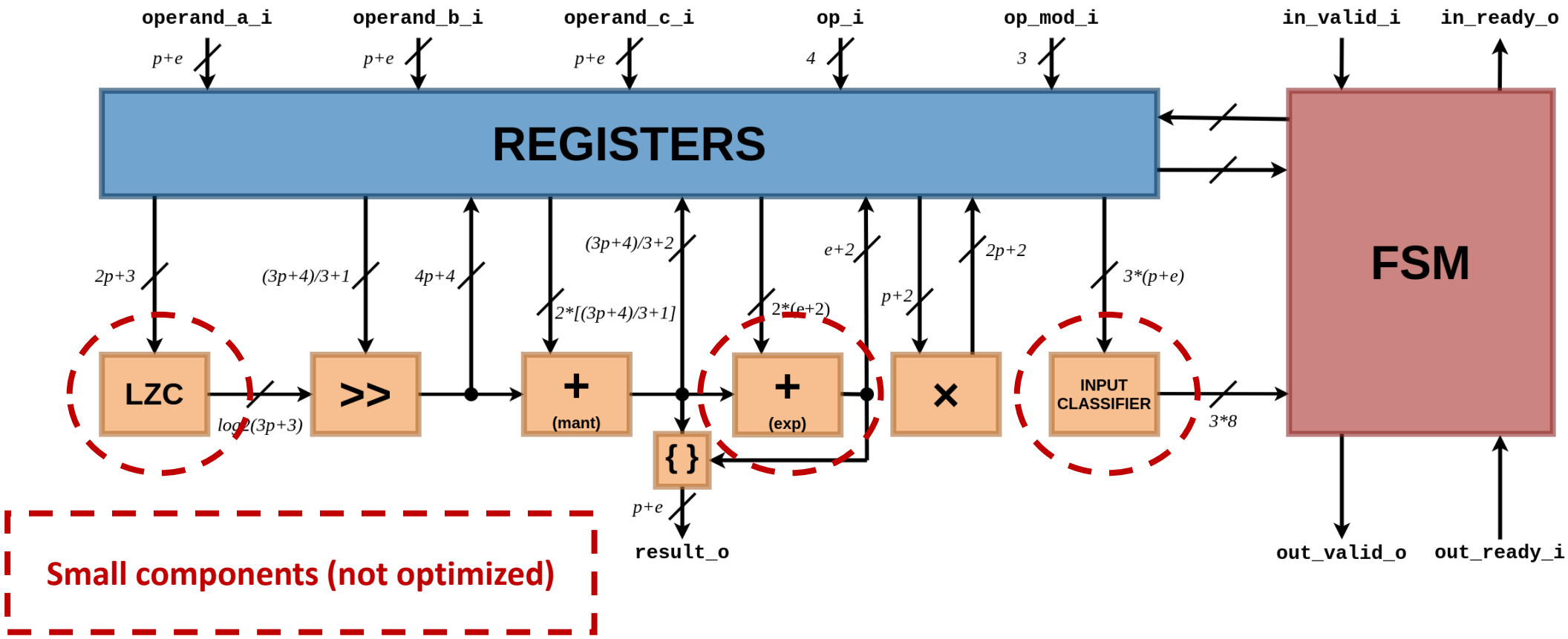
Tiny-FPU



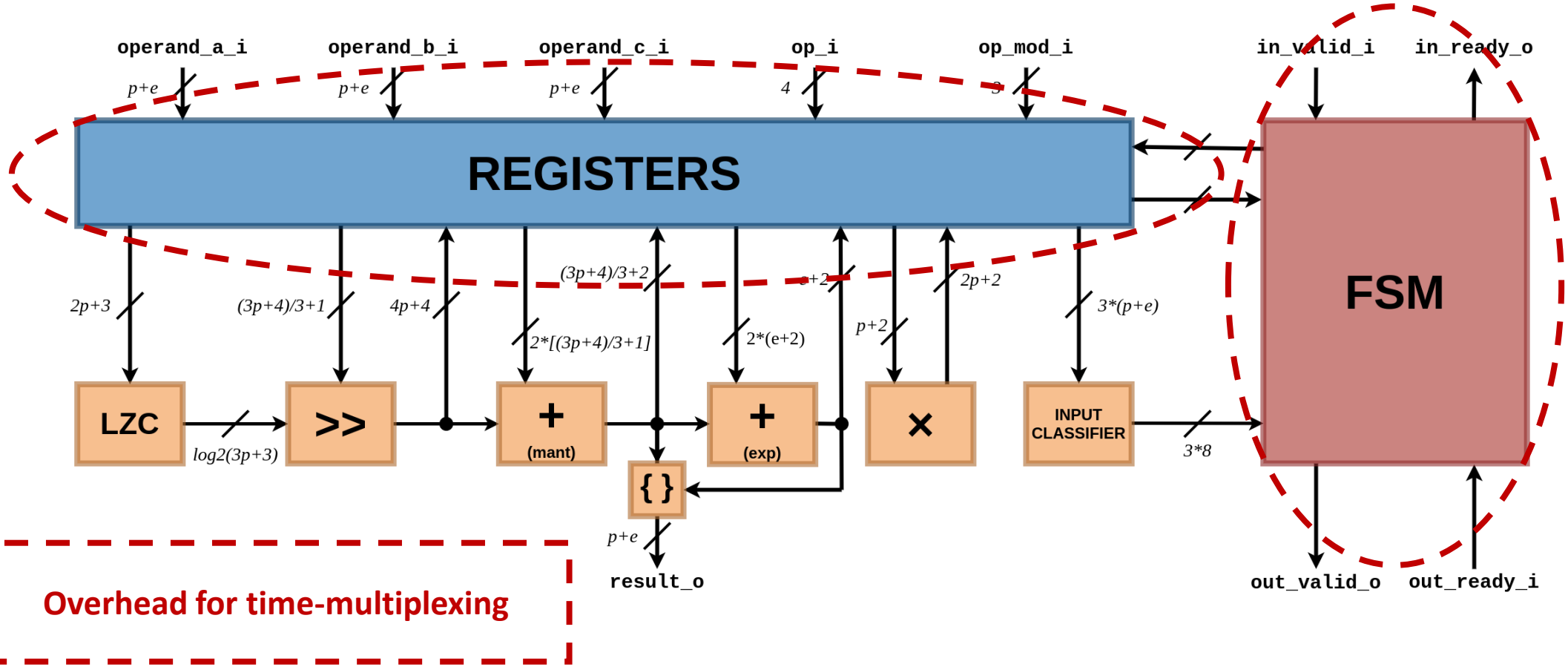
Tiny-FPU



Tiny-FPU

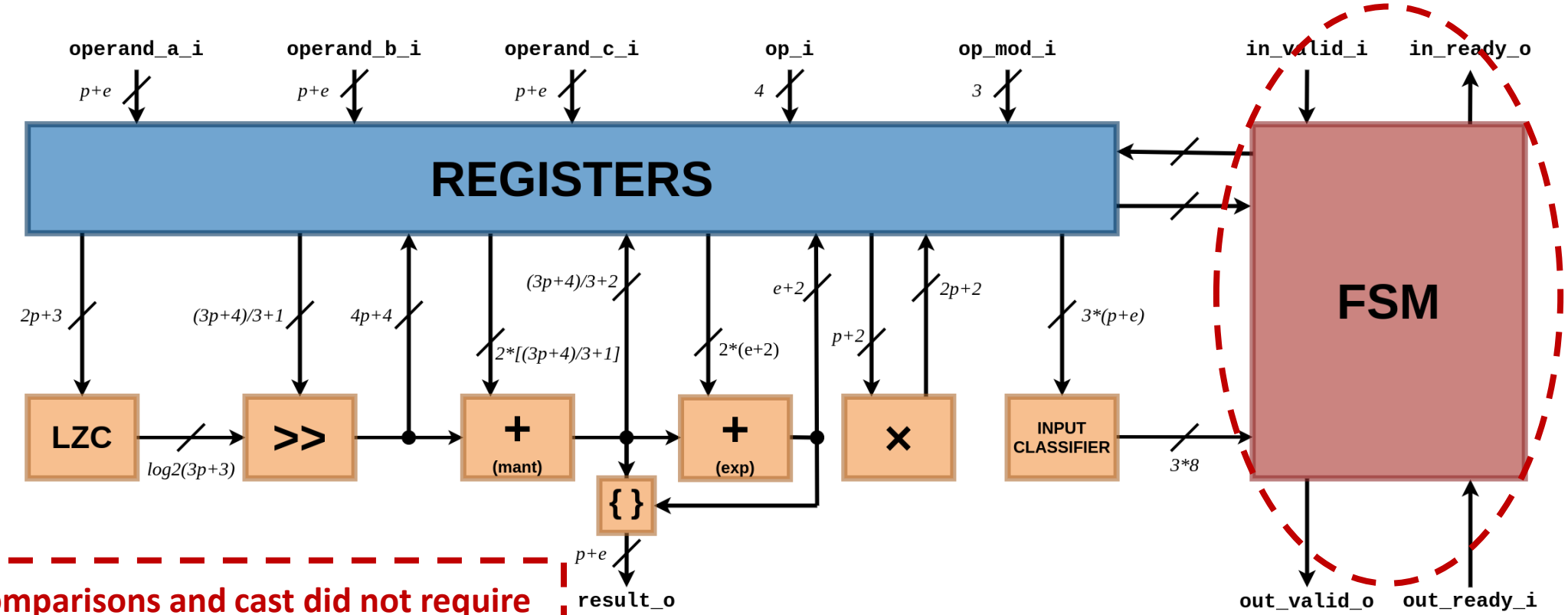


Tiny-FPU



Overhead for time-multiplexing

Tiny-FPU



Comparisons and cast did not require additional arithmetic components

Tiny-FPU - Performance

- **`fdiv/fsqrt`** not supported (**SW** emulated, option to GCC compiler `-mno-fdiv`)
- When emulating FP via SW (`libgcc` functions), even `fadd` and `fmul` can take hundreds of cycles

	Latency [cycles]					
	`fmadd`	**`fadd`**	**`fsub`**	**`fmul`**	**comparison**	**cast**
FP32	21-24	10-13	10-13	18	2	9
FP64	36-39	10-13	10-13	33	2	9
FP32 (on FP64 datapath)	22-25	10-13	10-13	19	2	9

Five Snitch Implementations

To evaluate our Tiny-FPU, we considered five Snitch implementations

• **Snitch-int :**	**Snitch**	+	`libgcc`
• **Snitch-tiny64:**	**Snitch**	+	**FP64 Tiny-FPU**
• **Snitch-fast64:**	**Snitch**	+	**FP64 Fast-FPU**
• **Snitch-tiny32:**	**Snitch**	+	**FP32 Tiny-FPU**
• **Snitch-fast32:**	**Snitch**	+	**FP32 Fast-FPU**

Benchmarks

Synthetic Benchmarks:

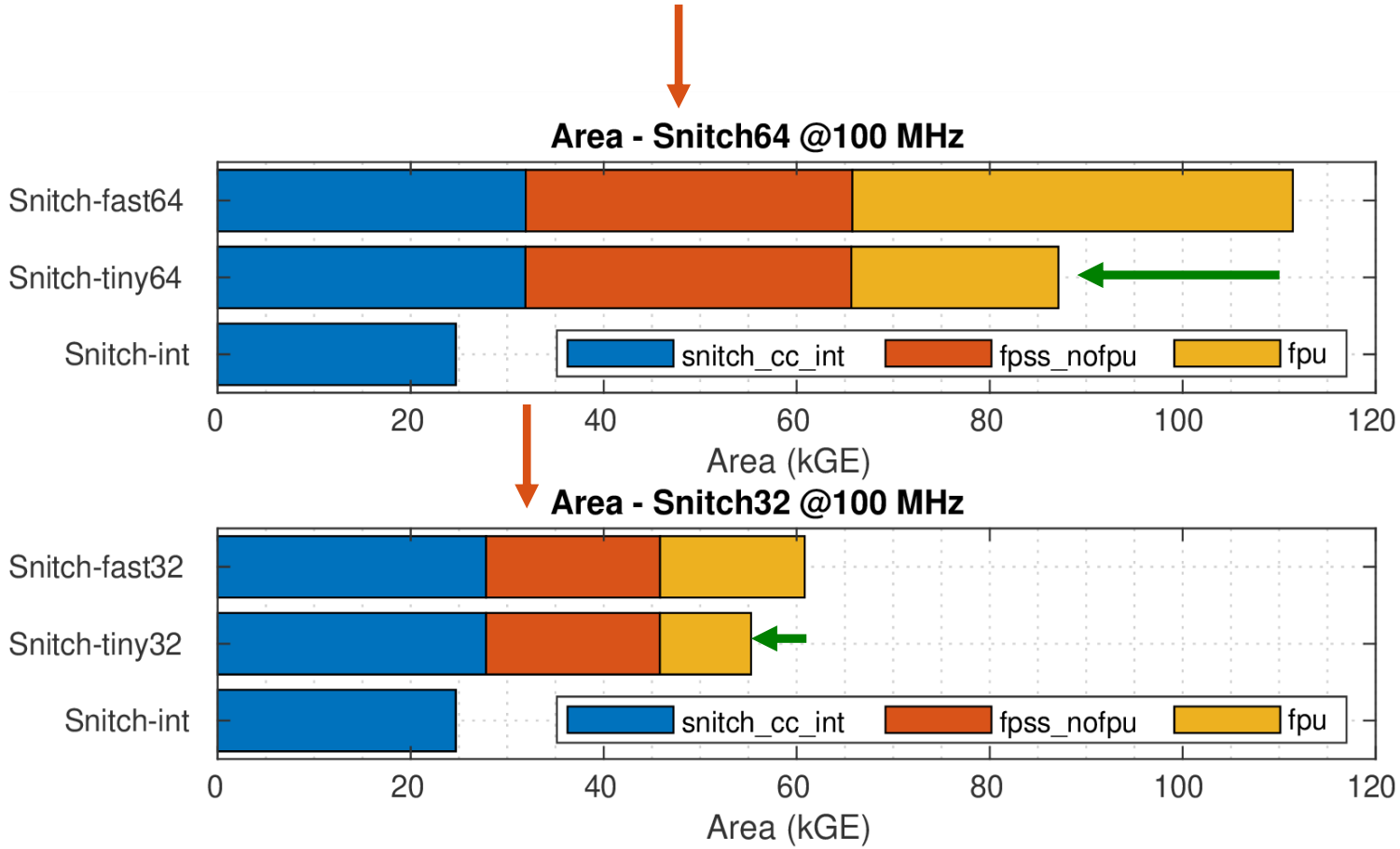
- Two matrix multiplications
- One integer, one FP
- Tuning FP intensity
- (%FP from *0.07% to 53%*)

Real Benchmarks:

- **`fann`** (*%FP = 21%*)
- **`conv2d`** (*%FP = 20%*)
- **`knn`** (*%FP = 7%*)
- **`fixed-point fann`** (*%FP=0%*)

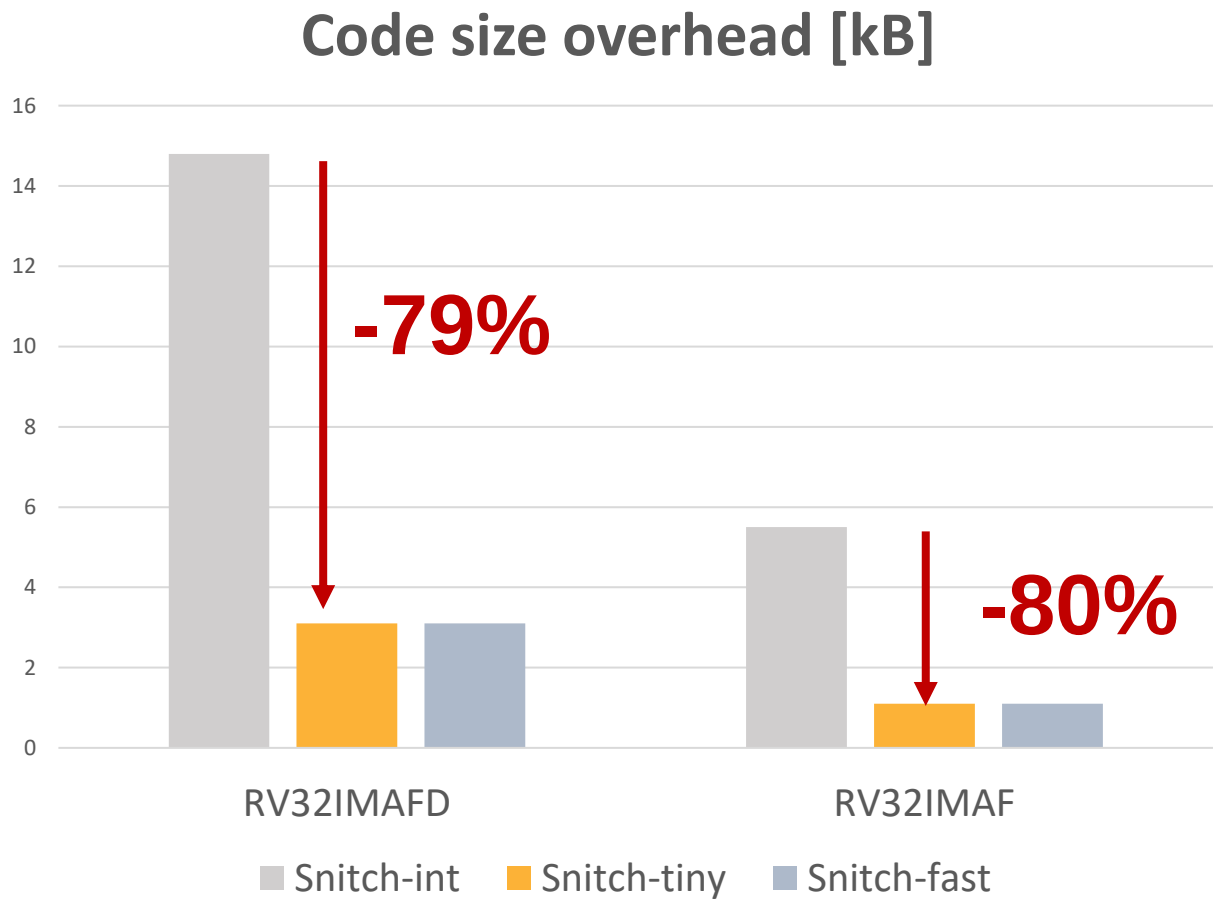
Results - Area

- GF22FDX @100MHz
- **Tiny-FPU** is **53%** (DP) and **37%** (SP) **smaller** than **Fast-FPU**
- RISC-V defines separate Register File (RF) for FP instructions
- **RF** occupies around **70%** of the area overhead to support D- and F-extension not dedicated to the FPU) and more than Tiny-FPU



Tiny FPU reduces Code Size

- Code size overhead for a full SW emulation library
- The FPUs need just the functions to emulate **`fdiv`** and **`fsqrt`** on the integer datapath
- Code size overhead up to **80% smaller** when implementing Tiny-FPU



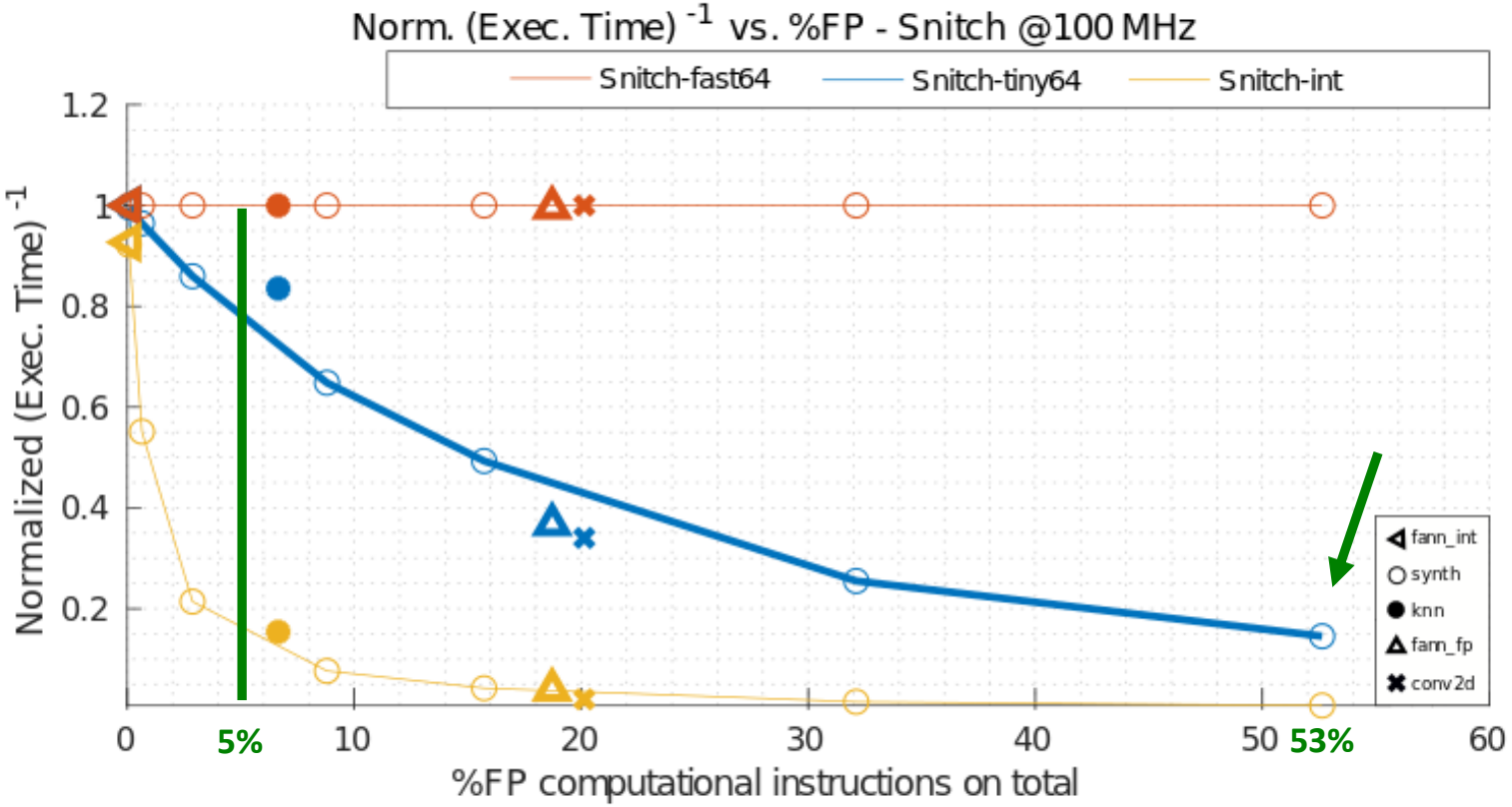
Results - Performance

High %FP

- **Snitch-tiny** up to **18.5x** (DP) and **15.5x** (SP) **faster** than **Snitch-int**

Low %FP (<5%)

- **Snitch-tiny** only **1.33x** (DP) and **1.18x** (SP) **slower** than **Snitch-fast,** while being **5x** (DP) and **3x** (SP) **faster** than **Snitch-int**



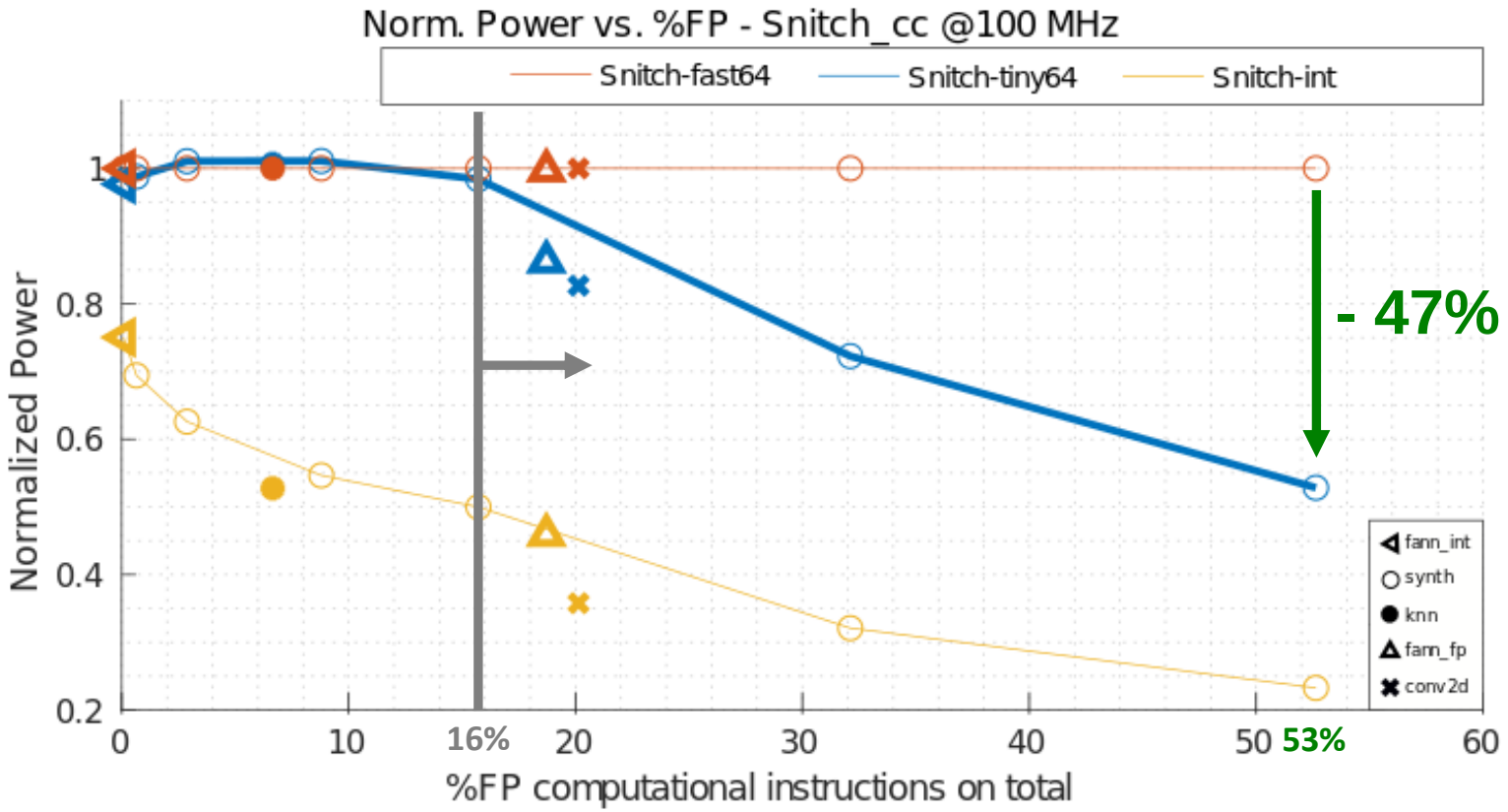
Results - Power

%FP > 16%

- Steep **increase** of **Snitch-fast power consumption** due to heavier system resources utilization

High %FP

- **Snitch-tiny** consumes up to **47%** (DP) and **33%** (SP) **less power** than **Snitch-fast**

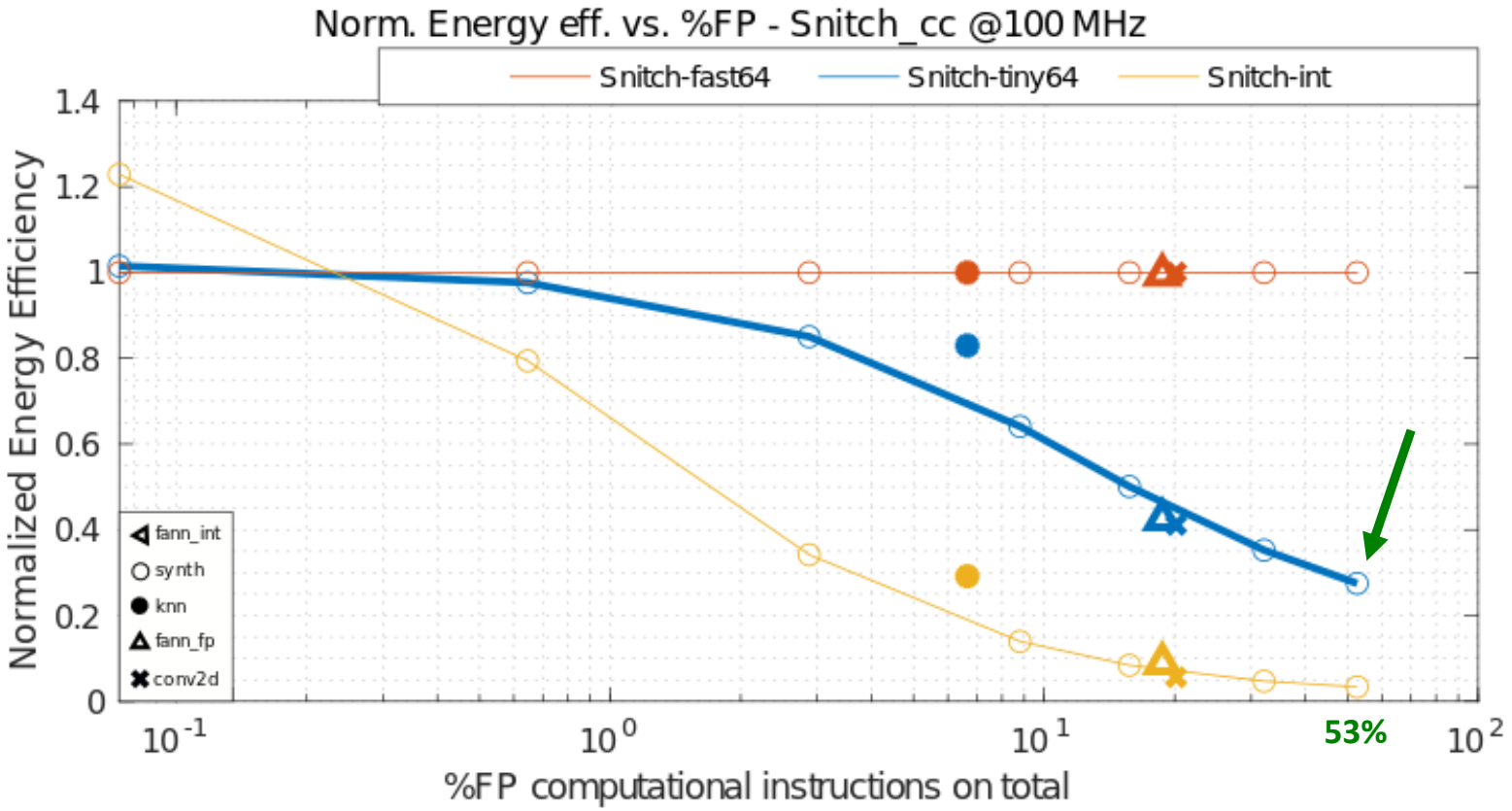


Results - Energy Efficiency

High %FP

- Snitch-tiny is not as energy-efficient as snitch-fast due to the multi-cycle execution

- **Snitch-tiny** is up to **8x more energy-efficient** than **Snitch-int**



Further Optimization

RISC-V defines a separate **register file** for FP instructions

Zfinx proposes to use just one register file when **FP** and **INT** share the **same word size**

Snitch-tiny32 would be just **1.7x larger** than **Snitch-int**

Conclusions

- **Tiny-FPU** is a new area-optimized **RISC-V** FPU:
  - **53%** (DP) and **37%** (SP) **smaller** than a high-performance and energy-efficient FPU

- We evaluated the costs and performance of **five** different **floating-point supports**, keeping Snitch as host system

- **Snitch coupled with Tiny-FPU** is:
  - up to **18.5x (**DP) and **15.5x** (SP) **faster** than **Snitch** employing **SW emulation**
  - up to **8x** more **energy-efficient** than **Snitch** employing **SW emulation**
  - up to **47%** (DP) and **33%** (SP) **less power-consuming** than **Snitch** coupled with **Fast-FPU**

- **Future work**: **Zfinx** version of Snitch-tiny32 to achieve the lowest area overhead to support FP in HW.

Open Source

Joining our open-source repositories soon!

https://pulp-platform.org

https://github.com/pulp-platform

Acknowledgement

ETH:

- Matteo Perotti
- Stefan Mach
- Pasquale Davide Schiavone
- Florian Zaruba
- Luca Benini

HUAWEI:

- Tariq Kurd
- Mark Hill
- Lukas Cavigelli

RISC-V® Summit

December 8-10 | Virtual Event

Thank you for your attention!

#RISCVSUMMIT @risc_v

Brought to you by informa tech

riscvsummit.com #RISCVSUMMIT