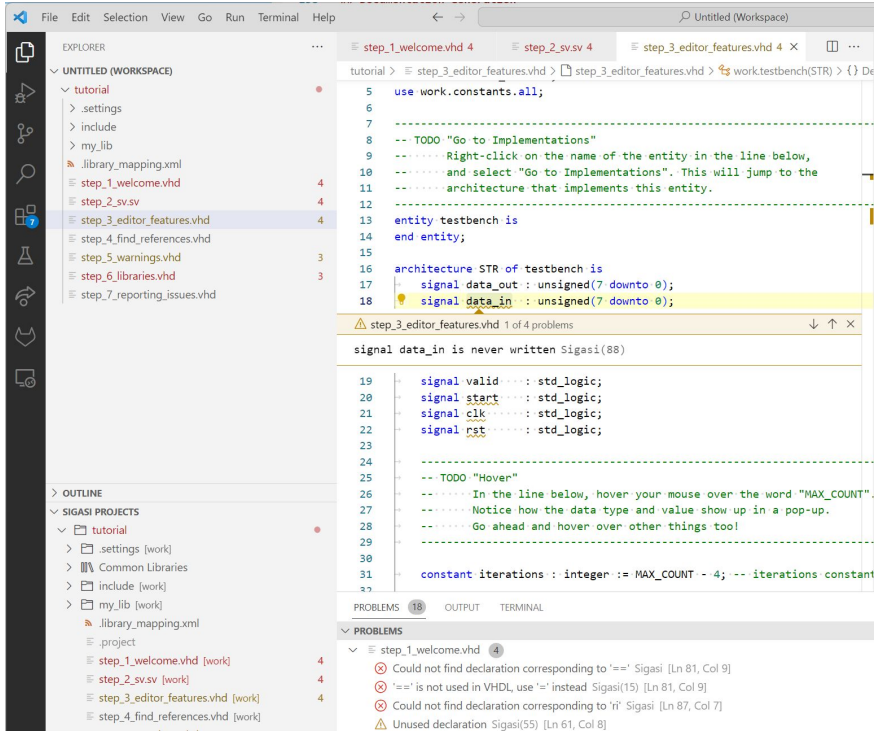
Exchange Format for HDL Build Systems

👋 Hi, I’m Lieven

- Software engineer with great love for program language design
- 15 years at Sigasi building front-end compilers and program analysis for VHDL & SystemVerilog
- Participated in the VHDL ~~2017~~ 2019 standardization
- Recently moved to product management

Quick commercial break

- Commercial VS Code extension & CLI for CI integration
- VHDL, Verilog, SystemVerilog
- Supercharged editor
  - Instant, actional feedback as you develop
  - Many visual representations
  - Navigation, quick fixes, formatting, …
- Instant, highly configurable linter
- Seamless integration into *your workflow*



Meet us at our booth, ask for a trial www.sigasi.com, try the “Community Edition” now!

Common Exchange Format for HDL Build Systems?

- What is the problem exactly? Is the problem real?
- A solution?
- Discussion

Common Exchange Format for HDL Build Systems?

Quick disclaimer

- Part of this talk is a bit of ‘~~a rant~~’ opinion piece
  This is an outside perspective
- I mean well and love all OSS projects that are mentioned
- Correct me if I’m wrong

The problem: Source/Project management

- Hardware design: ”Your HDL sources” ↔ numerous tools & frameworks
  - Tools: Simulation, Synthesis, Various linters, CDC/RDC tools, Formal, IDEs …
  - Verification libraries: OSVVM, UVVM, VUnit, UVM, Cocotb
  - Component libraries: Pile of Cores, Open Logic
  - Generators: registers and cores

- The same source files should be compiled the same way by tooling
  but not all tools see the same subset of sources
  not all tools need the same configuration
  most tools have a different interface for the same functionality

- How to make sure that all tools interpret the source the same/right way?
  - Theory: Single sourced via scripting
  - Practice: Messy, buggy, brittle tool specific scripting, configuration often partially duplicated

The problem

Tool “scripting”

- TCL, Python, Make, Bash, Perl …
  + some data in json, yaml, xml, csv, …

- Every company has a homegrown set of scripts
  - Specific to their use-case & available tools
  - Very opinionated
  - Sometimes full frameworks

- Hardware engineers are scripting junkies!
- Best practices?

The problem

What is scripted?

- How to compile the project (for vhdl ≃ associate list of files with libraries)

- How to run the tool
  Examples:
  - Simple simulation: toplevel, run length, log files, location of wave files, …
  - Verification framework: toplevel, parameters, …
  - Synthesis & cdc/rdc & formal: toplevel, constraints, …

So, making a list of files and associating them with a library?

- How hard can it be?
- Maybe duplication isn’t so bad?

The problem

Compiling files

- Simple folder structure: easy but the reality is often different

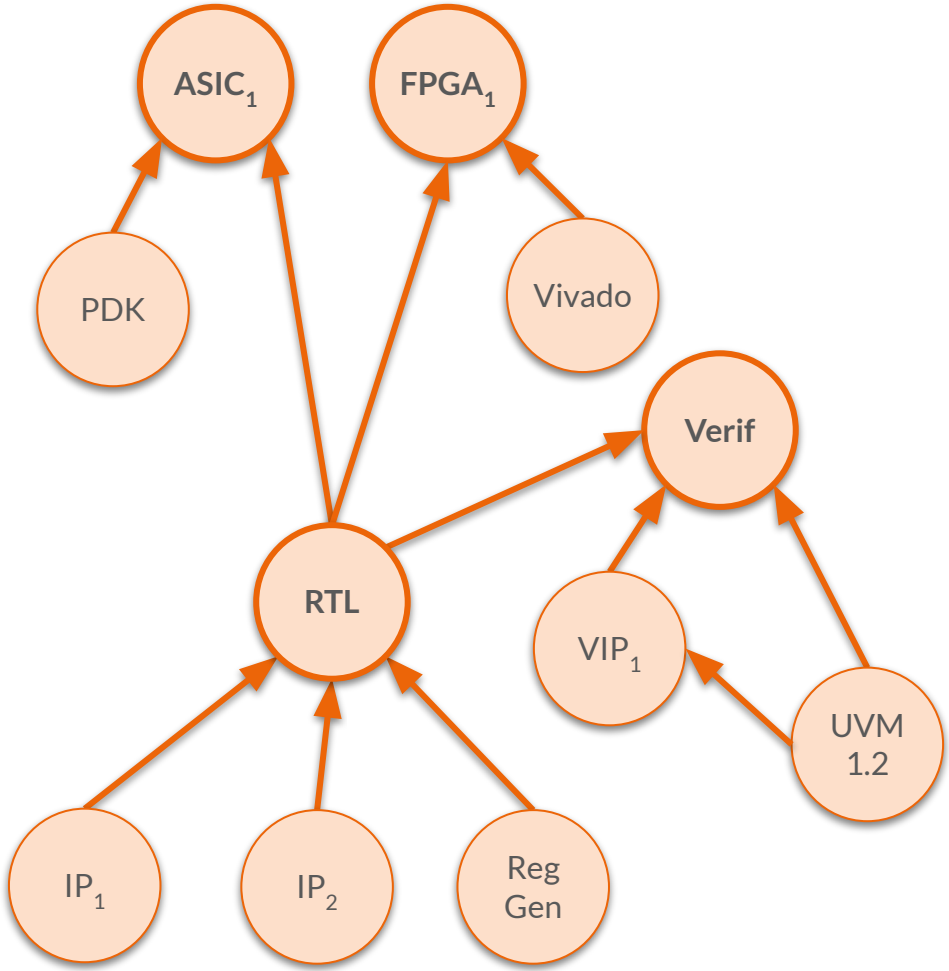
- Projects are heterogeneous
  - Internal IPs are built using an old scripting framework
  - External IPs or generated cores are built with their own scripts
    (that may need adaptation, if your tool is not supported)
    ↳ Not all sources are not under the users' control

- Sources are reused in various configurations
  - Verification framework specific toplevels (UVM, UVVM, OSVVM, VUnit)
  - ASIC toplevel + FPGA prototype toplevels
  - FPGA IP targeting various FPGA families

The problem

The configuration

- Heterogeneous
- Many configurations



The problem

- Too much time is spent
  setting up tools and writing halve-baked scripts

- Solution: Make the next OSS project to manage sources!
  - Pure build managers:
    HdlMake, Bender, FuseSoC, …
  - Verification framework oriented build managers:
    VUnit, HdlRegression (UVVM), OSVVM

The problem

Solution: Build managers, a breakdown

- Project definition
  - Managing file lists and properties for VHDL & SystemVerilog
  - “Executable” configuration method: python or tcl based
  - Modular projects + dependency management
    - Resolving dependencies with git submodules
    - Package (IP) manager with version conflict resolution
- Tool execution plan:
  - Core generation, Simulation, Synthesis, Lint
  - Generate script or directly run the tool
- Post-processing:
  - Generate reports for CI

The problem

Solution: Build managers, a breakdown

- Project definition
  - Managing file lists and properties for VHDL & SystemVerilog
  - “Executable” configuration method: python or tcl based
  - Modular projects + dependency management
    - Resolving dependencies with git submodules
    - Package (IP) manager with version conflict resolution

- Tool execution plan:
  - Core generation, Simulation, Synthesis, Lint
  - Generate script or directly run the tool

- Post-processing:
  - Generate reports for CI

- Flexibly defining hdl modules
- Composing modules
↳ Generate compilation ‘plan’

Turn compilation plan into tool execution

Process the results

The problem

Three ± orthogonal problems

- Project definition
  - Simple: VUnit, HdlRegression, OSVVM,
  - Full package management: HdlMake, FuseSoc, Bender
- Tool execution plan
  - YMMV
- Result postprocessing
  - Verification reporting: VUnit, HdlRegression, OSVVM

The problem

Three ± orthogonal problems

- Project definition
  - Simple: VUnit, HdlRegression, OSVVM,
  - Full package management: HdlMake, FuseSoc, Bender

  Problem with many opinionated solutions
- Tool execution plan
  - YMMV

  Why implement this again & again
- Result postprocessing
  - Verification reporting: VUnit, HdlRegression, OSVVM

  Problem with many opinionated solutions

The problem

- How can I package my libraries such that they can be effortlessly consumed by many package managers?

- How can I use my RTL, developed with FuseSoC and test it using the OSVVM scripting framework?

- If I build a new linter, should I make PRs for all package managers?

The answer should not always be more scripting!
We’re missing a generic way to exchange (sub-)project information.

The problem

Tool implementations for build managers:

- N build managers
- M tools

N*M implementations

Direct interoperability between build managers:

- N build managers = N*N implementations

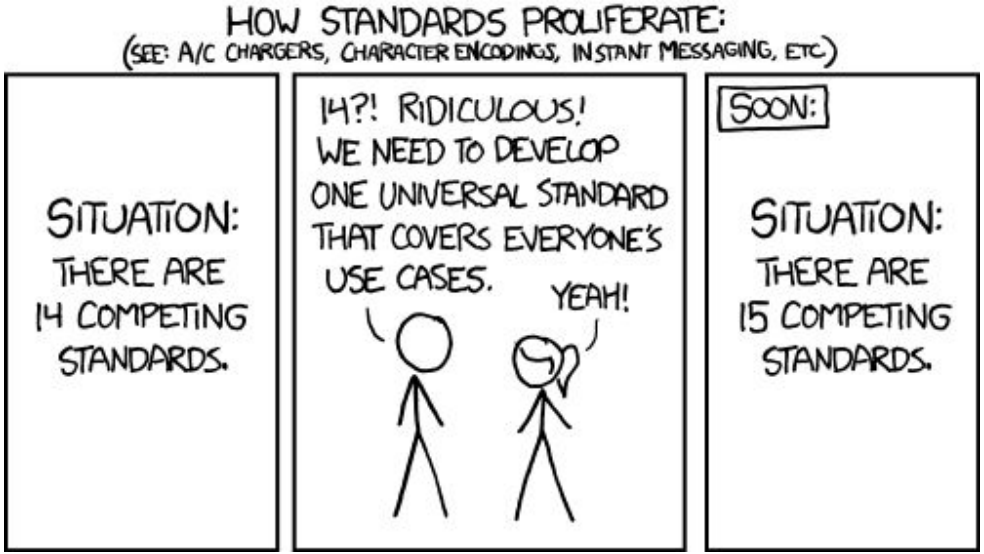
How to solve? We need *one* more layer of indirection!

A solution

- One OSS build manager to rule them all !

A Solution

- One OSS build manager to rule them all !



A Solution: build exchange format

- A way to package your HDL sources with a tool agnostic compilation plan
  - Does not mean it will work with any tool
  - Doesn’t contain all the information to simulate or synthesise
- Easy to ship with your library and generate
  - An easily accessible format, not an executable script
  - Covers all compilation options for vhdl ànd SystemVerilog
- Easy to consume by your build manager without additional scripting
- One OSS project that compiles that compilation plan for all tools

A Solution: build exchange format

Where exchange format?

We have a JSON format developed at Sigasi
that covers all of VHDL & SystemVerilog

but the format itself is not the difficult part

Discussion

- Keep open sourcing !

- Great OSS creates an ecosystem
  - Before you start a new project think about what is already there
  - Collaborate on OSS,
    expand ecosystems
    think about the bigger picture

- The siloed ecosystems for VHDL make me sad