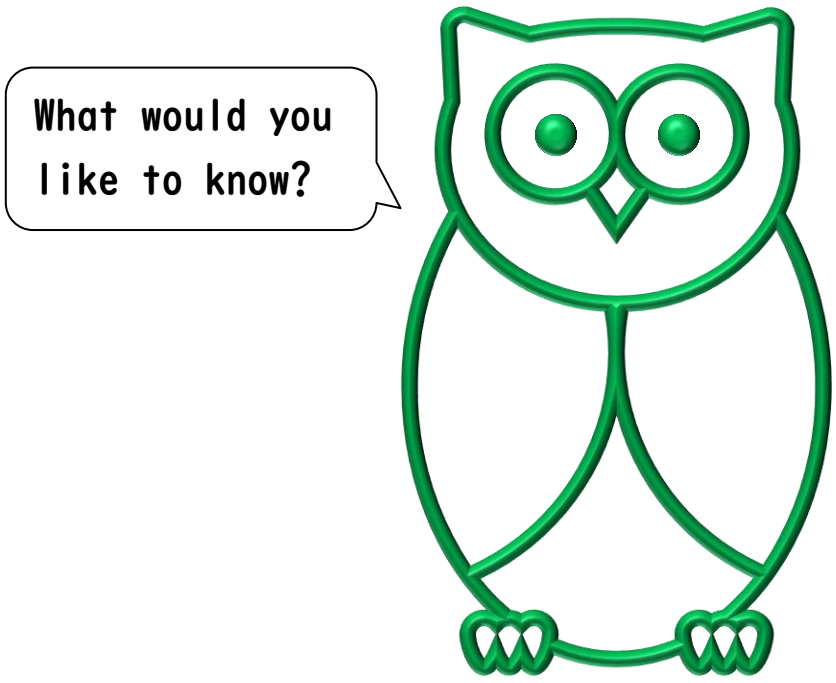
Artgraphics

SystemVerilog 豆知識

Document Revision: 1.0, 2024.04.15

アートグラフィックス
篠塚一也



SystemVerilog 豆知識

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

SystemVerilog Tips

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

はじめに

本書は、SystemVerilog の使用に関する耳よりな話を知識としてまとめた技術資料です。普段、何気なく使用している機能にも裏付けする理由があって正しい使用法となっていますが、いざ使用上での問題が発生すると解決の糸口さえつかめないという事が良くあります。本書は、仮にそのような状況に陥っても冷静に問題への対処策を検討できるように分析力を高めるための知識を提供します。また、本書は市販されている書物には書ききれない情報を補足する意味も持ちます。

例えば、以下のような変数宣言が与えられたとして、それらの使用法を以下に考察してみます。ここで、ushort_t は 16 ビット符号なし整数のデータタイプを意味します。

```
typedef logic [15:0] ushort_t;
ushort_t  a;
string    s;
```

先ず、次のような使用法をしても疑問を持つ事はありません。しかし、integral 型の変数 a に、何故、文字列を割り当てる事ができるかを知らない人も多い筈です。

```
a = "JP";
s = "JP";
```

そもそも、integral 型の変数 a に string 型の値を割り当てる事はできません。例えば、以下のように string 型の変数から integral 型への割り当てをするのは違法です。

```
a = s; // illegal
```

正しくは、以下のようにしなければなりません。

```
a = ushort_t'(s);
```

同様に、以下のような代入をする事はできません。

```
s = a; // illegal
```

正しくは、以下のようにしなければなりません。

```
s = string'(a);
```

しかし、文字列"JP"の場合には、この種の問題は起こりませんでした。そうすると、文字列には特別な性質が隠されている事になりますが、巻末の参考書を除き、その性質を解明している資料は極めて稀です。本書は、その稀な存在です。

もう一つ別の例を挙げてみます。例えば、以下のように変数 b が宣言されていると仮定します。すると、b に対して b[7:0]のような表現が可能ですが、b と b[7:0]の機能上の差異が明確でなければなりません。実は、両者の機能は全く異なります。

```
logic signed [7:0]  b;
```

もし、その差異を知らなければ b[7:0]の代わりに b と書き換える事は明らかで、本来の動作と異なる結果を得てしまいます。しかも、結果が異なる理由の解明には多大の時間が費やされます。

以上紹介した例で想像がつくと思いますが、何の疑問を持つことなく使用している機能にも理由があり、たまたま、正しい使用法になっている場合があれば、意図しない動作になっている場合も多々ある事が理解できたと思います。本書は、多くの人が頻繁に使用する機能で、しかも、深い理解が必要である機能に関してできるだけ詳しい解説を試みています。

内容的には SystemVerilog LRM の重要な機能の解説、誤解し易い機能に関する補足的な説明、および見落としやすい機能の解説に焦点を当てた意味のある資料であり、単なる雑談ではありません。本書は、実務者にとっての有益な技術資料です。なお、本書は徐々に内容を追加していくため、最初から完成した資料ではありません。

アートグラフィックス
篠塚一也

目次

1 字句ルール

1.1 整数型リテラル

サイズを指定しない整数型リテラルは 32 ビットとは限りません。符号付きのリテラルであれば、符号分のビットも確保されます。例えば、4294967296 (2^{32})は少なくとも 34 ビットで表現されます。

1.2 string リテラル

`string` リテラルは `string` 型ではなく、符号なし整数です。厳密に言えば、以下のようになります。

`string` リテラルは、文字列として表現されますが、機能的には、符号なしの `packed` ベクターです。一文字は 8 ビットで表現されるのでリテラルのビット長は 8 の倍数となります。そして、空文字列 "" は、`8'b0` として扱われます。

---

2 データタイプ

2.1 packed アレイ

SystemVerilog の packed アレイは、たとえ何次元として定義されていても、アレイ名称を単独で参照すると、一次元 packed アレイとして扱われます。

2.2 整数型と packed アレイ

SystemVerilog の整数型（byte, shortint, int, longint, integer, time）は、packed アレイではありませんが、packed アレイのように使用できます。

```
byte v1;
v1 = -2;
$display("v1 = %0d, v1[1:0] = 'b%b, v1[7:0] = %0d",
          v1, v1[1:0],v1[7:0]);
```

このように、v1[1:0] のようなパートセレクトや v1[4] のようなビットセレクトを使用できます。ただし、これらの表現は符号なしなので、v1[7:0] は 254 であり-2 ではありません。

2.3 ネット

SystemVerilog のネットは trireg を除き、値を保持する機能はありません。したがって、ネットは常にドライバーに接続されていなければなりません。例えば、ネットの宣言時に初期化をしても連続代入文（assign 文）に変換されます。

2.4 event

SystemVerilog のデータタイプは、一般に、値を持ちますが、値を持たないデータタイプも存在します。例えば、event は値を持ちません。しかし、一時的に値を持たせる事ができます。それが、triggered メソッドの機能です。値を持てば、レベルセンシティブなイベント制御に使用できます。つまり、triggered メソッドはエッジセンシティブからレベルセンシティブに変換する機能を持ちます。

3 メンバーで構成されるデータタイプ

3.1 アレイの初期化

データタイプの標準値以外でアレイ要素を初期化するには、キーワード default を使用すると便利です。

```
string    a[3] = '{ default:"red" };
int       v[5];
initial begin
   v = '{ default:10 };
end
```

3.2 packed アレイと unpacked アレイ

SystemVerilog の packed アレイと unpacked アレイは、大局的にはアレイであり同じ概念です。例えば、何れにも foreach を使用できます。根本的な差異は、データの格納法にあります。SystemVerilog では、packed アレイは右詰め、unpacked アレイは左詰めです。

packed アレイと unpacked アレイの差異

アレイの種別	格納法	動作例
packed アレイ	右詰め	logic [31:0] a; a = 16'habcd; // 32'h0000**abcd**

unpacked アレイ	左詰め	int src[3], dest1[], dest2[];
		src = '{2, 3, 4};
		dest1 = new[2](src); // dest1 = {**2**, **3**}
		dest2 = new[4](src); // dest2 = {**2**, **3**, **4**, 0}

4 オペレータと式

4.1 オペランド

SystemVerilog では、式の型はオペランドの型で決定されるので、予期しない結果が得られる事があります。例えば、`2.0**-1` は `0.5` ですが、`2**-1` は `0` になります。

`2.0**-1` は実数型なので `0.5` となりますが、**`2**-1`** は整数型なので切り捨てられて `0` となります。

4.2 {...}

結合オペレータ{...}はビットの `packed` ベクターとして扱われます。したがって以下のような記述をできます。

```
byte a, b ;
bit [1:0] c ;
c = {a + b}[1:0]; // 2 LSBs of sum of a and b
```

4.3 繰り返しを伴う {}

繰り返しを伴う {} のオペランドを評価する際、評価は一度だけ行われます。以下の場合、`f(w)`は一度だけしか評価されません。

```
result = {4{f(w)}};
```

4.4 オペレータの計算精度

`expr ? a : b` では、`a` と `b` のビット数の大きい方で決定されます。なお、`expr` は常に 1 ビットになる事に注意して下さい。

5　代入文

5.1　実行領域

以下の表現によると、a の値がどのように変化するかわかりません。

```
initial a = 3;
initial a = 2;
initial a = 1;
```

意図した通りに動作させるためには、代入の仕方を厳密に指定しなければなりません。以下のようにすれば、a は 1→2→3 の順に変化します。

```
initial a <= 3;      // NBA 領域
initial #0 a = 2;    // Inactive 領域
initial a = 1;       // Active 領域
```

参考 5-1

SystemVerilog の実行領域を学習過程の初期段階で正確に理解しておく必要があります。
□

6　プロセスとタイミング

6.1　final

initial や always プロシーシャと異なり、final プロシーシャは個別のプロセスとして実行しません。単なる function のようにシミュレータから呼び出され、シミュレーション時間を消費せずに戻ります。

6.2　always_comb と always_latch

always_comb や always_latch の内部でタイミング制御を伴わないタスクを呼び出す事ができますが、タスクの内部はデータフロー解析から除外されるので、タスクの内部で使用されている信号はセンシティビティリストに反映されません。

6.3　クロッキングブロックイベント

クロッキングブロックイベントはクロッキングイベントとは異なります。下記の、@(posedge clk)は Active 領域で解除されますが、@cb は Observed 領域で解除されます。

```
module test;
...
logic  clk;
clocking cb @(posedge clk); endclocking
always @(posedge clk) ...
always @cb ...
endmodule
```

7　クラス

7.1　メソッドと:extends

ベースクラスの virtual メソッドを拡張する際、引数リストが一致しないとサブクラスでは vaitual メソッドではななります。そのような致命的な間違いを防ぐために、サブクラスのメソッドには :extends を付けると良いです。

virtual メソッドを書き換えてしまう間違い

ベースクラス	サブクラス
class base_t; virtual function void print(**string** msg); ... endfunction endclass	class sub_t extends base_t; function void print(**int** msg); ... endfunction endclass

サブクラスの print()メソッドは間違いではありませんが、間違いをしている可能性はあります。上記のように print()メソッドを定義すると、ベースクラスのメソッドの引数リストと異なるため、サブクラス独自のメソッドとなり、virtual メソッドではなくなります。したがって、故意に別のメソッドとして定義しているのであれば、間違いではありません。しかし、ベースクラスの print()メソッドをサブクラス用に拡張する意図がある場合には、間違いとなります。このような間違いを未然に防ぐためには、以下のように:extends を使用できます。

```
function :extends void print(int msg);  // エラーが出ます
```

8　タスクとファンクション

8.1　サブルーティン呼び出しと実引数の評価順序

SystemVerilog では、サブルーティンの引数が `input` であれば、実引数には任意の式を指定できますが、`input` タイプの引数が複数個ある場合、それらの式を評価する順序を LRM は定めていません。例えば、サブルーチン呼び出し `f(a,b,c,z)` において、`a`、`b`、`c` が入力ポートに対応する場合、`a`、`b`、`c` を評価する順序はシミュレータにより異なります。

9 ファンクショナルカバレッジ

9.1 カバレッジビンの定義

SystemVerilog のファンクショナルカバレッジでは、制約の付いたランダム変数のカバレッジ計算をする場合、状況に応じたカバレッジビンの定義をすると良いです。例えば、以下の記述例では、カバーポイント a にビン定義がないため auto[0]~auto[7]の 8 個のビンが自動生成されますが、auto[6]と auto[7]はカバーされないので、最大 75%のカバレッジしか達成できません。

```
class sample_t;
rand bit [2:0]   a;

constraint C { a inside { [0:5] }; }

covergroup cg;
   coverpoint a;
endgroup

function new;
   cg = new;
endfunction
endclass
```

ランダム変数 a には制約が定義されている

カバーポイント a にはカバレッジビンが定義されていないので、100%カバレッジを達成できない

9.2 カバーポイント

SystemVerilog のオペレーションの演算精度は左辺を含むオペランドの精度で決定されるので、十分な計算精度で結果を得られます。しかし、式のカバレッジ計算には左辺が存在しないため計算精度の間違いに陥り易い傾向があります。

以下のカバーポイント(a+b)は左辺を持たないため、演算精度は 1 ビットです。したがって、(a+b)が 0 と 1 になる場合しか情報の収集がされません。つまり、auto[0]と auto[1]の二つのビンしか生成されません。

```
class simple_item_t;
rand logic    a, b;
bit           coverage_enabled;
covergroup cg;
   a_plus_b: coverpoint (a+b) iff (coverage_enabled);
endgroup
...
endclass
```

1 ビットの精度で計算されるので正しくない

ちなみに、上記の問題点を以下のように解消できます。

```
covergroup cg;
   a_plus_b: coverpoint (a+b+2'b0) iff (coverage_enabled)
   { bins fc_a_plus_b[] = {[0:2]}; }
endgroup
```

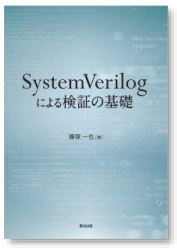
10　システムタスクとファンクション

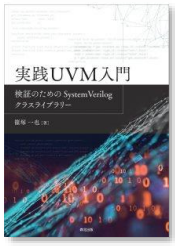
10.1　$stacktrace

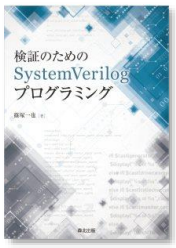
検証コードで異常が起きると原因を特定するのが難しい場合があります。そのような時に `$stacktrace` を使用すると便利です。`$stacktrace` は、コールスタックをを収集する機能を持ちますが、タスク版とファンクション版があるので柔軟性のある使用法ができます。

11　参考文献

かつての Verilog HDL の時代と異なり、近年ではチップの高集積度と高性能化の傾向と相まって設計および検証段階で厳密な追及が不可欠になりました。正に、『生兵法は大怪我のもと』とならないように、SystemVerilog に関する正しい知識を習得する必要があります。文献[3-7]は純正の SystemVerilog 参考書です。SystemVerilog が備えている機能を正しく使用するために有益な書物です。







[1] IEEE Std 1800-2017: IEEE Standard for SystemVerilog – Unified Hardware Design, Specification and Verification Language.
[2] IEEE Std 1800-2023: IEEE Standard for SystemVerilog – Unified Hardware Design, Specification and Verification Language.
[3] 篠塚一也、SystemVerilog 超入門、共立出版 2023.
[4] 篠塚一也、SystemVerilog 入門、共立出版 2020.
[5] 篠塚一也、SystemVerilog による検証の基礎、森北出版 2020.
[6] 篠塚一也、実践 UVM 入門、森北出版 2021.
[7] 篠塚一也、検証のための SystemVerilog プログラミング、森北出版 2022.
[8] SystemVerilog IEEE Std 1800-2023 の要約、アートグラフィックス 2024.