PySpike: Agile Co-Verification of RISC-V SoC's and Beyond

LIU Yu

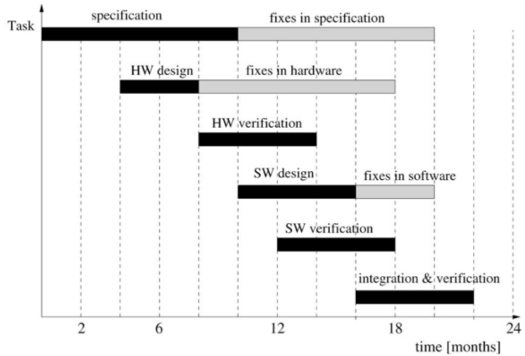
`<liuyu@huimtlab.org>`

Open Source @ Siemens 2025

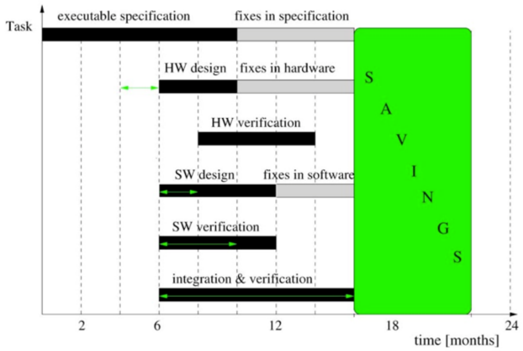
Collaborative (concurrent) engineering 101

中微亿芯 ESIONTECH

Sequential engineering



Collaborative engineering



. . . the executable specification may be used as a golden reference model for the software development process . . . . Indeed, such a specification and the use of cosimulation tools may also help to reveal errors in the test and firmware development cycle very early. [1]

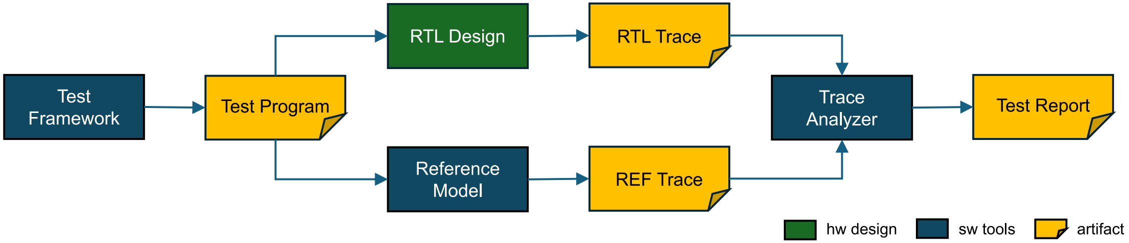
Agenda

Motivation

What Spike is all about

De facto standard RISC-V ISA simulator [2]

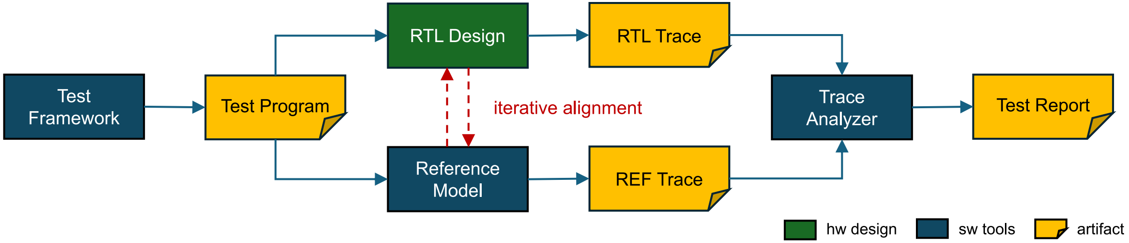
- functional model of RISC-V processors (harts) with state-of-the-art ISA support;
- originated from UC Berkeley, alongside RISC-V, actively maintained for 15+ years;
  `https://github.com/riscv-software-src/riscv-isa-sim`
  - reference model for hardware / software co-verification, e.g. *differential testing* [3]:



What Spike is all about

De facto standard RISC-V ISA simulator [2]

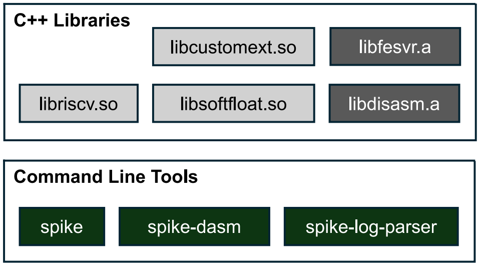
- ○ functional model of RISC-V processors (harts) with state-of-the-art ISA support;
- ○ originated from UC Berkeley, alongside RISC-V, actively maintained for 15+ years;
  `https://github.com/riscv-software-src/riscv-isa-sim`
- ● reference model for hardware / software co-verification, e.g. *differential testing* [3]:

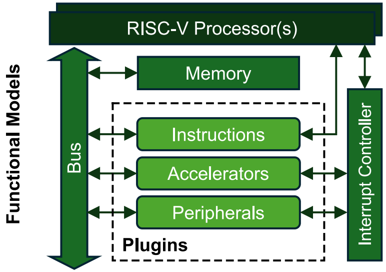


What Spike is all about (cont'd)

Spike [2], the upstream project at a glance

- C++ code base (40k+ lines), 190+ contributors, 2.9k stars on GitHub (2025/10);
- CLI tools (`spike`, `xspike`, `spike-dasm`, `...`) and C++ libs (`libriscv.so`, `...`);
  - plugin system based on dynamic loading (`dlopen`) of shared objects / libraries;

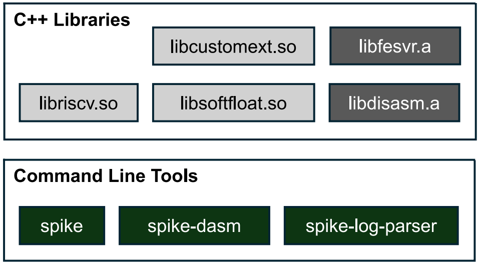


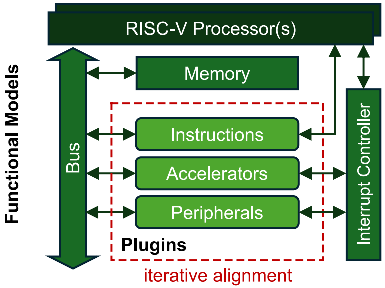


What Spike is all about (cont'd)

Spike [2], the upstream project at a glance

- ○ C++ code base (40k+ lines), 190+ contributors, 2.9k stars on GitHub (2025/10);
- ○ CLI tools (**`spike`**, **`xspike`**, **`spike-dasm`**, **`...`**) and C++ libs (**`libriscv.so`**, **`...`**);
- ● plugin system based on dynamic loading (**`dlopen`**) of shared objects / libraries;

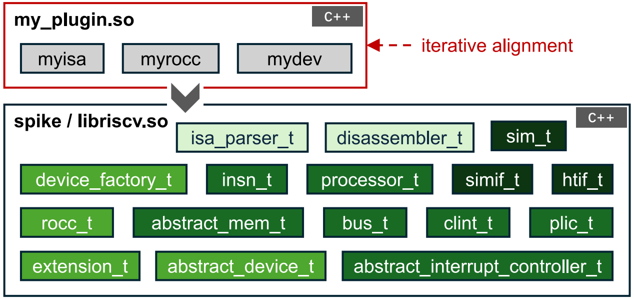




What Spike is all about (cont'd)

Spike [2], the upstream project at a glance

- C++ code base (40k+ lines), 190+ contributors, 2.9k stars on GitHub (2025/10);
- CLI tools (**`spike`**, **`xspike`**, **`spike-dasm`**, ...) and C++ libs (**`libriscv.so`**, ...);
- plugin system based on dynamic loading (**`dlopen`**) of shared objects / libraries;

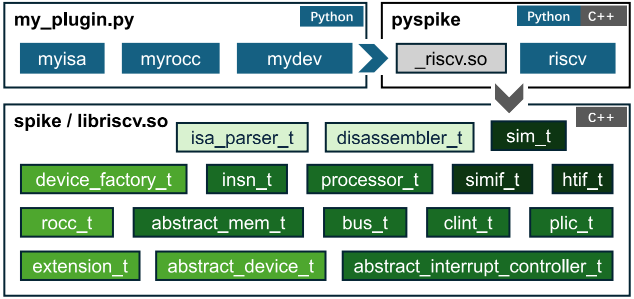


```
$ spike \
  --isa=rv64gc_xmyisa \
  --priv=msu \
  --extlib=my_plugin.so \
  --extension=myrocc \
  --device=mydev,0x20000000 \
  program.elf
```

Why Spike needs Python bindings

Agile co-verification calls for the feasibility of . . .

- fast-prototyping custom instructions, accelerators, peripherals, . . . ;
- programmatically manipulating Spike for interoperation with testbenches;
- reusing small goodies like ISA-string parser, RISC-V disassembler, . . . ;

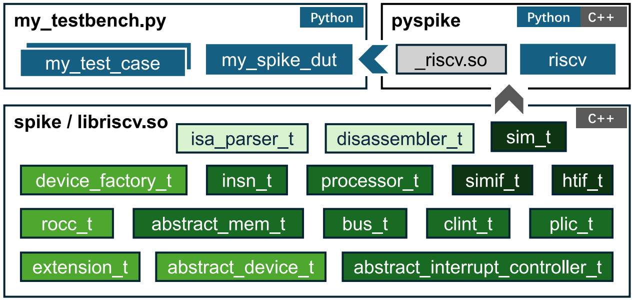


```
$ pyspike \
  --isa=rv64gc_xmyisa \
  --priv=msu \
  --extlib=my_plugin.py \
  --extension=myrocc \
  --device=mydev,0x20000000 \
  program.elf
```

Why Spike needs Python bindings

Agile co-verification calls for the feasibility of . . .

- ○ fast-prototyping custom instructions, accelerators, peripherals, . . . ;
- programmatically manipulating Spike for interoperation with testbenches;
- reusing small goodies like ISA-string parser, RISC-V disassembler, . . . ;



```
$ python -q
>>> from riscv.sim import *
>>> help(sim_t)
Help on class sim_t in module
↪  riscv._riscv.sim:

class sim_t(riscv._riscv.htif.h
↪  tif_t,
↪  riscv._riscv.simif.simif_t)
...
```

Notable Features

Getting started with PySpike

Install PyPI package: `spike`

1. setup virtual environment (optional):

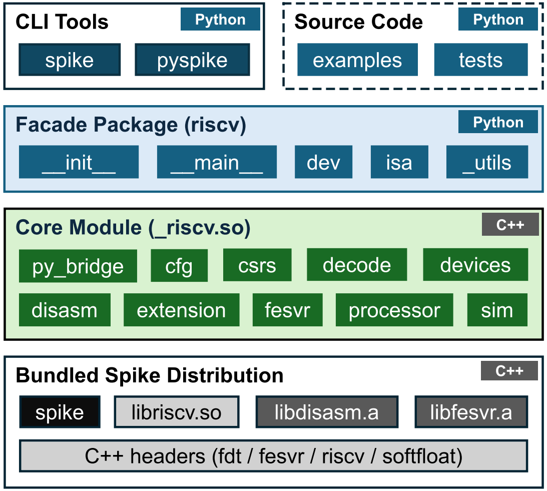
```
$ python3 -m venv venv
$ source venv/bin/activate
```

2. install **`spike`** package with **`pip`**:

```
(venv) $ pip install --pre spike
Collecting spike
  Downloading spike-0.0.5.dev...
```

3. check availability of **`pyspike`**:

```
(venv) $ pyspike --help
Spike RISC-V ISA Simulator 1.1.1-dev
  ...
usage: spike [host options] <target
↪   program> [target options]
  ...
```

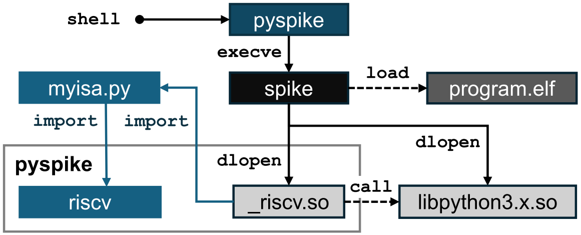


(Supports Linux / macOS; requires Python 3.8+)

What PySpike has to offer, exactly

Feature #1: Python in Spike (PIS)

- write instruction / peripheral models in Python, and plug them into Spike;



```
$ pyspike --isa=rv32gc_xmyisa
↪  --extlib=myisa.py program.elf
```

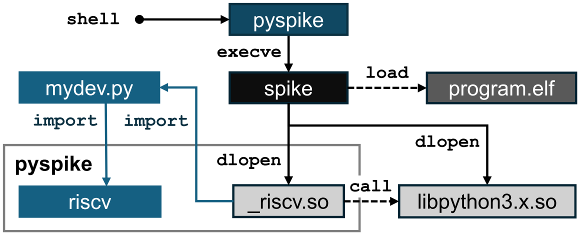
```
# @file myisa.pyi
from typing import List
from riscv import isa
from riscv.csrs import *
from riscv.disasm import *
from riscv.processor import *

@isa.register("myisa")
class MyISA(isa.ISA):
  def __init__(self): ...
  def get_instructions(self, proc)
  ↪  -> List[insn_desc_t]: ...
  def get_disasms(self, proc) ->
  ↪  List[disasm_insn_t]: ...
  def get_csrs(self, proc) ->
  ↪  List[csr_t]: ...
  def reset(self, proc) -> None: ...
```

What PySpike has to offer, exactly

Feature #1: Python in Spike (PIS)

- write instruction / peripheral models in Python, and plug them into Spike;



```
$ pyspike --isa=rv32gc --extlib=mydev.py
↪  --device=mydev,0x20000000
↪  program.elf
```

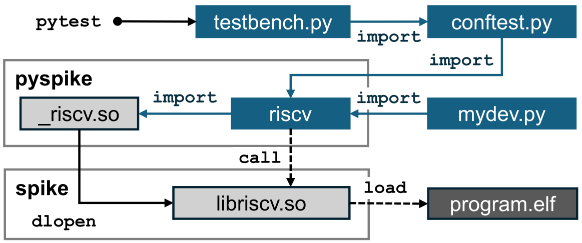
```python
# @file mydev.pyi
from typing import Optional
from riscv import dev
from riscv.sim import sim_t

@dev.register("mydev")
class MyDEV(dev.MMIO):
  def __init__(self, sim: sim_t,
  ↪  args: Optional[str]): ...
  def load(self, addr: int, size:
  ↪  int) -> bytes: ...
  def store(self, addr: int, data:
  ↪  bytes) -> None: ...
  def size(self) -> int: ...
  def tick(self, rtc_ticks: int) ->
  ↪  None:
```

What PySpike has to offer, exactly (cont'd)

Feature #2: Spike in Python (SIP)

- instantiate and seamlessly integrate Spike DUT into Python testbenches;



```
$ PYSPIKE_LIBS=mydev.py \
  pytest -v testbench.py
```

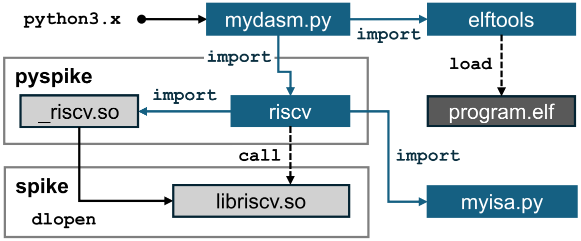
```python
# @file conftest.py
import pytest
from riscv.cfg import *
from riscv.debug_module import *
from riscv.sim import sim_t

@pytest.fixture
def mydut():
 yield sim_t(
  cfg=cfg_t(isa="rv32gc", priv="m",
   ↪  mem_layout=[mem_cfg_t(
   ↪  0x90000000, 0x40000)]),
  halted=False,
  plugin_device_factories=[("mydev",
   ↪  ("0x20000000", ))],
  args=["program.elf"],
  dm_config=debug_module_config_t())
```

What PySpike has to offer, exactly (cont'd)

Feature #2: Spike in Python (SIP)

- reuse Spike's internal gadgets in Python, i.e., RISC-V disassembler;



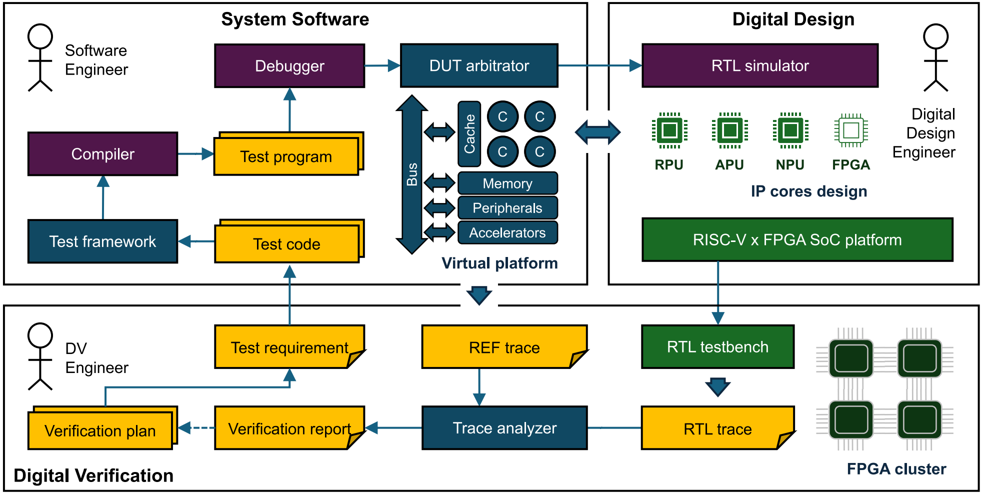
```
$ PYSPIKE_LIBS=myisa.py \
  python mydasm.py program.elf
```

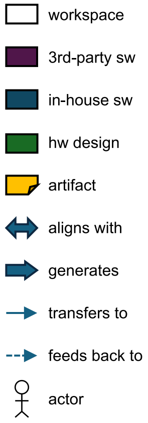
```python
#!/usr/bin/env python3
# @file mydasm.py
from sys import argv
from elftools.elf.elffile import *
from riscv.decode import *
from riscv.disasm import *
from riscv.isa_parser import *

d = disassembler_t(isa_parser_t(
↪   "rv64gc_xmyisa", "msu"))
e = ELFFile.load_from_path(argv[1])
s = e.get_section_by_name(".text")
pc = s.header.sh_addr
for op in insn_fetch_all(s.data()):
  print(f"0x{pc:08x} (0x{op:08x}):",
  ↪   d.disassemble(x:=insn_t(op)))
  pc += len(x)
```

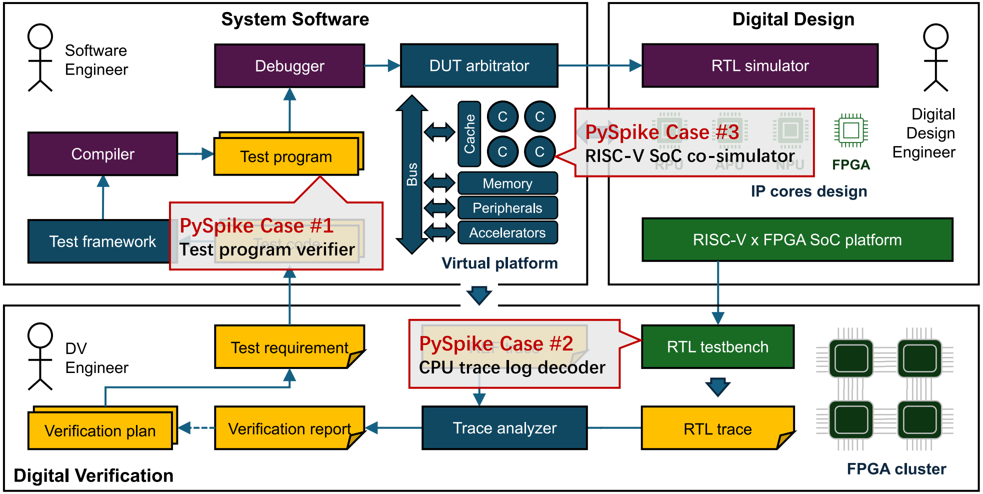
Real-World Examples

PySpike in the big picture of co-verification





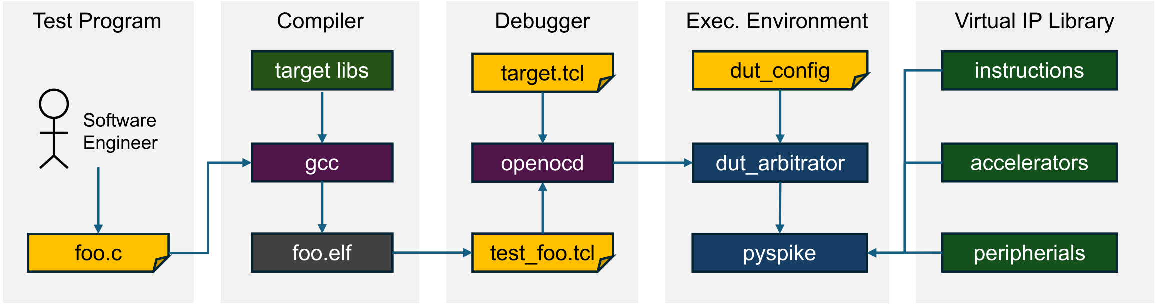
PySpike in the big picture of co-verification



▭	workspace
🟪	3rd-party sw
🟦	in-house sw
🟩	hw design
🟨	artifact
⇔	aligns with
⇒	generates
→	transfers to
--→	feeds back to
♀	actor

Case #1: Continuous test program verifier

- PySpike provides a reference DUT for verifying test-programs.

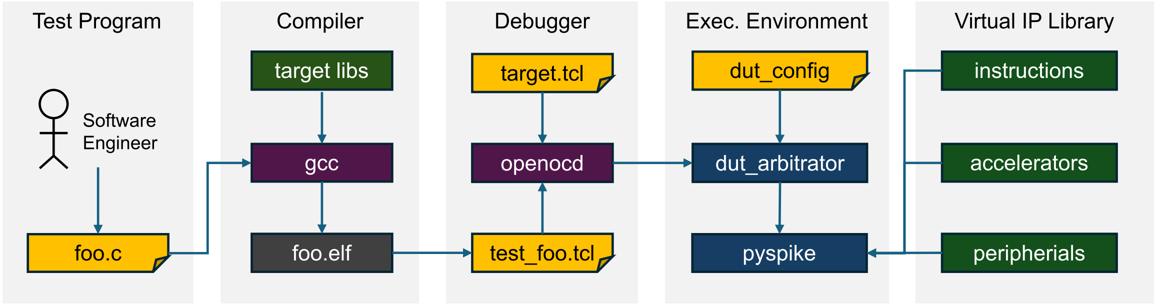


```
$ dut_arbitrator.py target-pyspike
Listening for remote bitbang
↪  connection on port 12345.
```

```
$ openocd \
  --file target.tcl \
  --file test_foo.tcl
...
```

Case #1: Continuous test program verifier

- PySpike provides a reference DUT for verifying test-programs.

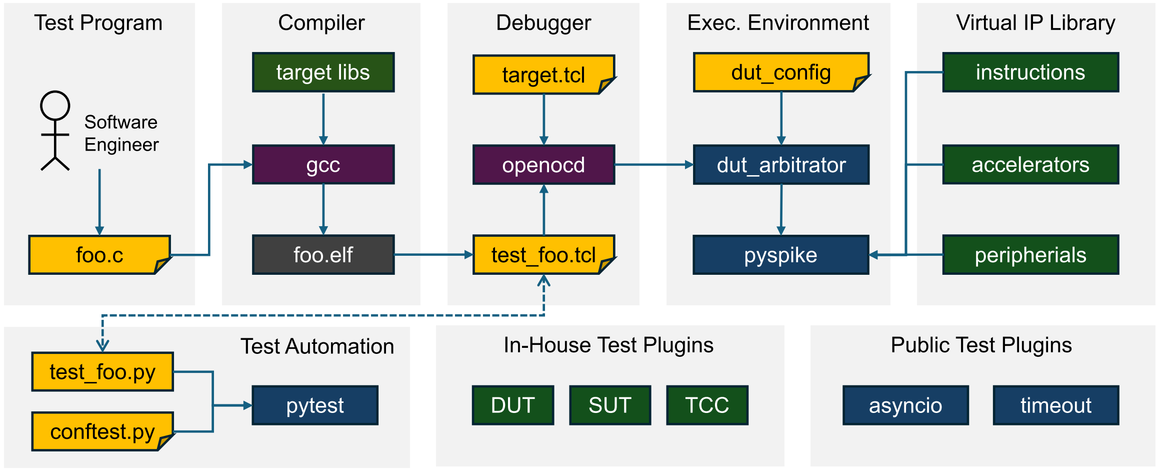


```
$ dut_arbitrator.py target-pyspike
Listening for remote bitbang
↪  connection on port 12345.
CPU1> Hello World!
```

```
$ openocd \
  --file target.tcl \
  --file test_foo.tcl
...
```

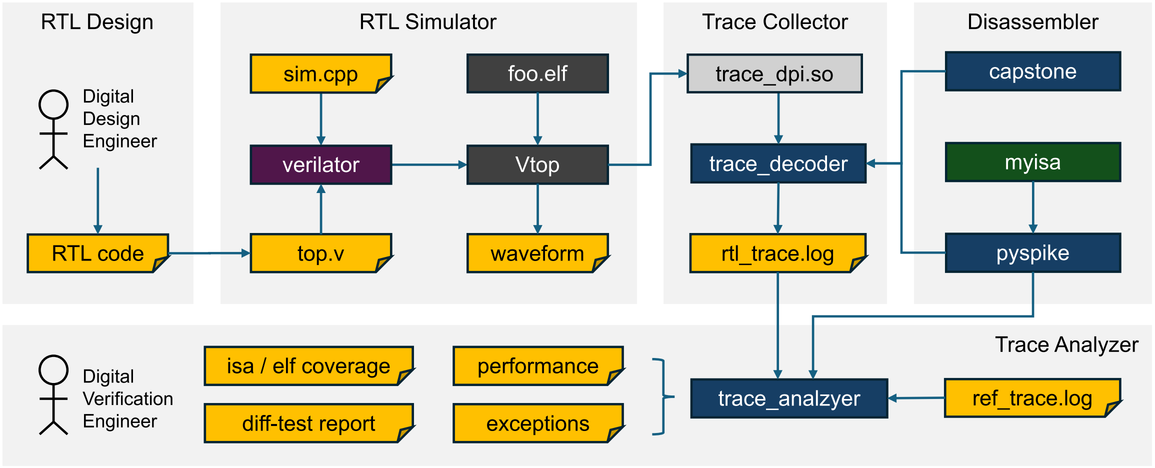
Case #1: Continuous test program verifier

- PySpike provides a reference DUT for verifying test-programs.



Case #2: CPU trace log decoder and beyond

- PySpike provides extensible RISC-V disassembler with top-norch ISA support.



Our Open-Source Playbook

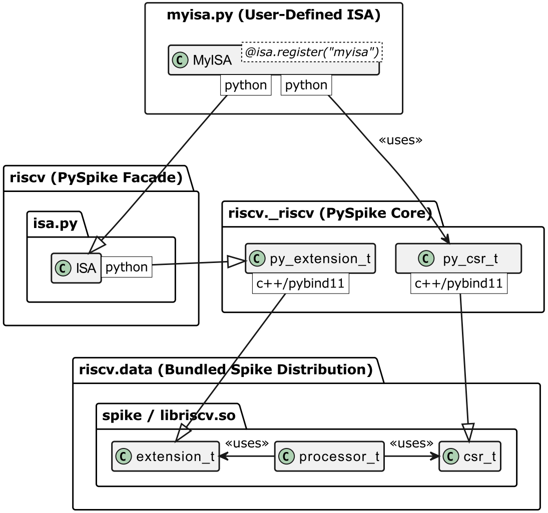
Strategy #1: Extend, don't modify

Concept

- design exclusively through Spike's **`dlopen`**-based plugin system, avoid instrusive patches to Spike's code;
- bundle Spike by compiling directly from upstream source at pinned commit;

Rationale

- zero-effort adoption for existing Spike users, w/ or w/o custom plugins;
- minimal maintenance overhead when syncing with upstream changes;



Strategy #1: Extend, don't modify

Concept

- design exclusively through Spike's **`dlopen`**-based plugin system, avoid instrusive patches to Spike's code;
- bundle Spike by compiling directly from upstream source at pinned commit;

Rationale

- zero-effort adoption for existing Spike users, w/ or w/o custom plugins;
- minimal maintenance overhead when syncing with upstream changes;

```
$ pyspike \
  --isa=rv64gc_xmyisa \
  --priv=msu \
  --extlib=myisa.py \
  --device=mydev,0x20000000 \
  program.elf
```

```
$ PYSPIKE_LIBS=myisa.py \
  spike \
  --isa=rv64gc_xmyisa \
  --priv=msu \
  --extlib=libpython3.8.so \
  --extlib=_riscv.so \
  --device=mydev,0x20000000 \
  program.elf
```

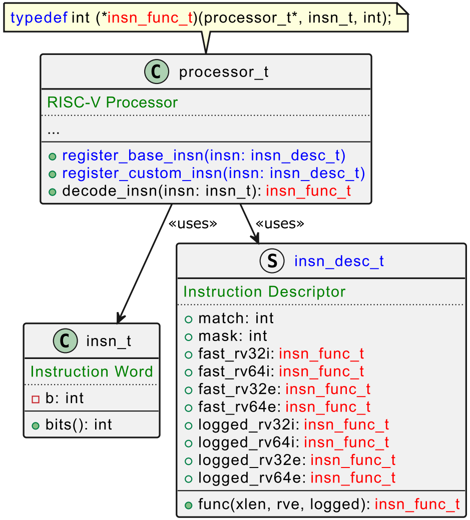
Strategy #1: Extend, don’t modify

Concept

- design exclusively through Spike’s **`dlopen`**-based plugin system, avoid instrusive patches to Spike’s code;
- bundle Spike by compiling directly from upstream source at pinned commit;

Rationale

- zero-effort adoption for existing Spike users, w/ or w/o custom plugins;
- minimal maintenance overhead when syncing with upstream changes;


typedef int (*insn_func_t)(processor_t*, insn_t, int);
C processor_t
RISC-V Processor
...
register_base_insn(insn: insn_desc_t)
register_custom_insn(insn: insn_desc_t)
decode_insn(insn: insn_t): insn_func_t
«uses»
«uses»
C insn_t
Instruction Word
b: int
bits(): int
S insn_desc_t
Instruction Descriptor
match: int
mask: int
fast_rv32i: insn_func_t
fast_rv64i: insn_func_t
fast_rv32e: insn_func_t
fast_rv64e: insn_func_t
logged_rv32i: insn_func_t
logged_rv64i: insn_func_t
logged_rv32e: insn_func_t
logged_rv64e: insn_func_t
func(xlen, rve, logged): insn_func_t

Strategy #2: Upstream, don’t fork

Concept

- proactively engage with upstream: identify hurdles early and resolve them through discussions or pull requests;
- strictly avoid maintaining divergent forks of Spike or PySpike, even internally;

Rationale

- prevent *technical debt* accumulation from conflicts and parallel development;
- strengthen the upstream foundation, which benefits the whole community;

Example (the ‘`--device`’ args fix)

2023/12: Someone proposed to *fix* Spike’s CLI option ‘**`--device`**’, i.e.,

> *Fix Spike **`--device`** option to pass on args to downstream plugins #1522*

```
$ spike --extlib=my_plugin.so \
-   --device=<name> \
+   --device=<name>,<args> \
  program.elf
```

The PR broke our *then-in-house* prototype PySpike for 1) the ‘fixed’ option assumed ‘**,**’-separated values, and 2) per-device **`args`** for the same **`name`** was impossible.

Strategy #2: Upstream, don't fork

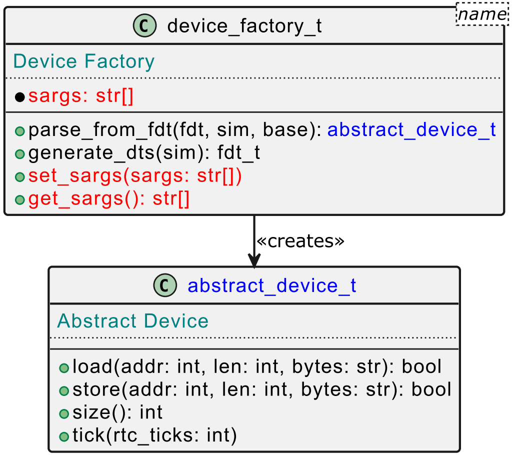
Concept

- proactively engage with upstream: identify hurdles early and resolve them through discussions or pull requests;
- strictly avoid maintaining divergent forks of Spike or PySpike, even internally;

Rationale

- prevent *technical debt* accumulation from conflicts and parallel development;
- strengthen the upstream foundation, which benefits the whole community;

Example (the '`--device`' args fix)



Strategy #2: Upstream, don't fork

Concept

- proactively engage with upstream: identify hurdles early and resolve them through discussions or pull requests;
- strictly avoid maintaining divergent forks of Spike or PySpike, even internally;

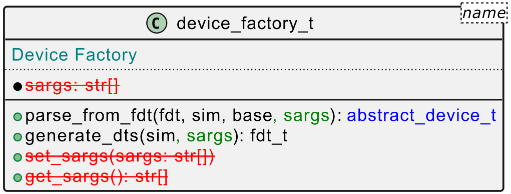
Rationale

- prevent *technical debt* accumulation from conflicts and parallel development;
- strengthen the upstream foundation, which benefits the whole community;

Example (the '`--device`' args fix)

2024/04: We raised our concern to the maintainers, received positive feedback, and got our PR merged in $\sim$1 week.

> *Counter-intuitive MMIO device arguments behavior #1652*
>
> *Support per-device arguments and device factory reuse #1655*


name
C device_factory_t
Device Factory
● ~~sargs: str[]~~
○ parse_from_fdt(fdt, sim, base, sargs): abstract_device_t
○ generate_dts(sim, sargs): fdt_t
○ ~~set_sargs(sargs: str[])~~
○ ~~get_sargs(): str[]~~

Strategy #3: Publicize workflows

Concept

- share PySpike's development, testing, and packaging workflows openly;
- leverage CI / CD services to automate and enforce quality adherence;

Rationale

- foster transparency and reproducibility for external contributors and users;
- open-source software thrive on community trust, even more so for quality-assuring tools like PySpike;

Example (developing workflow)

1. get source code:

```
$ git clone --recurse-submodules \
  git@github.com:huimtlab/pyspike
$ cd pyspike
```

2. setup virtual environment:

```
$ python3 -m venv venv
$ source venv/bin/activate
```

3. install in *editable* mode:

```
(venv) $ pip install -e '.[dev]'
```

4. lint & test & coverage:

```
(venv) $ pytest -v
...
76 passed, 8 skipped in 9.42s
```

Strategy #3: Publicize workflows

Concept

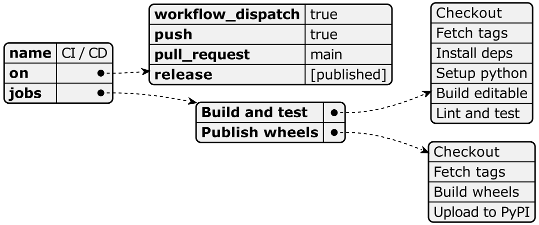
- share PySpike's development, testing, and packaging workflows openly;
- leverage CI / CD services to automate and enforce quality adherence;

Rationale

- foster transparency and reproducibility for external contributors and users;
- open-source software thrive on community trust, even more so for quality-assuring tools like PySpike;

Example (packaging workflow)

PySpike bundles Spike binaries as Python package data. We migrated internal build & package workflow to GitHub Actions.



Agile chip design methodology is still of limited usage due to lack of tool-chain and developing framework [4]

Remarks and Outlook

We are rolling out the first *Beta* release!

Closing remarks and key takeaways

- PySpike opens up Spike's C++ internals for interoperation with Python scripts; we showcase PySpike's potential in boosting agile co-verification with a few examples;
- The development of PySpike is guided by our *open-source playbook*, promising non-intrusive design, upstream engagement, and open engineering process;

Future outlook and call for collaboration

- Distributing embedded toolchains via the Python Package Index (PyPI) grants wider accessibility, and brings in extra benefits like dependency management, environment isolation, multi-version coexistence, . . . ; we hope to explore this area in the future;
- As a language binding, PySpike needs the community's help to keep up with Spike's changes to its evolving feature set and *unstable* C++ API's;

Who we are, and what we do

EsionTech (中微亿芯)

- subsidiary of CETC's Research Institute 58, founded in 2013;
- headquarter in Wuxi, R&D centers in Beijing, Shanghai, Wuhan, . . . ;
- vendor of all-programmable and heterogeneous computing chips;
- new player in the RISC-V ecosystem, since early 2023;

HuiMt Labs (惠山实验室)

- affiliated with EsionTech's Beijing R&D Center;
- small group of open-source software enthusiasts;
- veterans in HW / SW co-design and co-verification;
- contributors to RISC-V tools, i.e. **`gcc`**, **`spike`**, **`riscv-dv`**, . . . ;

Thank You!

References I

[1] Jürgen Teich. “Hardware/Software Codesign: The Past, the Present, and Predicting the Future”. In: *Proceedings of the IEEE* 100.Special Centennial Issue (2012), pp. 1411–1430. DOI: 10.1109/JPROC.2011.2182009.

[2] Andrew Waterman et al. *Spike RISC-V ISA Simulator*. https://github.com/riscv-software-src/riscv-isa-sim. 2010-2025. (Visited on 10/29/2025).

[3] William M. McKeeman. “Differential Testing for Software”. In: *Digital Technical Journal* 10.1 (1998).

[4] Yinan Xu et al. “Towards Developing High Performance RISC-V Processors Using Agile Methodology”. In: *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1178–1199. DOI: 10.1109/MICRO56248.2022.00080.