

# UVM 入門

---

---

Document Revision: 1.9, 2019.04.06

アートグラフィックス

篠塚一也

## UVM 入門

©2018, 2019 アートグラフィックス  
〒124-0012 東京都葛飾区立石 8-14-1  
[www.artgraphics.co.jp](http://www.artgraphics.co.jp)

## UVM Primer

©2018, 2019 Artgraphics. All rights reserved.  
8-14-1, Tateishi, Katsushika-ku, Tokyo, 124-0012 Japan  
[www.artgraphics.co.jp](http://www.artgraphics.co.jp)

## はじめに

この入門書は UVM の概要を簡単に纏めた学習用の素材です。UVM の基礎を把握する事を主眼にしている為、詳細の機能については触れていません。この入門書を読了後に、UVM User's Guide ([2]) を熟読する事を薦めます。或いは、UVM の文献 (例えば[4]) を読む事を薦めます。

誤字訂正、及び、説明の補足を随時行います。従って、常に本書の内容は更新を余儀なくされます。ご了承ください。

尚、本書の記述は UVM 1.2 をベースにしています。

アートグラフィックス

篠塚一也

### 注意事項

日本国内には UVM に関する良書が皆無な為、本書を執筆しました。UVM の知識を個人的に習得する目的として本書を活用して下さい。本書を通して、業務 (実践) で必要となる UVM 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) で本書を使用する事はご遠慮下さい。また、本書から学んだ知識を転載する場合等は出典が本書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。

本書は、弊社の設計・検証ツールの付属部品です。設計・検証ツールのライセンス保有者のみ使用して下さい。

## 変更履歴

| 日付         | Revision | 変更点   |
|------------|----------|---|
| 2018.03.10 | 1.0      | 第一版。  |
| 2018.04.05 | 1.1      | ① 第二章の構成を変更、及び、節の追加。<br>② 第五章を追加。   |
| 2018.05.31 | 1.2      | ① 第一章の内容に補足。  |
| 2018.07.25 | 1.3      | ① 1.7 を追加。<br>② 他の節に於いて一部記述を変更していますが、読み直す必要はありません。  |
| 2018.07.28 | 1.4      | ① 随所で表現を正確にしました。<br>② 「補足」の章を追加しました。  |
| 2018.08.07 | 1.5      | ① 第一章を再編成しました。<br>② フィールド・マクロ・フラグの記述を追加しました。<br>③ virtual sequencer 及び virtual sequence の記述を追加しました。 |
| 2018.10.20 | 1.6      | ① 第二章に「クラスの記述法」の節を追加しました。   |
| 2019.02.19 | 1.7      | ① 2.1.4 ファクトリの記述に説明を補足。   |
| 2019.03.10 | 1.8      | ① 1.1.4 節に説明補足。<br>② 1.8 節を追加。<br>③ 3.3.1 節に説明補足。<br>④ 4.2 節に説明補足。                                  |
| 2019.04.06 | 1.9      | ① 誤字訂正  |
| 2019.06.16 | 2.0      | ① 誤字訂正  |

## 目次

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>概要</b> .....                                      | <b>1</b>  |
| 1.1      | UVM 概要 .....   | 1         |
| 1.1.1    | UVM とは何か? .....                                      | 1         |
| 1.1.2    | TLM (Transaction-Level Modeling) .....               | 1         |
| 1.1.3    | 代表的な UVM クラス .....                                   | 1         |
| 1.1.4    | Virtual Interface .....                              | 2         |
| 1.1.5    | UVM テストベンチの構造 .....                                  | 3         |
| 1.2      | 検証技術のトレンド .....                                      | 4         |
| 1.3      | UVM の構成 .....  | 4         |
| 1.3.1    | uvm_pkg .....  | 4         |
| 1.3.2    | uvm_object と uvm_component .....                     | 5         |
| 1.3.3    | UVM マクロ .....  | 5         |
| 1.4      | シミュレーション・フェーズ (SIMULATION PHASES) .....              | 7         |
| 1.5      | ソース・コードの準備 .....                                     | 8         |
| 1.5.1    | \include .....                                       | 8         |
| 1.5.2    | import UVM .....                                     | 8         |
| 1.6      | UVM 使用例 .....  | 9         |
| 1.7      | UVM の DPI の意義 .....                                  | 11        |
| 1.8      | UVM に依る検証コード .....                                   | 11        |
| 1.8.1    | 記述 .....   | 11        |
| 1.8.2    | シミュレーション・フェーズ .....                                  | 12        |
| <b>2</b> | <b>UVM クラス・ライブラリーの基礎</b> .....                       | <b>14</b> |
| 2.1      | 基本的な概念 .....   | 14        |
| 2.1.1    | uvm_object と uvm_component .....                     | 14        |
| 2.1.2    | コンストラクタ .....  | 14        |
| 2.1.3    | フィールド・マクロ .....                                      | 14        |
| 2.1.4    | ファクトリ (factory) .....                                | 17        |
| 2.1.5    | シミュレーション・フェーズ (simulation phases) .....              | 18        |
| 2.1.6    | コンフィギュレーションの設定変更 .....                               | 19        |
| 2.2      | TLM (TRANSACTION-LEVEL MODELING) .....               | 21        |
| 2.2.1    | トランザクション .....                                       | 21        |
| 2.2.2    | コンポーネント間の通信 .....                                    | 21        |
| 2.2.3    | uvm_tlm_fifo .....                                   | 25        |
| 2.2.4    | analysis ports と exports .....                       | 26        |
| 2.3      | プリント機能 .....   | 27        |
| 2.3.1    | UVM プリンター .....                                      | 27        |
| 2.3.2    | アレイのプリント .....                                       | 30        |
| 2.4      | メッセージ機能 .....  | 32        |
| 2.5      | UVM シミュレーション .....                                   | 33        |
| 2.5.1    | raise_objection と drop_objection .....               | 33        |
| 2.5.2    | シミュレーション進行状況の確認 .....                                | 35        |
| 2.6      | クラスの記述法 .....  | 35        |
| <b>3</b> | <b>UVC (UNIVERSAL VERIFICATION COMPONENTS)</b> ..... | <b>38</b> |
| 3.1      | トランザクション .....                                       | 38        |
| 3.1.1    | トランザクションの定義 .....                                    | 38        |
| 3.1.2    | サブクラスによる制約 .....                                     | 39        |

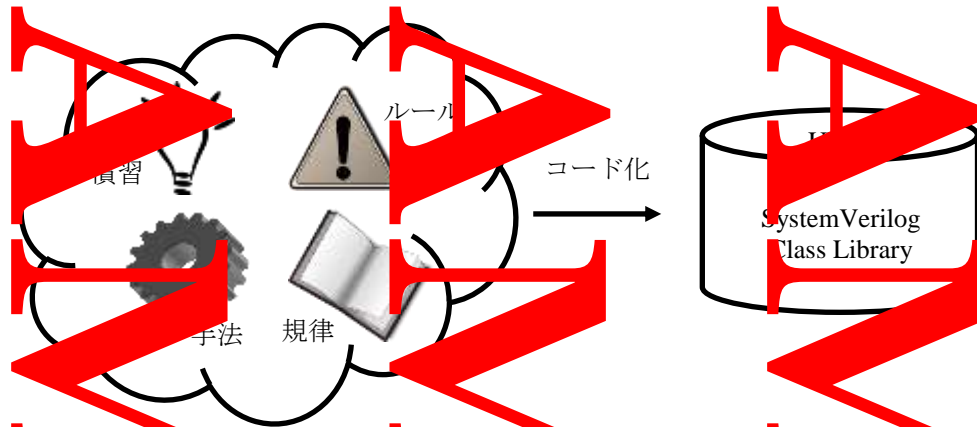
|          |  |           |
|----------|--|-----------|
| 3.2      | ドライバー .....  | 40        |
| 3.3      | シーケンサー .....   | 42        |
| 3.3.1    | シーケンサーの定義 .....  | 42        |
| 3.3.2    | シーケンサーとドライバーの基本的なハンドシェイク .....                               | 43        |
| 3.4      | モニター .....   | 43        |
| 3.4.1    | コレクター .....  | 43        |
| 3.4.2    | モニターの定義 .....  | 45        |
| 3.5      | コンポーネントのインスタンスを作成する方法 (UVM FACTORY) .....                    | 46        |
| 3.6      | AGENT .....  | 47        |
| 3.7      | ENVIRONMENT .....  | 48        |
| 3.8      | シナリオの作成 .....  | 51        |
| 3.8.1    | シーケンス (sequences) .....                                      | 51        |
| 3.8.2    | シーケンサー (sequencers) .....                                    | 52        |
| 3.8.3    | シーケンスの開始 .....   | 52        |
| 3.8.4    | Virtual Sequencer と Virtual Sequences .....                  | 53        |
| <b>4</b> | <b>テストベンチの作成 .....</b>                                       | <b>57</b> |
| 4.1      | トップ・レベルの ENVIRONMENT の作成 .....                               | 57        |
| 4.2      | ベース・テスト .....  | 57        |
| 4.3      | テスト .....  | 58        |
| 4.4      | テストの選択 .....   | 58        |
| 4.5      | PROGRAM ブロックとテストベンチ .....                                    | 59        |
| 4.6      | コマンド・ラインの操作 .....  | 59        |
| <b>5</b> | <b>UBUS .....</b>  | <b>61</b> |
| 5.1      | トップ・モジュール .....  | 61        |
| 5.2      | ベース・テスト .....  | 62        |
| 5.3      | トップ・レベルのエンバイロメント .....                                       | 63        |
| <b>6</b> | <b>補足 .....</b>  | <b>66</b> |
| 6.1      | 設計・検証作業の働き方改革 .....  | 66        |
| 6.2      | WISH LIST .....  | 66        |
| 6.2.1    | 環境管理及びファイル管理 .....   | 66        |
| 6.2.2    | SystemVerilog テキスト・エディタ (What You See Is What You Get) ..... | 66        |
| 6.2.3    | データ・タイプのサポート .....   | 68        |
| 6.2.4    | ウィザード .....  | 69        |
| 6.2.5    | コード・スニペットとブックマーク .....                                       | 71        |
| 6.2.6    | ルール・チェック .....   | 73        |
| 6.2.7    | コード・レビュー及び仕様書 .....  | 74        |
| 6.3      | 結論 .....   | 74        |
| <b>7</b> | <b>参考文献 .....</b>  | <b>76</b> |

## 1 概要

### 1.1 UVM 概要

#### 1.1.1 UVM とは何か？

Universal Verification Methodology (UVM) とは、業界で推奨されている技術、慣習、規律等をコード化して具体化し、検証技術の利用性を促進して生産性を高める為の SystemVerilog クラス ライブラリーです。UVM は Accellera System Initiative により開発されました。



UVM は SystemVerilog をベースにして記述されているので、SystemVerilog をサポートしている検証ツールの環境で使用する事が出来ます。

#### 1.1.2 Testbench-Level Model

UVM は TLM を採用し、シグナル・レベルよりも高位の記述法を用いて検証タスクを表現します。このアプローチはシステムの動作を考察する際にも有効な方法です。

UVM ではトランザクションはオブジェクトであり、UVM コンポーネントがトランザクションを操作します。特別なコンポーネントライブラリー (uvm\_\*) が存在します。トランザクションをシグナル・レベルに変換して DUT を探る役目を持ちます。

DUT 側シグナルの変化を検知する役目を持つ UVM コンポーネントが必要になります。そのコンポーネントは、一般的には、コレクター (collector) と呼ばれます。

ドライバライブラリーの存在により、トランザクション・システムを記述する事が出来るようになります。尚、UVM コンポーネントと DUT 間のデータ受受には SystemVerilog の Virtual Interface が使用されます。

#### 1.1.3 代表的な UVM クラス

UVM には多くのクラスが定義されていますが、ユーザが直接使用するのはその内の一部のクラスで、メソドロジー・クラス (methodology class) と呼ばれます。代表的なメソドロジー・クラスとトランザクション関連のクラスとしては、以下のクラスが存在します。

| UVM クラス           | 種別         |
|-------------------|------------|
| uvm_sequence_item | トランザクション関連 |
| uvm_sequence      | トランザクション関連 |

|                |            |
|----------------|------------|
| uvm_driver     | メソドロジー・クラス |
| uvm_sequencer  | メソドロジー・クラス |
| uvm_env        | メソドロジー・クラス |
| uvm_agent      | メソドロジー・クラス |
| uvm_test       | メソドロジー・クラス |
| uvm_monitor    | メソドロジー・クラス |
| uvm_scoreboard | メソドロジー・クラス |

uvm\_sequencer はトランザクションを生成するためのクラスで、データ・オブジェクトになります。uvm\_sequence もデータ・オブジェクトですが、トランザクションの生成を行います。

上記の表の uvm\_driver 以降はコンポーネント（データ・オブジェクトではありません。コンポーネントとは、シミュレーションの実行エンティティ。コンポーネントが実行する事によりシミュレーションが実行されます。この為、コンポーネントはデータ・オブジェクト。コンポーネントを明確に定義しています。

UVM コンポーネントはシミュレーションの対象となるので、デザインにおける module の様な役目を果たします。即ち、UVM において、コンポーネントは自然にコンポーネント・インスタンスの階層構造を構成します。階層のトップには、uvm\_testbench と呼ばれるインスタンスが存在します。UVM はその階層構造を使用してダイナミックなシミュレーションを制御します。UVM は、階層構造を使用してテストベンチの構成をシミュレーション開始直前に変更可能に出来ます。この機能により、一度のコンパイルで複数のテスト・ケースを実行する事が出来ます。階層構造は、シミュレーション開始直前に決定し、一度シミュレーションが開始（コンポーネントのタスク run\_phase が実行された時点）すると変更不可能なコンポーネント・インスタンスとして動作しません。

実行エンティティであるドライバー（uvm\_driver のサブクラス）は DUT を制御します。DUT はシグナル・オブジェクト。ドライバーは DUT をシミュレーション・レベルからコンポーネント・レベルへの変換を行い、DUT をドライヴします。ドライバーはトランザクションをシーケンサー（uvm\_sequencer のサブクラス）から受け取り、シーケンサーはこれらの機能を実際に定義する必要があります。ユーザはメソドロジー・クラスから継承したクラスの内部に、これらの変換手順、及び、制御処理を定義します。

モニター（uvm\_monitor のサブクラス）は DUT の信号値を監視する機能を提供します。モニター（通常はコレクター）は DUT の信号値をトランザクションに変換して、他のコンポーネントに引き渡します。それらのコンポーネントの一つとしてスコアボード（uvm\_scoreboard のサブクラス）が存在します。スコアボードは DUT の出力が期待する結果と一致するか否かのテストを行います。その他、カバレッジ情報等も収集します。ユーザはメソドロジー・クラスから継承したクラスの内部に、これらの変換、及び、制御手順を定義します。

シーケンサー、ドライバー、及び、モニターはエージェント（uvm\_agent のサブクラス）として纏められます。エンバィロメント（uvm\_env のサブクラス）はシーケンサー、スコアボード、エージェント、及び、下位のエンバィロメント（uvm\_env のサブクラス）から構成されます。

### 1.1.4 Virtual Interface

DUT とのデータ授受は virtual interface を介して行われます。virtual interface は interface のインスタンスへのポインターです。interface のインスタンスはテストベンチ（トップ・モジュール）で作られます。そのインスタンスが DUT と関連するコンポーネントの virtual interface に引き渡さなければなりません。その割り当てをハード・コーディングすると検証技術の再利用性に反する為、実行時（通常は、build\_phase、又は、connect\_phase）に virtual interface の情報を設定します。



その際、前述したコンポーネント・インスタンスの階層構造が使用されます。

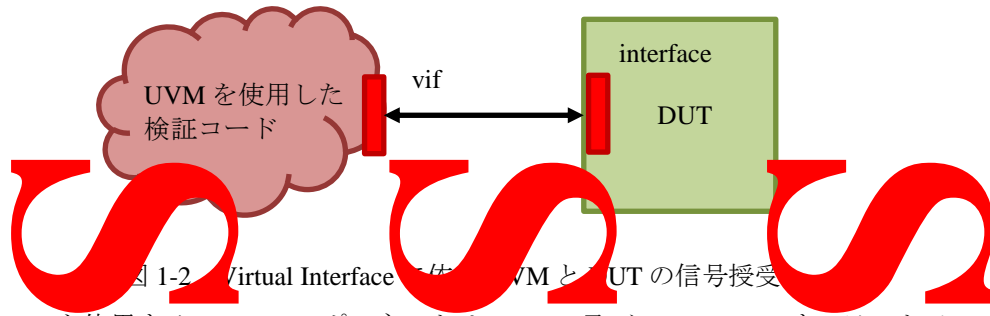


図 1-2 Virtual Interface 全体 (UVM と DUT の信号授受)

virtual interface を使用する UVM コンポーネントは driver、及び、collector です。テストベンチには interface のインスタンスが実在するので、テストベンチではそのインスタンスと UVM 側の virtual interface のメンバー vif との対応表を準備しておきます。この準備を run\_test() を呼ぶ前に完了しなければなりません。UVM 側は build\_phase、及び、connect\_phase 実行中に、その対応表を使用して vif に実在するインスタンスを割り当てます。ここで、vif に関する設定が実行されていないと実行が異常終了します。一般に、この時点でのエラー解析は複雑である為、vif の設定には注意が必要です。詳細は、3.2 節を参照して下さい。

### 1.1.5 UVM テストベンチの構造

下図において、図の包含関係はクラス、又は、module のインスタンスを持つという関係を示します。例えば、テストベンチは DUT のインスタンスを保持します。

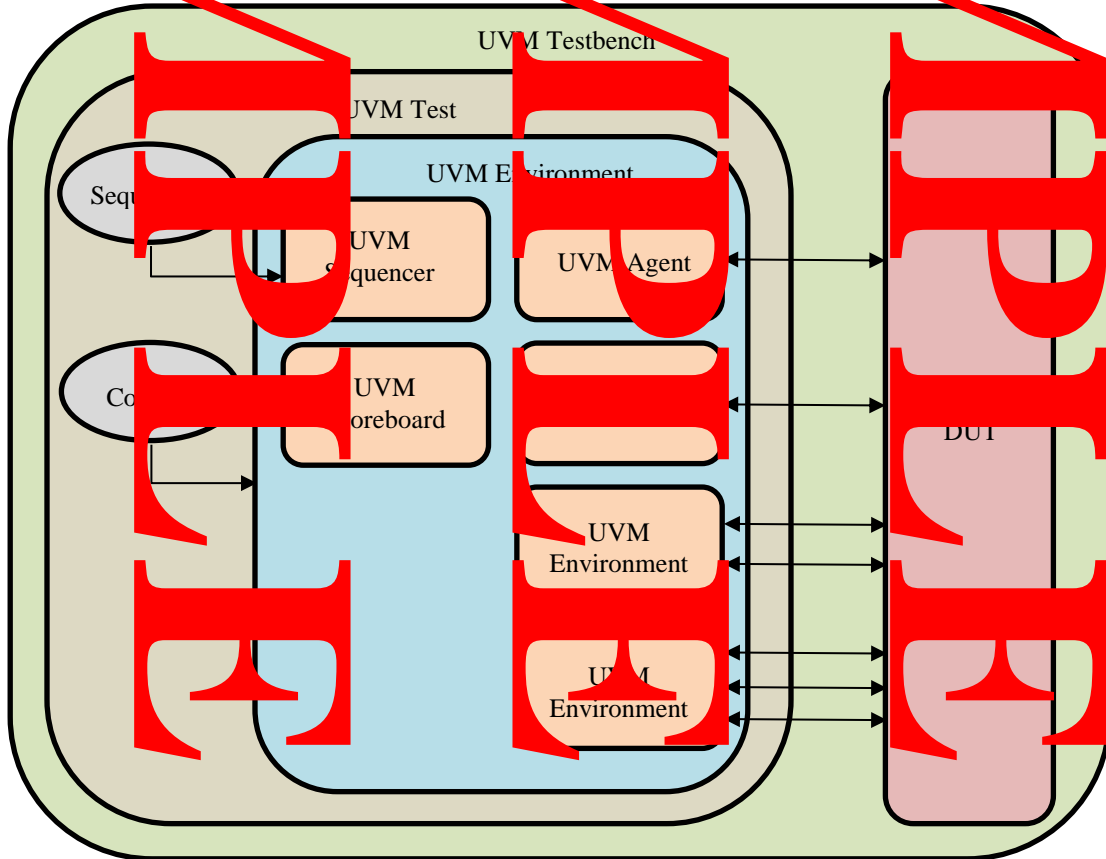


図 1-3 UVM テストベンチの構造 ([2])