FazyRV – A RISC-V Core that Scales to Your Needs

Meinhard Kissich

ORConf'24

Meinhard Kissich

www.meinhard-kissich.at
meinhard.kissich@tugraz.at

PhD Student
Graz University of Technology, Austria

FPGA Architectures & CAD Tools
Processor Architecture and ISAX
Applied Formal Verification

Another RISC-V core?

Little to moderate processing power, **high** focus on area

Little to moderate processing power, **high** focus on area

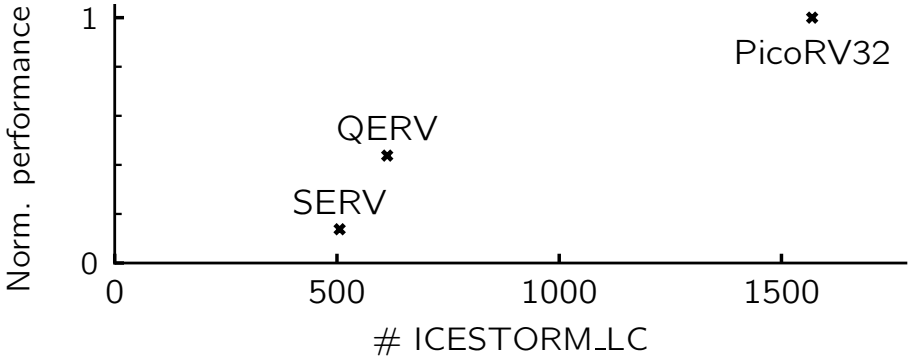
Control-oriented applications, IoT, ...

Little to moderate processing power, **high** focus on area

Control-oriented applications, IoT, ...

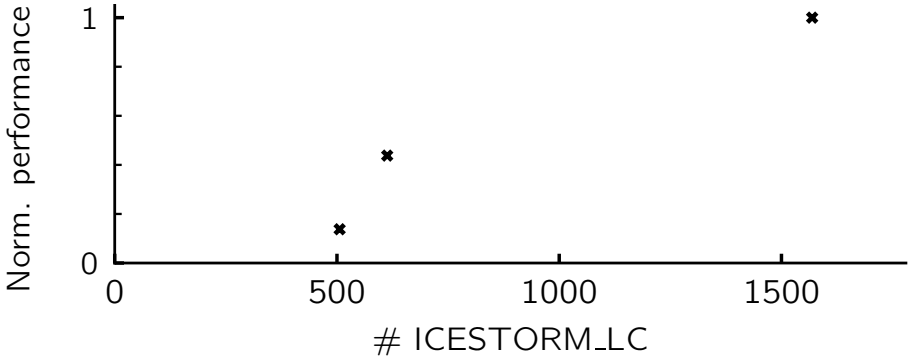
Fulfil perf. requirements with minimal area demand

Reference SoC with Core



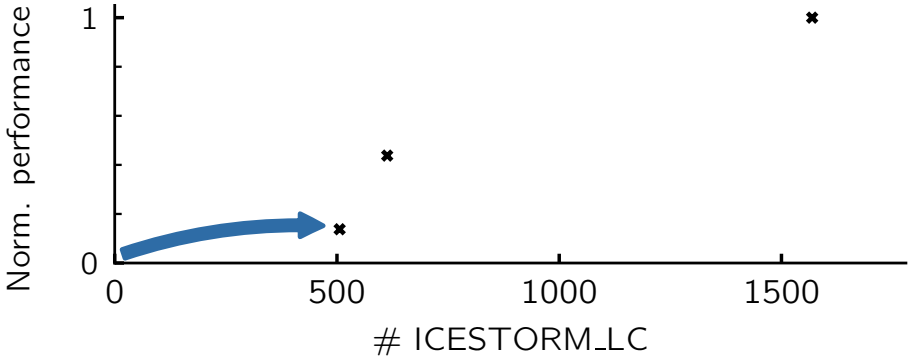
* frequency normalized

Reference SoC with Core



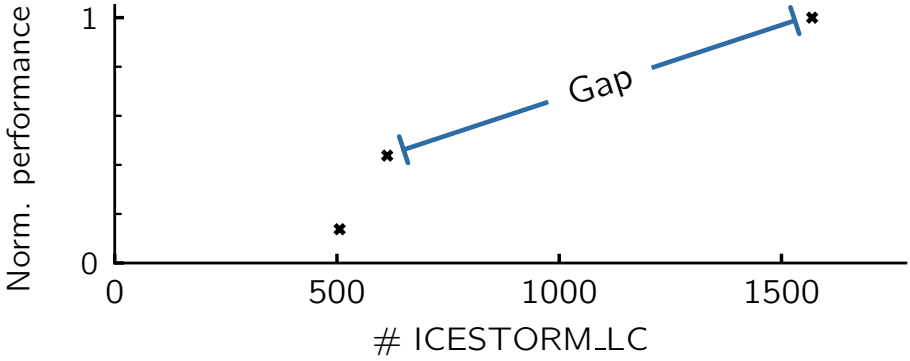
* frequency normalized

Reference SoC with Core



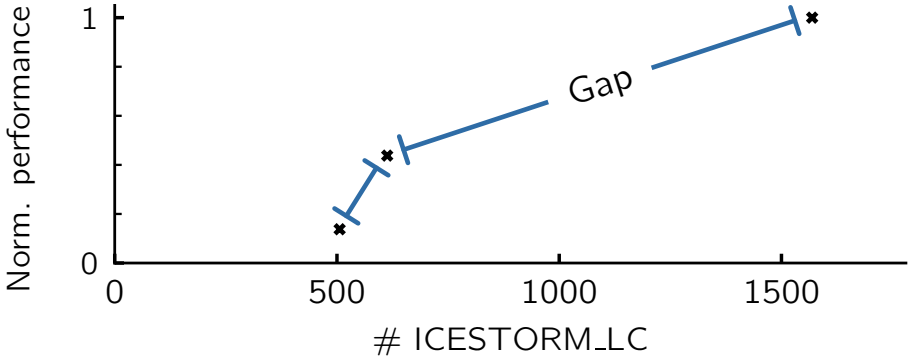
* frequency normalized

Reference SoC with Core



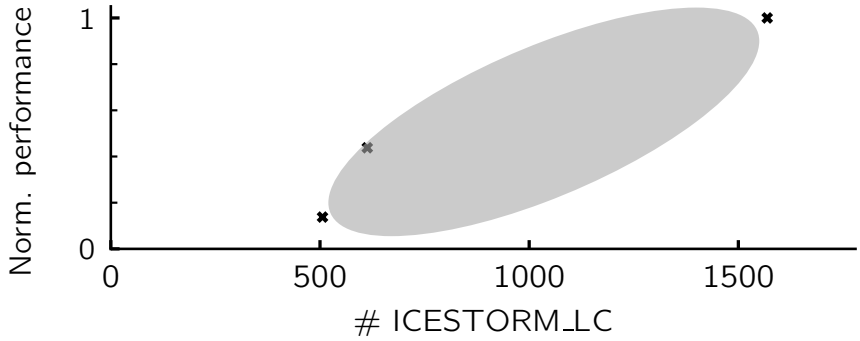
* frequency normalized

Reference SoC with Core



* frequency normalized

FazyRV: Closing the Gap between 32-Bit and Bit-Serial RISC-V Cores with a Scalable Implementation[1]



Slide set based on [16]

Design Objectives

- **Scalability** – Parametrizable 1, 2, 4, or 8-bit data path width (“*chunk size*”)

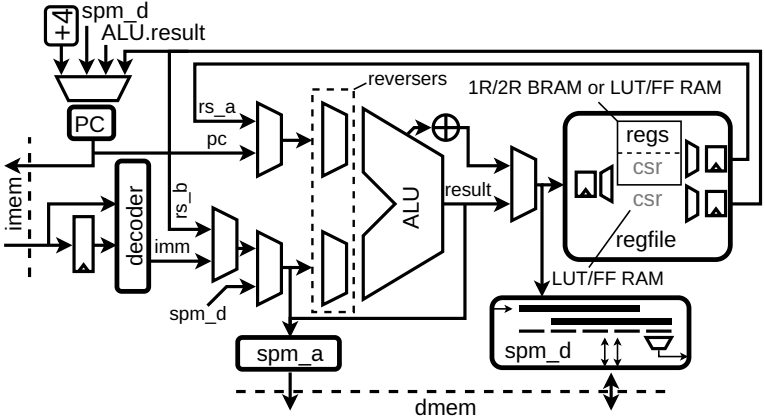
Design Objectives

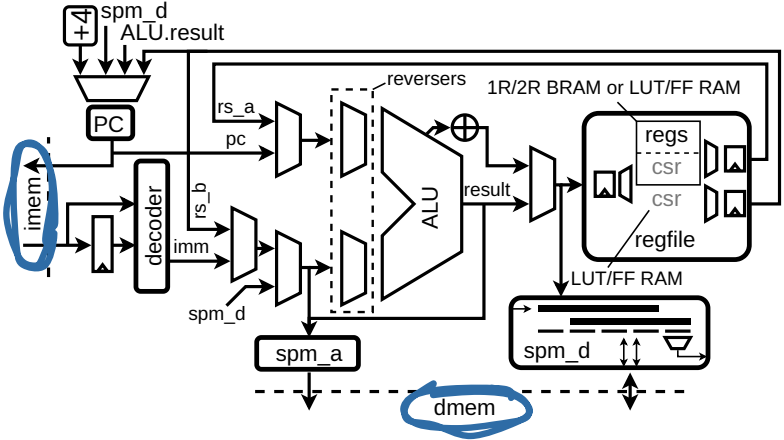
- **Scalability** – Parametrizable 1, 2, 4, or 8-bit data path width (“*chunk size*”)
- **Abstraction** – No hand optimization at the gate level

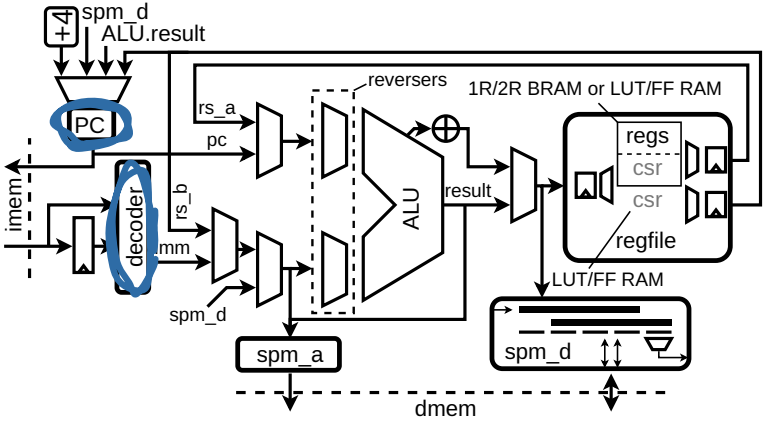
Design Objectives

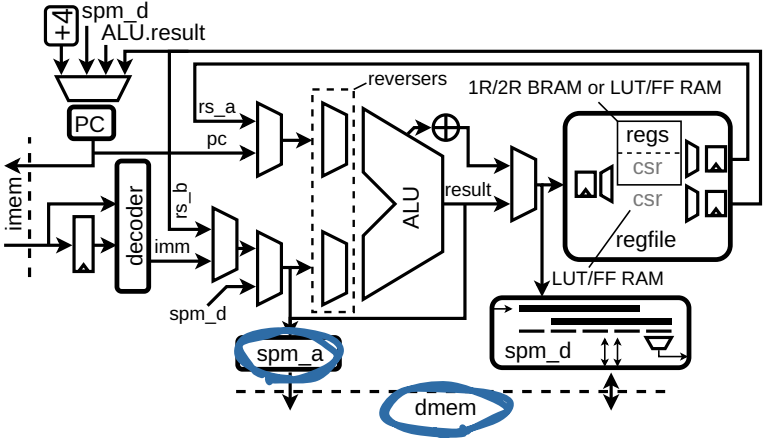
- **Scalability** – Parametrizable 1, 2, 4, or 8-bit data path width (“*chunk size*”)
- **Abstraction** – No hand optimization at the gate level
- **Variants** – Interrupts and CSRs: “*MIN*” $\subset$ “*INT*” $\subset$ “*CSR*”
  – Register file: single-port vs. dual-port RAM, BRAM vs. LUT RAM
  – Optional bypass MUX

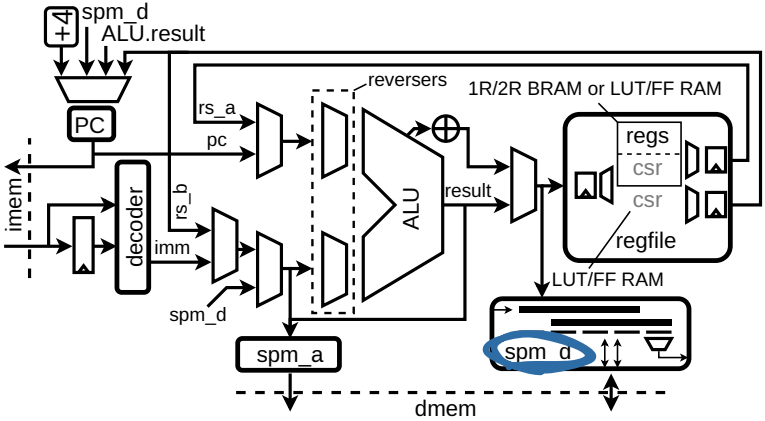
Core Insights

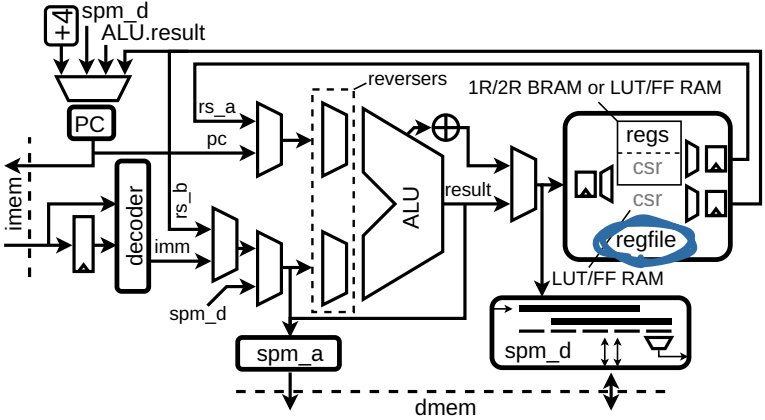

+4
spm_d
ALU.result
PC
reversers
1R/2R BRAM or LUT/FF RAM
rs_a
pc
regs
csr
csr
regfile
imem
decoder
rs_b
imm
ALU
result
LUT/FF RAM
spm_d
spm_d
spm_a
dmem

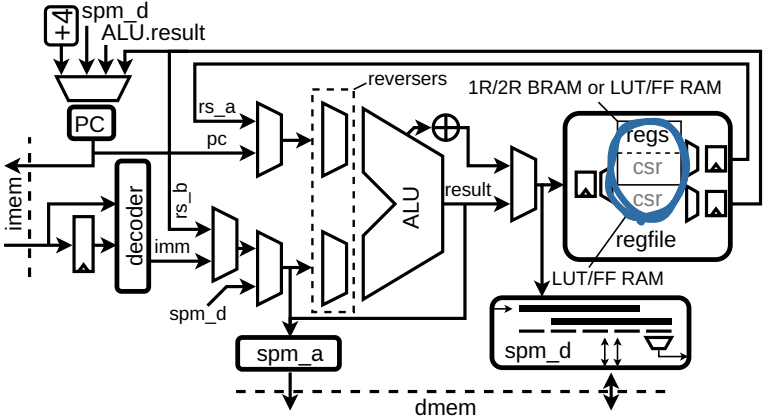

+4
spm_d
ALU.result
PC
imem
decoder
rs_a
pc
rs_b
imm
spm_d
reversers
ALU
result
1R/2R BRAM or LUT/FF RAM
regs
csr
csr
regfile
LUT/FF RAM
spm_d
spm_a
dmem

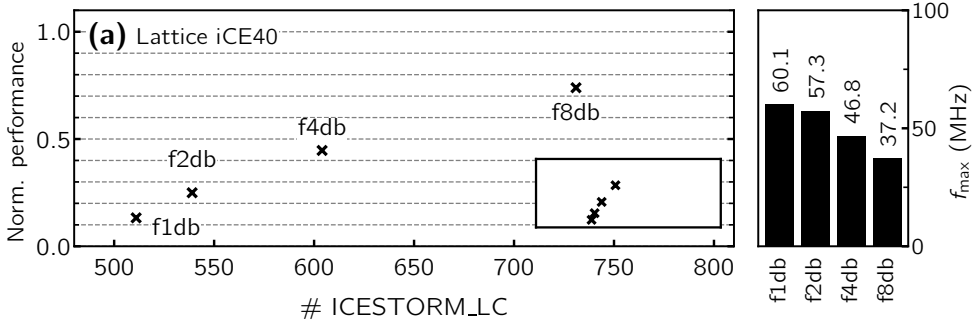

+4
spm_d
ALU.result
PC
reversers
1R/2R BRAM or LUT/FF RAM
rs_a
pc
regs
csr
csr
imem
decoder
rs_b
imm
ALU
result
regfile
LUT/FF RAM
spm_d
spm_d
spm_a
dmem


+4
spm_d
ALU.result
PC
reversers
1R/2R BRAM or LUT/FF RAM
rs_a
pc
imem
decoder
rs_b
imm
ALU
result
regs
csr
csr
regfile
LUT/FF RAM
spm_d
spm_a
spm_d
dmem

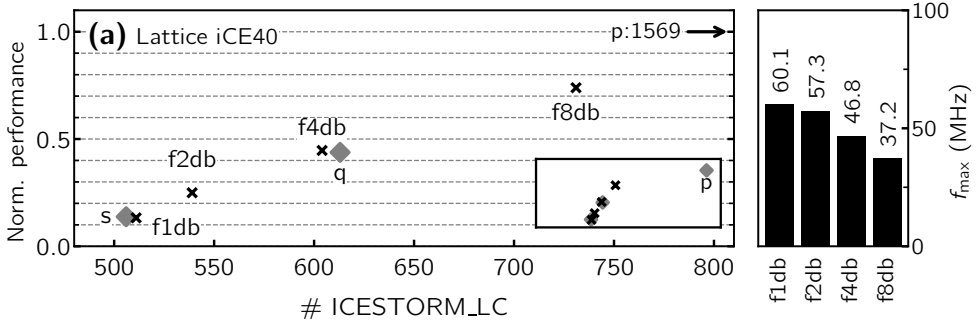

+4
spm_d
ALU.result
PC
reversers
1R/2R BRAM or LUT/FF RAM
rs_a
pc
regs
csr
csr
regfile
imem
decoder
rs_b
imm
ALU
result
LUT/FF RAM
spm_d
spm_a
spm_d
dmem


+4
spm_d
ALU.result
PC
reversers
1R/2R BRAM or LUT/FF RAM
rs_a
pc
regs
csr
csr
imem
decoder
rs_b
imm
ALU
result
regfile
LUT/FF RAM
spm_d
spm_d
spm_a
dmem

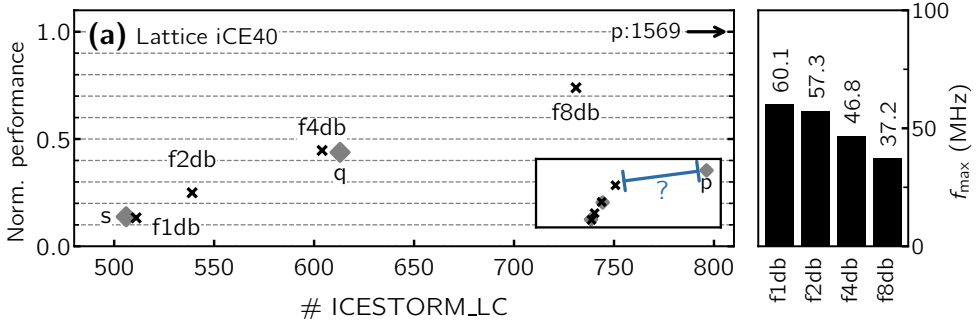

+4
spm_d
ALU.result
PC
reversers
1R/2R BRAM or LUT/FF RAM
rs_a
pc
regs
csr
csr
regfile
imem
decoder
rs_b
imm
ALU
result
LUT/FF RAM
spm_d
spm_d
spm_a
dmem



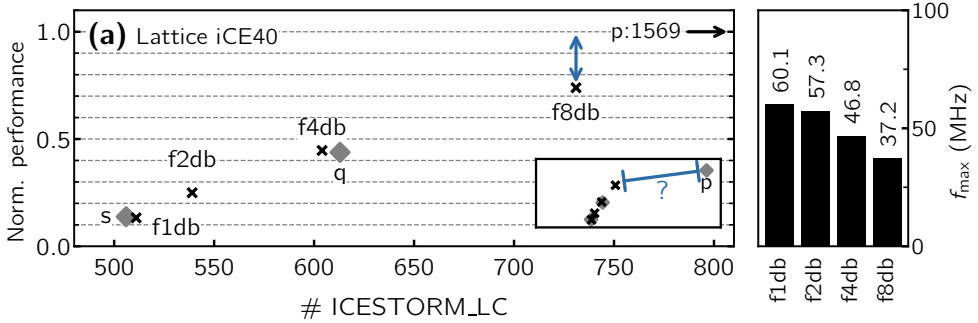
* frequency normalized



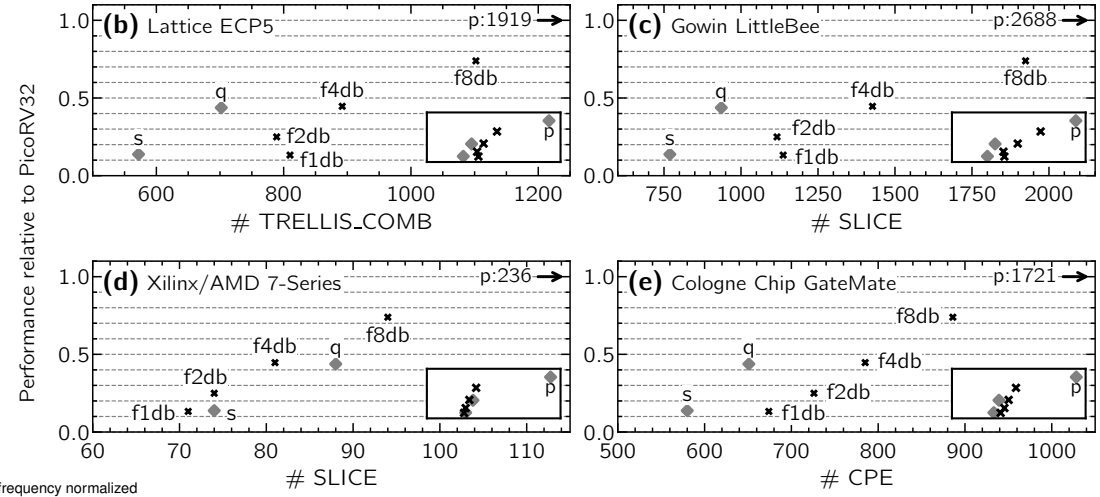
* frequency normalized



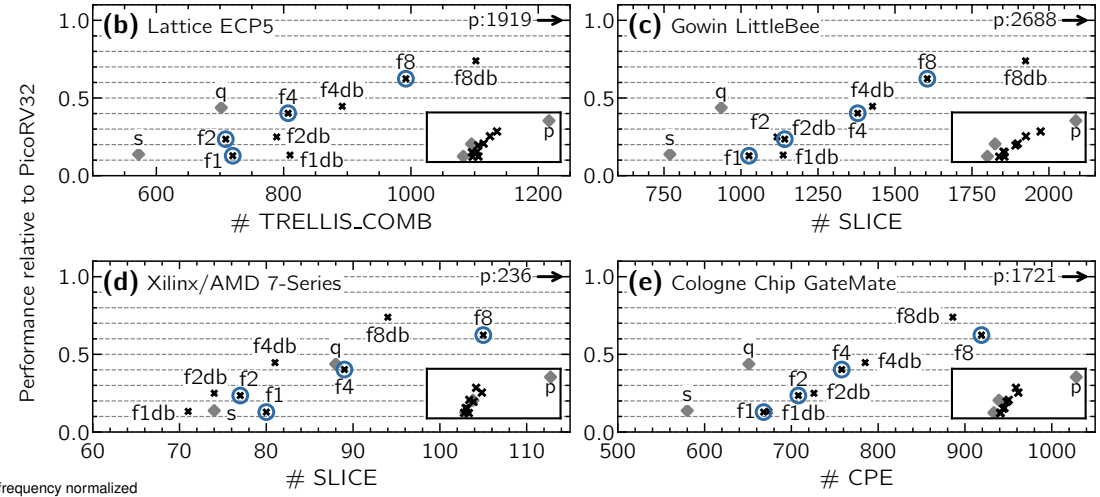
* frequency normalized



* frequency normalized



* frequency normalized



* frequency normalized

1,2,4,8 ... Why no 16-bit and 32-bit variants?

Shifts! And Zero/Sign-Extension!

1-bit Variant Shifts (Naïve Approach)

1-bit Variant Shifts (Naïve Approach)

1-bit Variant Shifts (Naïve Approach)

We Have to be More Clever

- Shifting 1 bit at a time does not scale

  ↳ inefficient for larger chunk sizes

We Have to be More Clever

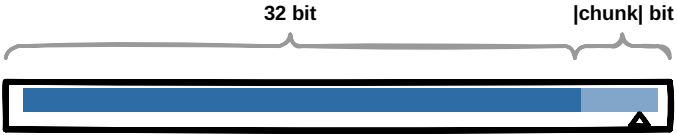
- Shifting 1 bit at a time does not scale
  - ↳ inefficient for larger chunk sizes
- Barrel shifter too costly
  - ↳ not justifiable for small chunk sizes

We Have to be More Clever

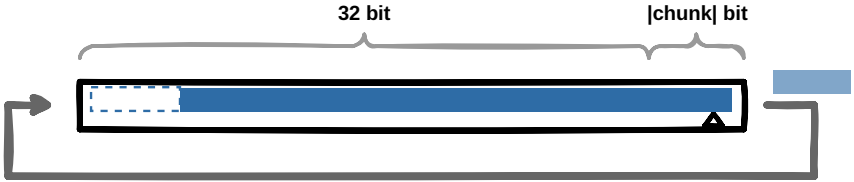
- Shifting 1 bit at a time does not scale
  - ↪ inefficient for larger chunk sizes
- Barrel shifter too costly
  - ↪ not justifiable for small chunk sizes

⇒ Introduce *Marco Steps*

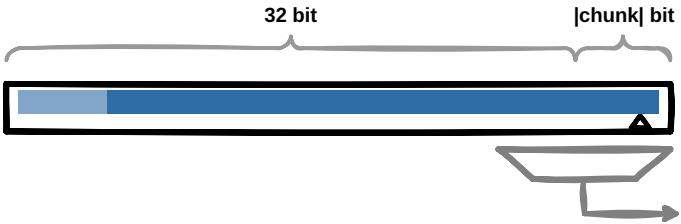
FazyRV Shifts



FazyRV Shifts

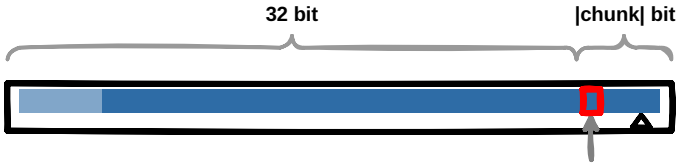


FazyRV Shifts



- ✅ Overhead scales with the chunk size
- ✅ Performance scales with the chunk size
- ... but beyond 8 bit chunk size: barrel shifter gets large

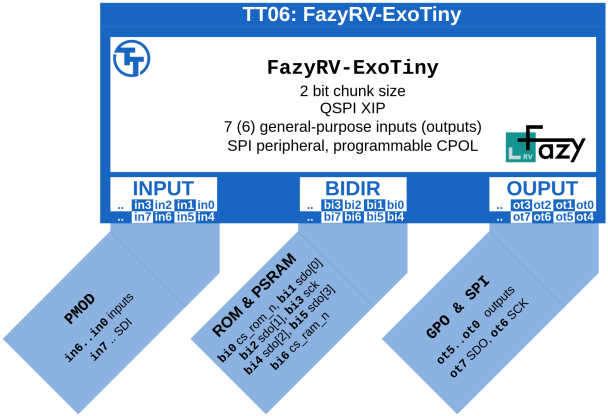
Zero/Sign Extension



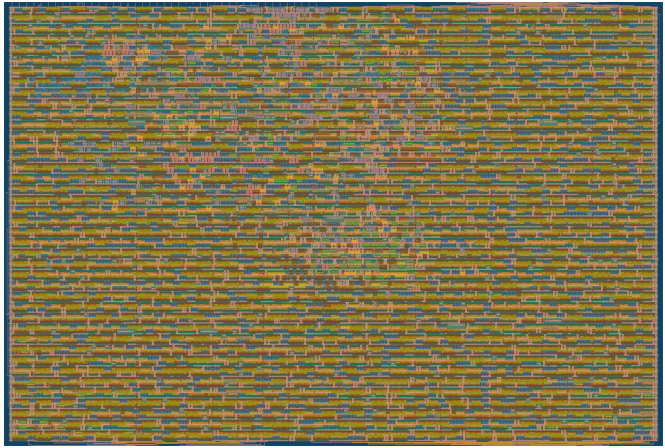
- Replicated bit always at same position
- ... but does not hold true for chunk sizes larger than 8 bit

FazyRV-ExoTiny [2]

approx. 334 μm x 216 μm, SkyWater SKY130 PDK, 6108 cells, OpenLane flow



FazyRV-ExoTiny [2]



What's next?

- (WiP) Custom instruction interface
- (WiP) Optimize decoder
- (~) Compressed ISA: code size & resources
- (~) Interrupts and privileged ISA
- (open) Improve verification framework: currently riscv-formal [3] + RISCOF [4]
- (?) CSR instructions in addition to `csrrw`

What's next?

(WiP) Custom instruction interface

(WiP) Optimize decoder

(~) Compressed ISA: code size & resources

(~) Interrupts and privileged ISA

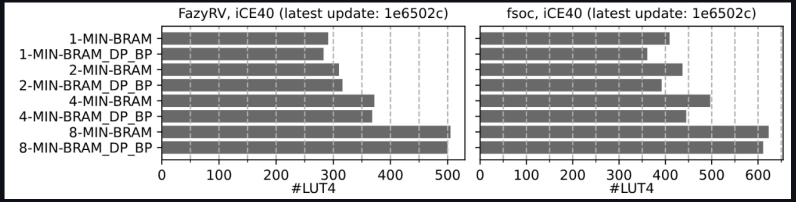
(open) Improve verification framework: currently riscv-formal [3] + RISCOF [4]

(?) CSR instructions in addition to `csrrw`

Interested in contributing to FazyRV? Feel free to contact me :)

Area / Resource Demand

The plot below tracks the resource demand of the FazyRV core (left) and a minimal reference SoC (right) for an iCE40 architecture. Note that only a few variants are plotted for brevity.



Organization

FazyRV is contained in `rtl` and licensed under a permissive MIT license. `rtl/fazyrv_top.sv` is the top module and instantiates the FazyRV core (`rtl/fazyrv_core.sv`) alongside the register file. The reference SoC (fsoc) is located in `soc/rtl` along with a constraint file for common architectures.

The data flow through the core can be outlined as follows:

?  ?

femto
atto
zepto
yocto

iti.tugraz.at/FazyRV

meiniKi / fazyrv 53

A minimal-area RISC-V core with a scalable data path to 1, 2, 4, or 8 bits and manifold variants.

meiniKi

Meinhard Kissich

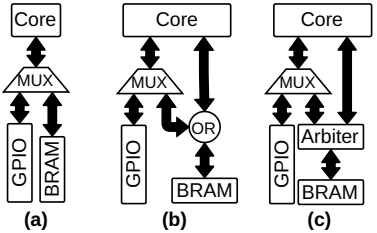
www.meinhard-kissich.at

meinhard.kissich@tugraz.at

Backup Slides

Comparison

- **iCE40** architecture
- Identical **open-source CAD flow**: including Yosys [5] and nextpnr [6], [7]
- Minimal **reference SoC**
  - 64 Bytes memory (BRAM, Wishbone)
  - 1 memory-mapped output
  - Minimal resources to realize SoC



Core in SoC	CSR	Synthesis # LUT-4	Synthesis # Cells	Implementation # LCs	Implementation f_{MAX} (MHz)	Ref.
SERV	✗	316	658	506	109	[8]
SERV	✓	395	783	590	98	[8]
QERV	✗	402	790	613	86	[9]
QERV	✓	500	942	717	71	[9]
VexRiscv*	✗	970	1699	1336	78	[10]
PicoRV32	✓	1360	2135	1569	71	[11]
VexRiscv (LiteX)	✓	1369	2241	1887	60	[12]

*smallest version in repo, (not marked productive)

Core in SoC	CSR	Synthesis # LUT-4	Synthesis # Cells	Implementation # LCs	Implementation f_{MAX} (MHz)	Ref.
SERV	✗	316	658	506	109	[8]
SERV	✓	395	783	590	98	[8]
QERV	✗	402	790	613	86	[9]
QERV	✓	500	942	717	71	[9]
VexRiscv*	✗	970	1699	1336	78	[10]
PicoRV32	✓	1360	2135	1569	71	[11]
VexRiscv (LiteX)	✓	1369	2241	1887	60	[12]

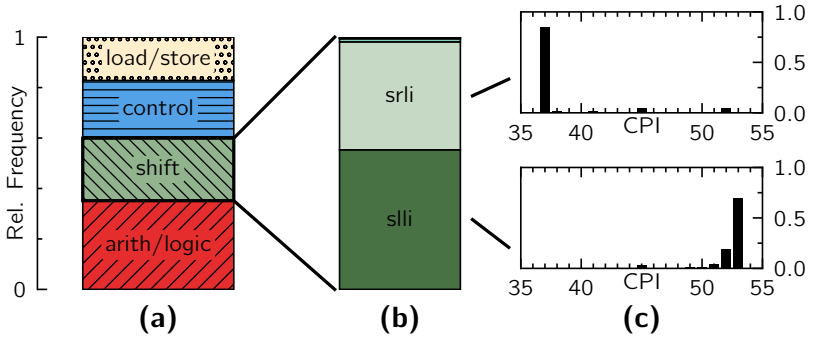
*smallest version in repo, (not marked productive)

$$\mathrm{CPI}_{\mathrm{min}} = N_{\mathrm{IF}} + N_{\mathrm{ID}} + \frac{32}{|\mathrm{chunk}|} \tag{1}$$

$$\mathrm{CPI}_{\mathrm{max}} = N_{\mathrm{IF}} + N_{\mathrm{ID}} + 2 \cdot \frac{32}{|\mathrm{chunk}|} + \left(1 + \frac{32}{|\mathrm{chunk}|}\right) \tag{2}$$

$$N_{\mathrm{IF}} = \begin{cases} 2 \text{ if Wishbone interface with 1 cycle delay} \\ 1 \text{ if fixed 1 cycle delay} \end{cases} \tag{3}$$

$$N_{\mathrm{ID}} = \begin{cases} 3 \text{ if single read port (1R) BRAM} \\ 2 \text{ if 1R BRAM + bypass, or 2R BRAM} \\ 1 \text{ if 2R BRAM + bypass} \end{cases} \tag{4}$$


1
0
Rel. Frequency
load/store
control
shift
arith/logic
(a)
srli
slli
(b)
1.0
0.5
0.0
35
40
CPI
50
55
(c)

Bibliography I

[1] M. Kissich and M. Baunach, **FazyRV: Closing the Gap between 32-Bit and Bit-Serial RISC-V Cores with a Scalable Implementation,** Proc. of the 21st ACM International Conference on Computing Frontiers (CF '24), Association for Computing Machinery, 2024, 240–248. DOI: 10.1145/3649153.3649195. [Online]. Available: https://doi.org/10.1145/3649153.3649195.

[2] Meinhard Kissich, **Tt06: Fazyrv-exotiny,** GitHub, (accessed 2024-04-16), 2024. [Online]. Available: https://github.com/meiniKi/tt06-FazyRV-ExoTiny (visited on 04/16/2024).

[3] YosysHQ, **Riscv-formal,** GitHub, (accessed 2023-11-28), 2023. [Online]. Available: https://github.com/YosysHQ/riscv-formal (visited on 11/28/2023).

[4] riscv-software-src, **RISCOF,** GitHub, (accessed 2023-11-28), 2023. [Online]. Available: https://github.com/riscv-software-src/riscof (visited on 11/28/2023).

[5] YosysHQ, **Yosys,** GitHub, (accessed 2023-11-27), 2023. [Online]. Available: https://github.com/YosysHQ/yosys (visited on 11/27/2023).

Bibliography II

[6] D. Shah, E. Hung, C. Wolf, S. Bazanski, D. Gisselquist, and M. Milanovic, **Yosys+nextpnr: An Open Source Framework from Verilog to Bitstream for Commercial FPGAs,** Proc. of the 27th IEEE Int. Symp. on Field-Programmable Custom Computing Machines (FCCM), IEEE, 2019, pp. 1–4.

[7] YosysHQ, **nextpnr,** GitHub, (accessed 2023-11-27), 2023. [Online]. Available: https://github.com/YosysHQ/nextpnr (visited on 11/27/2023).

[8] O. Kindgren, **SERV,** GitHub, (accessed 2023-11-17), 2023. [Online]. Available: https://github.com/olofk/serv.

[9] O. Kindgren, **QERV,** GitHub, (accessed 2023-11-30), 2023. [Online]. Available: https://github.com/olofk/qerv (visited on 11/30/2023).

[10] SpinalHDL, **VexRiscv,** GitHub, (accessed 2023-11-30), Nov. 2023. [Online]. Available: https://github.com/SpinalHDL/VexRiscv (visited on 11/30/2023).

[11] YosysHQ, **PicoRV32,** GitHub, (accessed 2023-11-18), 2023. [Online]. Available: https://github.com/YosysHQ/picorv32 (visited on 11/18/2023).

Bibliography III

[12] litex-hub, **Pythondata-cpu-vexriscv,** GitHub, (accessed 2023-12-14), 2023. [Online]. Available: https://github.com/litex-hub/pythondata-cpu-vexriscv.