SystemVerilog概論

第１部基礎編

篠塚一也
アートグラフィックス

目次

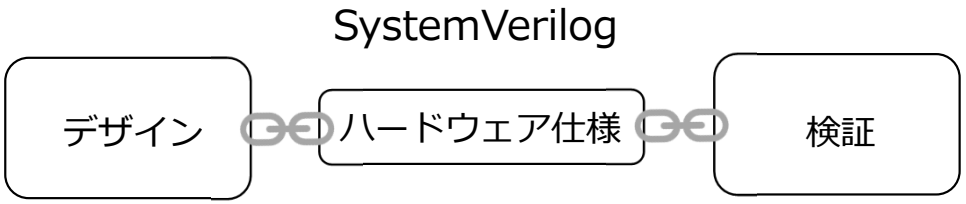
章	概要	章	概要
1	SystemVerilog概要	9	クラス
2	データタイプ	10	インターフェース
3	プロセス	11	パッケージ
4	オペレータ、式、代入文	12	モジュール
5	実行文	13	チェッカー、プログラム
6	タスクとファンクション	14	シミュレーション実行モデル
7	クロッキングブロック	15	システムタスクとファンクション
8	プロセス間通信機能		

第１章
SystemVerilog概要

SystemVerilogが備える特徴的な機能を紹介し、Verilog HDLとの差異を明確にします。その差異を通じてSystemVerilogが言語として目指す役割を解説します。本章の内容は、講義全体の概要でもあります。以降、 Verilog HDLをVerilogと略称する事があります。

SystemVerilogの特徴

- SystemVerilogはハードウェア設計、仕様、および検証を統一的に記述できる言語であり、単なる検証言語ではありません。



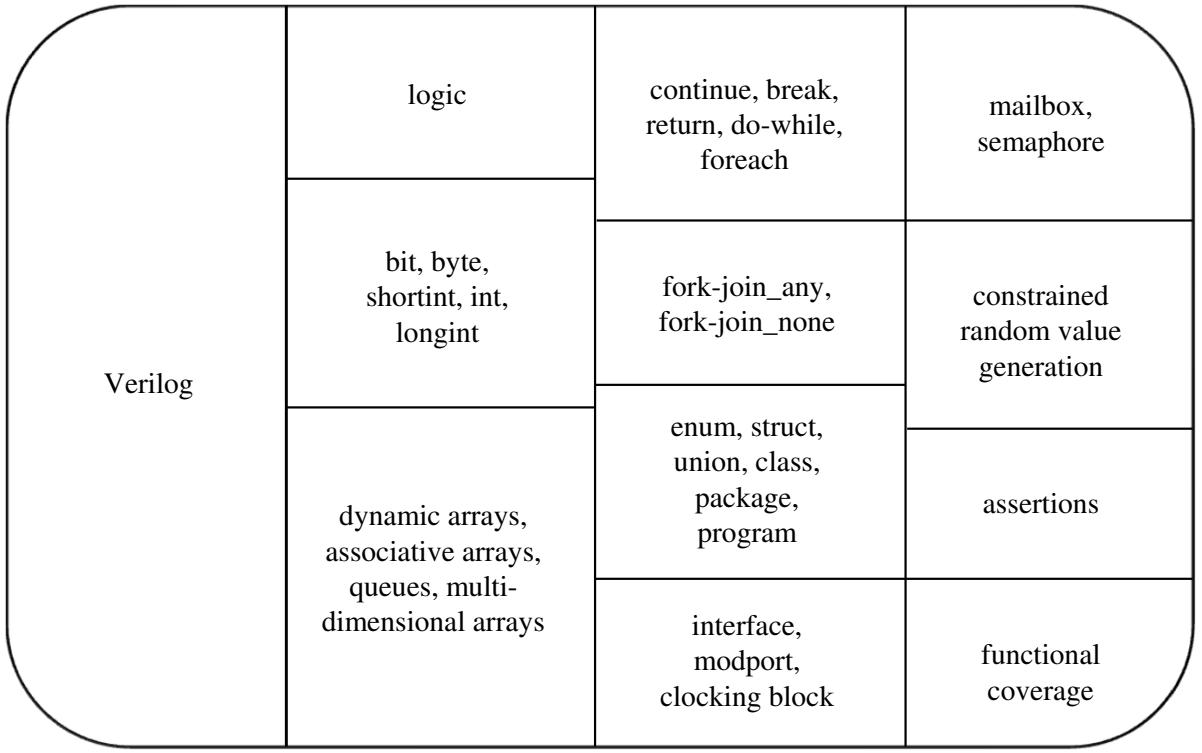
- デザインを検証するためには、仕様との比較が必要になります。その仕様が検証に使用される言語と同じ言語で記述されていれば、仕様を実行する事により効率的に、かつ正確に検証を行う事ができます。ここで、デザインとは仕様を基にして実装した内容を指します。
- 仕様記述機能はデザインと検証の橋渡しをする役割を果たします。アサーション、およびファンクショナルカバレッジは橋渡しの良い例です。

SystemVerilogの特徴

- SystemVerilogは、IEEE Std 1800-2023として規格化されているので、EDAツール間での相互互換性が保証されています。
- SystemVerilogは上位互換性の意味において、Verilogのスーパセットです。従って、従来のVerilogユーザは、そのままSystemVerilogのツール環境で作業を進める事ができます。
- RTL論理合成可能性を促進するための機能が充実しています。
- 検証に必要となる様々なデータ構造を支援するためにSystemVerilogは豊富なデータタイプを備えています。
- SystemVerilogはシミュレーション実行論理の厳密な規約を定義しています。
- 検証に必要なアサーション、ファンクショナルカバレッジ、ランダムスティミュラス 生成等の機能をSystemVerilogは標準的に備えています。

SystemVerilogの特徴

- SystemVerilogは下図に示すような機能を備えています。



SystemVerilogの機能（[10]）

Verilog HDLとの主な差異
変数の初期化順序

- Verilogでは、宣言時に指定した変数の初期化はシミュレーションの開始と同時に行われます。例えば、右記の①と②が同時に実行されるため、実行順序により結果が異なります。
- SystemVerilogでは、変数の宣言時に指定した初期化処理はシミュレーション開始前に行われるため、右記の例は正しく動作します。即ち、①の文が実行してから、②の文が実行します。

```
module test;
integer  width = 16;   ①
integer  nbits;

initial begin
     nbits = width;    ②
// ...
end
endmodule
```

Verilog HDLとの主な差異
シミュレーション実行モデル

- 例えば、右記の例では、変数aに1、２，３を割り当てていますが、Verilogでは変数aに値が設定されるタイミングを理解し難い問題があります。
- SystemVerilogはシミュレーションに関して厳密にタイミングを定義しているので、変数aに値が設定される時期を正確に把握できます。

```
logic [3:0]       a;

initial  a = 1;
initial  #0 a = 2;
initial  a <= 3;
// ...
```

実行命令	実行領域	実行順序
a = 1;	Active	1
#0 a = 2;	Inactive	2
a <= 3;	NBA	3

Verilog HDLとの主な差異
イベント制御の問題

- エッジセンシティブなイベント待ちは、クリティカルなタイミング状況ではイベント解除を見逃し易い傾向があります。例えば、Verilogでは右のような記述をすると、どちらのinitialプロシージャが先に実行しても、@ev1または@ev2の何れかが解除されません。
- SystemVerilogでは、triggeredメソッドを使用する事により、クリティカルなタイミング状況においても、イベント解除を見逃す事はありません。

```
event     ev1, ev2;
initial begin
     ->ev2;
     @ev1;
end
initial begin
     ->ev1;
     @ev2;
end
```

```
event     ev1, ev2;
initial begin
     ->ev2;
     wait( ev1.triggered );
end
initial begin
     ->ev1;
     wait( ev2.triggered );
end
```

Verilog HDLとの主な差異
アレイ

- Verilogでは一次元のアレイしか定義できませんが、SystemVerilogでは、多次元のアレイを定義する事ができます。
- しかも、SystemVerilog では、アレイには固定サイズのアレイだけでなく、ダイナミックアレイ、任意のデータ型のキーを持つアレイ、キュー等を定義する事ができます。

```
module test;
int      a1[10],       // 固定サイズのアレイ
         a2[],         // ダイナミックアレイ
         a3[string],   // 文字列をキーに持つアレイ
         a4[$],        // キュー
         a5[5][8];     // 二次元のアレイ
// ...
endmodule
```

設計言語としてのSystemVerilog

- Verilog HDLと比較して多くの機能がSystemVerilogに追加されました。特にRTL記述をより効率良く正確に行なうための機能が備えられています。例えば、以下のような機能が追加されています。
- モジュール間、およびテストベンチとDUT間の通信を簡素化するためのインターフェース機能
- enum型データタイプ
- ビット数に制限のない可変長リテラル（'0、'1、'x、'z）
- C/C++のようなストラクチャおよびユニオン等のデータタイプ

設計言語としてのSystemVerilog

- 値を戻さないファンクション
- 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、等）
- プログラミング機能の強化（continue、break、return文等）

検証言語としてのSystemVerilog

- Verilogには検証機能は皆無ですが、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、以下のような機能が追加されています。
- メモリー使用量と実行効率において優れた2-state型のデータタイプ（bit、byte、shortint、int、longint等）
- 文字列型データタイプ（string）
- 多次元アレイ
- 豊富なアレイタイプ（ダイナミックアレイ、associativeアレイ、キュー）
- クラス
- インターフェース

検証言語としてのSystemVerilog

- パッケージ
- テストベンチ機能（program）
- プロセス間通信機能（mailbox、semaphore等）
- プロセス制御機能（fork/join_any、fork/join_none）
- クロック信号に同期する信号の管理（clocking block）
- 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
- アサーション

SystemVerilog概論

第２部設計編

篠塚一也
アートグラフィックス

目次

章	概要
16	SystemVerilogによるモデリング
17	組み合わせ回路
18	シーケンシャル回路
19	FSM

第１６章
SystemVerilogによるモデリング

本章では、SystemVerilogによるモデリングンの仕方を以下に示す順に解説します。

(1) モデリングの仕方（組み合わせ回路、シーケンシャル回路）

(2) テストベンチの書き方（組み合わせ回路、シーケンシャル回路）

モデリングの仕方
組み合わせ回路

- alwaysプロシージャを使用してRTL論理合成可能な組み合わせ回路を記述する際に守るべきルールは、以下のようになります。
- 組み合わせ回路が依存する全ての信号をセンシティビティリストに指定します。always_combやalways_latchを指定する場合には、この手順を省略できます。
- 動作をブロッキング代入文だけを用いて記述します。
- ラッチの生成を避けるためには、必ず、出力信号に値を設定しなければなりません。
- 動作記述にディレーやイベント制御を使用する事はできません。

SystemVerilog概論

第３部検証編

篠塚一也
アートグラフィックス

目次

章	概要
２０	制約によるランダムスティミュラスの生成
２１	ファンクショナルカバレッジ
２２	アサーション
２３	UVM概説

第２０章 制約によるランダムスティミュラスの生成

SystemVerilogには、効率よく、かつ効果的にランダムスティムラスを生成する方法があります。制約を付けてランダムスティムラスを生成する事により、目的に即したテストデータを確実に準備する事ができます。現代の検証手法では、ランダムスティムラスを生成する方法はトランザクション生成の中心的な役割を担っています。本章では、ランダムスティムラス生成に関する基本的な知識を解説します。

概要

- 近年のテスト法は、CRT（Constrained Random Test）をベースしています。
CRTは、制約を満たすテストデータをランダムに生成して検証する手法です。
- SystemVerilogによるランダムスティミュラス生成の仕方は直接的で明解です。
テストデータに乱数発生情報を付加するだけで準備が終了します。例えば、
次のようにして乱数発生情報を持つパケットデータを準備します。

```
class packet_t;
rand bit [15:0]     addr;     // random variable
rand bit [31:0]     data;     // random variable
int unsigned        state;    // state variable
endclass
```

- クラスには乱数を発生するビルトインメソッドrandomize()が定義されてい
るので、次のようにして乱数を発生する事ができます。

```
packet_t  packet;
packet = new;
// ...
assert( packet.randomize() ); // assign random values
```

addrとdataに乱数を割り当てる

ランダム変数

- ランダム変数はrand、または、randcの修飾子を付けて宣言します。これらの修飾子をアレイ変数にも適用する事が出来ます。ランダム変数以外は、state変数と呼ばれます。
- ランダム変数はビルトインメソッドrandomize() により乱数が割り当てられます。
- ダイナミック・アレイをランダム変数に指定すると、アレイ・サイズもランダムに決定されます。アレイ・サイズに制約を設定しないと膨大なアレイが生成されて、殆どシミュレーションが終了しない状況に陥ります。必ず、アレイ・サイズに制約を設定する事が必要です。