社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2006－SLDM－127 (16)
2006／11／29

[特別講演] SystemVerilog チュートリアル

浜口 加寿美[1] 明石 貴昭[2] 湯井 丈晴[3] 後藤 謙治[4] 岡本 実幸[5]
杉浦 正志[6] 土屋 丈彦[7] 千綿 幸雄[8] 竹田津 弘州[9] 李 建道[10] 高嶺 美夫[11]

(1,9) 松下電器産業株式会社 〒571-8501 大阪府門真市大字門真 1006
(2) 日本シノプシス株式会社 〒140-0014 東京都品川区大井 1-28-1
(3) 株式会社沖ネットワークエルエスアイ 〒140-0031 東京都品川区五反田 2-15-7
(4) 日本ケイデンス・デザイン・システムズ社 〒222-0033 神奈川県横浜市港北区新横浜 3-17-6
(5) 三洋半導体株式会社 〒370-0596 群馬県邑楽郡大泉町坂田 1-1-1
(6) 株式会社図研 〒224-8585 神奈川県横浜市都筑区荏田東 2-25-1
(7) 株式会社東芝 〒105-8001 東京都港区芝浦 1-1-1
(8) 富士通株式会社 〒211-8588 神奈川県川崎市中原区上小田中 4-1-1
(10) メンター・グラフィックス・ジャパン株式会社 〒140-0001 東京都品川区北品川 4-7-35
(11) 株式会社ルネサステクノロジ 〒100-6334 東京都千代田区丸の内 2-4-1

E-mail: (1,9){hamaguchi.kasumi, taketazu.hirokuni}@jp.panasonic.com, (2) akashi@synopsys.com, (3) yui578@oki.com, (4) gotok@cadence.com, (5) okamoto@gf.hm.rd.sanyo.co.jp, (6) sugiura@zuken.co.jp, (7) takehiko2.tsuchiya@toshiba.co.jp, (8) chiwata.yukio@jp.fujitsu.com, (10) kun-do_Lee@mentor.com, (11) takamine.yoshio@renesas.com

あらまし JEITA (社団法人 電子情報技術産業協会) EDA 技術専門委員会／標準化小委員会傘下で, 10 社のメンバにより組織されている SystemVerilog タスクグループが, SystemVerilog の「設計のための構文」及び「検証のための構文」の中から「アサーション」をチュートリアル形式で説明します.

キーワード SystemVerilog, Verilog HDL, HDVL, RTL, アサーション

[Special Lecture] SystemVerilog Tutorial

Kasumi Hamaguchi[1] Takaaki Akashi[2] Takeharu Yui[3] Kenji Goto[4] Miyuki Okamoto[5]
Masashi Sugiura[6] Takehiko Tsuchiya[7] Yukio Chiwata[8] Hirokuni Taketazu[9] Kun Do Lee[10]
and Yoshio Takamine[11]

(1,9) Matsushita Electric Industrial Co., Ltd. 1600 Kadoma, Kadoma, Osaka, 571-8501 Japan
(2) Nihon Synopsys Co.,Ltd. 1-28-1 Oi, Shinagawa, Tokyo, 140-0014 Japan
(3) Oki Network LSI Co.,Ltd. 2-15-7 Gotanda, Shinagawa, Tokyo, 140-0031 Japan
(4) Sanyo Semiconductor Co., Ltd. 1-1-1 Sakata, Oizumi-machi, Ora, Gunma, 370-0596 Japan
(5) Zuken Inc. 2-25-1 Eda-Higashi, Tsuzuki, Yokohama, Kanagawa, 224-8585 Japan
(6) Cadence Design Systems, Japan B.V. 3-17-6 Shin-Yokohama, Kohoku, Yokohama, Kanagawa, 222-0033 Japan
(7) Toshiba Corporation. 1-1, Shibaura 1-chome, Minato-ku, Tokyo 105-8001 Japan
(8) FUJITSU LIMITED 1-1, Kamikodanaka 4-chome, Nakahara-ku, Kawasaki, kanagawa, 211-8588 Japan
(10) Mentor Graphics Japan Co.,Ltd. 4-7-35 Kita-Shinagawa, Shinagawa, Tokyo, 140-0001 Japan
(11) Renesas Technology Corp. 2-4-1 Marunouchi, Chiyoda, Tokyo, 100-6334 Japan

E-mail: (1,9){hamaguchi.kasumi, taketazu.hirokuni}@jp.panasonic.com, (2) akashi@synopsys.com, (3) yui578@oki.com, (4) gotok@cadence.com, (5) okamoto@gf.hm.rd.sanyo.co.jp, (6) sugiura@zuken.co.jp, (7) takehiko2.tsuchiya@toshiba.co.jp, (8) chiwata.yukio@jp.fujitsu.com, (10) kun-do_Lee@mentor.com, (11) takamine.yoshio@renesas.com

Abstract SystemVerilog Task Group which is formed by members from ten companies under the Standardization Support Sub-Committee of EDA Technical Committee, JEITA (Japan Electronics and Information Technology Industries Association) will present a tutorial style lecture of two feature of SystemVerilog; "design construct" and "assertion" in the "verification construct".

Keyword SystemVerilog, Verilog HDL, HDVL, RTL, Assertion

SystemVerilog (IEEE Std. 1800-2005) チュートリアル

2006/11/29 @デザインガイア2006

JEITA SystemVerilogタスク・グループ

JEITA SystemVerilog

アジェンダ

- JEITA SystemVerilogタスク・グループ紹介
- SystemVerilog概要
- 言語チュートリアル
  - 設計のための構文
  - 検証のための構文
- まとめ

SystemVerilog Task Groupメンバー

- JEITA：社団法人 電子情報技術産業協会
- 「JEITA EDA技術専門委員会／標準化小委員会傘下の SystemVerilog Task Group (SV-TG)
  - SystemVerilogの国際標準化活動に日本から参画
  - メンバー企業 10社

沖ネットワークエルエスアイ	日本シノプシス
三洋	松下電器産業
図研	メンター・グラフィックス・ジャパン
東芝	富士通
日本ケイデンス・デザイン・システムズ社	ルネサステクノロジ

(注)五十音順、敬称略

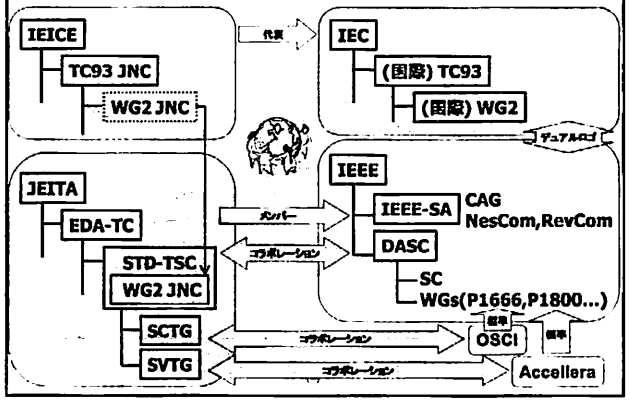
JEITA組織図

Japan Electronics and Information Technology Industries Association

- JEITA
  - Information Systems Board
  - Personal Information Board
  - Digital Home Appliances Board
  - Industrial Equipment and Social Systems Board
  - Display Devices Board
  - Electronic Components Board
  - **Semiconductor Board(JEITA-JSIA)**
    - Semiconductor Industrial Affairs Committee
    - Semiconductor International Affairs Committee
    - **Semiconductor Technology Committee**
      - Marketing Committee
      - Road Map Committee
      - ⋮
      - **EDA Technical Committee(EDA-TC)**

* EDA Technical Committee was formed to handle EDIF 2.0 standard as one of technical committees in JEITA (former EIAJ) in April 1993.

世界の標準機関とコラボレーション



アジェンダ

- JEITA SystemVerilogタスク・グループ紹介
- SystemVerilog概要
- 言語チュートリアル
  - 設計のための構文
  - 検証のための構文
- まとめ

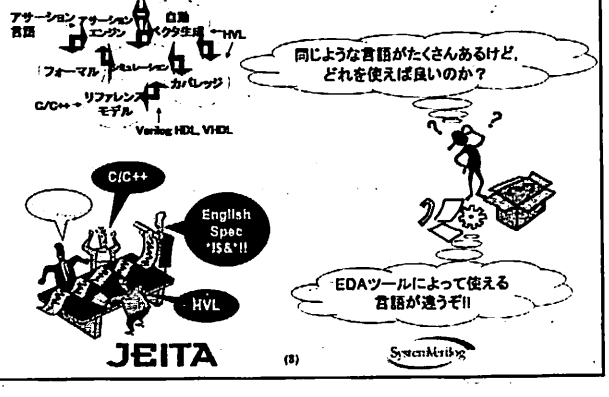
JEITA (6) SystemVerilog

何故SystemVerilogなのか?

- より大きな規模の回路をより短期間で設計しなくてはならない
  - より簡潔な記述で設計する必要 -> 記述量の削減
  - 曖昧の無い言語が必要 -> 不要なイタレーションの削減
  - 新しい検証テクニックが必要 -> より多くのテストを効率良く
  - 高抽象度化のための言語構造が必要 -> TLM構造のサポート
- しかし...
  - Verilog HDL / VHDLでは問題が多い...
    - Verilog HDLには,信号競合という大問題がある
    - HDLのピンモデルでは抽象度は上げられない
  - 過去の資産(デザイン,エンジニアのスキル等)は捨てたくない
  - 等々
- 新しい言語ではなく,Verilog HDLをエンハンスすることで,資産を継承

JEITA (7) SystemVerilog

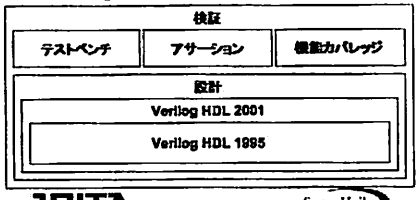
新しい技術の台頭と共に言語が乱立



JEITA (8) SystemVerilog

SystemVerilog の概要

- IEEE Std. 1364-2001 (Verilog HDL)の拡張
- IEEE Std. 1800-2005 標準
- デザインと検証のために言語を統合
  - HDVL (Hardware Description and Verification Language)
- SystemVerilog はVerilog HDL の新バージョン!



設計記述面の利点

- Verilog HDL記述の曖昧さの解消
- 可読性, 保守容易性の向上,記述量の削減
  (コンサイスRTL)
  - コード記述量を1/3～1/10 以下へと削減可能
  - 簡潔な記述 = 生産性の向上,バグの減少
- TLM記述を可能とする
  - `interface`構造を追加
  - ピンレベル・インターフェイス以外の`module`接続をサポート
- システムから ゲート・レベルまで
  - ゲート・レベル・ネットリストはVerilog HDLをそのまま継承

検証記述面の利点

- 検証記述の改善
  - 単一環境に組み込まれた検証
    - アドバンスド・テストベンチ構文を追加
    - アサーション構文を追加
    - カバレッジ構文を追加
- DPI(Direct Programing Interface)によるC/C++との親和性強化

他言語と比べるならば・・・

- vs 既存HDL
  - 検証面の充実
    - 制約付きランダム・スティミュラス
    - アサーション
    - 機能カバレッジ
  - 動的オブジェクトの追加
- vs ソフトウェア言語
  - 並列性
    - `always`, `fork-join`, ゲート・プリミティブ
  - 時間の概念
    - `#`(遅延), `##`(サイクル遅延), `@`(イベント遅延)

アジェンダ

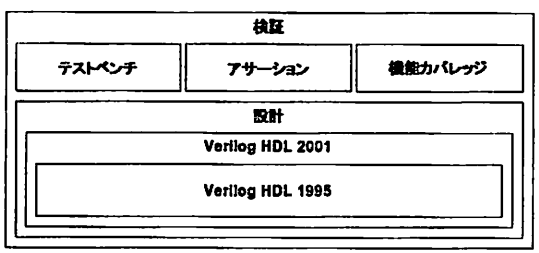
- JEITA SystemVerilogタスク・グループ紹介
- SystemVerilog概要
- 言語チュートリアル
  - 設計のための構文
  - 検証のための構文
- まとめ

言語特徴説明

- 設計のための構文
  - データ型
  - インタフェース
  - デザインインテント手続きブロック
  - デザインインテント条件文
- 検証のための構文
  - アサーション

※ テストベンチ構文とカバレッジ構文は今回は,割愛

設計のための構文



データ型 - 基本

- `logic` 4値 (0/1/X/Z)
  - `reg`, `wire`の区別なく記述可能
  - `wire`として使用する場合, ドライバを一つに限定
  - レース状態が起きないモデリングを可能に
  - リント・ツールでチェック可能(早期検出)
- `bit` 2値 (0/1)
  - シミュレーションの高速化
  - 初期値は0と規定されている
  - 全てのシミュレータの2値モードが整合

データ型 – ユーザ定義

- typedef
  - 可読性の向上
    - ユーザにとって意味のある型を定義できる
  - 保守容易性の向上
    - 記述部を変更せず，宣言部を変更するだけで定義を変更できる
- typedefを用いたワード定義の例

```
// 1ワードを32ビットとして定義
typedef logic [31:0] word_t;
word_t A, B;
```

```
// 1ワードを64ビットとして定義
typedef logic [63:0] word_t;
word_t A, B;
```


データ型 – 構造体

- struct
  - C言語に似た，簡潔な宣言
    - typedefを用いて，データ型名を指定可能
    - 構造体の全体に対し，一括して値を代入可能
  - 可読性，保守容易性の向上，記述量の削減

```
struct {
  bit [7:0]  opcode; // 8bit幅の2値
  bit [23:0] addr;   // 24bit幅の2値
  data_word  data;   // typedefされたユーザデータ型
} instruction;       // 名前付き構造体の宣言

instruction IR;      // 構造体の配置

IR = {6, 377, 45};   // 構造体への代入
```


データ型 – 列挙型

- enum
  - Verilog HDL 1995/2001
    - parameter定数宣言，もしくはdefineマクロ宣言
    - 値に名前をつけるにとどまる
  - SystemVerilog
    - enum宣言
    - 値の集合を定義，初期値設定，範囲チェック可能
    - 高抽象化モデリング
    - 記述量の削減に加え，可読性，保守容易性の向上

```
parameter RED = 0, YELLOW = 1, GREEN = 2;
reg [1:0] traffic_light;
```

Verilog HDL 1995/2001

```
enum {red, yellow, green} traffic_light;
```

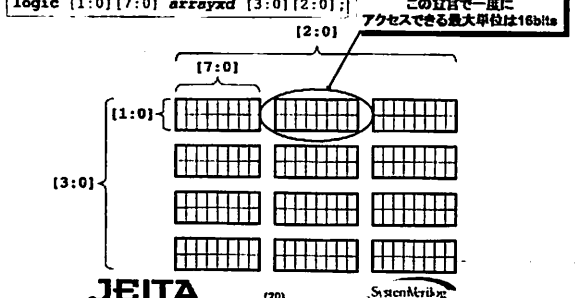
SystemVerilog

データ型 – 多次元配列

- 2次元パック型 x 2次元アンパック型

```
logic [1:0][7:0] arrayxd [3:0][2:0];
```

この宣言で一度にアクセスできる最大単位は16bits



データ型 – 多次元配列

- 可読性,保守容易性,記述量の削減

```
reg     [7:0]       mem0[0:1023];
reg     [7:0]       mem1[0:1023];
reg     [7:0]       mem2[0:1023];
reg     [7:0]       mem3[0:1023];
reg     [31:0]      q;

always @(posedge clk) begin
    if(ce) begin
        if(we[3]) mem3[a] <= d[31:24];
        if(we[2]) mem2[a] <= d[23:16];
        if(we[1]) mem1[a] <= d[15:8];
        if(we[0]) mem0[a] <= d[7:0];
    end
end

always @(posedge clk) begin
    if(ce)
        q <= {mem3[a], mem2[a],
              mem1[a], mem0[a]};
end
```

Verilog HDL 1995/2001

```
reg     [3:0][7:0]      mem[0:1023];
reg     [3:0][7:0]      q;
```

多次元化

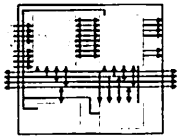
```
always @(posedge clk) begin
    if(ce) begin
        if(we[3]) mem[a][3] <= d[3];
        if(we[2]) mem[a][2] <= d[2];
        if(we[1]) mem[a][1] <= d[1];
        if(we[0]) mem[a][0] <= d[0];
    end
end

always @(posedge clk) begin
    if(ce)
        q <= mem[a];
end
```

同一変数をバイト単位でハンドリング

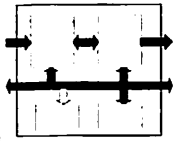
SystemVerilog

インタフェース

- Verilog HDL 1995/2001 モジュール・ポート接続
  - モジュール図の詳細な端子接続を知る必要がある
  - 入出力の端子群に変更があった場合記述の変更が煩雑
  - ポート宣言を多くのモジュールで何度も繰り返す必要がある



- SystemVerilog interface接続が可能
  - モジュール図の接続を一つに束ねて行う
  - 接続の内容の定義はモジュール定義からは独立して行う
  - 記述量の削減に多大な効果
  - ポートの追加・削除に伴うファイル修正作業を軽減



JEITA

Systembrikg

インタフェース

```
module memmod(
  input  logic        req,
         bit          clk,
         bit          start,
         logic [1:0]  mode,
         logic [7:0]  addr,
  inout  wire  [7:0]  data,
  output bit          gnt,
         bit          rdy );
         logic        ovall;
  always @(posedge clk) gnt <= req & ovall;
...
endmodule

module cpu(
  input  bit          clk,
         bit          gnt,
         bit          rdy,
  inout  wire [7:0]   data,
  output bit          req,
         bit          start,
       logic [7:0]    addr,
       logic [1:0]    mode );
...
endmodule

module top;
  logic req, gnt, start, rdy;
  logic clk = 0;
  logic [1:0] mode;
  logic [7:0] addr;
  wire  [7:0] data;
  memmod MEM(req, clk, start, mode, addr, data, gnt, rdy);
  cpu CPU(clk, gnt, rdy, data, req, start, addr, mode);
endmodule
```

interfaceを使わない場合

```
interface simple_bus;
  logic       req, gnt;
  logic [7:0] addr, data;
  logic [1:0] mode;
  logic       start, rdy;
endinterface

module memmod(simple_bus a, input bit clk);
  logic       ovall;
  always @(posedge clk) a.gnt <= a.req & ovall;
...
endmodule

module cpu(simple_bus b, input bit clk);
...
endmodule

module top;
  logic clk = 0;
  simple_bus sb_intf();
  memmod mem(sb_intf, clk);
  cpu cpu(.b(sb_intf), .clk(clk));
endmodule
```

interfaceを使う場合

Systembrikg

モジュール接続

- Verilog HDL 1995/2001 での接続:
  - 順序接続
  - 名前接続

```
module my_chip (input wire clock, reset,
  wire [63:0] a, b, c, d;
  alu u1 (clock, a, b, reset, c, d);
  alu u2 (.ck(clock), .reset(reset), .a(a), .b(b), .c(c), .d(d));
```

ポート宣言の順序で接続(誤りやすい記述)

名前同士で接続(冗長な記述)

- SystemVerilog で拡張された名前接続:
  - .name を用いて同一端子名省略
  - .* ワイルドカード

```
module my_chip (input wire clock, reset,
  wire [63:0] a, b, c, d;
  alu u1 (.ck(clock), .reset, .a, .b, .c, .d);
  alu u2 (.ck(clock), .*);
```

.name によって接続される端子とネット名が同一のときに記述を削減できる

.* は同一名の端子とネットを自動的に接続する

JEITA

Systembrikg

デザインインテント手続きブロック

- always_comb/latch/ff
  - 設計者が意図する回路を明示的に指定
  - ツール固有に依存しない回路が得られる
  - always_comb/latchはセンシティビティリストが不要
    - ツールは代入の右辺から導き出す
  - 詳細な規定によりツール間の不整合を防ぐ
    - initial, alwaysすべての手続き的ブロックがアクティブになった後,時間ゼロにおいて自動的に1回実行される
    - ⇒ 時間ゼロでの出力値は入力値に一致する
    - Verilog HDL 1995/2001のalways記述でeイベントがトリガされず、シミュレーションがロックする問題を解決

```
always_comb begin
  tmp1 = a & b;
  tmp2 = c & d;
  y = tmp1 | tmp2;
end
```

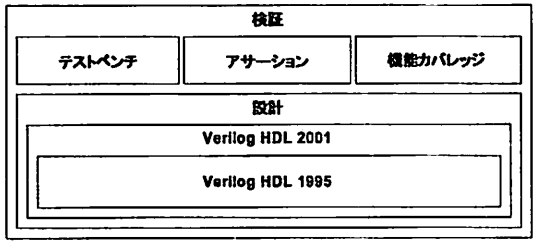

デザインインテント条件文

- unique/priority case/if
  - case/if文にて条件処理を順次/並列の何れかの解釈に指定
  - ツールベンダー依存のプラグマを排除
    - // full_case, parallel_case
  - シミュレーション,合成で統一した解釈を実現
    - RTL/ゲート間のシミュレーション・ミスマッチ防止

```
unique case (sel)
  3'b001 : muxo = a;
  3'b010 : muxo = b;
  3'b100 : muxo = c;
endcase
```

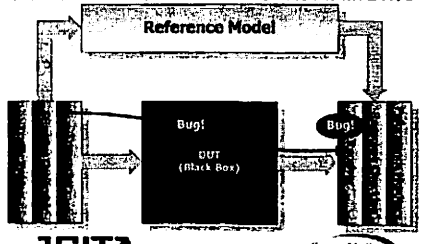
```
priority if (sel0) q = 3'b000;
    else if (sel1) q = 3'b011;
    else if (sel2) q = 3'b110;
    else if (sel3) q = 3'b111;
```


検証のための構文



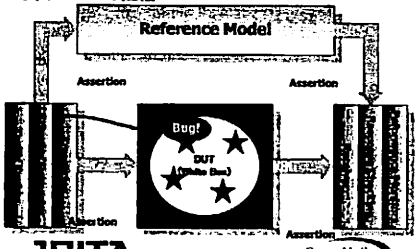
従来の検証手法

- バグの発見のためには. . .
  1. スティミュラスを加えて不正な動作を活性化
  2. 同じく不正な動作の結果を出力へ伝播させる
  3. 出力波形からバックトレースしてバグの原因箇所を特定



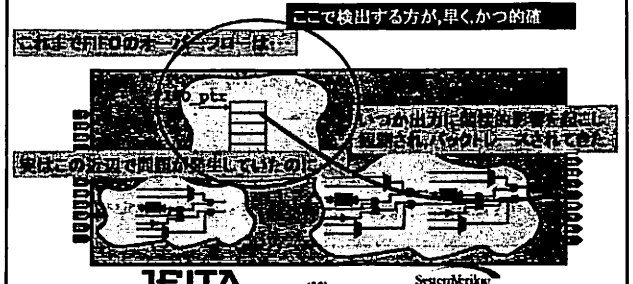
アサーションベース検証

- 観測性の向上
  - バグの早期発見
  - 出力に伝播させることが困難なバグの発見
  - バックトレースの短縮



アサーションでバグをすぐに見つける

- ブロック内ホワイトボックス検証
  - 設計のバグを,プライマリ出力に到達する前に観測



アサーションとは？ 何故使用するのか？

- アサーションは検証要素
  - 設計ブロックや, そのインタフェース上の禁止される動作を監視
  - アサーションで記述される, 期待される動作を追跡
- アサーションの利点
  - 観測性の向上
  - デバッグ効率向上
  - 検証期間短縮
  - カバレッジの一指標

アサーション使用例

- 禁止状態のチェック
  - プロパティの定義
  - プロパティの引数による再利用
  - not 演算子
- ワンホットのチェック
  - $onehot システム・ファンクション
- タイムアウト
  - シーケンスの定義
  - シーケンスの引数による再利用
  - プロパティからのシーケンス呼び出し
- 状態保持のチェック
  - throughout 演算子

禁止状態のチェック

- 意図していない状態になっていないか？
  - FIFOにおいて*push*と*pop*は同時に発生しない
  - FIFOにおいて*full*のときに書き込みはできない

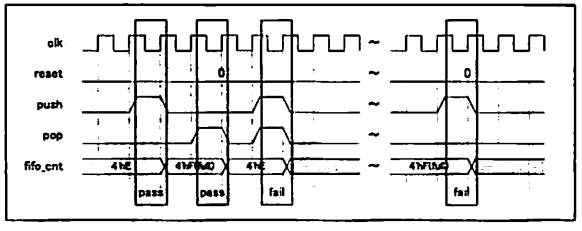
⇩

  - propertyを使用して期待する状態を定義
  - notを使用して禁止状態を指定

```
property p_expected([引数]);
        <期待する状態>
endproperty
property p_illegal([引数]);
        not (<禁止状態>);
endproperty
```


禁止状態の使用例

- FIFOにおいて*push*と*pop*は同時に発生しない
- FIFOにおいて*full*のときに書き込みはできない



禁止状態の記述例

- FIFOにおいて*push*と*pop*は同時に発生しない(禁止状態1)
- FIFOにおいて*full*のときに書き込みはできない(禁止状態2)

```
property p_fifo_push_pop;  // 禁止状態1を定義
      @(posedge clk) disable iff (reset)
                          // サンプルクロックと非同期リセット
      not (push && pop);
endproperty
property p_fifo_full_push; // 禁止状態2を定義
      @(posedge clk) disable iff (reset)
      not (full && push && !pop);
endproperty

// 定義したプロパティをアサート
a_fifo_rule_1 : assert property (p_fifo_push_pop);
a_fifo_rule_2 : assert property (p_fifo_full_push);
```


ワンホットのチェック

- ワンホットであるべき信号が,不正状態にならないか？
  - 対象とする信号の1ビットのみ"1"である

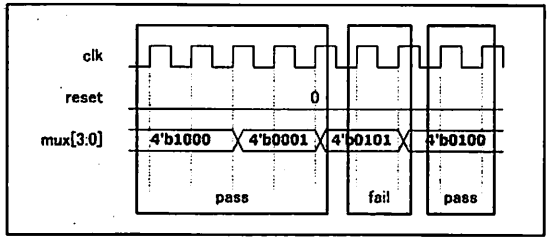
⇩

- $oneshotシステム・ファンクションを使用

```
$onehot(<チェック対象信号>)
```

ワンホットの使用例

- 信号中の1ビットのみ1である



ワンホットの記述例

- 信号中の1ビットのみ1である
  - プロパティ記述をassert

```
property p_onehot_check; // ワンホット指定を定義
        @(posedge clk) disable iff (reset)
        $onehot (mux);
endproperty
// 定義したプロパティをアサート
a_onehot_1 : assert property (p_onehot_check);
```

  - assert宣言内にプロパティを記述

```
// もしくはプロパティ定義を省略
a_onehot_2 : assert property (
        @(posedge clk) disable iff (reset)
        $onehot (mux));
```


タイムアウトのチェック

- 規定時間以内に動作が完了するか？
  - *req*のアサート後，既定サイクル以内に*ack*がアサート
  - *bus_trans*のデアサート後，次のサイクルで*bus_req*デアサート

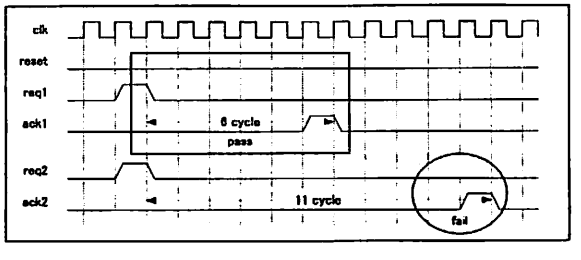
- 規定の動作をsequenceで定義

```
sequence s_sequence(引数);
        <動作シーケンス>
endsequence
```


タイムアウトの使用例

- *req*のアサート後，既定サイクル以内に*ack*がアサート

※ *req1-ack1*: 8サイクル以内，*req2-ack2*: 10サイクル以内



タイムアウトの記述例

- *req*のアサート後, 既定サイクル以内に*ack*がアサート

※ *req1-ack1*: 8サイクル以内, *req2-ack2*: 10サイクル以内

```
sequence s_response(ack, maxcycle); // req-ackのタイムアウトを定義
        @(posedge clk) ##[0:maxcycle-1] ack;
                                    // 0-maxcycleの間にackがアサート
endsequence
property p_response(req, ack, maxcycle); // req-ackのプロパティ
        @(posedge clk) disable iff (reset)// シーケンスをreq,ackに適用
        req |=> s_response (ack, maxcycle);
endproperty                                    <- sequence,property 再利用
// 定義したプロパティをreq1-ack1,req2-ack2それぞれにアサート
// (既定サイクルは異なる)
a_timeout_rule_1 : assert property (p_response(req1, ack1, 8));
a_timeout_rule_2 : assert property (p_response(req2, ack2, 10));
```


状態保持のチェック

- 特定シーケンス中, 状態が保持されているか？
  - *frame*はバスサイクル期間中アクティブである
  - *ready*が8クロックlowである期間中, *transmit*はアサートされる

- throughoutを使用

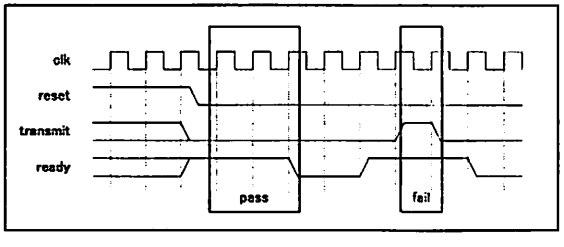
```
sequence s_throughout;
        <状態状態> throughout <保持期間>
endsequence;
```

※保持状態はブーリアン表現

状態保持のチェック使用例

- *reset*と*transmit*は, readyシーケンス中インアクティブである

※ readyシーケンス::= readyが連続的に3サイクルアクティブ



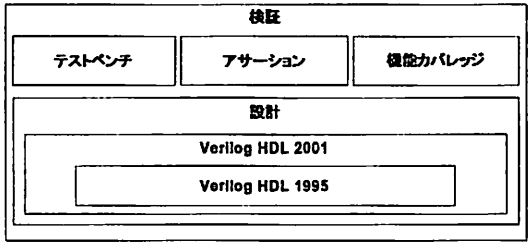
状態保持のチェック記述例

- *reset*と*transmit*は, readyシーケンス中インアクティブである

※ readyシーケンス::= readyが連続的に3サイクルアクティブ

```
sequence s_ready_3; // 保持すべきシーケンスを定義
        @(posedge clk)
          (!reset && !transmit ) throughout ready[*3];
endsequence

// シーケンスをclkドメインのプロパティとしてアサート
a_throughout_rule : assert property(
        @(posedge clk) disable iff (reset)
          s_ready_3);
```

テストベンチ・チュートリアルはSV-UF2007で！



JEITA (45) SystemVerilog

SystemVerilog ユーザ・フォーラム

「システム・デザイン・フォーラム 2007」

セッション 3： SystemVerilog ユーザ・フォーラム 2007
26 日(13:30～15:30)

Verilog HDL (IEEE Std. 1364) の次世代言語として、2005 年 11 月に標準化完了した SystemVerilog (IEEE Std. 1800-2005) は、LSI 設計者や検証エンジニアの間で急速に適用が広がっています。
本セッションでは、1) Accellera による次の SystemVerilog 改定に向けた取り組みの紹介、2) JEITA SystemVerilog タスクグループによる SystemVerilog テストベンチ・チュートリアルと技術紹介、3) 日本の SystemVerilog ユーザによる、SystemVerilog 検証事例発表、を行います。

司会 ：浜口 知寿美 氏 （JEITA SystemVerilog タスクグループ 三金/松下電器産業）
① SystemVerilog 標準化アップデート ：Dennis Brophy 氏 （Accellera）
② SystemVerilog テストベンチ言語チュートリアル ：SystemVerilog タスクグループ
③ 検証言語としての SystemVerilog 適用事例 ：細田 文良夫 氏 （ルネサステクノロジ）
④ SystemVerilog で構築したアレイプロセッサ検証環境とその効果 ：清水 圭典 氏 （ソナック）

日 時：1月25日(木) 13:30-17:30 セッション1 フィジカル・デザイン・フォーラム
1月26日(金) 10:00-12:00 セッション2 SystemC ユーザ・フォーラム 2007
13:30-15:30 セッション3 SystemVerilog ユーザ・フォーラム 2007
場 所：パシフィコ横浜 アネックスホール

まとめ

- 設計・検証言語としてSystemVerilogがIEEEで標準化
- SystemVerilogの特徴から
  - 設計のための構文紹介
  - アサーションを構文紹介
- テストベンチと機能カバレッジは,2007年1月26日の「SystemVerilogユーザ・フォーラム」(EDSフェア併設システム・デザイン・フォーラム内)にて紹介予定

JEITA (47) SystemVerilog

予約語一覧(221)

always	for	output	supply0	automatic	alias	endclass	join_none	string
and	force	parameter	supply1	cell	always_comb	endclocking	local	struct
assign	forever	pmos	table	config	always_ff	endgroup	logic	super
begin	fork	posedge	task	design	always_latch	endinterface	longint	tagged
buf	function	primitive	time	endconfig	assert	endpackage	matches	this
bufif0	highz0	pull0	tran	endgenerate	assume	endprogram	modport	throughout
bufif1	highz1	pull1	tranif0	generate	before	endproperty	new	timeprecision
case	if	pulldown	tranif1	genvar	bind	endsequence	null	timeunit
casex	ifnone	pullup	tri	incdir	bins	enum	package	type
casez	initial	rcmos	tri0	include	binsof	expect	packed	typedef
cmos	inout	real	tri1	instance	bit	export	priority	union
deassign	input	realtime	triand	liblist	break	extends	program	unique
default	integer	reg	trior	library	byte	extern	property	wait
defparam	join	release	trireg	localparam	chandle	final	protected	var
disable	large	repeat	vectored	noshowcancelled	class	first_match	pure	virtual
edge	macromodule	rnmos	wait	pulsestyle_ondetect	clocking	foreach	rand	void
else	medium	rpmos	wand	pulsestyle_onevent	const	forkjoin	randc	wait_order
end	module	rtran	weak0	showcancelled	constraint	iff	randcase	wildcard
endcase	nand	rtranif0	weak1	signed	context	ignore_bins	randsequence	with
endfunction	negedge	rtranif1	while	unsigned	continue	illegal_bins	ref	within
endmodule	nmos	scalared	wire	use	cover	import	return	
endprimitive	nor	small	wor		covergroup	inside	sequence	
endspecify	not	specify	xnor		coverpoint	int	shortint	
endtable	notif0	specparam	xor		cross	interface	shortreal	
endtask	notif1	strong0			dist	intersect	solve	
event	or	strong1			do	join_any	static	

Verilog HDL 2001 (211)

Verilog HDL 1995 (102)

SystemVerilog (93)

(48)