Veryl

A New Hardware Description Language Developed as Open Source Software

Naoya Hatta

PEZY Computing, K.K.

What is Hardware Description Language?

Hardware Description Language (HDL) is for digital circuit design

- Modern CPU/GPU/SoC are written in HDL
- Essential for large-scale circuit design

Industry standard HDLs are mainly used

- Verilog, VHDL, SystemVerilog

New HDLs have been developed

- Chisel, SpinalHDL, Bluespec
- Veryl

Veryl

A new HDL being developed as open source software

- Refined syntax based on SystemVerilog and Rust
- Compile into human-readable SystemVerilog

Veryl

```
/// Counter
module Counter #(
    param WIDTH: u32 = 1,
)(
    i_clk: input  clock       ,
    i_rst: input  reset       ,
    o_cnt: output logic<WIDTH>,
){
    var r_cnt: logic<WIDTH>;

    always_ff {
        if_reset {
            r_cnt = 0;
        } else {
            r_cnt += 1;
        }
    }
}
```

Compile

SystemVerilog

```
// Counter
module Counter #(
    parameter WIDTH = 1
)(
    input  logic             i_clk  ,
    input  logic             i_rst_n,
    output logic [WIDTH-1:0] o_cnt
);
    logic [WIDTH-1:0] r_cnt;

    always_ff @ (posedge i_clk or negedge i_rst_n) begin
        if (!i_rst_n) begin
            r_cnt <= 0;
        end else begin
            r_cnt <= r_cnt + 1;
        end
    end
endmodule
```

Agenda

Motivation

- Challenges in SystemVerilog Development
- Challenges of existing new HDLs

Veryl: A new HDL as an alternative to SystemVerilog

- Concept and vision
- Key features and benefits

Developing Veryl as Open Source Software

- Project status
- Actual usages

Conclusion

Agenda

Motivation

- Challenges in SystemVerilog Development
- Challenges of existing new HDLs

Veryl: A new HDL as an alternative to SystemVerilog

- Concept and vision
- Key features and benefits

Developing Veryl as Open Source Software

- Project status
- Actual usages

Conclusion

Challenges in SystemVerilog Development

Redundant and easy-to-mistake syntax

- SystemVerilog takes over old syntax from Verilog
- Unsynthesizable description can be mixed in easily

Challenges in SystemVerilog Development

Redundant and easy-to-mistake syntax

- SystemVerilog takes over old syntax from Verilog
- Unsynthesizable description can be mixed in easily

Limited compile-time check

- Type check depends on each EDA tools
- Some checks (e.g. CDC) need expensive tools

Challenges in SystemVerilog Development

Redundant and easy-to-mistake syntax

- SystemVerilog takes over old syntax from Verilog
- Unsynthesizable description can be mixed in easily

Limited compile-time check

- Type check depends on each EDA tools
- Some checks (e.g. CDC) need expensive tools

Less productivity tools

- No formatter, real-time diagnostics, dependency management

Challenges in SystemVerilog Development

Redundant and easy-to-mistake syntax

- SystemVerilog takes over old syntax from Verilog
- Unsynthesizable description can be mixed in easily

Limited compile-time check

- Type check depends on each EDA tools
- Some checks (e.g. CDC) need expensive tools

Less productivity tools

- No formatter, real-time diagnostics, dependency management

How about the existing new HDLs?

Challenges of existing new HDLs

Syntax is not optimal

- Take over the base language syntax (e.g. Chisel is based on Scala)
- HDL-specific syntax can’t be introduced

Challenges of existing new HDLs

Syntax is not optimal

- Take over the base language syntax (e.g. Chisel is based on Scala)
- HDL-specific syntax can’t be introduced

Semantics differences from Verilog

- Small code generates voluminous Verilog
- Low readability of generated Verilog

Challenges of existing new HDLs

Syntax is not optimal

- Take over the base language syntax (e.g. Chisel is based on Scala)
- HDL-specific syntax can’t be introduced

Semantics differences from Verilog

- Small code generates voluminous Verilog
- Low readability of generated Verilog

Generate Verilog, not SystemVerilog

- Integrating into existing SystemVerilog projects is difficult

Challenges of existing new HDLs

Syntax is not optimal

- Take over the base language syntax (e.g. Chisel is based on Scala)
- HDL-specific syntax can't be introduced

Semantics differences from Verilog

- Small code generates voluminous Verilog
- Low readability of generated Verilog

Generate Verilog, not SystemVerilog

- Integrating into existing SystemVerilog projects is difficult

A new HDL improving the above issues is necessary: Veryl

Agenda

Motivation

- Challenges in SystemVerilog Development
- Challenges of existing new HDLs

Veryl: A new HDL as an alternative to SystemVerilog

- Concept and vision
- Key features and benefits

Developing Veryl as Open Source Software

- Project status
- Actual usages

Conclusion

Introduction to Veryl Concept

Optimized syntax for synthesizable HDL

- Designed to enhance readability and reliability

Generate human-readable SystemVerilog

- Ensures generated code is easy to understand and debug

Productivity tools by default

- Incorporates tools that enhance developer productivity automatically

Introduction to Veryl Concept

Optimized syntax for synthesizable HDL

- Designed to enhance readability and reliability

Generate human-readable SystemVerilog

- Ensures generated code is easy to understand and debug

Productivity tools by default

- Incorporates tools that enhance developer productivity automatically

Optimized Syntax for Synthesizable HDL

1. Basic syntax
   - Streamlined for easier understanding and use
2. Clock and reset
   - Simplified handling of clock and reset signals
   - Quickly detection of errors around clock and reset
3. Generics
   - Enhanced support for generics to increase flexibility and reusability

Basic Syntax

Comparison between SystemVerilog and Veryl

Introduce language features of modern programming language

SystemVerilog

```
// Counter
module Counter #(
    parameter WIDTH = 1
)(
    input  logic             i_clk  ,
    input  logic             i_rst_n,
    output logic [WIDTH-1:0] o_cnt
);
    logic [WIDTH-1:0] r_cnt;

    always_ff @ (posedge i_clk or negedge i_rst_n) begin
        if (!i_rst_n) begin
            r_cnt <= 0;
        end else begin
            r_cnt <= r_cnt + 1;
        end
    end

    always_comb begin
        o_cnt = r_cnt;
    end
endmodule
```

Veryl

```
/// Counter
module Counter #(
    param WIDTH: u32 = 1,
)(
    i_clk: input  clock      ,
    i_rst: input  reset      ,
    o_cnt: output logic<WIDTH>,
){
    var r_cnt: logic<WIDTH>;

    always_ff {
        if_reset {
            r_cnt = 0;
        } else {
            r_cnt += 1;
        }
    }

    always_comb {
        o_cnt = r_cnt;
    }
}
```

Documentation comments

Trailing comma

Simplify idiomatic syntax in SystemVerilog

Bit width notation

Context-aware assignment

Clock and Reset

Dedicated clock and reset type

- Sensitivity list can be omitted in single-clock modules

Veryl

```
module Counter #(
    param WIDTH: u32 = 1,
)(
    i_clk: input  clock  ,
    i_rst: input  reset  ,
    o_cnt: output logic<WIDTH>,
){
    always_ff {
        if_reset {
            o_cnt = 0;
        } else {
            o_cnt += 1;
        }
    }
}
```

Clock and reset type

Automatically inferred clock and reset

Dedicated reset condition syntax

Clock and Reset

Polarity and synchronicity configurable during compiling

- Sensitivity list and reset condition are automatically adjusted
- Single Veryl code can be compiled for both ASIC and FPGA

Veryl

```
module ModuleA (
    i_clk: input clock,
    i_rst: input reset,
) {
    always_ff {
        if_reset {
        }
    }
}
```

for ASIC

SystemVerilog

```
always_ff @ (posedge i_clk or negedge i_rst) begin
    if (!i_rst) begin
    end
end
```

for FPGA

SystemVerilog

```
always_ff @ (posedge i_clk) begin
    if (i_rst) begin
    end
end
```

Clock and Reset

Clock domain annotation

- Annotation is mandatory in multi-clock modules
- Unexplicit clock domain crossings are detected as error

annotation

Veryl

```
module ModuleA (
    i_clk_a: input  'a clock,
    i_dat_a: input  'a logic,
    o_dat_a: output 'a logic,
    i_clk_b: input  'b clock,
    o_dat_b: output 'b logic,
) {
    always_ff (i_clk_a) {
        o_dat_a = i_dat_a;
    }

    unsafe (cdc) {
        assign o_dat_b = i_dat_a;
    }
}
```

Signals belonging clock domain 'a

Signals belonging clock domain 'b

Assignment in the same clock domain is OK

Clock domain crossing requires unsafe block

Generics

Type parameter to reduce code duplication

SystemVerilog

```
module SramQueueTypeA;
    SramTypeA u_sram();

    // queue logic
}

module SramQueueTypeB;
    SramTypeB u_sram();

    // queue logic
}

module Test {
    SramQueueTypeA u0_queue();
    SramQueueTypeB u1_queue();
}
```

Duplicated code

Veryl

```
module SramQueue::<T> {
    inst u_sram: T;

    // queue logic
}

module Test {
    // Instantiate a SramQueue by SramTypeA
    inst u0_queue: SramQueue::<SramTypeA>();

    // Instantiate a SramQueue by SramTypeB
    inst u1_queue: SramQueue::<SramTypeB>();
}
```

Parameterized module instance

Actual module can be specified here

Introduction to Veryl Concept

Optimized syntax for synthesizable HDL

- Designed to enhance readability and reliability

Generate human-readable SystemVerilog

- Ensures generated code is easy to understand and debug

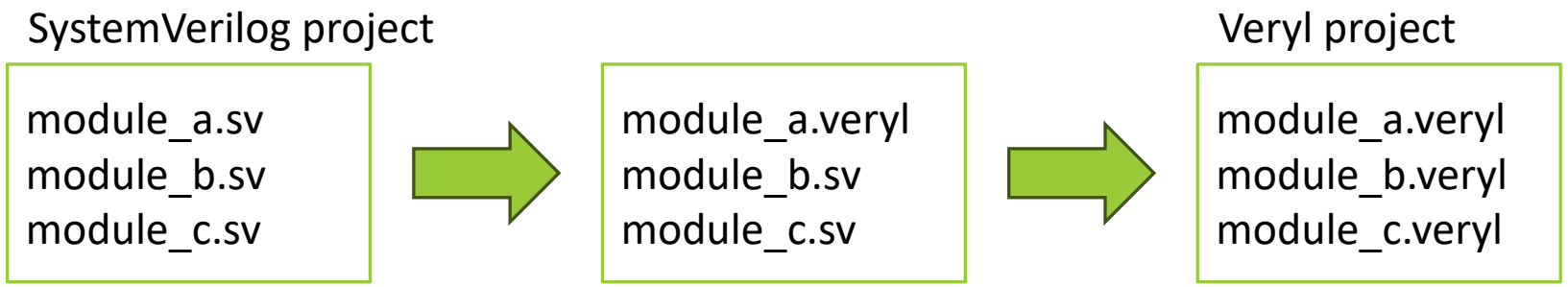
Productivity tools by default

- Incorporates tools that enhance developer productivity automatically

Generate Human-readable SystemVerilog

Interoperability with SystemVerilog

- Reuse the existing SystemVerilog codebase
- Introduce Veryl to the existing SystemVerilog project gradually



Generate Human-readable SystemVerilog

Debug with SystemVerilog features

- struct and interface can be used in waveform viewers

Generated SystemVerilog can be finely tuned

- Timing improvement and pre/post-mask ECO flow can be applied

Introduction to Veryl Concept

Optimized syntax for synthesizable HDL

- Designed to enhance readability and reliability

Generate human-readable SystemVerilog

- Ensures generated code is easy to understand and debug

Productivity tools by default

- Incorporates tools that enhance developer productivity automatically

Productivity Tools by Default

Real-time diagnostics

- Editor integration using standardized language server protocol

Visual Studio Code

```
41  module Test {
42      let a: logic = b + 1;
delay.veryl 1 of 2 problems
Semantic Error: b is undefined veryl-ls(undefined_identifier)
43  }
```

Vim

```
module Test {
W     let a: logic = b + 1;
}                    Diagnostics:
                     1. Semantic Error: b is undefined [undefined_identifier]
                     2. Semantic Warning: a is unused [unused_variable]
```

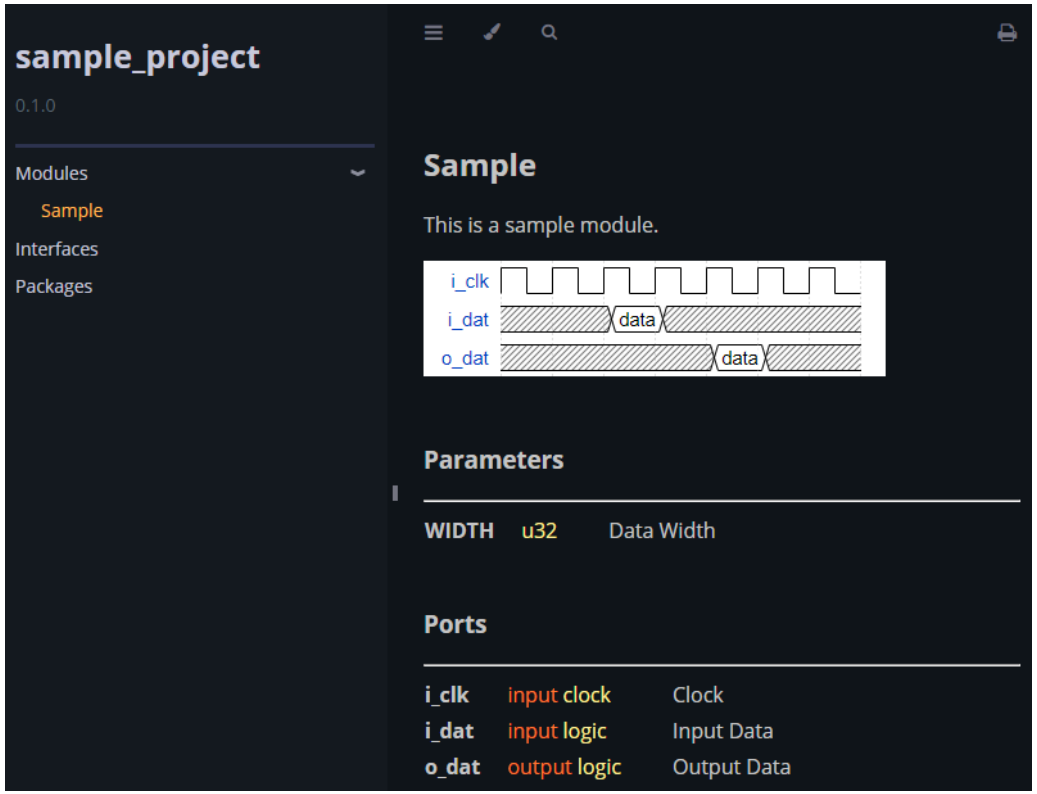
Productivity Tools by Default

Automatic document generation

- Supports Markdown format and waveform description

Veryl

```
/// This is a sample module.
///
/// ```wavedrom
/// {signal: [
///   {name: 'i_clk', wave: 'p.....'},
///   {name: 'i_dat', wave: 'x.=x...', data: ['data']},
///   {name: 'o_dat', wave: 'x...=x.', data: ['data']},
/// ]}
/// ```
pub module Sample #(
    /// Data Width
    param WIDTH: u32 = 1,
)(
    i_clk: input  clock       , /// Clock
    i_dat: input  logic<WIDTH>, /// Input Data
    o_dat: output logic<WIDTH>, /// Output Data
){}
```



Productivity Tools by Default

Automatic formatter

- Enables cleaner, standardized code layout

Veryl

```
module  Counter #(
param WIDTH  : u32 = 1,
) (
i_clk: input clock,
i_rst: input reset,
o_cnt: output   logic  < WIDTH >,
){
  always_ff{
      if_reset   {
          o_cnt = 0;
    }  else  {
    o_cnt += 1;
  }
}
}
```

Veryl

```
module Counter #(
    param WIDTH: u32 = 1,
)(
    i_clk: input  clock       ,
    i_rst: input  reset       ,
    o_cnt: output logic<WIDTH>,
){
    always_ff {
        if_reset {
            o_cnt = 0;
        } else {
            o_cnt += 1;
        }
    }
}
```

Productivity Tools by Default

Other features

- Integrated unit testing to streamline testing process
- Ability to publish projects as libraries for easy reuse
- Dependency management for efficient handling of project dependencies
- Toolchain manager to ease to update Veryl compiler

Agenda

Motivation

- Challenges in SystemVerilog Development
- Challenges of existing new HDLs

Veryl: A new HDL as an alternative to SystemVerilog

- Concept and vision
- Key features and benefits

Developing Veryl as Open Source Software

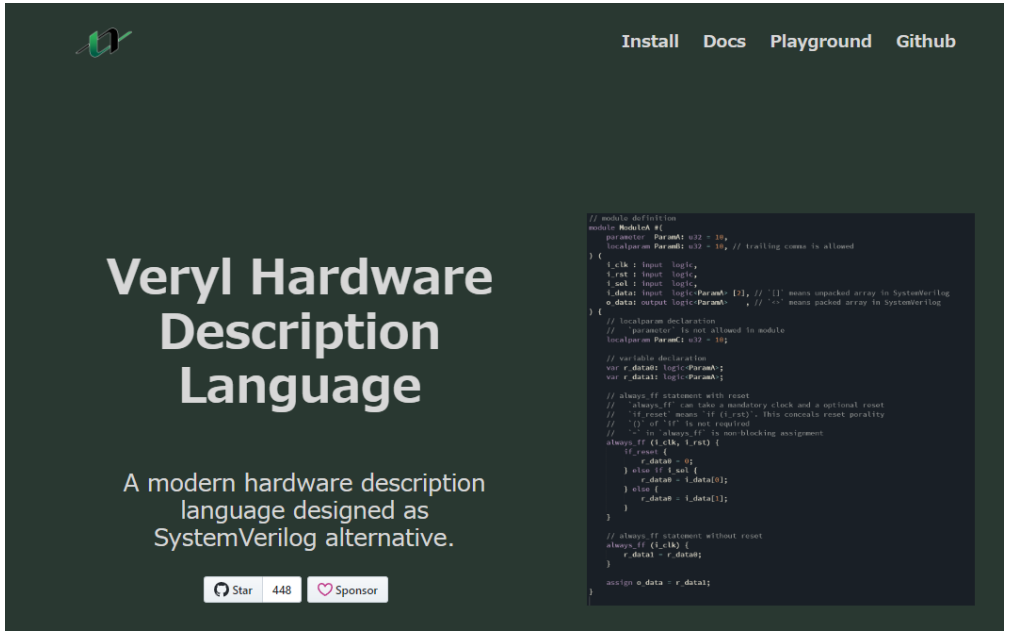
- Project status
- Actual usages

Conclusion

Project Status

GitHub

- Created : 2022/12
- Commits : 3032
- Releases : 51
- Pull Requests : 3 Open, 1114 Closed
- Contributors : 12

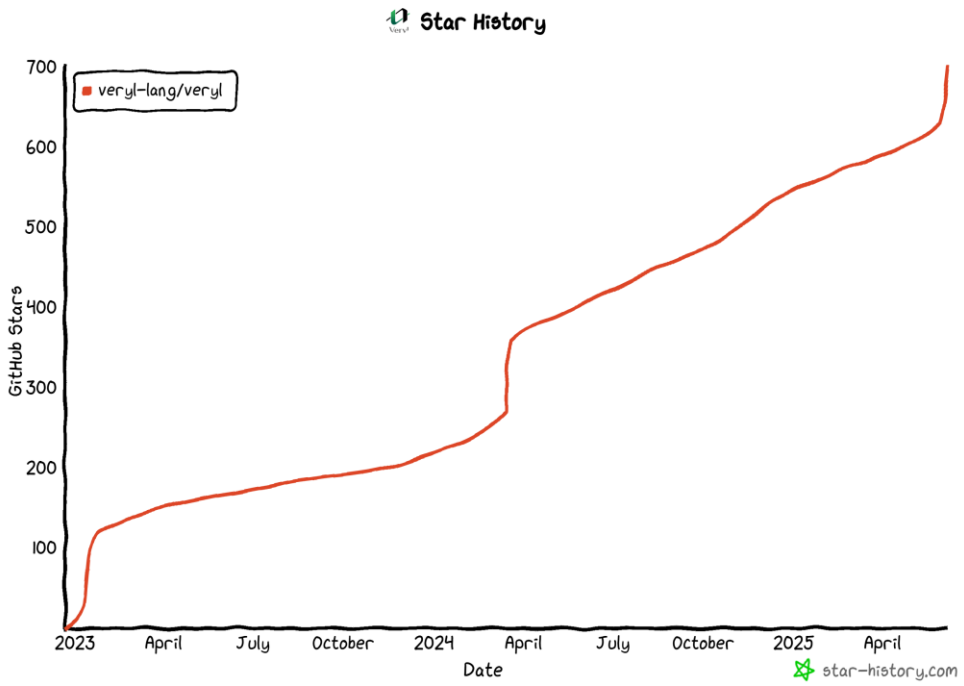


Resources

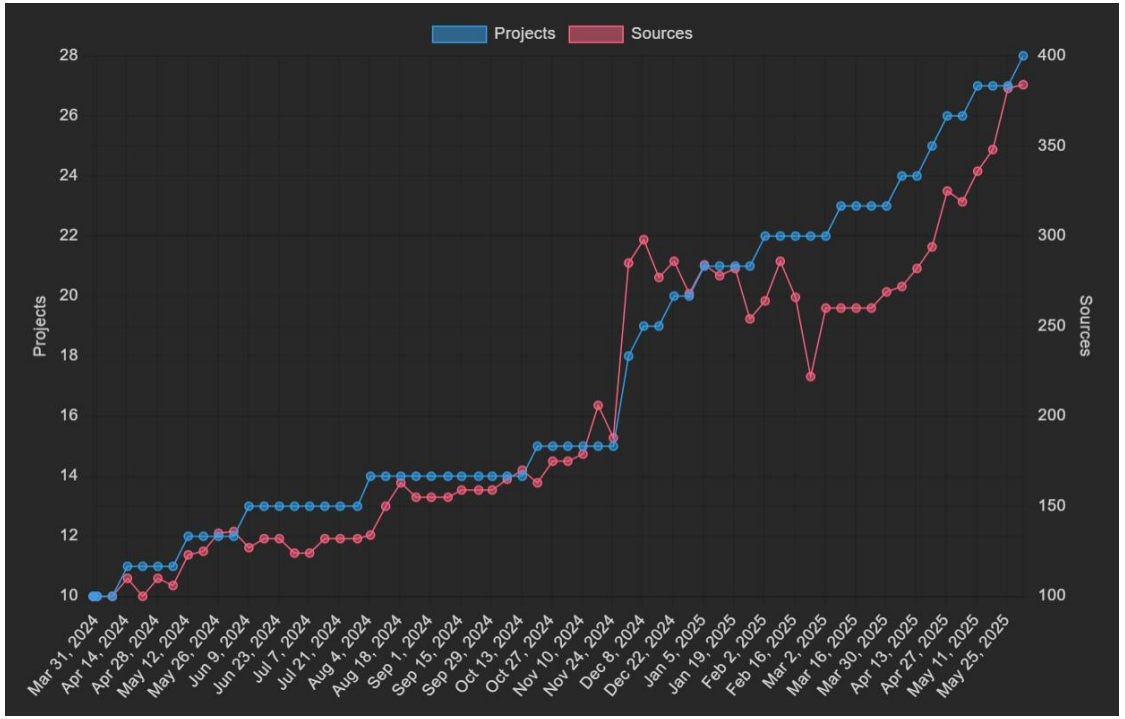
- Official site : https://veryl-lang.org
- Language reference : https://doc.veryl-lang.org/book/
- Playground : https://doc.veryl-lang.org/playground/

Project Status

GitHub Stars (2022/12~)



Projects on GitHub (2024/03~)



Actual Usages

HPC/AI accelerator

- the next generation chip developed in PEZY Computing
  - Closed source because it includes NDA-based information
- 50k lines in Veryl
  - with 6M lines in SystemVerilog

bluecore

- Open source, Linux bootable RISC-V
  - https://github.com/nananapo/bluecore
- 4k lines in Veryl
- Self-publishing books "Verylで作るCPU (Writing CPU in Veryl)"

Actual Usages

Digital design course at Luleå University of Technology Sweden

- MIPS32 subset implementation
- As an advanced task, Veryl can be selected

OSS processor implementations

- https://github.com/jbeaurivage/very-holy-core
- https://github.com/perlindgren/vips
- https://github.com/shinrabansyo/cpu

OSS tools supporting Veryl

- RgGen: CSR generator
- Marlin: Writing testbench in Rust

Agenda

Motivation

- Challenges in SystemVerilog Development
- Challenges of existing new HDLs

Veryl: A new HDL as an alternative to SystemVerilog

- Concept and vision
- Key features and benefits

Developing Veryl as Open Source Software

- Project status
- Actual usages

Conclusion

Conclusion

Veryl: A new HDL as an alternative to SystemVerilog

- Optimized syntax for synthesizable HDL
- Generate human-readable SystemVerilog
- Productivity tools by default

Developed as open source software

- Available on GitHub: https://github.com/veryl-lang/veryl
- Growing open source ecosystem

Future of Open Source Chip Design

Young developers are …

Familiar with modern programming languages

- They are interested in new HDLs

Familiar with open source culture

- Publishing code, contributing other projects

As they enter the industry, adoption of new HDLs and open source technologies will grow