Artgraphics

IEEE Std 1800-2023の追加機能

篠塚一也
アートグラフィックス
Document Revision: 1.1,2024.12.05

www.artgraphics.co.jp

注意事項
（Caveat）

SystemVerilogの知識を個人的に習得する目的として本資料を活用して下さい。本資料を通して、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）で本資料を使用する事はご遠慮下さい。また、本資料から学んだ知識を転載する場合等は出典が本資料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。本注意事項は現在及び過去に於ける弊社からの全てのフリーダウンロード資料に適用されます。

本注意事項に合意出来ない場合には、本資料を速やかに抹消して下さい。
尚、ダウンロード記録は、依然として残ります。

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。改訂版には前仕様での誤りの訂正、前仕様で誤解を招いた仕様の補足説明、新規に追加された機能の説明が含まれています。
- 本資料では、新規追加された機能の中で特記に値する機能を解説します。
- 本資料の目的は、改訂版が重要な機能追加をしている事を喚起する点にあります。本資料を読了後、更に詳細を知る必要性を感じた場合には、巻末の参考書を参照して下さい。

アレイ操作メソッドとイタレータ

- アレイ操作メソッドのシンタックスが拡張されて、アレイ要素のイタレータと要素のインデックスイタレータを指定できるようになりました。

```
typedef struct {int index; ...} idx_type;
idx_type q[$];
...
q = arr.find(item,iter_index) with (item.index != item.iter_index);
```

アレイ要素のイタレータ

インデックスイタレータ

map

- unpackedアレイから別のアレイを作り出す機能が追加されました。この機能はmapと呼ばれ、従来のアレイコピーを拡張した機能を持ちます。
- 一般に、アレイAをアレイBにコピーするには、以下のように記述します。

```
B = A;
```

- mapを使用すると以下のように書き換えられます。

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

- 以下のように、withクローズに複雑な計算式を指定できます。

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

- アレイBの領域がmapにより割り当てられる利点があります。

mapの使用例

```
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}
```

rand real

- 実数型の変数もランダム変数に指定できるようになりました。

```
class sample_t;
rand logic [7:0] a;
rand real        r;
constraint C1 {
         a inside {[0:7]};
         r > 0.0 && r < 2.0;
}
endclass
```


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。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
```

std::process

- 最も良く知られている :final の使用例は、std::processクラスの定義です。この定義より、ユーザがprocessクラスを拡張できない事は明白です。
- このように、拡張されて欲しくないクラスには、:final を付けておくと安全です。

```
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
```


コンストラクタとdefault

- 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。

```
class base_t;
string              m_name;
logic [3:0]         m_state;
function new(string name="name",logic [3:0] state);
          m_name = name;
          m_state = state;
endfunction
endclass
```

```
class sub1_t extends base_t;
byte        m_value;
function new(default,byte v);
          super.new(default);
          m_value = v;
endfunction
endclass
```

ベースクラスのコンストラクタの引数リストがdefaultの代わりに展開する

メソッド
:initial / :extends / :final

- 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、:initialはvirtualメソッドをnon-virtualメソッドに書き換える間違いを防ぐ手段です。
- サブクラスでベースクラスのvirtualメソッドを拡張する場合、:extendsを指定できます。:extendsを指定すると、その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ければエラーになります。
- サブクラスでメソッドが再定義されるのを防ぐために、:finalを指定できます。指定すると、サブクラスでは同じメソッド名を持つメソッドを定義できなくなります。

使用例
:initial / :extends / :final

```
virtual class base;
function void f1(); ... endfunction                    // non-virtual
virtual function void f2(); ... endfunction            // virtual
pure virtual function void f3();                       // pure virtual
endclass : base
```

```
virtual class A extends base;
 function :initial void f1(); ... endfunction
 // OK: base::f1 is not a virtual method

 virtual function :final :extends void f2(); ... endfunction
 // OK: f2 shall not be overridden in subclasses of A

 function :final void f4();...endfunction
 // OK: f4 shall not be overridden in subclasses of A

 virtual function :extends void f5(); ... endfunction
 // NOT OK: f5 is not a virtual override
endclass : A
```

制約
:initial / :extends / :final

- クラスのメソッドと同様に制約に関しても、:initial、:extends、:finalの属性を定義できます。
- サブクラスで定義する制約に:initialを付けると、同じ制約名称がベースクラスで定義されていればエラーになります。
- サブクラスで定義する制約に:extendsを付けると、同じ制約名称を持つ制約がベースクラスに定義されていなければエラーになります。
- 制約を定義する際に:finalを付けておくと、サブクラスではその制約を再定義できなくなります。

カバーグループとextends

- 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できるようになりました。以下のシンタックスを使用します。

```
covergroup extends covergroup_identifier ;
{ coverage_spec_or_option }
endgroup [: covergroup_identifier ]
```

```
class base;
...
covergroup g1;
...
endgroup
function new();
    g1 = new;
endfunction
endclass
```

```
class subclass extends base;
...
covergroup extends g1;
...
endgroup
function new();
    super.new();
endfunction
endclass
```

まとめ

- IEEE Std 1800-2023で追加された機能で注意に値する機能を紹介しました。その他にも重要な機能が多く追加されています。更に、詳しい解説は以下の資料を参照して下さい。
- 『SystemVerilog IEEE Std 1800-2023の概要』　　無償版
- 上記の資料を共立出版の『SystemVerilog超入門』または『SystemVerilog入門』の書籍ウェブサイトより無償でダウンロードできます。下記のいずれかのサイトに行き「関連情報タブ」をクリックし、「補足資料」をクリックすると資料を得られます。
- https://www.kyoritsu-pub.co.jp/book/b10003280.html
- https://www.kyoritsu-pub.co.jp/book/b10031708.html

参考文献

- 文献[1]は改訂版の正式な仕様書です。是非一読下さい。
- 文献[2]は前仕様です。
- 文献[6]は、改訂版の概要を詳しく解説した資料ですが有償です。
- SystemVerilogに関する知識の確認には、文献[3-5]を参照して下さい。

[1] IEEE Std 1800-2023: IEEE Standard for SystemVerilog – Unified Hardware Design, Specification and Verification Language.
[2] IEEE Std 1800-2017: IEEE Standard for SystemVerilog – Unified Hardware Design, Specification and Verification Language.
[3] 篠塚一也、SystemVerilogによる検証の基礎、森北出版 2020.
[4] 篠塚一也、SystemVerilog入門, 共立出版 2020.
[5] 篠塚一也、SystemVerilog超入門, 共立出版 2023.
[6] SystemVerilog IEEE Std 1800-2023の概要、アートグラフィックス 2024.

