BENDER

A dependency management tool for hardware design projects

Fabian Schuiki
**STATUS QUO**

**IPApprox / iptools**

- Distinction between IPs/chips
- No transitive dependencies
  - IPs don’t know their deps
  - Chips must list all deps
- Tool embedded into chip repository
- Mixes:
  - IPs worked on by the user
  - IPs checked out by the tool
STATUS QUO

IPApprox / iptools
- Distinction between IPs/chips
- No transitive dependencies
  - IPs don’t know their deps
  - Chips must list all deps
- Tool embedded into chip repository
- Mixes:
  - IPs worked on by the user
  - IPs checked out by the tool

A SHOT AT SOMETHING NEW

Bender
- A replacement tool to fix these issues
- A joint effort by: Andreas Kurth, Francesco Conti, Stefan Mach, Florian Zaruba
- Repository and binaries:
github.com/fabianschuiki/bender
- Or use cargo to build it
  > cargo install bender
- Or build it yourself:
  > git clone <url> bender
  > cd bender
  > cargo install
Design Goals
- Transitive dependencies
- Transitive dependencies
- Tier-based, hands-off, opt-in policy
  - **Tier 1**: Resolve package dependencies
  - **Tier 2**: Collect source files
  - **Tier 3**: Feed the tools
THE WISHLIST

Design Goals

- Transitive dependencies
- Tier-based, hands-off, opt-in policy
  - Tier 1: Resolve package dependencies
  - Tier 2: Collect source files
  - Tier 3: Feed the tools
- No central registry
THE WISHLIST

Design Goals

- Transitive dependencies
- Tier-based, hands-off, opt-in policy
  - Tier 1: Resolve package dependencies
  - Tier 2: Collect source files
  - Tier 3: Feed the tools
- No central registry
- Tailored to ASIC flow
  - Ultra conservative in updating IPs
  - Reproducible builds
THE WISHLIST

- Transitive dependencies
- Tier-based, hands-off, opt-in policy
  - **Tier 1**: Resolve package dependencies
  - **Tier 2**: Collect source files
  - **Tier 3**: Feed the tools
- No central registry
- Tailored to ASIC flow
  - Ultra conservative in updating IPs
  - Reproducible builds
- Written in compiled language for static checks
TRANSITIVE DEPENDENCIES

- IPs cannot declare their dependencies
- no standalone build for IPs
TRANSITIVE DEPENDENCIES

- IPs cannot declare their dependencies
- no standalone build for IPs
TRANSITIVE DEPENDENCIES

- IPs cannot declare their dependencies
- no standalone build for IPs

```
tosca  →  tosca-cluster
```
TRANSITIVE DEPENDENCIES

- IPs cannot declare their dependencies
- no standalone build for IPs

Diagram:
- tosca
  - tosca-cluster
  - riscv
- IPs cannot declare their dependencies
- no standalone build for IPs
TRANSITIVE DEPENDENCIES

- IPs cannot declare their dependencies
- no standalone build for IPs

Diagram:
- tosca
- tosca-cluster
- riscv
- axi
- axi_slice
TRANSITIVE DEPENDENCIES

- IPs cannot declare their dependencies
- no standalone build for IPs

Diagram:

- `tosca` -> `tosca-cluster`
- `riscv`
- `axi`
- `axi_slice`
- `common_cells`
TRANSITIVE DEPENDENCIES

- IPs cannot declare their dependencies
- no standalone build for IPs
TRANSITIVE DEPENDENCIES

- IPs cannot declare their dependencies
- no standalone build for IPs
TRANSITIVE DEPENDENCIES

- IPs cannot declare their dependencies
- no standalone build for IPs

Current | Bender
TRANSITIVE DEPENDENCIES

- IPs cannot declare their dependencies
- no standalone build for IPs

*Current*

*Tier 1*

*Bender*
SEMANTIC VERSIONING
SEMANTIC VERSIONING

- The problem with transitive dependencies:

```
riscv    382371f
    ↓
axi_slice 63e1b76
    ↓
common_cells
```
- The problem with transitive dependencies:

Are these commits compatible?
Which one do we pick?
The problem with transitive dependencies:

- Semantic Versioning (semver.org)

Are these commits compatible? Which one do we pick?

- The solution: Semantic Versioning (semver.org)

- Increment major version on breaking changes
- Increment minor version on backwards-compatible changes
- Increment patch version otherwise

Be careful with HDLs... many changes are breaking.
The problem with transitive dependencies:

The solution: Semantic Versioning (semver.org)

- riscv
- axi_slice
- common_cells
- 382371f
- 63e1b76

Are these commits compatible? Which one do we pick?

1.2.3

Major Version

Minor Version

Patch Version

- Increment major version on breaking changes
- Increment minor version on backwards-compatible changes
- Increment patch version otherwise

Be careful with HDLs… many changes are breaking.
- Make sure you know exactly what dependency versions were used for tape out.
- Make sure you know exactly what dependency versions were used for tape out
- Software faces this problem as well (e.g. composer, cargo, etc.)
- Make sure you know exactly what dependency versions were used for tape out
- Software faces this problem as well (e.g. composer, cargo, etc.)
- Solution: Use a lock file!
  - Tracks exact hash of each dependency

**Manifest**

```yaml
# Bender.yml
axi:       master
axi_slice: master
axi_node:  v1.0.1
riscv:     fixes
axi2mem:   master
mem2axi:   34e598c
jtag:      master
```

**Lock File**

```yaml
# Bender.lock
axi:       d1a671e
axi_slice: f2e4abb
axi_node:  ac692ad
riscv:     352a9c6
axi2mem:   ead844f
mem2axi:   34e598c
jtag:      2b5a6ca
```
REPRODUCIBLE BUILDS

- Make sure you know exactly what dependency versions were used for tape out
- Software faces this problem as well (e.g. composer, cargo, etc.)
- Solution: Use a lock file!
  - Tracks exact hash of each dependency
- Dependencies only update ...
  - to resolve version conflicts
  - when you ask for it

```
> bender update
> bender update axi_slice
```

---

**Manifest**

```
# Bender.yml
axi: master
axi_slice: master
axi_node: v1.0.1
riscv: fixes
axi2mem: master
mem2axi: 34e598c
jtag: master
```

**Lock File**

```
# Bender.lock
axi: d1a671e
axi_slice: f2e4abb
axi_node: ac692ad
riscv: 352a9c6
axi2mem: ead844f
mem2axi: 34e598c
jtag: 2b5a6ca
```
- Go through each dependency, determine which version to use
- Do the same for dependencies of dependencies, …
- Go through each dependency, determine which version to use
  - Do the same for dependencies of dependencies, …
- This is the tricky part
- Go through each dependency, determine which version to use
  - Do the same for dependencies of dependencies, …
- This is the tricky part
- Semantic versioning helps here:
  - Dependencies specified with a range of compatible versions
  - Can make a table of available versions and start crossing out
- Go through each dependency, determine which version to use
  - Do the same for dependencies of dependencies, …
- This is the tricky part
- Semantic versioning helps here:
  - Dependencies specified with a range of compatible versions
  - Can make a table of available versions and start crossing out
- Implemented as backtracking algorithm
DEPENDENCY RESOLUTION

- Go through each dependency, determine which version to use
  - Do the same for dependencies of dependencies, …
- This is the tricky part
- Semantic versioning helps here:
  - Dependencies specified with a range of compatible versions
  - Can make a table of available versions and start crossing out
- Implemented as backtracking algorithm

As a hardware developer, you don’t want to do this!
DEPENDENCY RESOLUTION

Simple Example

A
  v1.0
  B
  v1.1
  C
  v1.1
  D
We don’t have a registry with the dependency graph
- We don’t have a registry with the dependency graph
- Collecting graph a priori from git repositories not feasible
- We don’t have a registry with the dependency graph
- Collecting graph a priori from git repositories not feasible
- “Discover” dependencies on the fly
- We don’t have a registry with the dependency graph
- Collecting graph a priori from git repositories not feasible
- “Discover” dependencies on the fly
We don’t have a registry with the dependency graph
Collecting graph a priori from git repositories not feasible
“Discover” dependencies on the fly
We don’t have a registry with the dependency graph
- Collecting graph a priori from git repositories not feasible
- “Discover” dependencies on the fly
- We don’t have a registry with the dependency graph
- Collecting graph a priori from git repositories not feasible
- “Discover” dependencies on the fly
- We don’t have a registry with the dependency graph
- Collecting graph a priori from git repositories not feasible
- “Discover” dependencies on the fly
- We don’t have a registry with the dependency graph
- Collecting graph a priori from git repositories not feasible
- “Discover” dependencies on the fly
- We don’t have a registry with the dependency graph
- Collecting graph a priori from git repositories not feasible
- “Discover” dependencies on the fly
- We don’t have a registry with the dependency graph
- Collecting graph a priori from git repositories not feasible
- “Discover” dependencies on the fly
The dependency resolution diagram shows a simple example of how versions of different components interact:

- Component A depends on version v1.1 of Component B and C.
- Component B has two dependencies: version v1.3 of Component D and version v1.2 of Component C.
- Component C depends on version v1.0 of Component E.
- Component D depends on version v1.2 of Component B.
- Component E has no dependencies.

This diagram illustrates how version dependencies are resolved in a system.
DEPENDENCY RESOLUTION

Simple Example

A
v1.0

B
v1.1
v1.3: v2
v1.2: v1

C
v1.1
≤v1.2
v1.0

D

E
Simple Example

A → B → C → D → E

A

v1.0

B

v1.1

C

v1.1

D

≤ v1.2

v1.3: v2

v1.2: v1

E

v1.0
DEPENDENCY RESOLUTION

Simple Example

A

B
v1.3: v2
v1.2: v1

C
v1.3: v2
v1.2: v1

D
≤v1.2

E
v1.0

A  B  C
v1.0  v1.3  v1.1

v1.2  v1.0

v1.1

v1.0
DEPENDENCY RESOLUTION

Simple Example

A

B
v1.1
v1.3: v2
v1.2: v1

C
v1.1

D
≤v1.2

E
≤v1.0

A

B
v1.3

C
v1.1

D

E
v1.0
Simple Example

**A**

**B**

**C**

**D**

**E**

**A**

**B, C, D, E**

**A**

**B**

**C**

**D**

**E**

**A**

**B**

**C**

**D**

**E**
DEPENDENCY RESOLUTION

Simple Example

A: v1.0, v1.1
B: v1.3, v1.2, v1.1
C: v1.1, v1.0
D: v2.0, v1.0, v1.1
E: v1.0
Simple Example

A → B → C
D → E

Version dependencies:
- A: v1.0
- B: v1.3
- C: v1.1
- D: v2.0
- E: v1.0

- v1.0
- v1.3
- v1.1
- v2.0
- v1.0
- v1.0
- v1.0
DEPENDENCY RESOLUTION

Simple Example

A

B

C

D

E

v1.1

v1.1

v1.2

≤ v1.2

≤ v1.0

v1.3: v2
v1.2: v1

v1.0

v1.3

v1.1

v1.2

v1.0

v1.0

v1.1

v1.0

v1.0
## Simple Example

### Depency Graph

- **A**
  - **B**
    - **D**
      - **E**
        - **C**
          - **B**
            - **A**

### Version Numbers

- **A**: v1.0
- **B**: v1.3
- **C**: v1.1, v1.0
- **D**: v2.0, v1.0
- **E**: v1.0

- **D** depends on **B**: v1.2
- **B** depends on **A**: v1.1

- **C** depends on **B**: v1.2, v1.1

- **E** depends on **D**: v1.0

**Simple Example**

- **A**: Requires `v1.0` and `v1.1`
- **B**: Requires `v1.3`, `v1.2`, and `v1.1`
- **C**: Requires `v1.3`, `v1.2`, and `v1.1`
- **D**: Requires `v1.3`, `v1.2`, and `v1.0`
- **E**: Requires `v1.0`

**Dependency Graph**

- A → B
- A → C
- B → D
- B → E
- C → D
- C → E

**Version Numbers**

- **A**: `v1.0` and `v1.1`
- **B**: `v1.3`, `v1.2`, and `v1.1`
- **C**: `v1.3`, `v1.2`, and `v1.1`
- **D**: `v1.3`, `v1.2`, and `v1.0`
- **E**: `v1.0`
**DEPENDENCY RESOLUTION**

*Simple Example*

```
A  v1.0  v1.3  v1.2  v1.1  v1.0
B  v1.2  v1.1  v1.0
C  v1.1  v1.0
D  v2.0  v1.0
E  v1.0

A  v1.1  v1.1
B  v1.3  v1.1
C  v1.2  v1.0
D  v1.0
E  v1.0

v1.3: v2
v1.2: v1
≤ v1.2
v1.0
```
DEPENDENCY RESOLUTION

Simple Example

A

B

v1.1

C

D

E

v1.1

v1.3: v2

v1.2: v1

≥v1.2

v1.0

A

B

C

D

E

v1.0

v1.3

v1.1

v2.0

v1.0

v1.2

v1.0

v1.0

v1.1

v1.0
Simple Example

A

B

C

D

E

v1.1

v1.1

v1.2: v1

v1.3: v2

v1.2

v1.0

v1.1

v1.0

v1.0

v1.0

v1.0

v1.0

v1.3

v1.1

v2.0

v1.0

v1.1
DEPENDENCY RESOLUTION

Simple Example

A

B

v1.1

v1.3: v2
v1.2: v1

C

D

v1.1

≤v1.2

E

A

B

C

D

E

v1.0

v1.3

v1.1

v1.0

v2.0

v1.0

v1.1

v1.0

v1.0
- We have established dependency tracking as first tier
- We have established dependency tracking as first tier
- Let’s track source files as well

```yaml
# Bender.yml
dependencies:
  ...
sources:
  - src/axi_pkg.sv
  - src/axi_intf.sv
  - src/axi2mem.sv
  - src/mem2axi.sv
```
We have established dependency tracking as first tier
Let’s track source files as well
Allow for groups, include dirs, defines

```yaml
# Bender.yml
dependencies:
...
sources:
- src/axi_pkg.sv
- src/axi_intf.sv
- src/axi2mem.sv
- src/mem2axi.sv
```

```yaml
# Bender.yml
...
sources:
- src/axi_pkg.sv
- include_dirs:
  - src/include
defines:
  FPGA_EMUL: 1
  SKIP_TRACE: 0
files:
- src/axi_intf.sv
- src/axi2mem.sv
- src/mem2axi.sv
```
- Each dependency declares its source files
SOURCE FILES

- Each dependency declares its source files
- Build a compilation recipe:
  - Topologically sort the dependency graph
  - Concatenate source files in that order
SOURCE FILES

- Each dependency declares its source files
- Build a compilation recipe:
  - Topologically sort the dependency graph
  - Concatenate source files in that order
SOURCE FILES

- Each dependency declares its source files
- Build a compilation recipe:
  - Topologically sort the dependency graph
  - Concatenate source files in that order

**Topological Ordering**

```
  A 0
   ↙ 1
   B 1
     ↘ 2
     D 2
  ↘ 2
  C 2
     ↙ 3
     E 3
```
- Each dependency declares its source files
- Build a compilation recipe:
  - Topologically sort the dependency graph
  - Concatenate source files in that order
Each dependency declares its source files.

Build a compilation recipe:
- Topologically sort the dependency graph
- Concatenate source files in that order

Topological Ordering

Tier 2

SOURCE FILES

- E/top.sv
- E/pkg.sv
- C/foo.vhd
- C/bar.vhd
- D/ctrl.sv
- D/datapath.sv
- D/export.sv
- B/top.sv
- A/padframe.sv
- A/top.sv
Each dependency declares its source files

Build a compilation recipe:
- Topologically sort the dependency graph
- Concatenate source files in that order

Don’t want to do this by hand!
TARGETS

- We have the same source files for ...
TARGETS

- We have the same source files for ...
- ... different technologies:
  - ASIC (gf22, umc65, tsmc45, smic130, etc.)
  - FPGA (xilinx, altera)
- We have the same source files for ...
- ... different technologies:
  - ASIC (gf22, umc65, tsmc45, smic130, etc.)
  - FPGA (xilinx, altera)
- ... different use cases:
  - RTL simulation
  - RTL synthesis
  - Post-synthesis simulation
  - Post-layout simulation
  - Linting
TARGETS

- We have the same source files for ...
- ... different technologies:
  - ASIC (gf22, umc65, tsmc45, smic130, etc.)
  - FPGA (xilinx, altera)
- ... different use cases:
  - RTL simulation
  - RTL synthesis
  - Post-synthesis simulation
  - Post-layout simulation
  - Linting

```
sources:
  - src/queue.sv
  - target: fpga
    files:
      - src/fifo_fpga.sv
  - target: not(fpga)
    files:
      - src/fifo_generic.sv
```
TARGETS

- We have the same source files for ...
- ... different technologies:
  - ASIC (gf22, umc65, tsmc45, smic130, etc.)
  - FPGA (xilinx, altera)
- ... different use cases:
  - RTL simulation
  - RTL synthesis
  - Post-synthesis simulation
  - Post-layout simulation
  - Linting

sources:
- src/queue.sv
- target: fpga
  files:
    - src/fifo_fpga.sv
- target: not(fpga)
  files:
    - src/fifo_generic.sv

Target Syntax:
Names: fpga,asic,umc65
AND: all(fpga,xilinx)
OR: any(fpga,asic)
NOT: not(fpga)
- Let’s make a simple package without dependencies:

```yaml
# Bender.yml
package:
  name: common_cells
  author: ["John Doe <john@doe.com>"

sources:
  - src/generic_fifo.sv
  - src/round_robin.sv
  - src/leading_zero.sv
```
A RISC-V core that depends on a few other repositories:

```
# Bender.yml
package:
    name: riscv
    author: ["John Doe <john@doe.com>"]
dependencies:
    common_cells: { git: "../common_cells.git", version: 1.0.2 }
    tech_cells: { git: "../tech_cells.git", version: 0.5.3 }
sources:
    - src/riscv_core.sv
    - src/riscv_ctrl.sv
```

```bash
> tree
Bender.yml
LICENSE
README.md
src/
    riscv_core.sv
    riscv_ctrl.sv
```

- riscv
- common_cells
- tech_cells
EXAMPLE

- A chip repository that will be taped out
  - Before: Put *Bender.lock* in .gitignore
  - Here: **Commit** *Bender.lock* to get **reproducible** builds!

```yaml
# Bender.yml
package:
  name: tosca
  author: ["John Doe <john@doe.com>"]
dependencies:
  tosca-cluster: { git: ... }
  axi: { git: ... }
  axi_slice: { git: ... }
sources:
  - src/top.sv
  - src/padframe.sv
```

An entire chip

> tree
Bender.yml
Bender.lock
LICENSE
README.md
src/
  top.sv
  padframe.sv
FEEDING THE TOOLS

- We have all source files for an entire dependency graph.
WEEDING THE TOOLS

- We have all source files for an entire dependency graph.
- We need to feed many tools (different from software):
  - vsim
  - ncsim
  - synopsys
  - genus
  - spyglass
  - vivado
  - quartus
FEEDING THE TOOLS

- We have all source files for an entire dependency graph.
- We need to feed many tools (different from software):
  - vsim
  - ncsim
  - synopsys
  - genus
  - spyglass
  - vivado
  - quartus
- Can be done manually
FEEDING THE TOOLS

- We have all source files for an entire dependency graph.
- We need to feed many tools (different from software):
  - vsim
  - ncsim
  - synopsys
  - genus
  - spyglass
  - vivado
  - quartus
- Can be done manually
- Or have Bender do it for you…

```json
> bender sources
{
  "include_dirs": [...],
  "defines": {...},
  "files": [...],
}
```
Bender can maintain tool scripts for you
Bender can maintain tool scripts for you

Currently supported targets:
- Synopsys Design Compiler “analyze” scripts
- QuestaSim compile scripts
SCRIPT GENERATION

- Bender can maintain tool scripts for you
- Currently supported targets:
  - Synopsys Design Compiler “analyze” scripts
  - QuestaSim compile scripts

> bender script vsim > compile.tcl

```tcl
# compile.tcl
vlog +define+TARGET_VSIM \\
"src/riscv_top.sv" \\
...```
- Bender can maintain tool scripts for you
- Currently supported targets:
  - Synopsys Design Compiler “analyze” scripts
  - QuestaSim compile scripts
SCRIPT GENERATION

- Bender can maintain tool scripts for you
- Currently supported targets:
  - Synopsys Design Compiler “analyze” scripts
  - QuestaSim compile scripts
- Experimental support for edalize

```tcl
# analyze.tcl
lappend search_path "src/include"
analyze -format sv -define { \n    TARGET_SYNOPSYS \n    TARGET_SYNTHESIS \n} [list \n    "src/riscv_top.sv" \n]
```

```tcl
# compile.tcl
vlog +define+TARGET_VSIM \n    "src/riscv_top.sv" \n    ...
```
SCRIPT GENERATION

- Bender can maintain tool scripts for you
- Currently supported targets:
  - Synopsys Design Compiler “analyze” scripts
  - QuestaSim compile scripts
- Experimental support for edalize
- Scripts can be checked into version control
  - Takes bender out of the EDA loop
  - Can share environment with collaborators that do not have bender installed

Opt-in!
- Plugins allow Bender to be extended easily with custom commands
- A regular dependency
- Offers commands to the user:
  - can be simple scripts
  - can be entire executables
- Plugins allow Bender to be extended easily with custom commands
- A regular dependency
- Offers commands to the user:
  - can be simple scripts
  - can be entire executables

```yaml
# Bender.yml
package:
  name: bender-vsim
  author: ["John Doe <john@doe.com>"]
plugins:
  vsim: "do_stuff.sh"
```
**PLUGINS**

- Plugins allow Bender to be extended easily with custom commands
- A regular dependency
- Offers commands to the user:
  - can be simple scripts
  - can be entire executables

```bash
#!/bin/bash
# do_stuff.sh
SOURCES=`$BENDER sources`
for FILE in $SOURCES; do
    vlog-10.6b $FILE
done
echo "run -all" | vsim-10.6b -c
```

---

```
# Bender.yml
package:
    name: bender-vsim
    author: [“John Doe <john@doe.com>"

plugins:
    vsim: “do_stuff.sh”
```
TESTING

One-button testing

Future!
- I would like testing to be one button away

> bender test --all
All 4 tests passed.

Future!
TESTING

- I would like testing to be one button away
- Can be implemented as another plugin:
  1. Compilation tests with installed tools
     ➤ vsim/ncsim
     ➤ synopsys/genus
     ➤ spyglass/verilator
     ➤ vivado
  2. Run unit/regression tests
     ➤ vsim/ncsim

> bender test --all
All 4 tests passed.

Future!
TESTING

- I would like testing to be one button away
- Can be implemented as another plugin:
  1. Compilation tests with installed tools
     - vsim/ncsim
     - synopsys/genus
     - spyglass/verilator
     - vivado

  2. Run unit/regression tests
     - vsim/ncsim

Ensures that IP is compatible with different tools.

> bender test --all
All 4 tests passed.
TESTING

- I would like testing to be one button away
- Can be implemented as another plugin:
  1. Compilation tests with installed tools
     ➤ vsim/ncsim
     ➤ synopsys/genus
     ➤ spyglass/verilator
     ➤ vivado
  2. Run unit/regression tests
     ➤ vsim/ncsim
- Can be easily integrated into CI

> bender test --all
All 4 tests passed.

Ensures that IP is compatible with different tools.

One-button testing

Future!
I would like testing to be one button away
Can be implemented as another plugin:

1. Compilation tests with installed tools
   - vsim/ncsim
   - synopsys/genus
   - spyglass/verilator
   - vivado

2. Run unit/regression tests
   - vsim/ncsim

Can be easily integrated into CI

Ensures that IP is compatible with different tools.

> bender test --all
All 4 tests passed.

# Bender.yml

```yaml
package:
  name: bender-vsim
  author: ["John Doe <john@doe.com>"]

test:
  compile: [vsim, vivado, synopsys]
  benches:
    - test/tb_one.sv
    - test/tb_two.sv
  cases:
    a: [tb_one, NUM_MASTER=[1,2,3]],
    b: [tb_two, NUM_SLAVE=[3,4,9]],
```

Future!
REGISTRY for convenience and open source releases
- Makes it easy to find existing IPs (“Has anyone created a protocol adapter?”)
REGISTRY

- Makes it easy to find existing IPs (“Has anyone created a protocol adapter?”)
- Typing Git URLs for dependencies is tedious and error prone:

```json
common_cells: { git: "../common_cells.git", version: 1.0.2 }
tech_cells:   { git: "../tech_cells.git", version: 0.5.3 }
axi:          { git: "../axi.git", version: 0.2 }
```
REGISTRY

- Makes it easy to find existing IPs ("Has anyone created a protocol adapter?")
- Typing Git URLs for dependencies is tedious and error prone:

```
common_cells: { git: "../common_cells.git", version: 1.0.2 }
tech_cells:   { git: "../tech_cells.git",   version: 0.5.3 }
axi:          { git: "../axi.git",          version: 0.2  }
```

- Solution: Create a registry!
  - Simply a file on a web server which lists Git repositories
  - Can have multiple registries (pulp-restricted vs. pulp-open)
- Makes it easy to find existing IPs (“Has anyone created a protocol adapter?”)
- Typing Git URLs for dependencies is tedious and error prone:

```
common_cells: { git: "…/common_cells.git", version: 1.0.2 }
tech_cells:   { git: "…/tech_cells.git", version: 0.5.3 }
axi:          { git: "…/axi.git", version: 0.2 }
```

- Solution: Create a registry!
  - Simply a file on a web server which lists Git repositories
  - Can have multiple registries (pulp-restricted vs. pulp-open)

```
common_cells: 1.0.2
tech_cells:   0.5.3
axi:          0.2
```
- Helps with open-source releases
I HATE YOUR TOOL!

Alternatives
I HATE YOUR TOOL!

Alternatives

- Bazel
I HATE YOUR TOOL!

Alternatives

- Bazel
- The award-winning FuseSoC
I HATE YOUR TOOL!

Alternatives

- Bazel
- The award-winning FuseSoC
- npm?
I HATE YOUR TOOL!

Alternatives

- Bazel
- The award-winning FuseSoC
- npm?
- others?
FUTURE WORK

- Integration with FuseSoC/edalize? 😊
- Add support for more tools
- Features
- Automation/conventions for unit tests
CONCLUSION

Bender is here to help you!
Bender is here to help you!

1. Transitive dependency resolution
CONCLUSION

Bender is here to help you!

1. Transitive dependency resolution
2. Source file ordering and management
Bender is here to help **you**!

1. Transitive dependency resolution
2. Source file ordering and management
3. Registry and feeding the tools
Thanks!

https://github.com/fabianschuiki/bender

    > cargo install bender

    —— and ——

    > git clone <url> bender
    > cd bender
    > cargo install