Veryl: A Modern Hardware Description Language For Open Source Hardware Design

Naoya Hatta [1] Taichi Ishitani [1] Nathan Bleier [2]

June 19, 2025

[1]Pezzy Computing, K.K.

[2]University of Michigan

The Open Source Hardware Research Challenge

- Error-Prone Constructs
  - Sim/Synth semantic mismatches
  - Clock domain crossing errors
  - Non-synthesizable code

```
module reset_bug (
  input clk, rst,
  input [31:0] data_in,
  output reg [31:0] data_out
);
  // Malformed sensitivity list - common bug
  // Missing 'posedge' or 'negedge' for reset
  always_ff @(posedge clk or rst) begin
    if (rst)
      data_out <= 0;
    else
      data_out <= data_in;
  end
endmodule
```

The Open Source Hardware Research Challenge

- Error-Prone Constructs
  - Sim/Synth semantic mismatches
  - Clock domain crossing errors
  - Non-synthesizable code
- Academic Productivity Issues
  - No real-time error checking
  - Manual dependency management
  - Poor integration with modern editors

```
// No real-time feedback on errors
module mystery_bug (
  input clk, rst,
  input [31:0] data_in,
  output [31:0] data_out
);
  // Bug only found during simulation
  always @(posedge clk or rst) begin
    if (rst)
      data_out <= 0;
    else
      data_out <= data_in + 1;
  end
endmodule
```

The Open Source Hardware Research Challenge

- Error-Prone Constructs
  - Sim/Synth semantic mismatches
  - Clock domain crossing errors
  - Non-synthesizable code
- Academic Productivity Issues
  - No real-time error checking
  - Manual dependency management
  - Poor integration with modern editors
- Collaboration Friction
  - Complex build setup for newcomers
  - Ad-hoc component re-use
  - Ad-hoc documentation workflow

```
# Typical research project setup
$ git clone mysterious_repo
$ cd mysterious_repo
# Now what? No clear build instructions
$ make # Command not found
$ vivado # Need license, specific version
# Email the author for build instructions
```

The Open Source Hardware Research Challenge

- Error-Prone Constructs
  - Sim/Synth semantic mismatches
  - Clock domain crossing errors
  - Non-synthesizable code
- Academic Productivity Issues
  - No real-time error checking
  - Manual dependency management
  - Poor integration with modern editors
- Collaboration Friction
  - Complex build setup for newcomers
  - Ad-hoc component re-use
  - Ad-hoc documentation workflow
- Integration with Existing Codebases
  - Unreadable generated SV
  - Hard to debug generated code

```
// Generated by SpinalHDL - hard to debug!
assign _zz_176 = 3'b100;
assign _zz_177 = execute_INSTRUCTION[19:15];
assign _zz_180 = ($signed(_zz_181) + $signed(_zz_184));
assign _zz_181 = ($signed(_zz_182) + $signed(_zz_183));
assign _zz_184 = (execute_SRC_USE_SUB_LESS ? _zz_185 : _zz_186);
assign _zz_185 = 32'h00000001;
assign _zz_186 = 32'h0;
// What does this actually do?
// Can't integrate with existing SV modules easily
// Can't effectively modify generated SV manually
// Can't intuitively map generated SV to Spinal source-code
```

The Research Productivity Gap

Software Research Today:

- Language servers and Tree-Sitter
- Package managers for dependencies
- Auto-formatters for consistency
- Integrated docs (e.g., rust-docs)
- CI/CD for reproducible builds

Hardware Research Reality:

- Basic syntax highlighting
- Manual `` `include `` statements
- "House style" varies by lab
- Separate documentation tools
- Complex vendor-specific flows

The Research Productivity Gap

Software Research Today:

- Language servers and Tree-Sitter
- Package managers for dependencies
- Auto-formatters for consistency
- Integrated docs (e.g., rust-docs)
- CI/CD for reproducible builds

Hardware Research Reality:

- Basic syntax highlighting
- Manual `` `include `` statements
- "House style" varies by lab
- Separate documentation tools
- Complex vendor-specific flows

Young researchers expect modern tooling!

The Research Productivity Gap

Software Research Today:

- Language servers and Tree-Sitter
- Package managers for dependencies
- Auto-formatters for consistency
- Integrated docs (e.g., rust-docs)
- CI/CD for reproducible builds

Hardware Research Reality:

- Basic syntax highlighting
- Manual `` `include `` statements
- "House style" varies by lab
- Separate documentation tools
- Complex vendor-specific flows

Young researchers expect modern tooling!
And they should!

Why This Matters for Open Source Computer Architecture Research

- **Barrier to Entry:** New students struggle with archaic tooling

Why This Matters for Open Source Computer Architecture Research

- **Barrier to Entry:** New students struggle with archaic tooling
- **Collaboration Friction:** Difficult to share and build upon others' work

Why This Matters for Open Source Computer Architecture Research

- **Barrier to Entry:** New students struggle with archaic tooling
- **Collaboration Friction:** Difficult to share and build upon others' work
- **Research Velocity:** Custom, manual, & buggy processes slow down iteration

Why This Matters for Open Source Computer Architecture Research

- **Barrier to Entry:** New students struggle with archaic tooling
- **Collaboration Friction:** Difficult to share and build upon others' work
- **Research Velocity:** Custom, manual, & buggy processes slow down iteration
- **Reproducibility:** Complex build systems make reproducibility difficult

Why This Matters for Open Source Computer Architecture Research

- **Barrier to Entry:** New students struggle with archaic tooling
- **Collaboration Friction:** Difficult to share and build upon others' work
- **Research Velocity:** Custom, manual, & buggy processes slow down iteration
- **Reproducibility:** Complex build systems make reproducibility difficult

Open source hardware research needs a modern language infrastructure

Veryl: Research-Focused Design Principles

1. Gradual Adoption in Research Projects
    - Bidirectional SystemVerilog interoperability
    - Generate safe, readable SystemVerilog for debugging & integration

Veryl: Research-Focused Design Principles

1. Gradual Adoption in Research Projects
   - Bidirectional SystemVerilog interoperability
   - Generate safe, readable SystemVerilog for debugging & integration
2. Modern Research Infrastructure
   - Language server for real-time error checking
   - Package manager for sharing research components
   - Automated formatting for consistent collaboration
   - Integrated documentation with diagrams

Veryl: Research-Focused Design Principles

1. Gradual Adoption in Research Projects
   - Bidirectional SystemVerilog interoperability
   - Generate safe, readable SystemVerilog for debugging & integration
2. Modern Research Infrastructure
   - Language server for real-time error checking
   - Package manager for sharing research components
   - Automated formatting for consistent collaboration
   - Integrated documentation with diagrams
3. Academic Workflow Support
   - ASIC/FPGA portability for prototyping
   - Built-in testing framework integration
   - Reproducible project configuration with version management
   - CI/CD Pipeline to automate testing & artifact evaluation

Gradual Adoption: Bidirectional SystemVerilog Interoperability

Veryl seamlessly integrates with existing SystemVerilog ecosystem

Top-level SystemVerilog:

```sv
// cpu_core.sv - Existing research CPU
module cpu_core (
  input logic clk, rst_n,
  input logic [31:0] instr,
  input logic [31:0] fp_reg_a,
  input logic [31:0] fp_reg_b,
  input logic [3:0] fp_op,
  output logic [31:0] result
);
  // Instantiate new Veryl FPU
  fp_alu u_fpu (
    .clk(clk), .rst_n(rst_n),
    .op_a(fp_reg_a), .op_b(fp_reg_b),
    .operation(fp_op),
    .result(fp_result)
  );
endmodule
```

Veryl FPU:

```
// fp_alu.veryl
module fp_alu (
    clk       : input  clock               ,
    rst       : input  reset_async_low     ,
    op_a      : input  logic          <32>,
    op_b      : input  logic          <32>,
    operation: input  logic           <3> ,
    z         : output logic          <32>,
) {
    inst u_macc: $sv::DW_fp_mac (
        clk  : clk   , rst_n: rst   ,
        a    : op_a  , b    : op_b  ,
        c    : 32'h0 , rct  : 3'b000,
        z    : z     ,
    );
    // ...
}
```

Generated SystemVerilog:

```sv
// fp_alu.sv - Generated from Veryl
module fp_alu (
  input logic clk,
  input logic rst_n,
  input logic [31:0] op_a,
  input logic [31:0] op_b,
  input logic [2:0] operation,
  output logic [31:0] z
);
  // Clean, readable instantiation
  DW_fp_mac u_macc (
    .clk(clk), .rst_n(rst_n),
    .a(op_a), .b(op_b), .c(32'h0),
    .rct(3'b000), .z(z)
  );
  // ...
endmodule
```

Gradual Adoption: Bidirectional SystemVerilog Interoperability

Veryl seamlessly integrates with existing SystemVerilog ecosystem

Top-level SystemVerilog:

```
// cpu_core.sv - Existing research CPU
module cpu_core (
  input logic clk, rst_n,
  input logic [31:0] instr,
  input logic [31:0] fp_reg_a,
  input logic [31:0] fp_reg_b,
  input logic [3:0] fp_op,
  output logic [31:0] result
);
  // Instantiate new Veryl FPU
  fp_alu u_fpu (
    .clk(clk), .rst_n(rst_n),
    .op_a(fp_reg_a), .op_b(fp_reg_b),
    .operation(fp_op),
    .result(fp_result)
  );
endmodule
```

Veryl FPU:

```
// fp_alu.veryl
module fp_alu (
    clk       : input  clock               ,
    rst       : input  reset_async_low     ,
    op_a      : input  logic          <32>,
    op_b      : input  logic          <32>,
    operation: input  logic           <3> ,
    z         : output logic          <32>,
) {
    inst u_macc: $sv::DW_fp_mac (
        clk  : clk   , rst_n: rst   ,
        a    : op_a  , b    : op_b  ,
        c    : 32'h0 , rct  : 3'b000,
        z    : z     ,
    );
    // ...
}
```

Generated SystemVerilog:

```
// fp_alu.sv - Generated from Veryl
module fp_alu (
  input logic clk,
  input logic rst_n,
  input logic [31:0] op_a,
  input logic [31:0] op_b,
  input logic [2:0] operation,
  output logic [31:0] z
);
  // Clean, readable instantiation
  DW_fp_mac u_macc (
    .clk(clk), .rst_n(rst_n),
    .a(op_a), .b(op_b), .c(32'h0),
    .rct(3'b000), .z(z)
  );
  // ...
endmodule
```

- ✓ Veryl uses SV IP
- ✓ SV uses Veryl modules
- ✓ Clean generated code
- ✓ Readable interfaces

Gradual Adoption: Safety for Debugging & Integration

Veryl catches errors early and generates debuggable SystemVerilog

SystemVerilog - Runtime Errors:

```
module unsafe_alu (
  input logic clk, rst,
  input logic [31:0] a, b,
  output logic [31:0] result);
  always_ff @(posedge clk or rst)
    if (rst) result <= 0;
    else result <= a + b;
endmodule
```

Veryl - Compile-time Safety:

```
module safe_alu (
  clk: input clock, i_rst: input reset_async_low,
  a: input logic<32>, b: input logic<32>,
  o_result: output logic<32>,
) {
  // Clock and reset types prevent sensitivity error
  // Reset polarity determined by type or config
  always_ff {
    if_reset { o_result = 0; }
    else { o_result = a + b; }
  }
}
```

Gradual Adoption: Safety for Debugging & Integration

Veryl catches errors early and generates debuggable SystemVerilog

SystemVerilog - Runtime Errors:

```
module unsafe_alu (
  input logic clk, rst,
  input logic [31:0] a, b,
  output logic [31:0] result);
  always_ff @(posedge clk or rst)
    if (rst) result <= 0;
    else result <= a + b;
endmodule
```

- ✓ Clock/reset types prevent sensitivity list errors
- ✓ Reset polarity determined by type system
- ✓ if_reset syntax eliminates malformed constructs
- ✓ Generated code maps clearly to source

Veryl - Compile-time Safety:

```
module safe_alu (
  clk: input clock, i_rst: input reset_async_low,
  a: input logic<32>, b: input logic<32>,
  o_result: output logic<32>,
) {
  // Clock and reset types prevent sensitivity error
  // Reset polarity determined by type or config
  always_ff {
    if_reset { o_result = 0; }
    else { o_result = a + b; }
  }
}
```

Modern Research Infrastructure: Developer Tooling

Language Server

```
module direction_error (
  i_data: input logic<32>,
  o_result: output logic<32>,
) {
  // Error: Cannot assign to input port
  assign i_data = 32'h0;
      // ^^^^^^
      // LSP: Cannot assign to input port
  assign o_result = i_data + 1;
}
```

Modern Research Infrastructure: Developer Tooling

Language Server

```
module direction_error (
  i_data: input logic<32>,
  o_result: output logic<32>,
) {
  // Error: Cannot assign to input port
  assign i_data = 32'h0;
      // ^^^^^^
      // LSP: Cannot assign to input port
  assign o_result = i_data + 1;
}
```

Package Management:

```
[project]
name = "research_cpu"
version = "0.1.0"
license = "Apache-2.0"
repository = "https://github.com/..."
[dependencies]
std = "1.0"
riscv_common = { git = "https://github.com/...",
                 version="0.2.0" }
```

Modern Research Infrastructure: Developer Tooling

Language Server

```
module direction_error (
  i_data: input logic<32>,
  o_result: output logic<32>,
) {
  // Error: Cannot assign to input port
  assign i_data = 32'h0;
      // ^^^^^^
      // LSP: Cannot assign to input port
  assign o_result = i_data + 1;
}
```

Package Management:

```
[project]
name = "research_cpu"
version = "0.1.0"
license = "Apache-2.0"
repository = "https://github.com/..."
[dependencies]
std = "1.0"
riscv_common = { git = "https://github.com/...",
                 version="0.2.0" }
```

Automatic Formatting:

```
$ veryl fmt
$ veryl check
```

Consistent code style & lint checking

Modern Research Infrastructure: Developer Tooling

Language Server

```
module direction_error (
  i_data: input logic<32>,
  o_result: output logic<32>,
) {
  // Error: Cannot assign to input port
  assign i_data = 32'h0;
      // ^^^^^^
      // LSP: Cannot assign to input port
  assign o_result = i_data + 1;
}
```

Package Management:

```
[project]
name = "research_cpu"
version = "0.1.0"
license = "Apache-2.0"
repository = "https://github.com/..."
[dependencies]
std = "1.0"
riscv_common = { git = "https://github.com/...",
                 version="0.2.0" }
```

Automatic Formatting:

```
$ veryl fmt
$ veryl check
```

Consistent code style & lint checking

Publishing & Sharing:

```
$ veryl publish
$ veryl build
```

Share with community & reproducible builds

Modern Research Infrastructure: Documentation

Integrated documentation with diagrams and auto-generation

Rich Documentation Comments:

```
/// # RISC-V ALU
/// Arithmetic Logic Unit for research processor
/// ```wavedrom
/// { signal: [
///   { name: 'clk', wave: 'p.....' },
///   { name: 'op', wave: 'x3.4.5' },
///   { name: 'result', wave: 'x.3.4.' }
/// ]}
/// ```
pub module alu (
  /// ALU operation selector
  i_op: input alu_op_t,
  o_result: output logic<32>,
) { /* ... */ }
```

```
$ veryl doc
```

Modern Research Infrastructure: Documentation

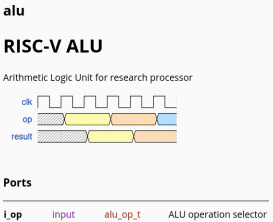
Integrated documentation with diagrams and auto-generation

Rich Documentation Comments:

```
/// # RISC-V ALU
/// Arithmetic Logic Unit for research processor
/// ```wavedrom
/// { signal: [
///   { name: 'clk', wave: 'p.....' },
///   { name: 'op', wave: 'x3.4.5' },
///   { name: 'result', wave: 'x.3.4.' }
/// ]}
/// ```
pub module alu (
  /// ALU operation selector
  i_op: input alu_op_t,
  o_result: output logic<32>,
) { /* ... */ }
```

```
$ veryl doc
```

Generated Documentation:



- Doc for packages, interfaces, modules
- Wavedrom & Mermaid diagram support
- Markdown formatting
- Searchable interface

Academic Workflow Support

Clock Domain Safety:

```
module cdc_example (
  i_clk_a: input 'a clock,
  i_data_a: input 'a logic<32>,
  i_clk_b: input 'b clock,
  o_data_b: output 'b logic<32>,
) {
  // Error: Clock domain crossing detected
  assign o_data_b = i_data_a;
                //      ^^^^^^^^^
                // LSP: Clock domain 'a -> 'b crossing
                //       Use unsafe(cdc) block
                // unsafe (cdc) {
                //   assign o_data_b = i_data_a;
                // }
}
```

Academic Workflow Support

Clock Domain Safety:

```
module cdc_example (
  i_clk_a: input 'a clock,
  i_data_a: input 'a logic<32>,
  i_clk_b: input 'b clock,
  o_data_b: output 'b logic<32>,
) {
  // Error: Clock domain crossing detected
  assign o_data_b = i_data_a;
             //     ^^^^^^^^^
             // LSP: Clock domain 'a -> 'b crossing
             //      Use unsafe(cdc) block
             // unsafe (cdc) {
             //   assign o_data_b = i_data_a;
             // }
}
```

ASIC/FPGA Portability:

```
module clocked (
  i_clk: input clock, i_rst: input reset,
) { /* ... */ }
```

Veryl.toml determines EDA toolchain compatibility:

```
[build]
clock_type = "posedge"
reset_type = "async_low"
flatten_array_interface = true
emit_cond_type = true
implicit_parameter_types = ["string"]
expand_inside_operation = true
```

Academic Workflow Support: Testing Integration

Built-in testing framework integration

Multi-Simulator Support:

```
# Works with multiple simulators
$ veryl test --sim verilator
$ veryl test --sim vcs
$ veryl test --sim vivado
```

Integrated Test Discovery:

```
$ veryl test
[INFO] Compiling test (test_counter)
[INFO] Executing test (test_counter)
[INFO] Succeeded test (test_counter)
[INFO] Completed tests : 1 passed, 0 failed
```

Embedded SystemVerilog Tests:

```
// In counter.veryl
#[test(test_counter)]
embed (inline) sv{{{
  module counter_tb;
    logic clk, rst, [7:0] count;
    counter dut(.*);
    initial begin
      rst = 1; #10; rst = 0;
      repeat(10) @(posedge clk);
      assert(count == 8'h0A);
    end
  endmodule
}}}
// Or include external file:
include(inline, "counter_tb.sv");
```

Academic Workflow Support: Testing Integration

Built-in testing framework integration

Multi-Simulator Support:

```
# Works with multiple simulators
$ veryl test --sim verilator
$ veryl test --sim vcs
$ veryl test --sim vivado
```

Integrated Test Discovery:

```
$ veryl test
[INFO] Compiling test (test_counter)
[INFO] Executing test (test_counter)
[INFO] Succeeded test (test_counter)
[INFO] Completed tests : 1 passed, 0 failed
```

Embedded Cocotb Tests:

```
// In counter.veryl
#[test(test_counter, counter)]
embed (cocotb) py{{{
  import cocotb
  from cocotb.triggers import RisingEdge
  @cocotb.test()
  async def test_counter(dut):
      dut.rst.value = 1
      await RisingEdge(dut.clk)
      dut.rst.value = 0
      assert dut.count.value == 0
}}}
```

Academic Workflow Support: Reproducible Configuration

Verylup - Toolchain Version Manager:

```
# Install specific Veryl toolchain version
$ cargo install verylup
$ verylup install 0.12.0
$ verylup default 0.12.0

# Per-project version pinning
$ verylup override set 0.11.5
$ veryl --version
veryl 0.11.5 (pinned for this project)

# List available versions
$ verylup show
0.10.0
0.11.5
0.12.0 (default)
nightly
```

Academic Workflow Support: Reproducible Configuration

Verylup - Toolchain Version Manager:

```
# Install specific Veryl toolchain version
$ cargo install verylup
$ verylup install 0.12.0
$ verylup default 0.12.0

# Per-project version pinning
$ verylup override set 0.11.5
$ veryl --version
veryl 0.11.5 (pinned for this project)

# List available versions
$ verylup show
0.10.0
0.11.5
0.12.0 (default)
nightly
```

Veryl.toml - Project Config:

```
[project]
name = "research_cpu"
version = "1.2.0"
description = "RISC-V core for ISCA 2025 paper"
[build]
clock_type = "posedge"
reset_type = "async_low"
[dependencies]
std = "1.0"
axi_common = { version = "2.1", git = "..." }
```

Academic Workflow Support: Reproducible Configuration

Verylup - Toolchain Version Manager:

```
# Install specific Veryl toolchain version
$ cargo install verylup
$ verylup install 0.12.0
$ verylup default 0.12.0

# Per-project version pinning
$ verylup override set 0.11.5
$ veryl --version
veryl 0.11.5 (pinned for this project)

# List available versions
$ verylup show
0.10.0
0.11.5
0.12.0 (default)
nightly
```

Veryl.toml - Project Config:

```
[project]
name = "research_cpu"
version = "1.2.0"
description = "RISC-V core for ISCA 2025 paper"
[build]
clock_type = "posedge"
reset_type = "async_low"
[dependencies]
std = "1.0"
axi_common = { version = "2.1", git = "..." }
```

- ✓ Veryl toolchain version pinned
- ✓ Project & dependency versions locked
- ✓ Build configuration documented
- ✓ Reproducible across labs
- ✓ No docker required

Academic Workflow Support: CI/CD Integration

GitHub Actions integration for automated testing and deployment

GitHub Actions Workflow:

```yaml
# .github/workflows/veryl.yml
name: Veryl CI
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: veryl-lang/setup-veryl@v1
      - run: veryl fmt --check
      - run: veryl check
      - run: veryl test --sim verilator
      - run: veryl build
      - run: veryl publish --dry-run
```

Automated Workflow:

- ✓ **Lint checking** on every commit
- ✓ **Test execution** with multiple simulators
- ✓ **Build verification** across platforms
- ✓ **Package validation** before publishing
- ✓ **Multi-version testing** for compatibility

Research Benefits:

- Catch errors before paper submission
- Ensure reproducibility for reviewers
- Automate artifact validation

Advanced Language Features: Quality of Life Improvements

Modports with Converse:

```
interface axi_if {
  var awvalid: logic; var awready: logic;
  var wdata: logic<32>;
  modport Manager {
    awready: input, ..output
  }
  modport Subordinate { ..converse(Manager) }
  modport Monitor {..input}
}
```

Advanced Language Features: Quality of Life Improvements

Modports with Converse:

```
interface axi_if {
  var awvalid: logic; var awready: logic;
  var wdata: logic<32>;
  modport Manager {
    awready: input, ..output
  }
  modport Subordinate { ..converse(Manager) } }
  modport Monitor {..input}
}
```

Generics:

```
package PackageA::<T: u32> {
    const X: u32 = T;
    struct StructA::<Y: u32> {
      x: logic<X>,
      y: logic<Y>,
    }
```

Generics supported for:
function, **module**, **interface**, **package**, **struct**, **union**

Advanced Language Features: Quality of Life Improvements

Modports with Converse:

```
interface axi_if {
  var awvalid: logic; var awready: logic;
  var wdata: logic<32>;
  modport Manager {
    awready: input, ..output
  }
  modport Subordinate { ..converse(Manager) } }
  modport Monitor {..input}
}
```

Generics:

```
package PackageA::<T: u32> {
    const X: u32 = T;
    struct StructA::<Y: u32> {
      x: logic<X>,
      y: logic<Y>,
    }
```

Generics supported for:
function, **module**, **interface**, **package**, **struct**, **union**

Trait-like Prototypes:

```
proto package Arithmetic {
  const T: type;
  fn add(a: T, b: T) -> T;
  fn sub(a: T, b: T) -> T;
}
// Implement for different types
package ArithmeticInt::<N: u32> {
  const T: type = logic<N>;
  fn add(a, b) -> T { return a + b; }
  fn sub(a, b) -> T { return a - b;}
}
// Use the proto package
module Alu::<PKG: Arithmetic>
{ /*...*/ }
```

Prototypes supported for:
module, **interface**, and **package**

Current Veryl Adoption and Usage

Growing ecosystem with real-world deployments in industry and academia

Industry Adoption:

- **PEZY Computing** - HPC/AI accelerator
  - 50k lines of Veryl code
  - Mixed with 6M lines of SystemVerilog
  - Next-generation chip development
- **Open Source Tools**
  - RgGen: CSR generator with Veryl support
  - Marlin: Rust-based testbench framework

Academic Usage:

- **Luleå University of Technology** (Sweden)
  - Digital design course
  - MIPS32 subset implementation
  - Veryl as advanced option
- **Open Source Processors**
  - bluecore: Linux-bootable RISC-V (4k lines)
  - very-holy-core: Holy Core Course in Veryl

The Future of Academic Hardware Research

Join us in building modern infrastructure for open-source architecture research

Where We're Going:

- **Modern Research Velocity**
  - Real-time error detection and feedback
  - Seamless collaboration and reproducibility
  - Integrated testing and CI/CD workflows
- **Open Research Ecosystem**
  - Easy sharing of research components
  - Standardized documentation and diagrams
  - Cross-lab compatibility and reuse
- **Academic-First Design**
  - Built for research constraints and workflows
  - Gradual adoption without disruption
  - Safety and debuggability as priorities

How You Can Help Shape This Future:

The Future of Academic Hardware Research

Join us in building modern infrastructure for open-source architecture research

Where We're Going:

- **Modern Research Velocity**
  - Real-time error detection and feedback
  - Seamless collaboration and reproducibility
  - Integrated testing and CI/CD workflows
- **Open Research Ecosystem**
  - Easy sharing of research components
  - Standardized documentation and diagrams
  - Cross-lab compatibility and reuse
- **Academic-First Design**
  - Built for research constraints and workflows
  - Gradual adoption without disruption
  - Safety and debuggability as priorities

How You Can Help Shape This Future:

- ✓ **Try Veryl in Your Research**
  - Start with one module in an existing project
  - Share your experience and pain points
- ✓ **Contribute to the Ecosystem**
  - Language features, standard library
  - Documentation, tutorials, examples
  - Tool integrations (EDA, simulators)
- ✓ **Build the Community**
  - Use in courses and research projects
  - Publish artifacts and share experiences
  - Connect with other OSCAR researchers

Getting Started With Veryl

Creating a Veyrl project:

```
$ cargo install verylup
$ verylup install latest
$ veryl new my_project
```

Veryl Resources

- Website: veryl-lang.org
- Book: doc.veryl-lang.org/book/
- GitHub: github.com/veryl-lang/