Veryl

SystemVerilogの代替となる
新しいハードウェア記述言語の設計と実装

初田 直也

PEZY Computing

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

アジェンダ

01 背景

02 Verylの概要

03 言語機能

04 ツールチェインと評価

05 エコシステムとまとめ

背景

なぜ新しいHDLが必要なのか

01

背景

SystemVerilog開発における課題

- **冗長でミスを招きやすい構文**
  Verilog由来の古い構文を引き継ぎ、合成不可能な記述も混入しやすい
- **限定的なコンパイル時チェック**
  型チェックはEDAツール依存で、CDCなどの検証には高価なツールが必要
- **開発支援ツールの不足**
  フォーマッタ・リアルタイム診断・依存管理などが標準で備わっていない

背景

既存の新しいHDL（Alt-HDL）の課題

- **構文が最適でない**

  プログラミング言語に埋め込むDSL型のため、HDL固有の構文を導入しにくい（例: ChiselはScala）

- **生成Verilogの可読性が低い**

  少量の記述から膨大なVerilogが生成され、デバッグや波形観測が難しい

- **既存フローと共存しにくい**

  pre/post-mask ECOなど、Verilogを直接編集する設計フローに組み込みづらい

Verylの概要

SystemVerilogの代替を目指す新しいHDL

02

CONCEPT

Verylのコンセプト

01

HDLに特化した構文

合成可能性を保証し、可読性と信頼性を高める。SystemVerilog経験者にも馴染む基本構文

02

可読なSystemVerilog生成

既存のSystemVerilog資産と相互に接続できる。従来の設計フローと共存し、波形観測やデバッグも容易

03

単一ツールチェイン

フォーマッタ・リンタ・リアルタイム診断からシミュレーション・論理合成までを標準で提供

CONCEPT

Verylのビルドフロー

CONCEPT

SystemVerilogとほぼ1対1に対応

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
  assign  o_cnt =  r_cnt;
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;
    }
  }
  assign  o_cnt =  r_cnt;
}
```

ドキュメントコメント

末尾カンマを許容

複合代入が使える

言語機能

設計を安全かつ簡潔にする主要機能

03

言語機能

クロック・リセットの抽象化

```
module  ModuleA (
  i_clk: input clock,
  i_rst: input reset,
) {
  always_ff {
    if_reset {
      // リセット時の処理
    } else {
      // 通常時の処理
    }
  }
}
```

- **感度リストの省略**
  単一クロックのモジュールでは記述を省略できる
- **`if_reset` が極性を隠蔽**
  同期／非同期・正／負をコードから分離
- **単一ソースでASIC・FPGA両対応**
  極性・同期方式はビルド時の構成で切り替え

言語機能

クロックドメイン注釈とCDC検出

```
module  ModuleA (
  // クロックドメイン 'a
  i_clk_a: input  'a clock,
  i_dat_a: input  'a logic,
  o_dat_a: output 'a logic,
  // クロックドメイン 'b
  i_clk_b: input  'b clock,
  o_dat_b: output 'b logic,
) {
  // 同一ドメイン内は安全
  assign  o_dat_a =  i_dat_a;

  // ドメインを跨ぐため明示が必要
  unsafe(cdc) {
    assign  o_dat_b =  i_dat_a;
  }
}
```

- **クロックドメインを型で明示**
  信号ごとに `'a` / `'b` を付与
- **跨ぐ箇所は `unsafe(cdc)` 必須**
  レビュー対象を明確化できる
- **未明示のCDCをエラー検出**
  高価なEDAツールに頼らずコンパイル時に発見

言語機能

ジェネリクスとプロトタイプ

```
// SRAMモジュールのプロトタイプ
proto module  SramProto #(
  param  WIDTH: u32 = 8,
)( /* ポート定義（略）*/ );

// SRAM種別に依存しないキュー
module  SramQueue::<T: SramProto> {
  inst  u_sram: T;
  // キューのロジック（略）
}

module  Top {
  inst  u0:  SramQueue::<SramTypeA>;
  inst  u1:  SramQueue::<SramTypeB>;
}
```

- **型パラメータで重複を排除**
  SRAM種別ごとにモジュールを書き分けない
- **`proto` で境界を規定**
  満たすべきインタフェースを型で保証
- **高い再利用性**
  SystemVerilogのgenerateより明快に記述

ツールチェインと評価

単一ツールチェインとシミュレーション性能

04

TOOLCHAIN

IRを核にした単一ツールチェイン

Veryl
ソースコード

構文解析 →

CST
構文木

意味解析 →

IR
型・接続・CDC検査

→

SystemVerilog生成
可読なSVを生成

フォーマッタ
スタイルを統一

言語サーバ
LSPで即時診断

シミュレータ
ネイティブ実行

論理合成
ゲートレベル解析

依存管理
Gitで依存を取得

TOOLCHAIN

シミュレータ

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

- **二段階のJITコンパイル**

  Craneliftで即座にシミュレーションを開始
  並行してCコンパイラで最適化し動的に切り替え

- **`veryl test` でテストを統合**

  単体テストを外部ツールなしで実行

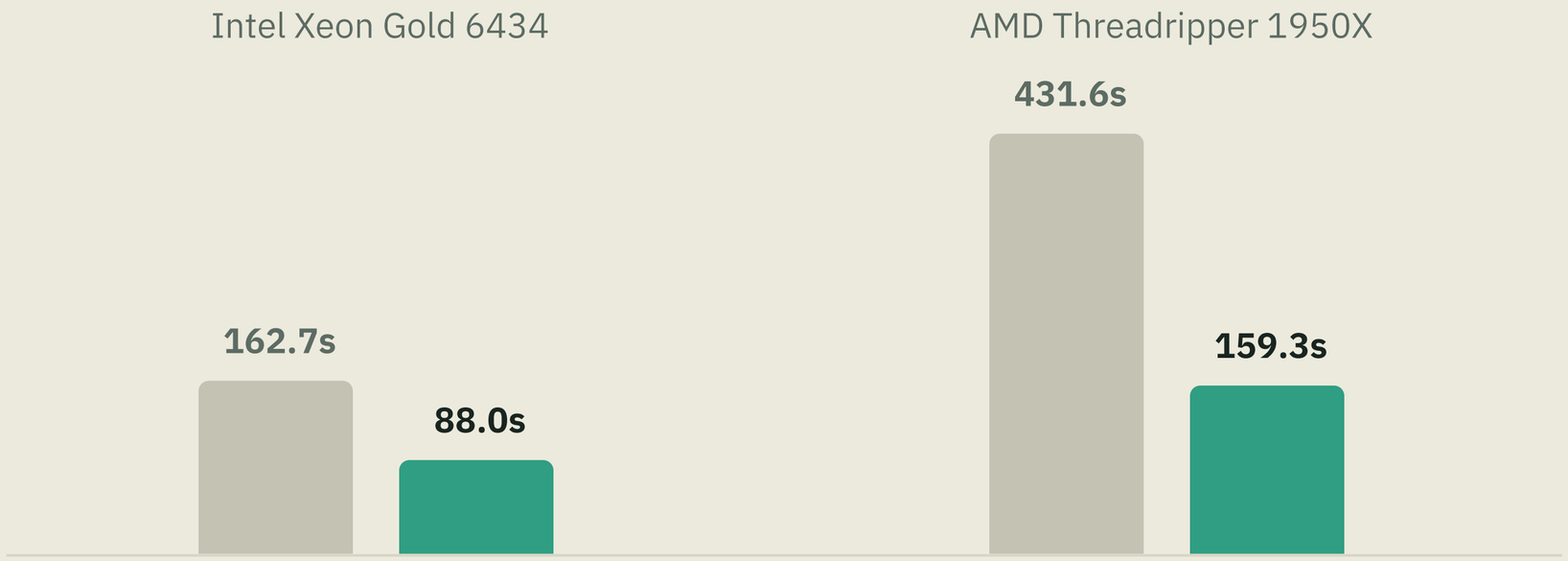
- **外部シミュレータ連携**

  Verilatorなど既存のシミュレータも利用できる

TOOLCHAIN

Linuxブートのシミュレーション性能

RVA23準拠RISC-VコアHeliodor ／ 単一スレッド・キャッシュ未使用（短いほど速い）



Verilator v5.050

Veryl 内蔵シミュレータ

-41〜63%

コンパイル時間を含んだ値であり、Craneliftによる高速コンパイルが効いている

シミュレーション時間のみでもVerilatorより20〜30%高速

TOOLCHAIN

論理合成とオープンPDK対応

```
$  veryl synth

synth: TopModule — 123 gates, 17 FFs
library: sky130_fd_sc_hd

summary:
  area:    1234.56 um²
  timing:    2.345 ns   8 levels
  power:    0.1234 mW
```

- **`veryl synth` でゲートレベルへ**
  面積・タイミング・電力を概算して報告
- **外部合成器なしで即座に確認**
  設計検討中の素早いフィードバックに
  精度はYosys比±10〜20%程度
- **オープンPDKに対応**
  SKY130 ／ ASAP7 ／ GF180MCU ／ IHP SG13G2

エコシステムとまとめ

オープンソースとしての広がり

05

ECOSYSTEM

活発に開発が続くプロジェクト

2022年12月に開始（2026年8月時点）

5,332	996	65	31
Commits	GitHub Stars	Releases	Contributors

ECOSYSTEM

実際の採用事例

クローズド

PEZY 次世代HPC/AIチップ

約50万行のSystemVerilogと約10万行のVerylを併用
実チップ開発でVerylを採用

オープンソース

Heliodor

RVA23準拠のアウトオブオーダRISC-Vコア
8コアSMPでLinuxが起動、ACT v4準拠

オープンソース

bluecore

Linuxが起動するRISC-Vコア
技術同人誌『Verylで作るCPU』で詳細を解説

教育

ルレオ工科大学

ディジタル回路設計の講義でVerylを採用

CONCLUSION

まとめ

01

HDLに特化した構文

クロックドメイン注釈・ジェネリクスなど設計を安全・簡潔にする機能

02

単一ツールチェイン

内蔵シミュレータはVerilator比で実行時間を41〜63%短縮

03

オープンソースで開発

実チップ・OSS・教育へ広がるエコシステム

WEB

veryl-lang.org

GITHUB

github.com/veryl-lang/veryl