クイズ　：　下記ソース内には、文法エラーでは無いが問題となる構造を含んでいます。どこが問題か見つけられますか？

VHDL

```
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.STD_LOGIC_UNSIGNED.all;

entity sample is
  port (CLK              : in std_logic ;
        RST              : in std_logic ;
        LD               : in std_logic ;
        DATAIN           : in std_logic_vector (7 downto 0) ;
        COUT             : out std_logic ;
        DATAOUT          : buffer std_logic_vector (7 downto 0)
  ) ;
end sample ;

architecture RTL of sample is
  signal c_out           : std_logic ;
  begin
    process (CLK, RST) begin
      if RST='0' then
        DATAOUT <= (others=>'0') ;
      elsif (CLK'event and CLK='1') then
        if LD='1' then
          DATAOUT <= DATAIN ;
          c_out <= '0' ;
        elsif DATAOUT = "11111111" then
          DATAOUT <= (others=>'0') ;
          c_out <= '1' ;
        else
          DATAOUT <= DATAOUT + '1' ;
          c_out <= '0' ;
        end if ;
      end if ;
    end process ;


    process (c_out, RST) begin
      if RST='0' then
        COUT <= '0' ;
      elsif c_out = '1' then
        COUT <= '1' ;
      end if ;
    end process ;

end RTL ;
```

VHDL 答え
・buffer 宣言を使用
(STARC_VHDL_2.1.3.2)
・COUTのProcess文はラッチを生成
(STARC_VHDL_2.2.1.1)
・c_out のリセット条件抜け
(STARC_VHDL_2.3.6.1)

Verilog

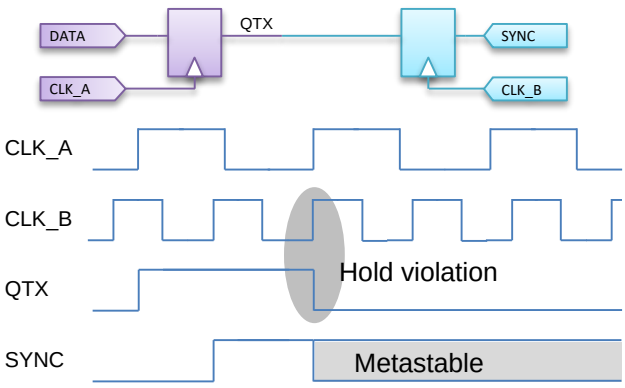
```
`timescale 1ns/10ps

module sample (CLK, RST, LD, DATAIN, COUT, DATAOUT) ;

input          CLK;
input          RST;
input          LD;
input    [8:0]  DATAIN;
output         COUT;
output   [7:0]  DATAOUT;

reg            c_out;
reg            COUT;
reg    [7:0]   DATAOUT;

always @(posedge CLK or negedge RST) begin
  if(!RST)
    DATAOUT <= 8'h00;
  else begin
    if (LD) begin
      DATAOUT <= DATAIN;
      c_out <= 1'b0;
    end
    else if (DATAOUT == 8'hFF) begin
      DATAOUT <= 8'h00;
      c_out <= 1'b1;
    end
    else begin
      DATAOUT <= DATAOUT + 1;
      c_out <= 1'b0;
    end
  end
end

always @(posedge c_out or negedge RST) begin
  if(!RST)
    COUT <= 1'b0;
  else if (c_out)
    COUT <= 1'b1;
end

endmodule
```

Verilog 答え
・COUTのalways文は入力値固定のFFを生成
(STARC_VLOG.2.3.5.1)
・c_out のリセット条件抜け
(STARC_VLOG.2.3.6.1)
・DATAIN/DATAOUTのバス幅が異なる
(STARC_VLOG.2.10.3.3)

さらに見過ごせない、省電力化、デバイスの高速化に伴うクロックドメインクロッシング（CDC）とリセットドメインクロッシング（RDC）問題

下記にある従来の検証手法では、発見が困難！！

1. デザインレビューなどの人手による解析
   - CDC/RDCポイント全ての洗い出し、およびチェックには多大な時間が必要
2. シミュレーション／タイミング解析
   - 通常シミュレーションでは発見不可
   - 予め決め打ちで検証すれば見つかる可能性はある → 現実的では無い
3. 実機検証
   - CDC/RDC問題の発見は運任せ！！
   - 問題が発生しても、再現性は極端に低い数時間、数日、数週間に1回

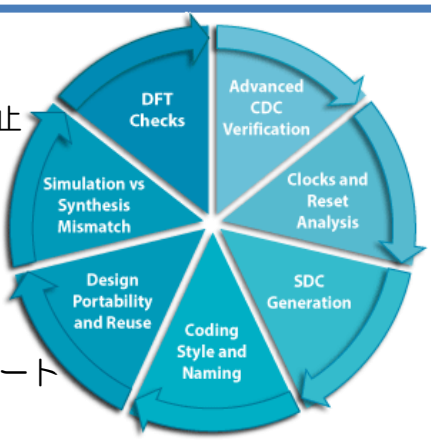


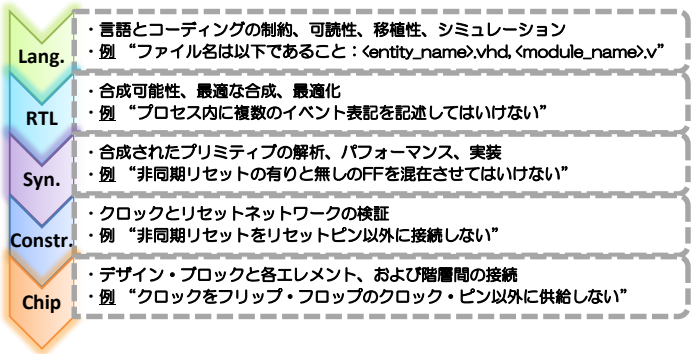
ALINT-PRO はデザイン内の各種問題を早期に発見！！

Windows/Linux で動作可能なスタティックデザイン解析ツール

●ALINT-PRO は、RTL コードを解析する検証ソリューション
- ➢クロック・リセット・ネットワークの解析
- ➢RTLシミュレーションと合成後シミュレーションのミスマッチを防止
- ➢コードのポータビリティと再利用性
- ➢ALDEC_CDCルール・プラグインによる幅広いCDCチェック
- ➢DFTチェック
- ➢IP記述用のデザイン制約拡張

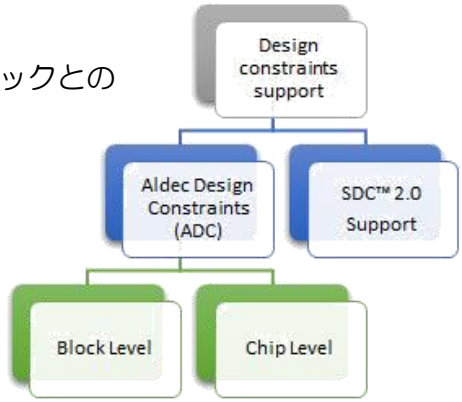
●デザイン・フローの早期にバグを検出
●ベンダツール（Xilinx, Altera）からのコンバートをサポート
●実行時の妨げになる各FPGAの基本ベンダのライブラリを標準サポート
●ブラックボックスファイルと制約を自動生成

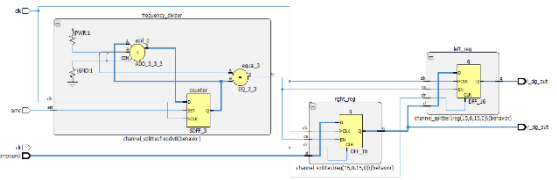




●業界最先端の設計手法に基づいた包括的なルールライブラリ
- ➢Aldec CDC/RDC ルール
- ➢Aldec Basic/Premium ルール
- ➢Aldec SystemVerilog ルール
- ➢STARC ルール
- ➢RMM(Reused Methodology Manual)ルール
- ➢DO-254 ルール

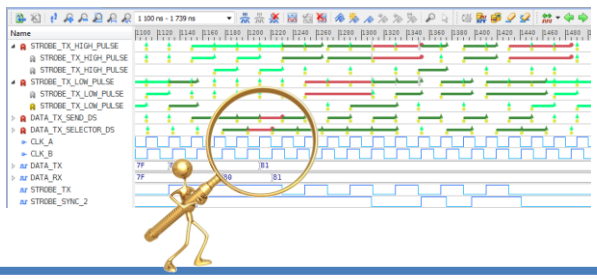
●デザイン制約をサポート
- ➢Synopsys Design Constraints (SDC) をサポート
  クロック宣言とそれらの関係性、および入力・出力ポートとクロックとの相互関係
- ➢Aldec Design Constraints (ADC) をサポート
  - ✓ **ブロック・レベル・デザイン制約**
    FPGAベンダ・プリミティブ、ビヘビアモデル、暗号化済みIP などの合成できないモジュールへの制約
  - ✓ **チップ・レベル・デザイン制約**
    リセット・ネットワーク記述、およびネットリストから情報
- ➢実行結果からSDC/ADCを自動生成可能

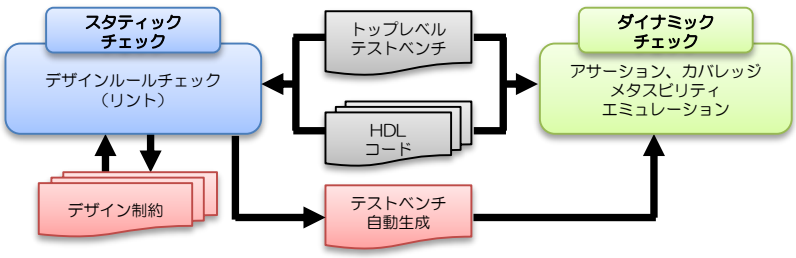




●スタティック検証によるチェックで問題個所を明確化
- ➢CDC/RDC 問題箇所を検出しレポート
- ➢クロック、およびデザイン構造をツリー表示
- ➢問題個所を回路図で表示することにより、解析時間を短縮
- ➢ステートマシーン記述をグラフィカル表示

●ダイナミック検証用のメタスビリティエミュレーション、およびCDC ポイントのアサーション/カバレッジ（SVA, PSL）を自動生成





・商品ページはコチラから

・お問合せはコチラから