

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

©2018, 2019 アートグラフィックス
〒124-0012 東京都葛飾区立石 8-14-1
www.artgraphics.co.jp

SystemVerilog Primer

©2018, 2019 Artgraphics. All rights reserved.
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けています。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あります。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
1.1	SYSTEMVERILOG の歴史.....	1
1.2	設計、仕様、及び検証記述言語としての SYSTEMVERILOG	1
1.3	VERILOG との主な差異.....	3
1.3.1	SystemVerilog は Verilog の superset.....	3
1.3.2	シミュレーションの実行タイミング (simulation semantics)	4
1.3.3	初期化設定と時刻 0 におけるシミュレーション・イベント	4
1.3.4	豊富なデータ・タイプ	5
1.3.5	信号値の初期値.....	6
1.4	設計及び検証の為にビルディング・ブロック	6
1.4.1	設計要素.....	6
1.4.2	module	7
1.4.3	program	7
1.4.4	interface	8
1.4.5	checker	10
1.4.6	package.....	10
1.4.7	primitive.....	11
1.4.8	configuration.....	12
1.5	ガーベッジ・コレクション (GARBAGE COLLECTION)	12
1.6	検証パッケージ	13
1.7	本書で用いるシンタックス記法.....	13
2	データ・タイプ	14
2.1	データ・タイプとデータ・オブジェクト	14
2.2	LOGIC 型	14
2.3	単数系と集合系 (SINGULAR AND AGGREGATE TYPES)	15
2.4	ネットと変数 (NETS AND VARIABLES)	15
2.5	NET 型.....	16
2.6	TWO-STATE 型	16
2.7	FOUR-STATE 型.....	17
2.8	REAL、SHORTREAL と REALTIME.....	18
2.9	VOID 型	18
2.10	CHANDLE 型	18
2.11	CLASS	18
2.12	STRING データ・タイプ	18
2.13	EVENT データ・タイプ	21
2.14	TYPEDEF.....	22
2.15	ENUM	23
2.16	CONSTANTS	26
2.17	CAST オペレータ	27
2.18	\$CAST ダイナミック型変換.....	28
2.19	便利な初期値設定	29
2.19.1	リテラルの拡張	29
2.19.2	インデックス指定.....	30
3	メンバーで構成されるデータ・タイプ (AGGREGATE DATA TYPES)	32
3.1	ストラクチャ (STRUCTURES)	32
3.1.1	Packed ストラクチャ (packed structures)	33

3.1.2	ストラクチャへの値の設定.....	33
3.2	ユニオン (UNIONS)	34
3.2.1	Packed ユニオン (packed unions)	35
3.2.2	タグ付きユニオン (tagged unions)	36
3.3	PACKED アレイと UNPACKED アレイ (PACKED AND UNPACKED ARRAYS)	36
3.3.1	Packed アレイ (packed arrays)	36
3.3.2	Unpacked アレイ (unpacked arrays)	37
3.3.3	アレイの操作 (operations on arrays)	37
3.3.4	アレイのアクセス (indexing and slicing of arrays)	39
3.4	ダイナミック・アレイ (DYNAMIC ARRAYS)	40
3.4.1	New[]コンストラクタ	40
3.4.2	size()メソッド.....	41
3.4.3	Delete()メソッド.....	42
3.5	アレイのコピー (ARRAY ASSIGNMENTS)	42
3.6	ASSOCIATIVE アレイ.....	44
3.6.1	Associative アレイの要素の登録.....	45
3.6.2	Associative アレイの関数.....	46
3.6.3	Associative アレイ・リテラル (associative array literals)	47
3.7	キュー (QUEUES)	48
3.7.1	キューの操作 (queue operators)	49
3.7.2	キューを操作する関数 (queue methods)	50
3.8	アレイ情報取得関数 (ARRAY QUERYING FUNCTIONS)	52
3.9	アレイ操作関数 (ARRAY MANIPULATION METHODS)	52
3.9.1	アレイ検索関数 (array locator methods)	52
3.9.2	アレイ要素の順序を操作する関数 (array ordering methods)	54
3.9.3	アレイを計算する関数 (array reduction methods)	55
4	クラス (CLASSES)	58
4.1	クラスの概要.....	58
4.2	クラス・オブジェクト (クラス・インスタンス)	59
4.3	クラス・プロパティ及びメソッドへのアクセス.....	59
4.4	コンストラクタ	59
4.5	タイプ指定のコンストラクタ呼び出し.....	60
4.6	STATIC クラス・プロパティ	61
4.7	STATIC クラス・メソッド	62
4.8	STATIC メンバーの参照.....	62
4.9	THIS.....	63
4.10	ハンドルのアレイ	63
4.11	クラスのコピー	64
4.12	クラスの継承とサブクラス	64
4.13	\$CAST.....	65
4.14	VIRTUAL メソッド (VIRTUAL METHODS)	65
4.15	アブストラクト・クラスとピュア・バーチャル・メソッド.....	66
4.16	INTERFACE クラス (INTERFACE CLASSES)	67
4.17	クラススコープ・オペレータ (CLASS SCOPE RESOLUTION OPERATOR ::)	68
4.18	メンバーへのアクセス制限 (LOCAL AND PROTECTED)	70
4.19	メソッドをクラスの外に記述する方法	71
4.20	パラメータによる汎用クラスの定義 (PARAMETERIZED CLASSES)	71
4.21	クラスのフォワード宣言 (TYPEDEF CLASS)	74
4.22	クラス・ハンドルの初期化	76

5	プロセス (PROCESSES)	78
5.1	シミュレーション・プロシージャ (STRUCTURED PROCEDURES)	78
5.1.1	initial プロシージャ	78
5.1.2	always プロシージャ	79
5.1.3	final プロシージャ	82
5.2	ブロック文 (BLOCK STATEMENTS)	82
5.2.1	begin-end ブロック	82
5.2.2	fork-join ブロック	83
5.2.3	ブロック名 (block names)	86
5.3	タイミングによる実行制御 (PROCEDURAL TIMING CONTROLS)	86
5.3.1	遅延による制御 (delay control)	86
5.3.2	イベント制御 (event control)	86
5.3.3	レベル・センシティブ・イベント制御 (level-sensitive event control)	87
5.3.4	シーケンスによるイベント制御	87
5.4	プロセス制御 (PROCESS CONTROL)	88
5.4.1	wait fork 文	88
5.4.2	disable fork 文	89
5.5	ユーザ固有のプロセス制御 (FINE-GRAIN PROCESS CONTROL)	90
6	代入文 (ASSIGNMENT STATEMENTS)	93
6.1	パターン指定による代入 (ASSIGNMENT PATTERNS)	93
6.2	アレイ・パターン指定代入文	94
6.3	ストラクチャ・パターン指定代入文	95
6.4	UNPACKED ARRAY の初期化	96
7	演算子と式 (OPERATORS AND EXPRESSIONS)	99
7.1	演算子	99
7.1.1	代入演算子 (assignment operators)	99
7.1.2	インクリメント及びデクリメント演算子 (++, --)	99
7.1.3	冪乗	99
7.1.4	比較演算子 (equality operators)	99
7.1.5	ワイルドカード比較演算子 (wildcard equality operators)	100
7.1.6	結合演算子 (concatenation operators)	102
7.1.7	inside オペレータ (set membership operator)	102
7.1.8	ビット・ストリーム・オペレータ (<<, >>)	103
7.2	オペランド (OPERANDS)	105
7.2.1	パートセレクト (part-select)	105
7.3	タグ付きメンバーの操作	106
8	実行文 (PROCEDURAL PROGRAMMING STATEMENTS)	108
8.1	IF 文	108
8.1.1	全ての条件を列挙 (fully specified)	108
8.1.2	unique-if、unique0-if 文	108
8.1.3	priority-if 文	109
8.2	CASE 文	109
8.2.1	unique-case、unique0-case 文	110
8.2.2	priority-case 文	110
8.2.3	inside オペレータと if 文及び case 文	111
8.3	ループ文 (LOOP STATEMENTS)	112

8.3.1	for 文	112
8.3.2	repeat 文	113
8.3.3	foreach 文	113
8.3.4	while 文	115
8.3.5	do-while 文	115
8.4	FOREVER 文	116
8.5	RETURN 文	116
8.6	BREAK 文	117
8.7	CONTINUE 文	118
9	タスクとファンクション (TASKS AND FUNCTIONS)	119
9.1	タスク	119
9.2	ポート・リスト	119
9.3	タスク内の記述	119
9.4	ファンクション	120
9.4.1	ファンクションの制限	120
9.4.2	ポート・リスト	120
9.4.3	ファンクション内でタイミング制御を行う方法	121
9.5	引数に標準値を指定する方法	122
9.6	値を戻すファンクションの使用	124
9.7	再帰呼び出し	124
9.8	クラスのメソッドと再帰呼び出し	125
9.9	メソッド内での変数の初期化	125
9.10	引数としてのアレイ	127
9.11	IMPORT 及び EXPORT	128
10	クロッキング・ブロック (CLOCKING BLOCKS)	130
10.1	最も簡単なクロッキング・ブロック	130
10.2	クロッキング・スキュー (CLOCKING SKEW)	130
10.3	サイクル・ディレイ (CYCLE DELAY)	132
11	プロセス間の同期と交信	134
11.1	セマフォ (SEMAPHORES)	134
11.2	メール・ボックス (MAILBOXES)	135
11.3	パラメータ化したメール・ボックス (PARAMETERIZED MAILBOXES)	137
11.4	名称付きイベント (NAMED EVENTS)	138
11.4.1	triggered	139
11.4.2	引数としてのイベント・オブジェクト	140
11.4.3	wait_order 文	141
11.4.4	イベント資源の解放	142
11.4.5	別名 (alias)	143
11.4.6	イベントの比較 (events comparison)	143
12	アサーション (ASSERTIONS)	145
12.1	概要	145
12.2	即時実行型アサーション (IMMEDIATE ASSERTIONS)	145
12.3	並列型アサーション (CONCURRENT ASSERTIONS)	146
12.3.1	アサーションはマルチ・スレッド	147
12.3.2	サンプリング (sampling)	147
12.3.3	アサーション・クロック	147

12.4	ブーリアン式 (BOOLEAN EXPRESSIONS)	147
12.5	シーケンス (SEQUENCES) とプロパティ (PROPERTIES)	147
12.6	シーケンスの操作 (SEQUENCE OPERATIONS)	150
12.6.1	##m	151
12.6.2	##[m:n]	151
12.6.3	[*m]	152
12.6.4	[*m:n]	154
12.6.5	[=m]	155
12.6.6	[=m:n]	156
12.6.7	[->m]	158
12.6.8	[->m:n]	158
12.6.9	sig throughout seq	159
12.6.10	seq1 within seq2	160
12.6.11	seq1 and seq2	161
12.6.12	seq1 or seq2	162
12.6.13	seq1 intersect seq2	163
12.6.14	first_match (seq)	164
12.7	シーケンスへのパラメータ	164
12.8	便利なサンプル関数	165
12.8.1	サンプル関数	165
12.8.2	レベル・センシティブ とエッジ・センシティブ	166
12.9	メソッド TRIGGERED	169
12.10	DEFAULT クロッキング	171
12.11	プロパティ	172
13	チェッカー (CHECKERS)	181
13.1	概要	181
13.2	チェッカー・インスタンス	181
14	制約によるランダム・スティムラスの生成	183
14.1	概要	183
14.2	ランダム変数 (RANDOM VARIABLES)	184
14.2.1	rand 修飾子	184
14.2.2	randc 修飾子	184
14.3	制約 (CONSTRAINTS)	185
14.3.1	inside オペレータ	186
14.3.2	dist オペレータ	188
14.3.3	unique オペレータ	189
14.3.4	implication (->)	190
14.3.5	foreach	191
14.3.6	乱数決定順序 (solve a before b)	193
14.4	乱数発生関数	194
14.5	実行時に制約を定義する方法 (IN-LINE CONSTRAINTS)	195
14.6	ランダム変数の制御	196
14.7	制約の制御	198
14.8	RANDOMIZE 関数によるランダム変数の制御	199
14.9	STD::RANDOMIZE()	200
14.10	システム関数とメソッド	200
14.11	ストラクチャ	201
14.12	条件の否定 !(EXPRESSION INSIDE { SET })	202
14.13	キューに乱数を発生	203

14.14	チェッカーとしての制約 (IN-LINE CONSTRAINT CHECKER)	204
14.15	制約をテストケース毎に指定する方法	206
14.16	RANDCASE	207
15	ファンクショナル・カバレッジ (FUNCTIONAL COVERAGE)	209
15.1	カバレッジ・モデルの定義 (COVERGROUP)	209
15.1.1	covergroup の定義	209
15.1.2	covergroup への引数	210
15.1.3	サンプリングのタイミング指定	210
15.1.4	サンプリング関数 (sample)	210
15.1.5	簡単な使用例	211
15.2	クラス内の COVERGROUP	212
15.3	カバー・ポイントの定義 (COVERAGE POINTS)	215
15.3.1	ビンの定義	215
15.3.2	カバー・ポイントの使用	217
15.3.3	信号値の遷移 (transitions)	220
15.4	クロス・カバレッジ (CROSS COVERAGE)	222
15.5	カバレッジ・オプション	226
16	システム・タスクとシステム関数	227
16.1	値変換 (CONVERSION FUNCTIONS)	227
16.2	情報取得関数	227
16.3	ビット VECTOR システム関数 (BIT VECTOR SYSTEM FUNCTIONS)	229
16.4	エラー処理タスク	230
16.5	\$SFORMAT と \$SFORMATF	231
16.6	確率分布関数	231
16.7	その他のシステム・タスク及びシステム関数	231
17	プログラム (PROGRAMS)	233
17.1	概要	233
17.2	プログラムの制御	233
17.3	シミュレーションの終了	235
18	インターフェース (INTERFACES)	236
18.1	概要	236
18.2	GENERIC インターフェースに依る接続	237
18.3	MODPORT	237
18.4	クロッキング・ブロック	237
18.5	パラメータ化したインターフェース (PARAMETERIZED INTERFACES)	237
18.6	VIRTUAL インターフェース (VIRTUAL INTERFACES)	239
18.7	クロッキング・ブロックの使用例	241
19	パッケージ (PACKAGES)	243
19.1	概要	243
19.2	パッケージ内のリソースの参照	245
19.3	パッケージをモジュール・ヘッダで IMPORT する方法	245
19.4	STD パッケージ	245
20	モジュール (MODULES)	249
20.1	概要	249

20.2	ポート・リスト	249
20.2.1	ポートの方向	249
20.3	パラメータ化したモジュール (PARAMETRIZED MODULES)	250
20.4	トップ・レベル・モジュール	251
20.5	モジュール・インスタンス	251
20.6	未定義モジュールの宣言 (EXTERN MODULES)	254
20.7	階層名称	255
21	プリプロセッサ (COMPILER DIRECTIVES)	257
21.1	`INCLUDE 文	257
21.2	`DEFINE 文	257
21.2.1	定数を定義する場合	257
21.2.2	接頭辞及び接尾辞を持つ名称の創成	259
21.3	文字列内のパラメータ展開	259
21.4	`ENDIF 文	260
21.5	`_FILE_、`_LINE_	260
22	シミュレーション実行モデル (SIMULATION SEMANTICS)	262
22.1	シミュレーション領域 (SIMULATION REGIONS)	262
22.2	#0 デイレーの効果	263
22.3	PROGRAM とクロック・ジェネレータ	264
23	参考文献	266

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 とマージされました。SystemVerilog 3.0 は第三世代の Verilog 言語と誕生しました。第一世代は広く知られている Verilog-1995、第二世代は Verilog-2001 があります。

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 はその後 SystemVerilog 2.0 及び、SystemVerilog 3.0 の改訂版を経て、2005年11月に IEEE Std 1800-2005 として正式に公開されました。これが最初の 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、PSI アサーション等の技術が使用されています。

SystemVerilog の仕様は数年に一度の改訂が行われており、IEEE Std 1800-2005 は後に公開されました。SystemVerilog 言語仕様は 2018年2月21日に規格 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 設計仕様及び検証記述言語としての SystemVerilog

SystemVerilog はハードウェア設計、仕様の記述に特化した言語として設計された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ぬ」という仕様の検証は SystemVerilog では以下の様に記述出来ます。

```
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な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);
    deassert 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
```


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;
    cov_sample;
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 `pre_randomize()`が呼ばれると必ずこの関数も呼ばれます。

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

1.3 Verilog との差異

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。

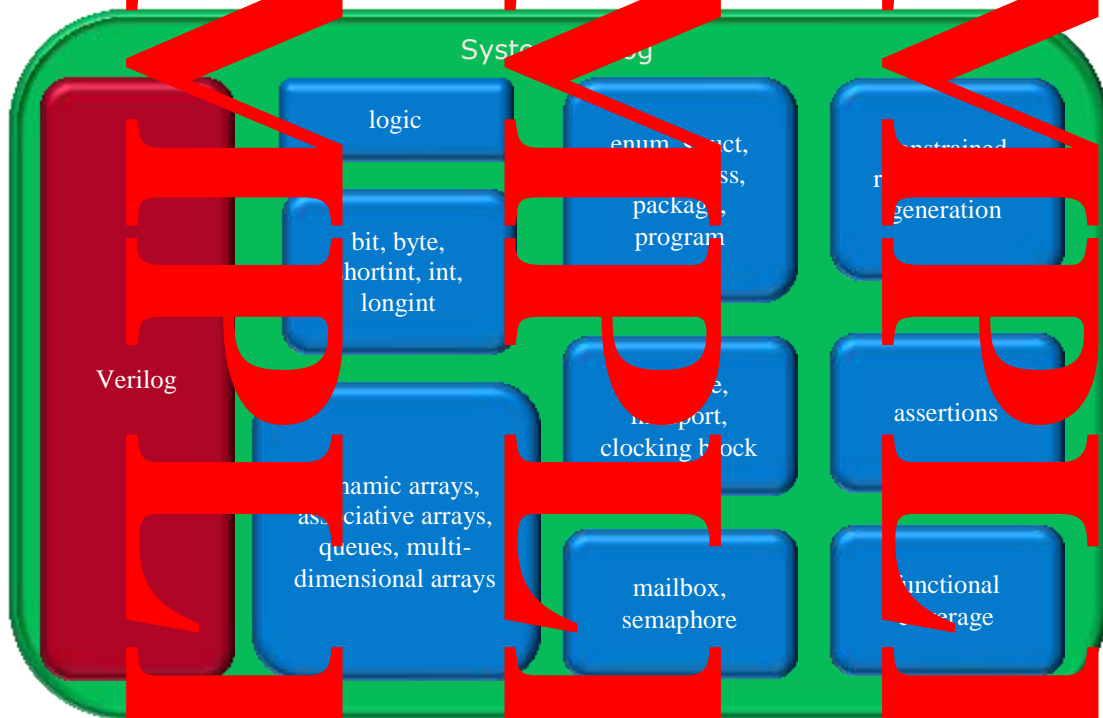


図 1-1 SystemVerilog と Verilog の関係

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。