2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION JAPAN

Veryl: A New HDL as an Alternative to SystemVerilog

Naoya Hatta
PEZY Computing K.K.

accellera SYSTEMS INITIATIVE

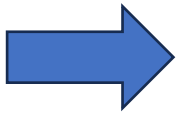
Veryl

- A new HDL being developed as open-source software
  - Refined syntax based on SystemVerilog
  - 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
  - Enhancing SystemVerilog development
- Existing approach: Alternative HDLs
  - Overview and challenges
- Veryl: A new HDL as an alternative to SystemVerilog
  - Concept and vision
  - Key features and benefits
- Conclusion
  - Development status

Agenda

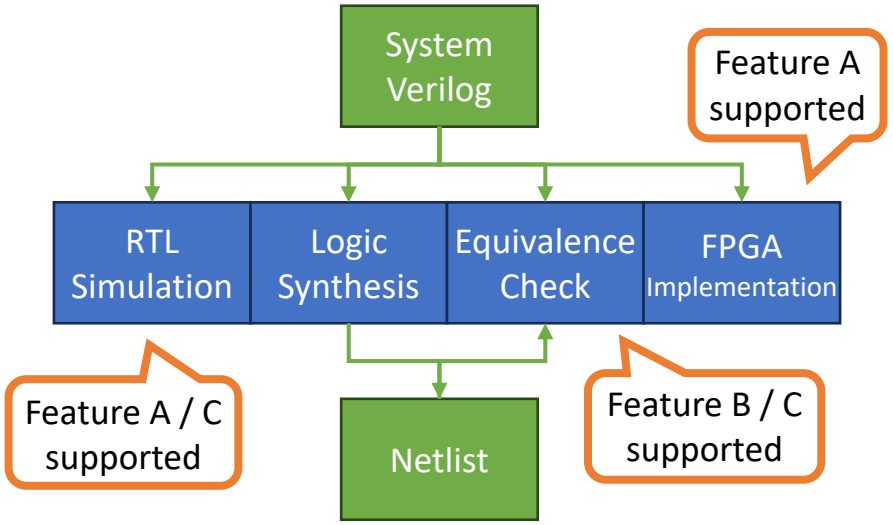
- Motivation
  - Enhancing SystemVerilog development
- Existing approach: Alternative HDLs
  - Overview and challenges
- Veryl: A new HDL as an alternative to SystemVerilog
  - Concept and vision
  - Key features and benefits
- Conclusion
  - Development status

Enhancing SystemVerilog Development

- Possible ways
  1. Introduce useful SystemVerilog features
  2. Introduce new tools

Introduce Useful SystemVerilog Features

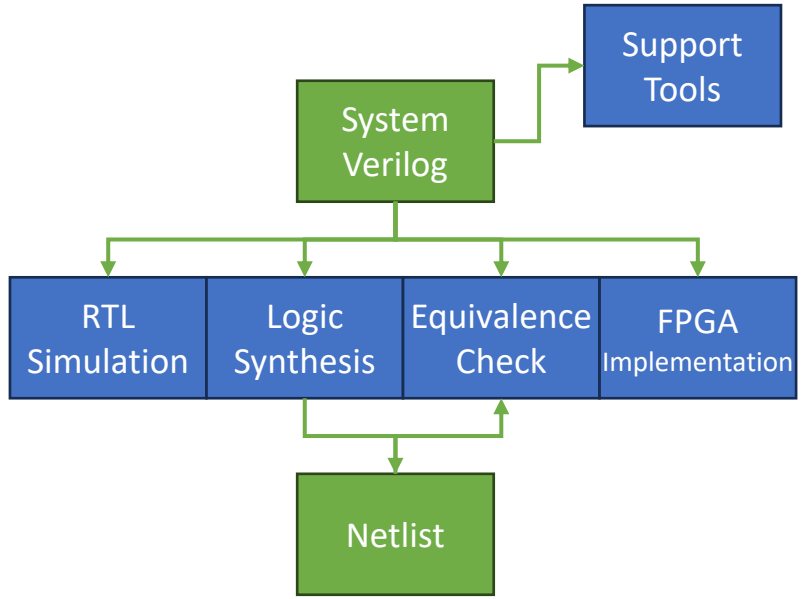
- No tool supports all features completely
  - Each tool has a different support area for features
- Synthesizability is not guaranteed



- Introducing inexperienced features is challenging
  - Need to explore usable feature combinations

Introduce New Tools

- A variety of support tools for programming languages
  - e.g., linters, formatters, and specialized editors
- In contrast, it is difficult to develop such support tools for SystemVerilog
  - Complex syntax and semantics make it very challenging



- Few options for support tools in SystemVerilog
  - Very limited choice of both commercial and open-source software

Enhancing SystemVerilog Development

- Possible ways
  1. Introduce useful SystemVerilog features
  2. Introduce new tools

- Both approaches are difficult
- How about the existing alternative HDLs?
  - For example, Chisel

Agenda

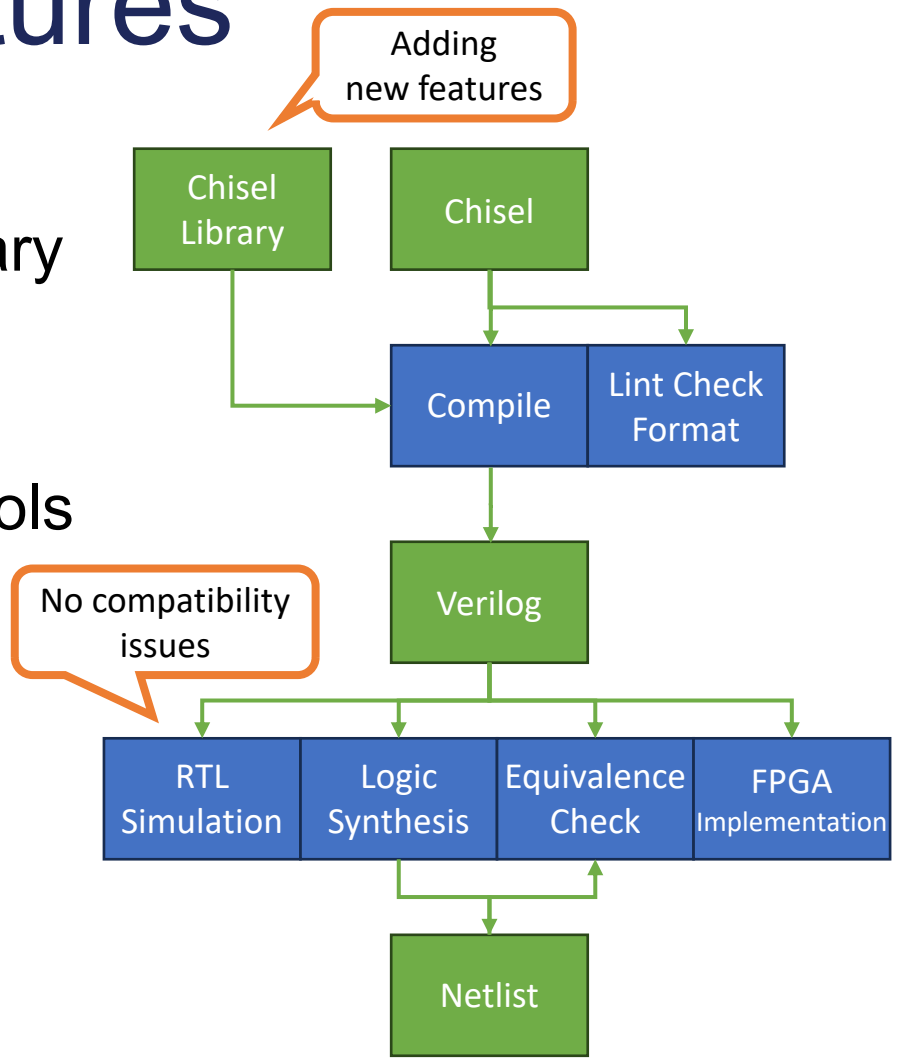
- Motivation
  - Enhancing SystemVerilog development
- Existing approach: Alternative HDLs
  - Overview and challenges
- Veryl: A new HDL as an alternative to SystemVerilog
  - Concept and vision
  - Key features and benefits
- Conclusion
  - Development status

Overview of Existing Alternative HDLs

- Advantages over SystemVerilog
  1. Extensible language features
  2. Reuse the existing ecosystem

Extensible Language Features

- Alternative HDLs as software libraries
  - For example, Chisel is developed as a library in Scala
- Generate standard Verilog
  - Ensuring compatibility with existing EDA tools

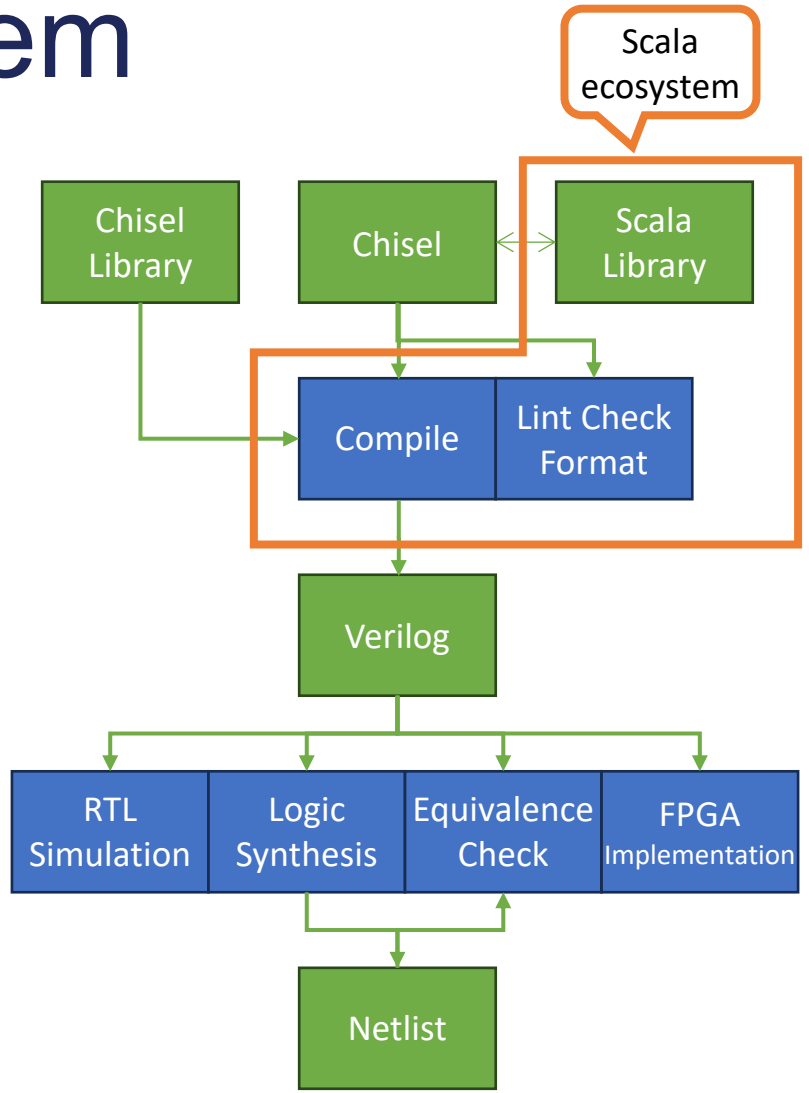


- Flexible features integration
  - New features can be added freely without compatibility issues

Reuse the Existing Ecosystem

- Based on programming language
  - Compiler and tooling ecosystem can be used as is
  - Sophisticated language features compared to traditional HDLs
  - Software libraries can be easily integrated

- Quick development can be achieved
  - Both compiler and logic design



Overview of Existing Alternative HDLs

- Advantages over SystemVerilog
  1. Extensible language features
  2. Reuse the existing ecosystem

- There are some good points
- Can alternative HDLs be used instead of SystemVerilog?

Challenges in Using Alternative HDLs

1. Syntax is not optimal
2. Semantics differences from Verilog
3. Interoperability with SystemVerilog

Syntax is not optimal

- Take over the base language syntax
  - Extensibility is limited
  - HDL-specific syntax can't be introduced

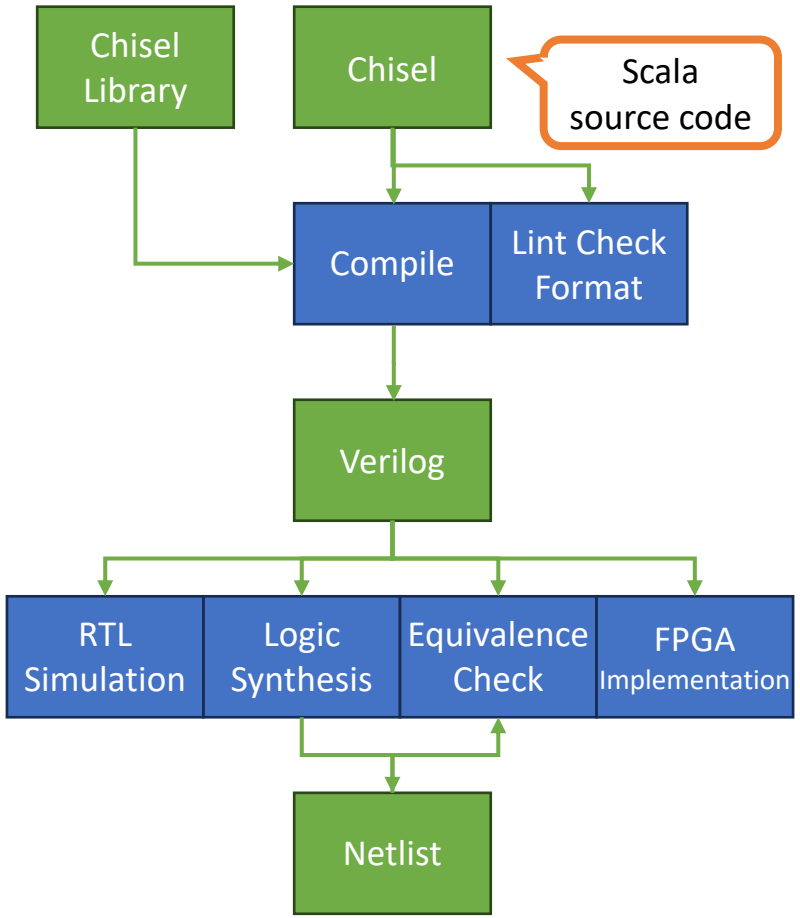
Chisel

```
import chisel3._

class Mux2IO extends Bundle {
  val sel = Input(UInt(1.W))
  val in0 = Input(UInt(1.W))
  val in1 = Input(UInt(1.W))
  val out = Output(UInt(1.W))
}

class Mux2 extends Module {
  val io = IO(new Mux2IO)
  io.out := (io.sel & io.in1) | (~io.sel & io.in0)
}
```

Bit width notation

Nested function calls are required

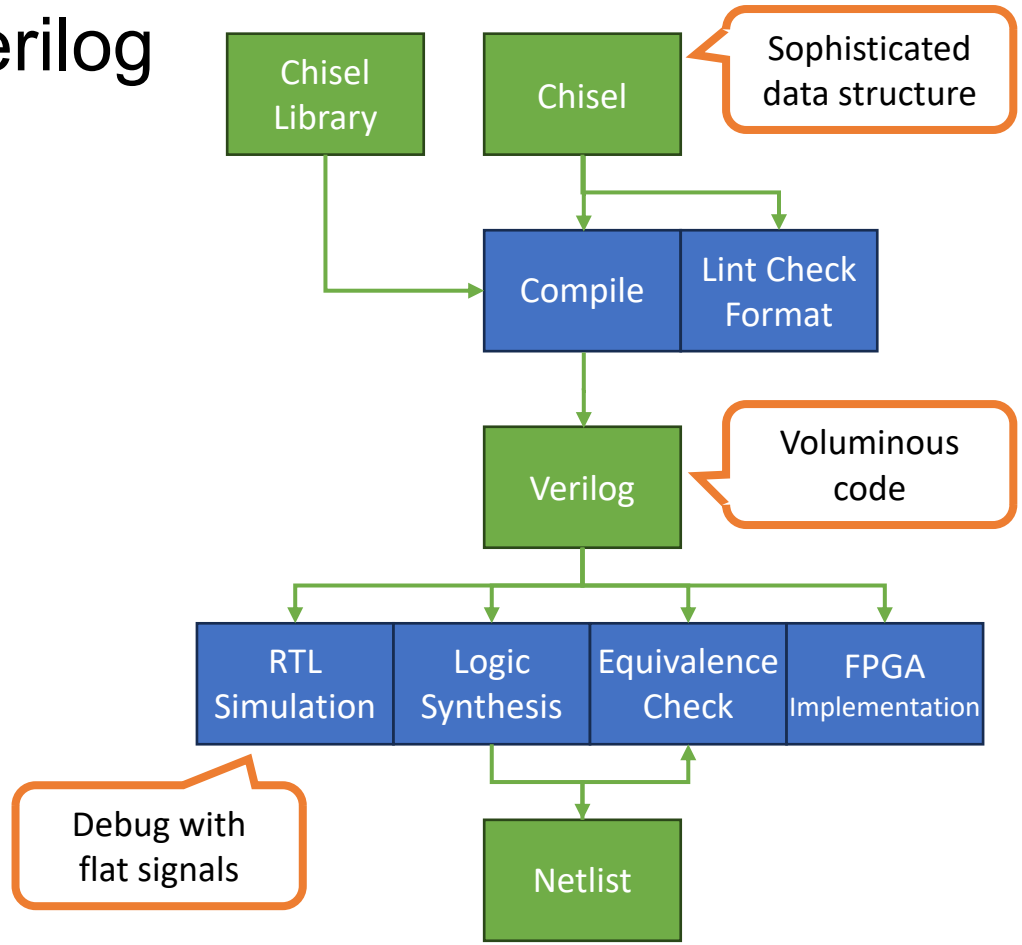
IO is a general variable



Semantics differences from Verilog

- Small code generates voluminous Verilog
  - Low readability of generated Verilog
  - Debug with generated flat signals

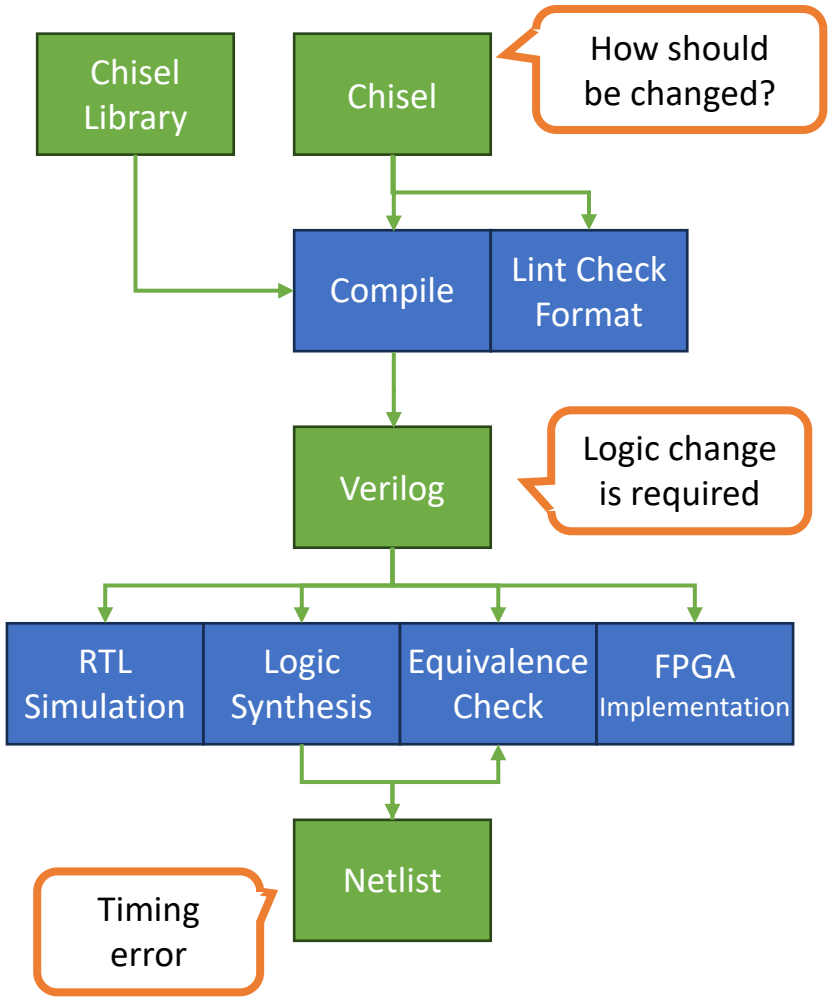
- Debug in Verilog side is difficult



Semantics differences from Verilog

- Small changes in code cause large-scale changes in Verilog
  - Predicting induced changes is difficult

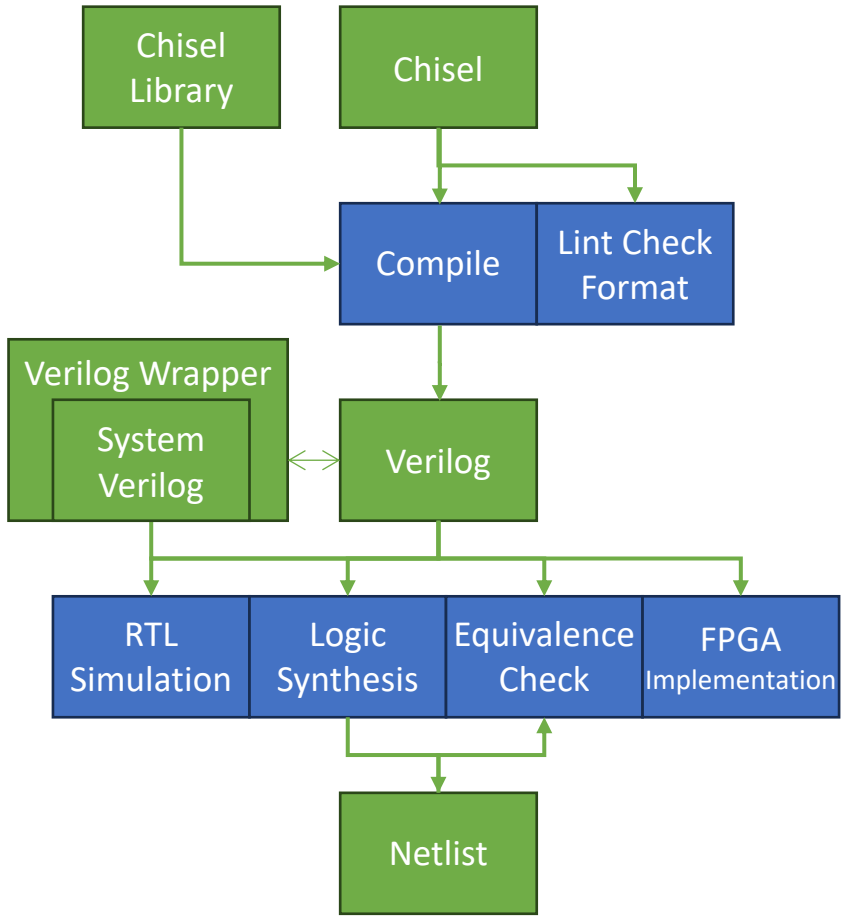
- Changing generated Verilog in detail is difficult
  - Timing improvement
  - pre/post-mask ECO flow



Interoperability with SystemVerilog

- SystemVerilog can’t be used directly
  - A Verilog wrapper is required

- Integrating into existing SystemVerilog projects is difficult



Challenges in Using Alternative HDLs

1. Syntax is not optimal
2. Semantics differences from Verilog
3. Interoperability with SystemVerilog

- The existing alternative HDLs are difficult to use as alternatives to SystemVerilog
- A new HDL is required: Veryl

Agenda

- Motivation
  - Enhancing SystemVerilog development
- Existing approach: Alternative HDLs
  - Overview and challenges
- Veryl: A new HDL as an alternative to SystemVerilog
  - Concept and vision
  - Key features and benefits
- Conclusion
  - Development status

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;
        }
    }
}
```

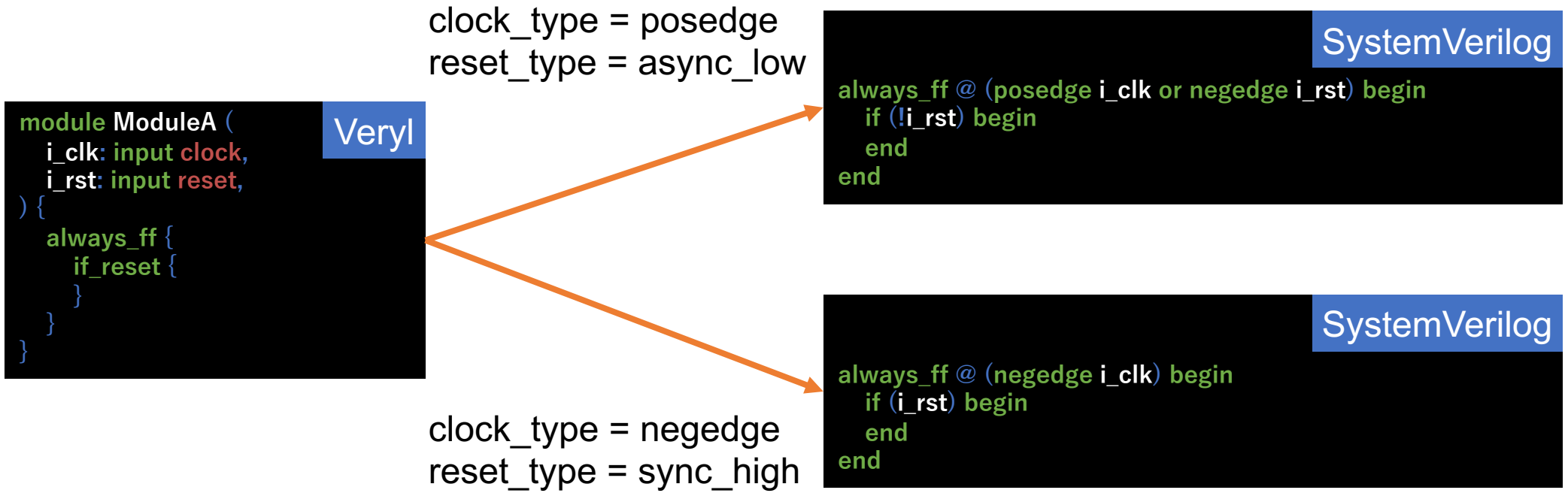
Clock and reset type

Automatically inferred clock and reset

Dedicated reset condition syntax

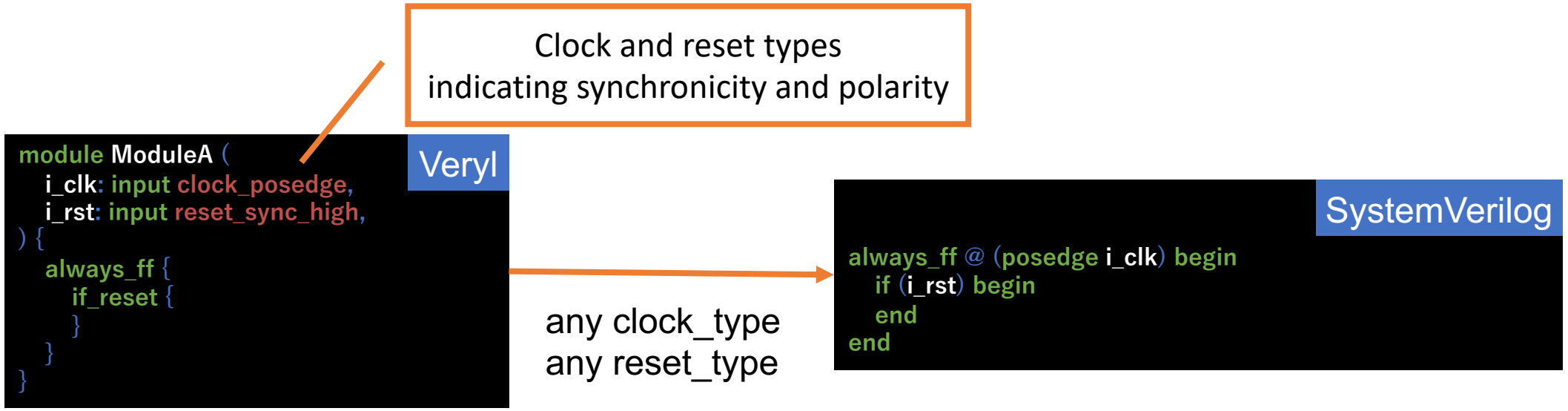
Clock and Reset

- Synchronicity and polarity configurable during compiling
  - Sensitivity list and reset condition are automatically adjusted
  - Single Veryl code can be compiled for both ASIC and FPGA



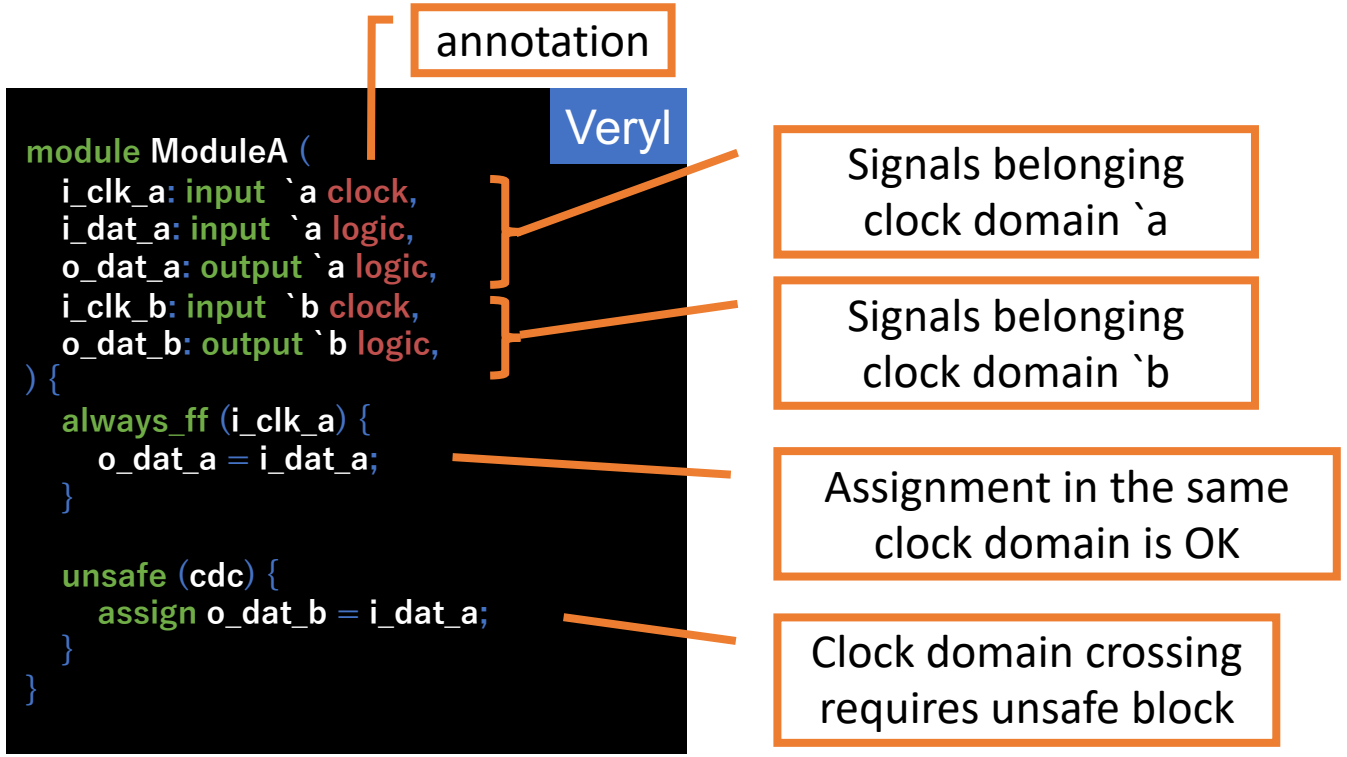
Clock and Reset

- Synchronicity and polarity configurable during compiling
  - Special types indicating synchronicity and polarity are supported too



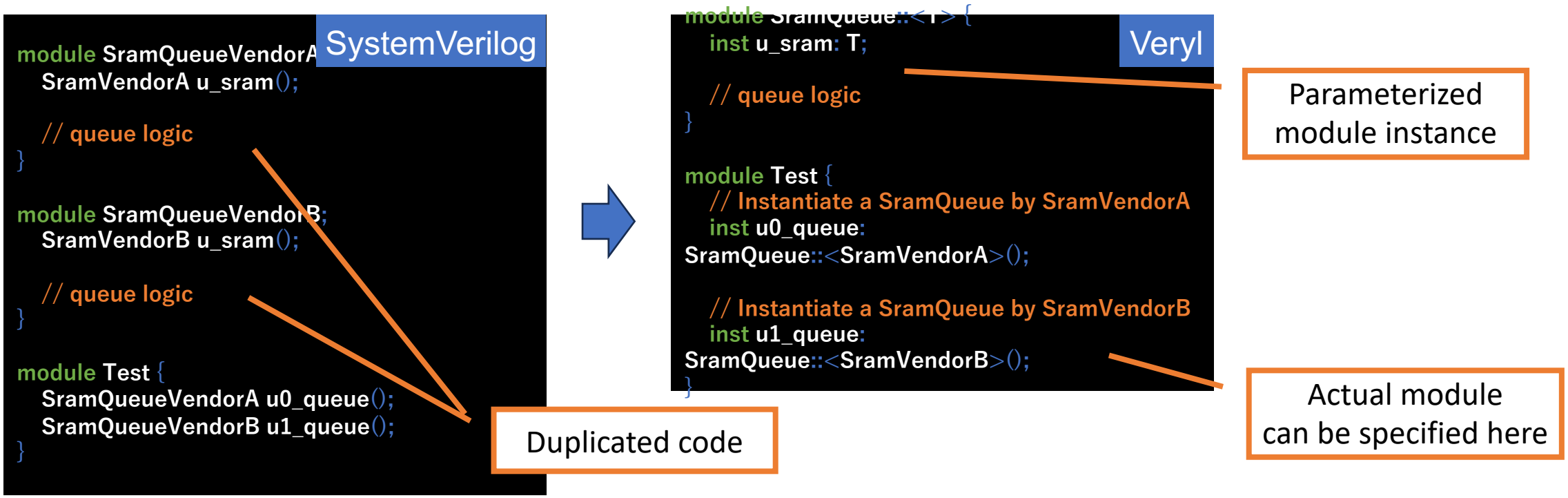
Clock and Reset

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



Generics

- Type parameter to reduce code duplication

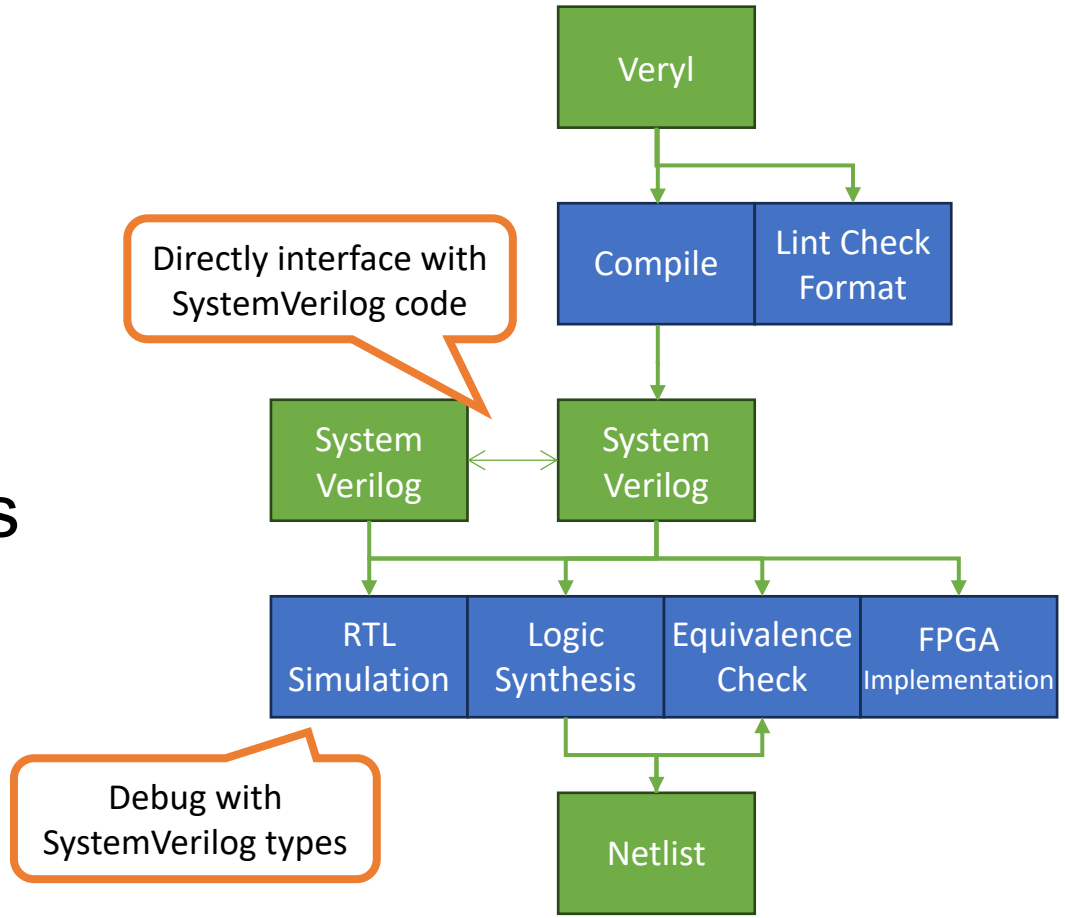


Introduction to Veryl Concept

- 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

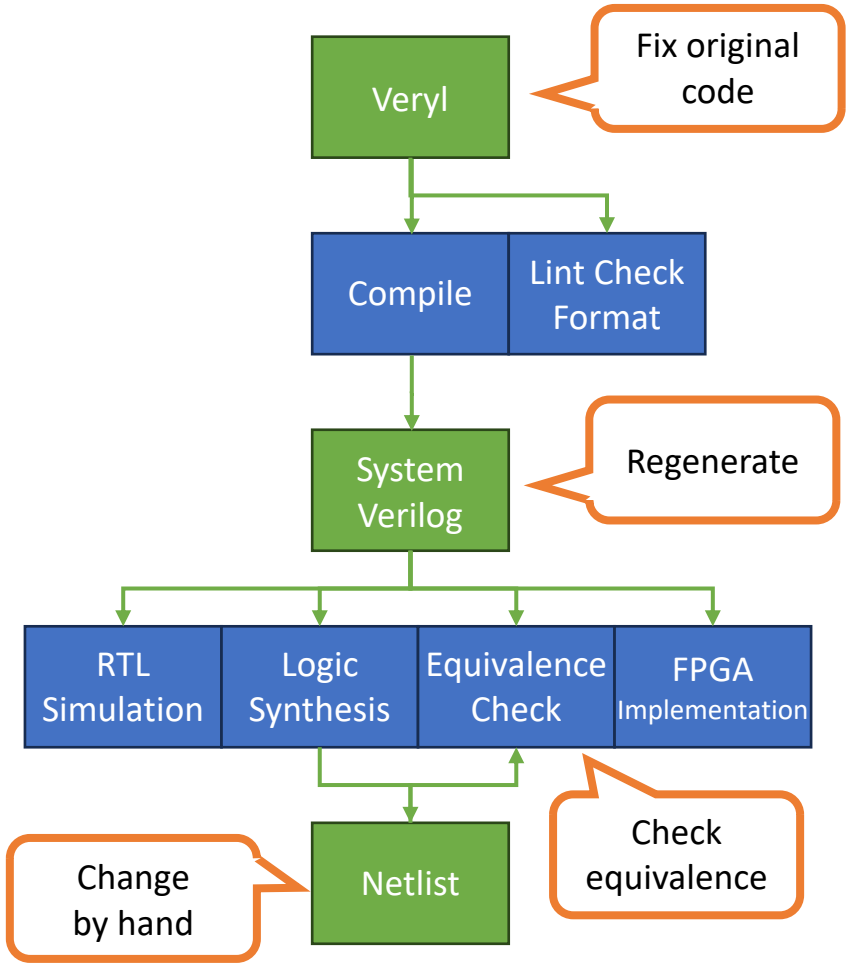
- Interoperability with SystemVerilog
  - Reuse the existing SystemVerilog codebase
  - Introduce Veryl to the existing SystemVerilog project gradually
- Debug with SystemVerilog features
  - struct and interface can be used in waveform viewers



Generate Human-readable SystemVerilog

- Consistent semantics with SystemVerilog
  - Predictable changes in generated SystemVerilog when modifying Veryl code

- Generated SystemVerilog can be finely tuned
  - Timing improvement and pre/post-mask ECO flow can be applied



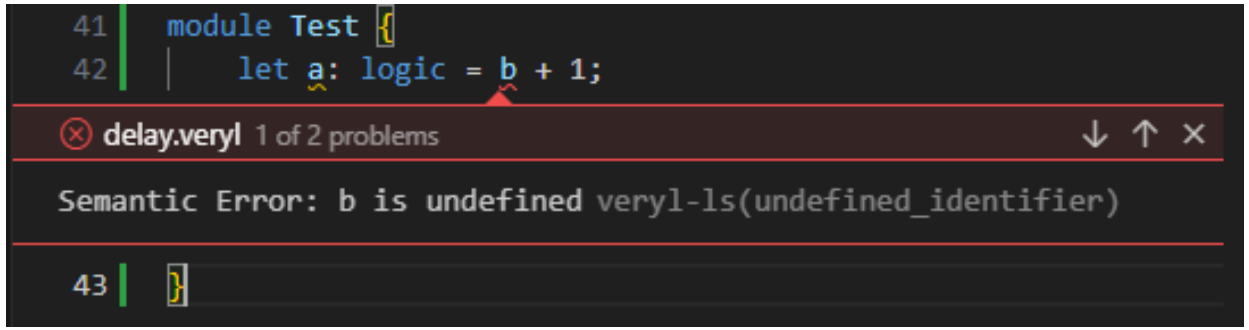
Introduction to Veryl Concept

- 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

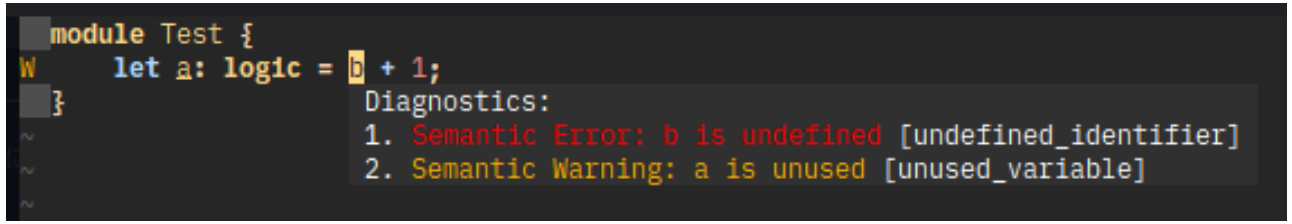
Productivity Tools by Default

- Real-time diagnostics
  - Editor integration using standardized language server protocol

Visual Studio Code



Vim

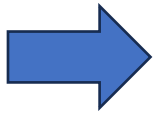


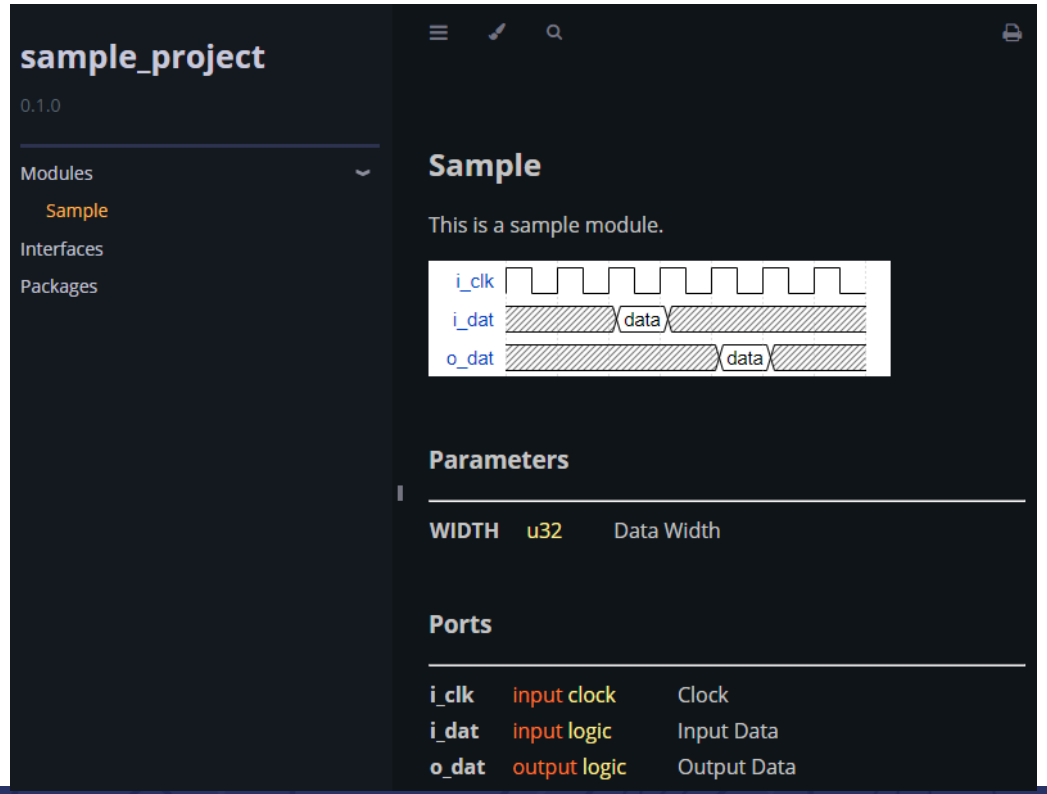
Productivity Tools by Default

- 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

- Other features
  - Auto formatting for cleaner, standardized code layout
  - Integrated unit testing to streamline testing process
  - Ability to publish projects as libraries for easy reuse
  - Dependency management for efficient handling of project dependencies

Agenda

- Motivation
  - Enhancing SystemVerilog development
- Existing approach: Alternative HDLs
  - Overview and challenges
- Veryl: A new HDL as an alternative to SystemVerilog
  - Concept and vision
  - Key features and benefits
- Conclusion
  - Development status

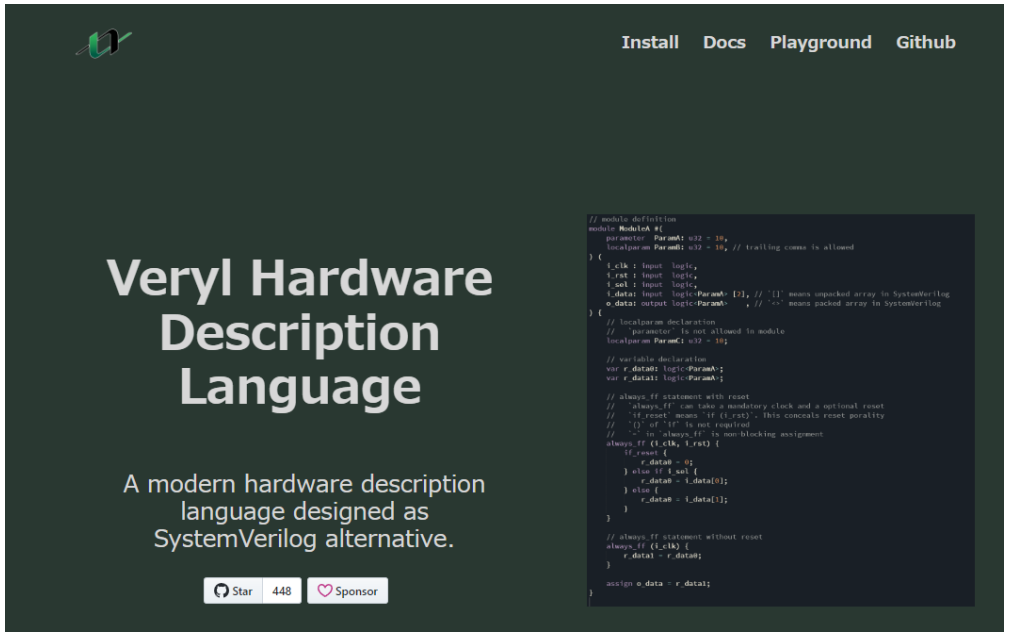
Conclusion

- Veryl: A new HDL as an alternative to SystemVerilog
  - Optimized syntax for synthesizable HDL
  - Generate human-readable SystemVerilog
  - Productivity tools by default

- Open-source development
  - Developed as open-source software
  - Available on GitHub: https://github.com/veryl-lang/veryl

Development Status

- Project Status
  - GitHub Stars : 448
  - Commits : 1778
  - Issues : 54 Open, 211 Closed
  - Pull Requests : 0 Open, 563 Closed
  - Contributors : 9
- Resources
  - Official site : https://veryl-lang.org
  - Language reference : https://doc.veryl-lang.org/book/
  - Playground : https://doc.veryl-lang.org/playground/



Questions