Artgraphics

SystemVerilog 入門

Document Revision: 2.0, 2019.06.16

アートグラフィックス
篠塚一也

SystemVerilog 入門

〒124-0012 東京都葛飾区立石 8-14-1
www.artgraphics.co.jp

SystemVerilog Primer

8-14-1, Tateishi, Katsushika-ku, Tokyo, 124-0012 Japan
www.artgraphics.co.jp

はじめに

この入門書は SystemVerilog の基本的な知識を提供する為に書かれた学習用の素材です。短期間に SystemVerilog の基礎知識を習得したいと思う方々に向いています。特長は、SystemVerilog 特有の機能を出来るだけ明確に、且つ、正確に記述している事にあります。本書を通じて、設計及び検証作業に役立つ知識を取得する事を主眼にしています。アサーション、及び、ファンクショナル・カバレッジの基礎知識の習得にも役立ちます。

本文書中に引用する例は弊社の設計・検証ツールにより確認してありますが、絶対的に正しい記述であるかは保障の限りではありません。それらの例を実践で使用する場合には、使用者の責任において実行して下さい。

本書の記述対象である SystemVerilog は、IEEE Std 1800-2017 標準規格を基にしています。この規格を以下では原書（文献[1]）と呼ぶ事にします。本書に記述されている記述に関しての更に詳しい説明、又は、質問・疑問を解消する為には、原書をお読み下さい。

また、本書では原書で使用されている用語をそのままの形で引用する場合が多くなっています。理由は、対応する適切な日本語訳が見つからないからです。読者自身で適切な翻訳を試みて下さい。

誤字訂正、及び、説明の補足を随時行います。従って、常に本書の内容は更新を余儀なくされます。ご了承ください。

アートグラフィックス
篠塚一也

注意事項

日本国内には SystemVerilog に関する良書が皆無な為、本書を執筆しました。SystemVerilog の知識を個人的に習得する目的として本書を活用して下さい。本書を通して、業務（実践）で必要となる SystemVerilog に関する知識を習得して頂くのが本来の目的です。

転用目的（本来の目的と違った他の用途に使う事）で本書を使用する事はご遠慮下さい。また、本書から学んだ知識を転載する場合等は出典が本書である事を明記して下さい。但し、他の著者の文書にも書かれている内容は、この限りではありません。

本書は、弊社の設計・検証ツールの付属部品です。設計・検証ツールのライセンス保有者のみ使用して下さい。

変更履歴

日付	Revision	変更点
2018.02.28	1.0	第一版。
2018.04.30	1.1	① SystemVerilog と Verilog の差異の記述を追加。 ② その他、誤字を訂正。
2018.06.06	1.2	① 各章に於いて説明を補足。
2018.07.25	1.3	① IEEE Std 1800-2012 を最新 IEEE Std 1800-2017 に更新。 ② 第五章にシーケンスによるイベント制御を追加。
2018.09.30	1.4	① 第一章に加筆。
		① 第九章に C/C++とのインターフェース機能の記述を追加。 ② 誤字訂正。
2018.12.01	1.5	① fork/join_none の例 5-4 を変更。 ② 節 9.10 を追加。
2018.12.16	1.6	① 節 1.3.1 を追加
2019.01.20	1.7	① 「SystemVerilog による検証の実践」の章を追加。
2019.04.21	1.8	① 誤字訂正
2019.05.12	1.9	① 第四章に Interface クラスの解説を追加。 ② 誤字訂正
2019.06.16	2.0	① enum に関する特殊な機能を追加。 ② 誤字訂正

目次

1 概要

1.1 SystemVerilog の歴史

SystemVerilog 言語仕様の主要部分は SUPERLOG を基にして Accellera standard groups により開発され、2002 年に SystemVerilog 3.0 としてリリースされました。SystemVerilog 3.0 は第三世代の Verilog 言語として誕生しました。第一世代は広く知られている Verilog-1995 で、第二世代は Verilog 2001 です。

言語の世代	言語
第一世代	Verilog-1995
第二世代	Verilog-2001
第三世代	SystemVerilog 3.0

SystemVerilog 3.0 までの後 SystemVerilog 3.1、3.1a、SystemVerilog 3.1 等の改訂を経て、2005 年 11 月に IEEE Std 1800-2005 として正式に公開されました。これが初めての SystemVerilog 言語仕様です。尚、SystemVerilog には SUPERLOG だけでなく、OpenVERA 検証言語、OpenVERA アサーション、PSL アサーション等の技術が使用されています。

SystemVerilog 言語仕様は数年に一度の改訂が行われており、IEEE Std 1800-2012 として公開されました。現在の SystemVerilog 言語仕様は、2018 年 2 月 21 に規格 IEEE Std 1800-2017 として公開されています。本書の解説は、最新仕様に準拠しています。

1.2 設計、仕様、そして検証記述言語としての SystemVerilog

SystemVerilog はハードウェア設計、仕様、及び検証を統一的に記述する為の言語です。単なる検証言語ではありません。デザインを検証する為には仕様との比較が必要になります。その仕様が検証に使用される言語と同じ言語で記述されていれば、効率的に、且つ、正確に検証を行う事が出来ます。

例えば、「req 信号が発生したら、1~3 クロック以内に ack 信号が発生しなければならない」という仕様の検証は、SystemVerilog では以下の様に記述が出来ます。

```
assert property (@(posedge clk) req ##[1:3] ack)
        else $display("@%0t: FAIL",$time);
```

これは SystemVerilog でアサーションと呼ばれる機能です。この仕様をソース・コード中に記述する事により、設計した内容が仕様に従っているかを検証する事が出来ます。しかも、仕様に従っていない設計箇所を容易に特定する事が出来ます。

```
module test(input clk,req,ack);

    default clocking cb @(posedge clk); endclocking

    sequence check_req_ack;
        req ##[1:3] ack;
    endsequence

    assert property (check_req_ack)
        else $display("@%0t: FAIL",$time);
...
endmodule
```

SystemVerilog では多くのデータ型を使用する事が出来ますが、それらは主として検証作業で使用されます。例えば、トランザクションの要素を以下の様に定義する事が出来ます。

```
typedef enum bit { READ, WRITE } direction_e;
class packet_t;
rand bit [31:0]          addr;
rand bit [31:0]          data;
rand direction_enum      direction;
rand integer             transmit_delay;
      constraint word_boundary
      {
            addr[1:0] == 2'b00;
      }
      covergroup cg;
            coverpoint  addr;
            coverpoint  data;
      endgroup
function new;
      cg = new;
endfunction
endclass
```

トランザクションのデータ項目値に対して乱数を割り当てる事が出来ます。クラスのメソッド randomize()関数を呼ぶと rand 属性の付いた上記のメンバーに乱数が割り当てられます。また、制約を与える事も出来ます。SystemVerilog では addr・バウンダリーと制限する事は容易です。上記の様に制約をクラス内に定義します。必要な時に必要なだけメソッド randomize()関数を呼ぶ事により効果的にトランザクションを作成する事が出来ます。しかも、制約を有効にする事が出来る機能を利用して、様々なパターンのトランザクションを簡単に作成する事が出来ます。

ランダムに発生した値を基にしたトランザクションを使用して検証を行う場合、その検証も自動的に行なわれなければなりません。所謂、スコアボーカー（DUT の結果と期待される結果との比較を行う機能）の概念が必要になります。SystemVerilog にはそれをサポートする為の機能があります。

更に、重複したトランザクションの作成を避け、無駄の無い様に検証を進めなければなりません。幅広い範囲のトランザクションを効果的に定義する為には、検証が既にカバーした範囲を正しく認識し、且つ未検査の範囲を確実に把握する事が必要です。所謂、スコアボード機能が必要になります。SystemVerilog の functional coverage 機能はその要望に答える機能を提供します。

上記の例では、変数、data のカバレッジを取っています。カバレッジ情報はクラスの外に定義する事も出来ます。然し、カバレッジ情報をクラス内部に定義するとカバレッジ変数（この例では cg）が自動的に確保される利点があります。その他、カバレッジ・ポイントを自由に参照する事が出来る利点があります。カバレッジは covergroup クラスのメソッド sample()関数を呼ぶ事により収集する事が出来ます。例えば、次の様にしてカバレッジ情報を集めます。

```
packet_t    packet;
...
assert(packet.randomize());
packet.cg.sample();
```

この様に、指定したタイミングでカバレッジ情報を集める事が出来ますが、トランザクションが発生した時は何時でもカバレッジを収集する事も出来ます。例えば、クラス内のメソッド post_randomize()関数にカバレッジ収集機能を追加する事が出来ます。

```
class packet_t;
...
function void post_randomize;
      cg_sampl();
endfunction
endclass
```

この関数 post_randomize()は virtual 関数ではありませんが、クラス毎に異なる定義をする事が出来ます。関数 ...domize()が呼ばれると必ず...関数も呼ばれます。

以上の例から検証に必要な...くの機能が SystemVerilog に備わっている事が分かります。

1.3 Verilog と...異

1.3.1 SystemVerilog は Verilog の superset

SystemVerilog は機能的に Verilog を含んでいます。次の図は SystemVerilog が備えている主な機能を示しています。

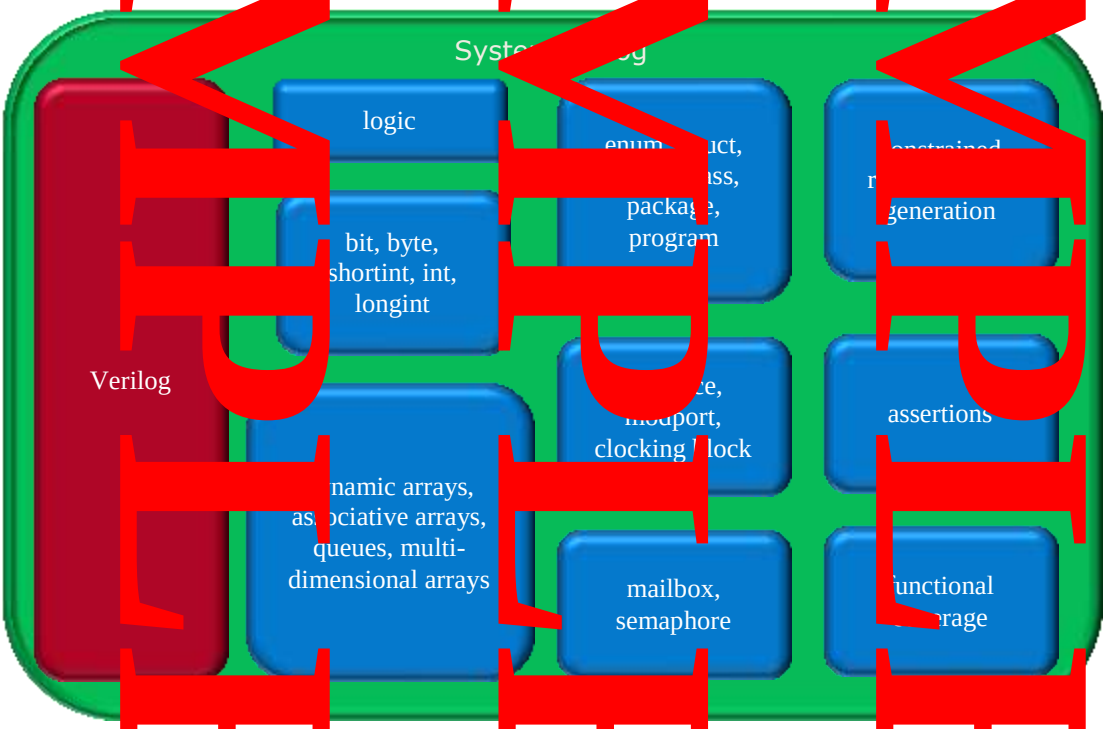

図 1-1 System...erilog と Verilog の関係

SystemVerilog はこの他にも多くの機能を含みますが、上の図では代表的な機能を記載しました。Verilog は上位互換性として SystemVerilog に含まれますが、完全に互換性がある訳ではありません。Verilog はタイミングに関して曖昧性がある為、シミュレーション結果が SystemVerilog と一致するとは限りません。例えば、次の様な機能を考えてみます。