Artgraphics

SystemVerilog IEEE Std 1800-2023 の概要

Document Identification Number: ARTG-TD-003-2024
Document Revision: 1.3, 2024.12.12
アートグラフィックス
篠塚一也

SystemVerilog IEEE Std 1800-2023 の概要

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

SystemVerilog IEEE Std 1800-2023 Summary

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

はじめに

SystemVerilog 言語仕様の改訂版が，2024 年 2 月 28 日に IEEE Std 1800-2023（以下、改訂版または LRM と略称）として公開されました。本資料は、前仕様（IEEE Std 1800-2017）との差異を要約した資料です。本資料には改訂版で追加された機能の解説も含まれています。

多くの人が先ず疑問に思うのは、「改訂版と前仕様との言語上の差異」だと思います。従来と同様に、今回の改訂版は以下のような改良を含んでいます。

- 前仕様書における誤りの訂正
- 誤解し易い仕様に対する補足説明
- 拡張された機能と追加された機能の説明

従来と異なり、今回の改訂版には機能的な拡張と追加は僅かしか含まれていません。この点において、今回の改訂が既存のユーザに大きな影響をもたらす事が無いのは確実です。ただし、開発するコードの品質を高めるための便利な機能が追加されているので、今後は開発手法が大きく変化していく可能性があります。早い段階で新機能を実務に活用できるように努力が必要です。例えば、以下の記述は UVM を使用したドライバーの定義ですが、前仕様にはなかった機能が使用されています。

```
class simple_driver_t extends uvm_driver #(simple_item_t);
virtual simple_if   vif;

`uvm_component_utils(simple_driver_t)

function new(default);
   super.new(default);
endfunction

extern function :extends void build_phase(uvm_phase phase);
extern task :extends run_phase(uvm_phase phase);
extern task :initial get_and_drive();
extern task :initial drive_dut(input simple_item_t item);
endclass
```

`:initial` が追加されていますが、ベースクラスの virtual メソッドを不注意に書き換えてしまう間違いを未然に防ぐ役割を果たしています。また、`:extends` はベースクラスに同名の virtual メソッドが同じ仕様で定義されている事を確認する働きをしています。また、サブクラスのコンストラクタに `default` を使用する事により、ベースクラスの引数リストを簡略表現できます。この指定により、ベースクラスの引数に変更が発生してもサブクラスのコンストラクタには影響が及びません。

しかし、一方、どのような機能が拡張され、または追加されたかを知るためには改訂版の隅から隅までを注意深く読まなければなりません。膨大な LRM のうち僅か十数ページのために多大の努力を費やさなければならないのは苦痛であると共に生産的であるとは言えません。本資料は、そのような負担を軽減するために準備されました。十分な時間が無いにもかかわらず業務を遂行する上で一通り LRM に目を通さなければならない実務者に最適な資料です。以下の質問に答えられない場合には、LRM または本資料を読むべきです。

- `soft packed` ユニオンとは何ですか？また、なぜ必要なのですか？
- `type(this)` とは何ですか？また、使用する利点は何ですか？
- `foreach` を使用せずにアレイ要素のイタレータを定義するにはどうすれば良いですか？
- `foreach` を使用せずにアレイ要素のインデックスイタレータを定義するにはどうすれば良いですか？
- アレイの `map` 機能とは何ですか？

- クラスが拡張されるのを防ぐにはどうすれば良いですか？
- `super.new(default)`とは何ですか？
- `super.new(default)`を使用する利点は何ですか？
- virtual メソッドを間違いなく拡張するにはどうすれば良いですか？
- サブクラスで定義するメソッドが、ベースクラスで virtual メソッドとして定義されていない事を確認するにはどうすれば良いですか？
- `[ expression +/- expression ]`の機能は何ですか？
- `[ expression +%- expression ]`の機能は何ですか？
- ベースクラスで定義されているカバーグループをサブクラスで拡張するには、どうすれば良いですか？
- 制約に`:initial`、`:final`、`:extends` を付けるとどうなりますか？
- 弱い参照と何ですか？弱い参照はなぜ必要なのですか？

前仕様書の誤りの訂正に関しては、改訂版と比較しなければ相違を見出す事ができませんが、訂正された内容で注目に値する内容に関しては本資料で詳しく解説しています。したがって、本資料を精読する事は、前仕様書の曖昧性から生じる誤解を解消する機会となります。例えば、改訂版では、クロッキングイベントとクロッキングブロックイベントの相違を明確にしていますが、本資料でもその意図をくみ取り念入りな解説を提供しています。

誤解し易い機能の説明に関しても同様に、前仕様書と比較しながら読まなければわからないのですが、本資料では、どのような説明が補足されているかを分かり易く報告しています。

機能の拡張と新機能の追加に関しては、LRM を熟読する事により理解する事ができますが、本資料でも機能概要を解説しています。したがって、本資料を精読すれば、前仕様を理解している読者は IEEE Std 1800-2023 の全容を把握する事ができます。

本資料では、特に顕著な差異を中心に纏めてあります。意味上の変化が認められない差異に関しては報告していません。例えば、IEEE Std 1800-2023 では太文字に変化している表現がありますが差異として報告をしていません。また、改訂版の Part 3 と Part 4 は調査対象外です。

最後に、調査結果が完全である保証はありませんが、殆どの重要な差異を網羅していると確信しています。なお、細心の注意を払い調査をしましたが、解説が不十分な仕様があるかも知れません。本資料で理解し難い説明があれば、LRM を紐解いて下さい。

アートグラフィックス
篠塚一也

変更履歴

日付	Revision	変更点
2024.03.17	1.0	初版。
2024.03.27	1.1	写像機能の解説を補足。
2024.07.07	1.2	全体的な校正を実施。
2024.12.12	1.3	第 4 章を校正。

目次

1　概要

本資料では、IEEE Std 1800-2023 を改訂版、または LRM と略称します。また、IEEE Std 1800-2017 を前仕様と略称します。

1.1　改訂版の意義

冒頭でも述べましたが、機能的な拡張は僅かであり、ユーザに影響を及ぼす変更は見当たりません。しかし、ユーザが誤解している可能性のある機能に関しては、多くの追加説明が提供されているので、改訂版は現ユーザにとっても有用な資料となります。

例えば、以下の記述おいて、`@(posedge phi1)`と`@(dram)`は全く同じではないという事が力説されています。

```
module foo (input phi1, input [7:0] data);
clocking dram @(posedge phi1);
input data;
output negedge #1 address;
endclocking
always @(posedge phi1) $display("clocking event");
always @(dram) $display("clocking block event");
endmodule
```

`@(posedge phi1)`と`@(dram)`が異なる機能を持つ事は前仕様にも書いてあるのですが、前仕様の他のページでは二つの記述が同値であるとも説明しています。したがって、誤解を招く説明が散乱しているために改訂版では厳密な説明に統一されています。

なお、機能的な拡張は僅かではありますが存在するので、本資料では拡張された機能をできるだけ詳しく解説をしています。ただし、最終的には LRM で機能を確認する事をすすめます。

1.2　変更・改善・追加点のまとめ

今回の改訂版により変更・改善・追加された仕様を以下にまとめます（表1-1）。勿論、これ以外にも変更されている説明がありますが、見易さのために顕著な変更点だけを表にまとめました。

表 1-1　改訂事項の概要

対象機能	特記事項
string リテラル	三重引用符文字列（"""..."""）が追加されました。三重引用符内では、改行や引用符（"）をエスケープせずに使えます。
{multiplier{Str1,Str2,...,Strn}}	少なくとも一つの Stri が必要です。Stri は string リテラルか、string 型の式です。
type(this)	現在のスコープのクラスの型を参照できるようになりました。
union	union に soft packed が追加されています。 struct_union ::= struct | union [soft | tagged]
array manipulation	index_argument のオプションが追加されました。アレイを with 条件で操作する際に、アレイ要素のインデックス名を指定できます。
iterator index querying	メソッド名が変更されました。 function int_or_index_type index_method_name (int dimension = 1);

array map() method	unpacked アレイを作る機能を持ちます。with で条件を指定できます。
クラスと:final	クラスを拡張できないようにするための指定が追加されました。
コンストラクタと default	サブクラスのコンストラクタでは、ベースクラスのコンストラクタの引数リストを default で参照できるようになりました。
メソッドと:initial	virtual メソッドを保護する仕組みが追加されました。サブクラスで定義するメソッドに:initial を付けておくと、そのメソッドがベースクラスで virtual に宣言されていればエラーになります。つまり、virtual メソッドを non-virtual メソッドに書き換える間違いを防ぐための手段です。
メソッドと:extends	サブクラスでベースクラスの virtual メソッドを再定義するために使用できるオプションです。メソッドに:extends を添えると、そのメソッドはベースクラスの virtual メソッドを再定義する意味を持ちます。したがって、ベースクラスに定義されているメソッドが virtual でなければエラーになります。また、:extends で指定されるメソッドは virtual なので、キーワード virtual を省略できます。
メソッドと:final	サブクラスでメソッドが再定義されないようにするために、:final オプションが追加されました。:final が付けられているメソッドをサブクラスで再定義するとエラーになります。
weak reference	ガーベッジコレクションの機能の一部として追加された。
inside	シンタックスが拡張されました。 inside_expression ::= expression inside { range_list } range_list ::= value_range { , value_range } value_range ::= expression | [expression : expression] | [$: expression] | [expression : $] | [expression +/- expression] | [expression +%- expression]
+/- +%-	absolute tolerance : +/- [expression +/- expression] [A +/- B] := [A-type(A)'(B) : A+type(A)'(B)] relative tolerance : +%- [expression +%- expression]
function	ファンクション内で、プロセスを kill(), suspend(), resume()できるようになっています。
constraint	クラスメソッドと同様に、 :initial、:final、:extends を使用できるようになりました。
rand	real 型の変数をランダム変数に指定できるようになりました。
default :/ expression	:/ に対して default が追加されました。
covergroup	covergroup を extends できるようになりました。

bins	ビンの定義で +/- や +%- を使えるようになりました。 [covergroup_expression +/- covergroup_expression] [covergroup_expression +%- covergroup_expression]
timescale tasks and functions	$timeunit と$timeprecision が追加されました。 $stacktrace が追加されました。
compiler directives	&&、||が使えるようになりました。

1.3 本資料の記法

LRM の変更点には、以下の三種類があります。

- 新機能の追加
- 仕様の明確化
- 仕様の訂正

本資料での説明をわかり易くするために以下の記法ルールを設けます（表 1-2）。

表 1-2 本資料での記法ルール

記法	意味
説明	新機能のシンタックスや機能の説明をしています。
説明	仕様の明確化および仕様の変更・訂正の説明をしています。
コード	主として、SystemVerilog による使用例を示します、
結果	主として、実行結果を意味します。
説明	前仕様および LRM からのシンタックスの引用を示します。

第 2 章以降に IEEE Std 1800-2023 の要約を解説しますが、本資料の章のタイトルは LRM の章のタイトルを反映しています。

1.4 改訂版の取得

IEEE GET Programにより無料で改訂版のPDFを取得する事ができます。ただし、IEEEのアカウント（無料）を作成する必要があります。 取得したマニュアルは、ダウンロードした本人のみが使用する事ができます。原則として、他人にコピーを渡す事、または、他人からコピーを取得する事は正しい取得法ではありません。

4　メンバーで構成されるデータタイプ

以下の機能追加があります。

- soft packed ユニオン機能
- イタレータの指定機能（アレイ要素、要素のインデックス）
- アレイマッピング機能（map）

4.1　soft packed ユニオン

ユニオンが使用し易くなりました。

4.1.1　機能概要

SystemVerilog の packed アレイ同士の代入文で左辺が右辺よりも長い場合、右辺が左辺の長さに拡張されて代入文が実行されますが、その拡張を抑止する機能が追加されました。この機能は、soft packed ユニオンと呼ばれます。

例えば、以下のように soft packed ユニオンを定義します。キーワード soft を付けると自動的に soft packed ユニオンになるので、packed を省略できます。

```
typedef union soft {
    logic [15:0]  m_word;
    logic [7:0]   m_byte;
} word_u;

word_u     w;
```

このように定義すると、w は図 4-1 のような構成になります。m_word と m_byte は同じ場所を共有しますが、m_word は 16 ビットを保有し、m_byte は LSBs の 8 ビットを保有します。そして、m_byte を操作する時に、m_word[15:8]は使用されません。つまり、m_byte の左側の 8 ビットの内容は保護されます。

図 4-1　word_u のレイアウト

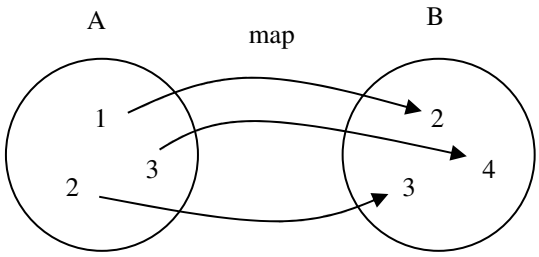
例えば、以下のようにwを定義すると、w.m_byte では m_byte の部分だけを参照するため、w[15:8]の内容は不変です。

```
word_u     w;
w = 16'habcd;
w.m_byte = 8'h56;
```

4.1.2　機能詳細

ユニオンは、複数の名称を持つメンバーから構成されますが、そのうちのどれか一つのメンバーしか同時に参照できないデータタイプです。また、ユニオンには packed ユニオンと unpacked ユニオンがありますが、packed ユニオンの場合には、全てのメンバーは integral 型を持たなければなりません。そして、tagged が付いていなければ packed ユニ

```
B = A.map(x) with (x+1);
```




```
B = A.map(x) with (x.toupper);
```

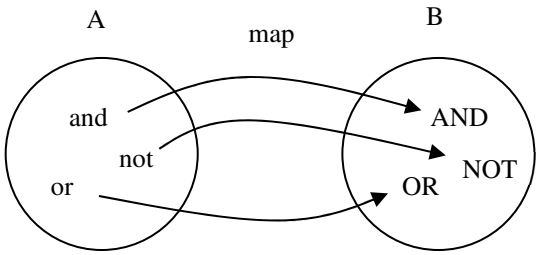


図 4-2　写像機能

4.7.2　機能詳細

unpacked アレイから別の unpacked アレイを作り出す機能がアレイ map メソッドとして追加されました。with クローズで条件を指定する事により、アレイ要素の値を置き換える事ができます。

LRM 178 ページ

アレイ map()メソッドは、元のアレイと同じ範囲の unpacked アレイを作り出します。アレイ要素の値は with クローズにより元のアレイ要素から再計算する事ができます。

例 4-5　map()メソッドの使用例（[1]）

map()メソッドを以下のように使用して別のアレイを構築します。

```
int A[] = {1,2,3}, B[] = {2,3,5}, C[$];

// Add one to each element of an array
A = A.map() with (item + 1'b1);          // A = {2,3,4}

// Add the elements of 2 arrays
C = A.map(a) with (a + B[a.index]);      // C = {4,6,9}

// Element by element comparison
bit Compare[];
Compare = A.map(a) with (a == B[a.index]);  // Compare = {1,1,0}
```

■

5　クラス

クラスはベースクラスを定義した後に拡張して汎用性を高める重要な機能を持ちますが、クラスを拡張する場合には危険な作業も伴います。例えば、ベースクラスに virtual メソッドが定義していたにも関わらず、別の目的を持つメソッドとして再定義してしまう危険性も潜んでいます。そうすると、ベースクラスの virtual メソッドは、正しく継承されなくなってしまいます。そのような危険性を未然に防ぐために、クラスには保護機能が追加されました。その他、サブクラスのコンストラクタを定義する際に、ベースクラスのコンストラクタの引数を簡単に参照できる仕組みとしてキーワード default が追加されました。本章では、これらの拡張機能を重点的に解説します。

5.1　クラスと:final

クラスを拡張できなくするためのオプションが追加されました。

LRM 190 ページ

final 属性を持つクラスは拡張できなくなります。キーワード class の後にコロン（:）と final を添える事で、拡張できなくなります。

例えば、以下のように定義されたクラス TopPacket は final なので、拡張する事はできません。BrokenPacket の定義は、エラーになります。

```
class :final TopPacket extends LinkedPacket;
...
endclass

class BrokenPacket extends TopPacket; // ILLEGAL
```

最も適切な例は、std::process クラスの定義になると思えます。std::process クラスは、以下のように定義されているので、ユーザは process クラスを拡張する事はできません。

```
class :final process;
    typedef enum {FINISHED, RUNNING, WAITING, SUSPENDED, KILLED} state;
    static function process self();
    function state status();
    function void kill();
    task await();
    function void suspend();
    function void resume();
    function void srandom(int seed);
    function string get_randstate();
    function void set_randstate(string state);
endclass
```

5.2　コンストラクタの default 引数

サブクラスのコンストラクトを定義する際、ベースクラスのコンストラクタの引数のリストを default で参照できるようになりました。この機能により、サブクラスのコストラクタは、ベースクラスのコンストラクタの仕様変更による影響を受け難くなります。

7　オペレータと式

オペレータでは、以下のような変更と機能追加があります。

- 関係演算子（<、>、<=、>=）の評価結果が 1 ビットになりました。
- 等価演算子（===、!==、==、!=）の評価結果が 1 ビットになりました。
- inside 演算子の右オペランドに指定できる値域の記述法が拡張しました。

演算結果の精度に関しての差は表 7-1 のようになります。

表 7-1　IEEE Std 1800-2017 と IEEE Std 1800-2023 の演算結果の差異

演算子	IEEE Std 1800-2017	IEEE Std 1800-2023
関係演算子 (<、>、<=、>=)	1、0、1'bx	1'b1、1'b0、1'bx
等価演算子 (==、!=)	1、0、1'bx	1'b1、1'b0、1'bx
等価演算子 (===、!==)	1、0	1'b1、1'b0

参考 7-1

関係演算子と等価演算子の評価結果がそれぞれ 0 と 1 から 1'b0 と 1'b1 に変更されましたが、実害はありません。寧ろ、実行効率が良くなる利点があります。
□

7.1　関係演算子（relational operators）

関係演算子としては、a < b、a > b、a <= b、a >= b があります。演算結果は、前仕様では、1、0、および 1'bx と解説されていましたが、次のように変わりました。

LRM 278 ページ

関係演算子の評価結果が偽であれば 1'b0、真であれば 1'b1、何れかのオペランドが x または z を含んでいれば結果は 1'bx となります。

参考 7-2

従来の式の結果に影響を及ぼす事はありません。
□

7.2　等価演算子（equality operators）

等価演算子としては、a === b、a !== b、a == b、a != b があります。関係演算子と同様に、演算結果は 1 ビットなりました。

LRM 279 ページ

関係演算子の評価結果が偽であれば 1'b0、真であれば 1'b1 となります。==と!=の場合には、何れかのオペランドが x または z を含んでいれば結果は 1'bx となります。

参考 7-3

従来の式の結果に影響を及ぼす事はありません。

□

7.3 inside 演算子と[+/-]と[+%-]

inside 演算子の右オペランドで指定する値域の指定法が拡張されました。以下のように[expression +/- expression]と[expression +%- expression]を指定できるようになりました。

LRM 290 ページ

値域を絶対許容範囲（absolute tolerance）トークン（+/-）または相対許容範囲（relative tolerance）トークン（+%-）で表現する事ができます。

シンタックスは以下のようになります。

LRM 289 ページ

```
inside_expression ::= expression inside { range_list }
range_list ::= value_range { , value_range }
value_range ::=
 expression
  | [ expression : expression ]
  | [ $ : expression ]
  | [ expression : $ ]
  | [ expression +/- expression ]
  | [ expression +%- expression ]
```

絶対許容範囲と相対許容範囲は以下のように定義されています。

LRM 290 ページ

絶対許容範囲は以下のように定義されています。
`[A +/- B] ::= [A-type(A)'(B) : A+type(A)'(B)]`

相対許容範囲は以下のように定義されています。
`[A +%- B] ::= [A-type(A)'(A*B/100.0) : A+type(A)'(A*B/100.0)]`

- A と B の符号にかかわらず、値域は[low_bound:high_bound]のようになります。ただし、もし low_bound > high_bound であれば、値を入れ替えるものとします。
- 計算式より、結果の値の型は、常に、A と同じ型になります。
- A が整数型の場合には、実数を整数型に変換しなければなりませんが、四捨五入ではなく切り捨てになります。

参考 7-4

絶対許容範囲と相対許容範囲の計算で実数を整数に変換する場合には、四捨五入ではなく、切り捨てる事に注意して下さい。
□

相対許容範囲の計算はやや複雑なので、以下に計算例を紹介します。