Artgraphics

SystemVerilog 序説

ハードウェア記述言語入門

Document Identification Number: ARTG-TD-001-2026
Document Revision: 1.0, 2026.02.10
アートグラフィックス
篠塚一也

SystemVerilog 序説
ハードウェア記述言語入門

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

SystemVerilog for Beginners
Introduction to Hardware Description Languages

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

はじめに

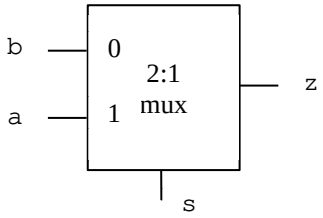
本書は、ハードウェア記述言語（HDL）に関する知識が全くない初心者向けのハードウェア記述言語の入門書です。かつては Verilog HDL や VHDL など複数の HDL が使われていましたが、現在ではその役割は SystemVerilog に集約されています[1]。本書では、HDL として SystemVerilog を仮定し、ハードウェアモデリングに必要な基礎知識と基礎技術を初心者に理解できるように丁寧かつ厳密に解説しています。

今日では、多くの人が何らかのプログラミング言語を経験しているので HDL が全く別世界の存在であるという印象はないと思います。例えば、SystemVerilog の if-then-else の構文に違和感を覚える人は皆無であろうと思います。同様に、`for` 文や `foreach` 文の構造を誤解なく類推できると思います。しかし、その構文のセマンティクスは言語により大きく異なる事に気付いている人は少ないかも知れません。

例えば、HDL を初めて学ぶ初心者にとって、次のような SystemVerilog による RTL[2]記述 (a) がハードウェアの 2:1 マルチプレクサ (b)を表現していると理解するのは簡単ではありません。

```
module mux2(input a,b,s,output logic z);
always @(a,b,s)
    if( s )
        z = a;
    else
        z = b;
endmodule
```

(a) 2:1 マルチプレクサの RTL モデリング



(b) 2:1 マルチプレクサ

ましてや、以下のような別の記述が、全く同じ 2:1 マルチプレクサ(b)を表現していると気づくのはさらに難しいかも知れません。

```
module mux2_df(input a,b,s,output z);
wire [1:0] w = {a,b};
assign z = w[s];
endmodule
```

そもそも多くの初心者にとっては、なぜ HDL がハードウェアを記述できるのか？という根本的な疑問が最初の壁になります。なぜ if-then-else が 2:1 マルチプレクサに相当するのか？なぜ配列アクセスがマルチプレクサになるのか？こうした素朴で本質的な疑問を解消しない限り、SystemVerilog を自在に使いこなす事はできません。本書は、このような「最初の疑問」に正面から答え、初心者が SystemVerilog を使って自由自在にハードウェアをモデリングできるようになる事を目的とした入門書です。

HDL はハードウェアを表現するための言語ですが、HDL の記述だけでは回路は実現しません。HDL で書かれたデザインを論理回路へ変換する 論理合成[3]（logic synthesis） が不可欠です。したがって、SystemVerilog の文法だけでなく、論理合成ツールを正しく使うための知識と技術も同時に習得する必要があります。

本書は全くの初心者を対象としているため、HDL 経験者への入門書とは異なるアプローチを採用しています。本来であれば、基礎となる用語を一つずつ説明して順に知識を積み重ねて

[1] 実際、Verilog HDL は SystemVerilog のサブセットとして位置づけられています。
[2] Register Transfer Level の略で、ゲートレベルよりも高位（より抽象的）な記述スタイルで、レジスタおよびレジスタ間の組み合わせ回路をハードウェア記述言語で記述する手法です。一般的には、レジスタ転送レベルと訳されます。
[3] 論理合成とは、RTL 記述を論理回路のネットワークに変換する EDA ツールです。

行く方法が望ましいのですが、その方法では初心者にとっては先がいつまでも見えずに時間だけが過ぎ去ってしまう結果になりがちです。寧ろ、本書の解説法は正反対の立場をとっています。すなわち、RTL モデリングを最初に提示し全体像をつかんでから学べる構成を採用しています。

第 1 章は、典型的な RTL デザインを例にとり SystemVerilog の基本機能を解説する事から始まります。このモデリング例は非常に簡単ですが、モジュール定義、組み合わせ回路、シーケンシャル回路といったデザインの基本要素を含んでいるので、実務で遭遇するデザインの縮図としての役割を果たします。このモデリングに使用されている SystemVerilog の構文や概念を丁寧に説明することで、読者は簡単な RTL モデリングであれば自身で解読できるレベルに到達します。続いて、AND、OR、NAND、NOR、XOR、XNOR といった基本回路の SystemVerilog による記述法を解説し、真理値表を用いてハーフアダーが AND と XOR の基本回路から構成されることを示します。さらに、RTL モデリングの例として、2:1 マルチプレクサと簡単な ALU を紹介し、読者を SystemVerilog によるモデリングに慣れ親しむ状態に導きます。この段階で、読者は 冒頭にあった疑問（if-then-else が 2:1 マルチプレクサを表現している理由）に答えられるようになります。また、第 1 章には SystemVerilog の歴史と概要が含まれているので、広い視野で SystemVerilog 言語を捉える事ができるようになります。

第 2 章以降では、第 1 章で得た知識を発展させ、1 ビットのデザインから任意ビット数を扱うデザインへと進化するための SystemVerilog の技術を解説します。特に、第 2 章から第 4 章では記述のビルディングブロックであるデータタイプ、次元定義の仕方、オペレータの機能を具体例と共に詳しく解説しています。続く第 5 章から第 8 章では、ハードウェアの動作を表現するために不可欠なプログラミング機能を解説しています。そして、第 9 章ではモデリング情報を共有するための手段としてのパッケージを解説して第 10 章の準備をします。第 10 章では、モジュールの書き方および階層構造の作り方を解説し、組み合わせ回路とシーケンシャル回路のモデリング法を紹介しています。シーケンシャル回路としては、FSM を例として採用しているので、読者は一段高いレベルに導かれます。そして、第 11 章と第 12 章では、汎用的なプログラミング機能の解説とソースコードを操作するための便利なコンパイラディレクティブを学び、本書を終えます。

要約すると、本書を読むことにより、読者は基本回路から 組み合わせ回路とシーケンシャル回路 の記述法まで段階的に理解できます。しかし、本書は単なる入門書ではありません。SystemVerilog の知識を「使える知識」にするため、新しい視点に基づくモデリング手法も紹介しています。これらの手法は、読者の思考力を高め、より柔軟で創造的なデザインができるようになる事を目指しています。

しかし、初心者向けの入門書であっても将来の学習に不可欠な基礎概念は省略できません。そのため本書では、やや難易度の高い内容も一部含めています。これらには ＊印 を付けてありますので、初読では飛ばしても構いません。本書全体を読み終えた後に、改めて取り組む事を勧めます。また、各章末には少量ですが練習問題があり、本文の解説補足および読者の理解を確認する役割をします。第 13 章には全ての問題に対する解答が準備されているので、できるだけ問題を解いて下さい。

アートグラフィックス
篠塚一也

変更履歴

日付	Revision	変更点
2026.02.10	1.0	初版。

目次

本書で使用する略語一覧

略語	定義
ALU	Arithmetic Logic Unit
DUT	Design Under Test、または、Device Under Test
EDA	Electronic Design Automation
FSM	Finite State Machine
HDL	Hardware Description Language
LHS	Left Hand Side
LRM	Language Reference Manual、すなわち、IEEE Std 1800-2023
LSB	Least Significant Bit
MSB	Most Significant Bit
NBA	Nonblocking Assignment
RHS	Right Hand Side
RTL	Register Transfer Level

1 概要

今日では、HDL はハードウェア設計・検証に欠かせない手段になっています。現在では HDL の役割は SystemVerilog に集約されています。したがって、ハードウェアの設計・検証に SystemVerilog を採用するのは時代の趨勢です。本章では、SystemVerilog を概説する目的を持ちますが、HDL 全体に共通して適用できる機能や性質を強調する場合には、SystemVerilog の代わりに HDL という用語を使用します。SystemVerilog の理解を深めるために、SystemVerilog 言語の歴史から解説を始めます。

1.1 SystemVerilog の歴史

SystemVerilog は、Verilog HDL（以降、Verilog とも略称）を拡張した言語ですが、実際には、その他の多く言語から影響を受けています。設計分野においては、SUPERLOG、および C、検証分野においては、SUPERLOG、VERA C、C++、VHDL、OVA、PSL 等の影響を受けています。文法的には Java の影響も受けています。

SystemVerilog 言語仕様の主要部分は SUPERLOG を基にして Accellera[4]の標準化グループにより開発され、2002 年に SystemVerilog 3.0 としてリリースされました。SystemVerilog 3.0 は、その後、Accellera による SystemVerilog 3.1、および SystemVerilog 3.1a 等の改訂版を経て、2005 年 11 月に IEEE Std 1800-2005 として正式に公開されました。これが初版の SystemVerilog 言語仕様です。SystemVerilog 言語仕様は数年に一度の改訂が行われており IEEE 規格になってからの SystemVerilog 言語仕様の改訂版の変遷は表 1-1 のようになります。本書の解説は、IEEE Std 1800-2023 に準拠しています。

表 1-1 SystemVerilog の IEEE 標準規格としての言語仕様の変遷

IEEE Std 1800-2005	このリリースの SystemVerilog は IEEE Std 1364-2005 (Verilog HDL)の拡張言語であると宣言しているだけであり、SystemVerilog が Verilog HDL を基礎言語として包含しているとは明記していません。
IEEE Std 1800-2009	SystemVerilog は IEEE Std 1364-2005 (Verilog HDL)と IEEE Std 1800-2005 を統合した言語として定義されています。
IEEE Std 1800-2012	このリリースは SystemVerilog として機能が充実していますが、望ましくない機能も追加されています。例えば、現在では非推奨となっているオペレータオーバーローディングの機能が含まれています。
IEEE Std 1800-2017	非推奨機能等を除外して非常に良く書かれた LRM です。このリリースに書かれた知識を持てば、世界で通用する技術者と言えます。
IEEE Std 1800-2023	前仕様の解説上の改善と僅かですが新機能が含まれています。ただし、非常に難解な英文で書かれています。

1.2 SystemVerilog の概要

本節では SystemVerilog の概要を機能的な全体像として解説します。そして、Verilog との機能範囲的な差異も明確にします。本節では SystemVerilog の持つ機能を列挙しますが、本書ではそれらの機能の一部だけを解説します。以下の SystemVerilog 機能概説は、本書を読了後に今後の進路を定める際に役立つと思います。

1.2.1 設計・仕様・検証言語としての SystemVerilog

SystemVerilog はハードウェア設計、仕様、および検証を統一的に記述する事ができる言語です。単なる設計言語でも検証言語でもありません。仕様記述機能はデザインと検証の橋渡し

[4] Accellera は EDA 分野で使用されるツール間の互換性を促進するために設計言語や検証ツール等の開発支援や仕様標準化を進めている NPO です。この組織で議論され吟味された標準化案は IEEE に提案されて具体的な規格となるのが一般的です。

をする役割を果たします。アサーション、およびファンクショナルカバレッジは橋渡しの良い例です。アサーションは、デザインの動作と仕様が一致する事を確認します。また、ファンクショナルカバレッジはテストの中に仕様の情報を記録して、その情報がどれだけ検証で使用されたかを計測します。

1.2.2 設計言語としての SystemVerilog

SystemVerilog には RTL 記述をより効率良く、かつ正確に行なうための機能が多く備えられています。具体的には、以下のような機能があります。

- デザインを記述するための機能（module、UDP、基本ゲート）
- 信号およびネットを表現するためのデータタイプ（logic、integer、time、bit、byte、shortint、int、longint 等）
- モジュール間、およびテストベンチと DUT[5]間の接続を簡素化するためのインターフェース機能
- 値を戻さないファンクション
- ユーザデータタイプを定義するための typedef
- C/C++の様なストラクチャとユニオンデータタイプ
- enum 型データタイプ
- パッケージ
- 便利な複合オペレータ（++、--、+=、&=、等）
- 論理合成可能性を促進する機能（always_comb、always_latch、always_ff、unique-if、priority-if、unique-case、priority-case 等）
- プログラミング機能の強化（continue、break、return 文等）

これらの機能により生産性を向上できると共に、保守性に優れたコードを製造できます。

1.2.3 検証言語としての SystemVerilog

Verilog HDL には検証機能は皆無であり、SystemVerilog には多くの検証機能、および検証に必要な機能が付け加わりました。具体的には、以下の様な機能が備えられています。

- 文字列型データタイプ（string）
- 多次元アレイ
- アレイタイプ（ダイナミックアレイ、associative アレイ、キュー）
- クラス（class）
- インターフェース（interface）
- パッケージ（package）
- テストベンチ機能（program）
- プロセス間通信機能（event 機能の拡張、mailbox、semaphore 等）
- プロセス制御機能（fork-join、fork-join_any、fork-join_none）
- クロック信号に同期する信号の管理（clocking block）
- 信号変化の状態取得（`$rose`、`$fell` 等）
- ランダムスティミュラス生成機能
- ファンクショナルカバレッジ（functional coverage）
- アサーション（assertions）

これらの拡張機能は検証作業の生産性を向上し、検証技術の蓄積に大きな役割を果たします。特に、クラスは再利用可能な検証環境を構築するために必要不可欠な機能です。

1.3 シミュレーションと論理合成

シミュレーションは、一般的には、機能的な検証やタイミングの検証を行う過程です。しかし、RTL ではクロック信号に同期してデザインが機能的に正しく動作する事を確認します。シミュレーションはイベントドリブン方式を採用し、イベントが発生する事により論理のシミュレーションが進行していきます。ここで、イベントとは、信号値が変化する状態を意味

[5] テスト対象デザイン、または、テスト対象デバイスを指します。

します。SystemVerilog では、信号が単純に変化する状態と特別な値に変化する状態を区別して表現します。例えば、イベント待ちを@(a)と書くと a の信号値が変化するとイベント待ちが解除されます。一方、@(posedge clk)と書くと、clk が特別な変化（この場合には、クロックの立ち上がり）をするとイベント待ちが解除されます。

シミュレーションにより機能的に正しく動作する事が確認された RTL デザインは、論理合成により論理回路のネットワークに変換されます。その後、物理設計の過程を得て回路の配置と配線接続が決定されます。最終的にはウェハ上に集積回路として実装されます。

1.4 SystemVerilog によるデザイン記述法の概説

本書は設計分野で必要とされる SystemVerilog の機能を解説する目的を持ちますが、それらの機能を解説するにはデザインを主体とした記述例を示すのが最も有効な方法です。しかし、SystemVerilog の知識を持たない読者にとって見慣れない構文やまだ学習していないキーワードが説明に出現してくると説明を十分に理解できない可能性があります。それでは本書の目的を達成するには程遠くなるので、SystemVerilog の基礎知識を持たない初心者にも理解し易くなるように SystemVerilog による記述法を概説する事から始めます。それぞれの機能に関する詳しい解説は第 2 章以降で行われますが、本節では次節以降で解説する記述例を理解するために充分な予備知識を提供します。

1.4.1 デザイン構造の概要

SystemVerilog によるデザインの対象は組み合わせ回路（combinational circuits）およびシーケンシャル回路（sequential circuits）です。デザインの記述は、キーワード module で始まりキーワード endmodule で終了します。それらのキーワードの間に以下の仕様を記述します。

- モジュール名称
- パッケージの参照
- パラメータの定義
- ポートリスト
- ネットおよび変数の定義
- 連続代入文（式の記述による組み合わせ回路の記述）
- 組み合わせ回路の記述（always、always_comb）
- シーケンシャル回路の記述（always、always_ff）

パッケージの参照とパラメータの定義を除く上記の機能を使用した簡単なデザインを以下に紹介します。このような記述法は RTL モデリング（RTL Modeling）と呼ばれます。このデザインは simple_design と呼ばれ以下のような仕様を持ちます。

---

simple_design の仕様

- simple_deign は、a、b、c、sel、clk を入力信号、q を出力信号として持つシーケンシャル回路です。ここで、clk はクロック信号です。何れの信号も 1 ビットです。
- シーケンシャル回路は組み合わせ回路とエッジトリガなフリップフロップから構成されます。
- 組み合わせ回路はフリップフロップの入力となる信号値を生成します。
- クロッキングイベント（posedge clk）時に、組み合わせ回路が生成した信号値からフリップフロップの出力 q の値が生成されます。

例 1-1　SystemVerilog によるデザイン例

以下の記述例（simple_design）には組み合わせ回路とシーケンシャル回路（フリップフロップの記述）が含まれているので、簡単な記述ですが多くの SystemVerilog 機能を紹介しています。

```
module simple_design(input clk,a,b,c,sel,output logic q);
logic  tmp;

assign w = a&b;

always @(sel,c,w)
    if( sel )
        tmp = c;
    else
        tmp = w;

always @(posedge clk)
    q <= tmp;

endmodule
```

ポートリスト

組み合わせ回路の記述

シーケンシャル回路の記述
（フリップフロップの記述）

この SystemVerilog 記述は図 1-1 に示すような回路に合成されます。

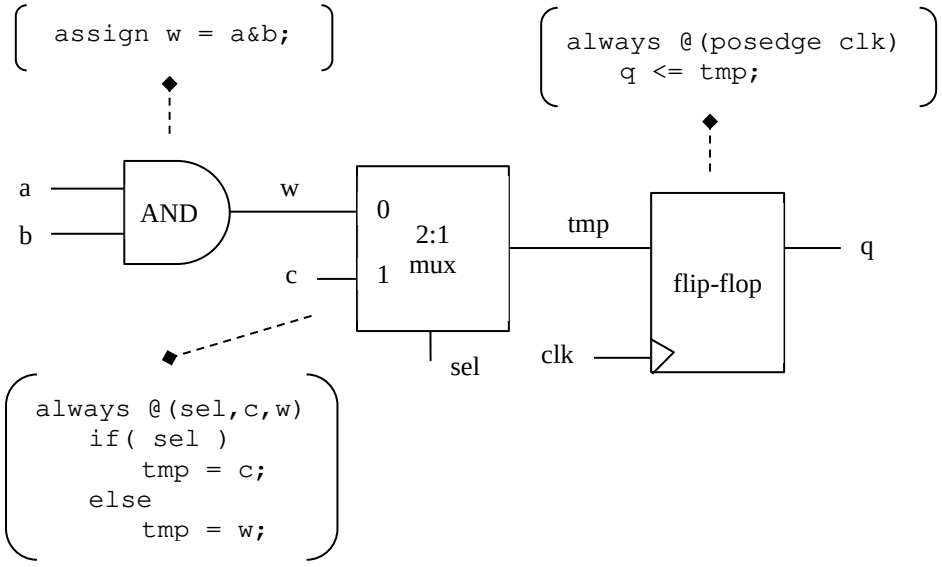


図 1-1　simple_design の回路構成

■

参考 1-1

always @(sel,c,w) で記述されているプロシージャから 2:1 マルチプレクサが合成される仕組みは、1.7.2.2 で詳しく解説されます。
□

次に、例 1-1 で使用されている SystemVerilog の機能を順に解説します。

1.4.2　モジュール名称

キーワード module に続いてモジュール名称を指定します。以下の記述における simple_design はモジュール名称でデザイン名称としてユーザが自由に命名できます。

```
module simple_design(input clk,a,b,c,sel,output logic q);
```

自由に名称を指定できますが、SystemVerilog のルール（1.9.4 項）に従わなければなりません。つまり、モジュール名称は、英文字、数字、$、アンダースコア（_）から構成されます。ただし、最初の文字は英文字またはアンダースコアでなければなりません。

1.4.3 ポートリスト

例 1-1 では指定されていませんが、モジュール名称の後にパッケージの参照とパラメータの定義を指定できます。その後に、（）で囲まれたポートリストが続きます。以下の例にはポートリストが指定されています。

```
module simple_design(input clk,a,b,c,sel,output logic q);
```

キーワード input は入力ポートを指定するために使用され、output は出力ポートを指定するために使用されます。SystemVerilog には、この他に inout ポートを指定する機能がありますが、ここでは使用されていません。input、inout、output はポートの方向と呼ばれます。ポートリストの宣言効果は表 1-2 のようになります。

表 1-2 simple_design のポートリスト

宣言	ポートの種類	意味
input clk,a,b,c,sel	ネット	• 入力ポート（clk、a、b、c、sel）を宣言しています。何れのポートも 1 ビットです。 • clk の後には input が指定されていませんが、他のポートに対しても input が有効になっています。 • input に対してデータタイプが指定されていませんが、データタイプとして logic が仮定されます。つまり、全ての入力ポートは logic 型のネットになっています。
output logic q	変数	• 出力ポート（q）を宣言しています。ポートは 1 ビットになります。 • output が指定されているので、それ以前の input の指定が output に切り替わります。 • output にデータタイプ（この場合には logic）が指定されると変数になります。

これらのポート定義を基にすると図 1-2 のようなブロックダイアグラムを得ます。

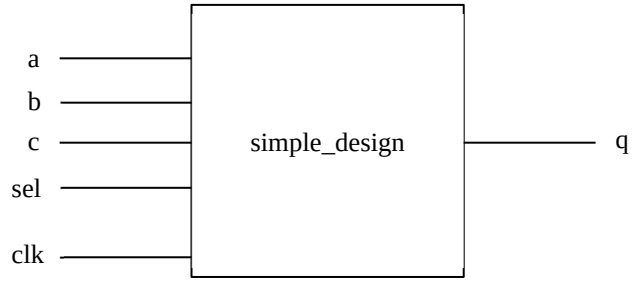


図 1-2 simple_design のブロックダイアグラム

参考 1-2

ポートがネットであるか変数であるかに従い使用制限が異なります。always プロシージャ内では、代入文の左辺に指定できる信号は変数に限られます。ネットと変数に関しては、2.3 節で詳しく解説されます。

□

参考 1-3

モジュールのポートはグローバルです。つまり、モジュールの外からも参照できます。外からも参照できるので、論理合成はポート名称を変えずにそのまま合成するのが一般的です。
□

1.4.4 ネットおよび変数の定義

モジュール内でのみ使用する信号があれば、ポートを宣言した後に定義する必要があります。simple_design では、変数 tmp を以下のように定義しています。この定義により、tmp は 1 ビットの変数となります。

```
logic  tmp;
```

参考 1-4

tmp はモジュール内でローカルに定義された変数なので、一般的には、モジュールの外から直接参照する事ができません。したがって、論理合成は名称を保存する義務がないため、tmp という名称は論理合成結果に反映されない可能性があります。
□

1.4.5 連続代入文

simple_design では、組み合わせ回路を記述するために代入文を以下のように使用しています。キーワード assign を使用して書かれた代入文は連続代入文（continuous assignments）と呼ばれます。連続代入文は always プロシージャの外に存在しなければなりません。

```
assign w = a&b;
```

この代入文は、a と b のビット演算 AND（&）を記述しているので、図 1-1 の AND 回路に合成されます。そして、この連続代入文は以下のような機能を持ちます。

① 2 つの信号 a、b の何れかの信号値が変化するまで待ちます。
② 2 つの信号 a、b の何れかの信号値が変化すると右辺が評価されます。
③ 評価された結果が以前の式の値と異なれば、新しい値が左辺の w に設定されます。その後、①に戻り同じ処理を繰り返します。
④ もし評価結果が以前の値と同じであれば、①に戻り同じ処理を繰り返します。この場合には、w に変化が起こりません。

参考 1-5

右辺に現れている信号a、b の変化か起こらない限り右辺の評価が発生しない事に注意して下さい。このような処理方式はイベントドリブンと呼ばれます。
□

参考 1-6

信号 w はどこにも宣言されずに使用されている事に注意して下さい。SystemVerilog では、1 ビットのネットであれば、宣言せずに使用できるルールがあります。w にはそのルールが適用されています。
□