



## Open Verification Methodology

### Open Verification Methodology (OVM)

メンター・グラフィックスのAdvanced Verification Methodology (AVM) とケイデンスのUniversal Reuse Methodology (URM) の成果を基に、それらの長所を組み合わせることで作成されたOVMは、SystemVerilogのメリットを最大限に引き出す検証メソッドロジです。

メンター・グラフィックスとケイデンス・デザイン・システムズ（以下ケイデンス）の共同開発によるOpen Verification Methodology（OVM）は、業界初のオープンで相互運用性のあるSystemVerilogベースの検証メソッドロジです。OVMが提供するベース・クラスのライブラリを使用し、モジュール化され再利用可能な検証環境を構築することができます。この環境下では、標準のトランザクション・レベル・モデリング（TLM）インターフェースを介してコンポーネント同士が通信可能となります。さらに、高度なテスト・シナリオや、ブロックからシステムへの再利用を可能にする共通メソッドロジとクラス・ライブラリにより、社内外を問わない再利用を実現します。

OVMは、様々な検証プラットフォームをサポートする、検証の初心者にも熟練者にも最適なデファクト・スタンダードのメソッドロジです。メンター・グラフィックスのAdvanced Verification Methodology（AVM）、およびケイデンスのUniversal Reuse Methodology（URM）の成功を基に構築されたOVMは、業界をリードする両社の共同開発力で、SystemVerilogの利点を最大限に引き出します。検証IP（VIP）、トランザクション・レベルとRTLモデルのためのツール互換メカニズムを提供し、開発フローで一般的に使用されている他の言語とも完全に統合することができます。

## SystemVerilogへの期待

複雑な電子機器の設計検証には、スタンダードへの準拠が望まれます。標準の言語、ライブラリ、ツール間でのデータ交換の書式、インターフェース、コネクタなどを使用することで、設計記述やEDAツール、最終製品の互換性や再利用性を向上することが可能です。従って、新しいスタンダードが現実のものとなったとき、それを使用することによる大きな利点を得ることができます。

SystemVerilogはそのようなスタンダードの1つです。数年にわたる集中的な標準化作業を経て、IEEE Std. 1800-2005として策定・認可されています。業界初のハードウェア設計検証言語（HDVL）として、SystemVerilogは記述性の高い書式として多くの設計者に使用され、かつ多くのEDAツールがサポートする言語となっています。このことから、あるツール上で作成されたデザイン、モデル、検証コンポーネントを、他のツール上で使用することが可能となります。

ただし実際には、言語それ自体だけでは、環境非依存で使用できることを保証するには不十分です。ツールの互換性は極めて重要ですが、それに加えて検証コンポーネントが広く対応できるメソッドロジに基づいていることが重要です。言語の機能は、洗練されたテストベンチ／検証環境を構成するための検証コンポーネント作成に使用されます。SystemVerilogを用いた検証におけるオブジェクト指向の考え方では、これらの検証コンポーネントはクラス・ライブラリとして定義されます。これらのライブラリを活用して、再利用可能な検証環境を構築する方法については、サンプル・コードやメソッドロジが助けになります。図1では、言語やライブラリとシミュレータの関連について、典型的な階層構造を示します。

主要なEDAベンダーは、図1に示されるような階層構造に類似した手法を採用していますが、その詳細については若干の差異があります。各ベンダーはそれぞれ異なるメソッドロジ、クラス・ライブラリ、クラス・ライブラリ内で使用されメソッドロジで推奨される言語機能を提案しています。

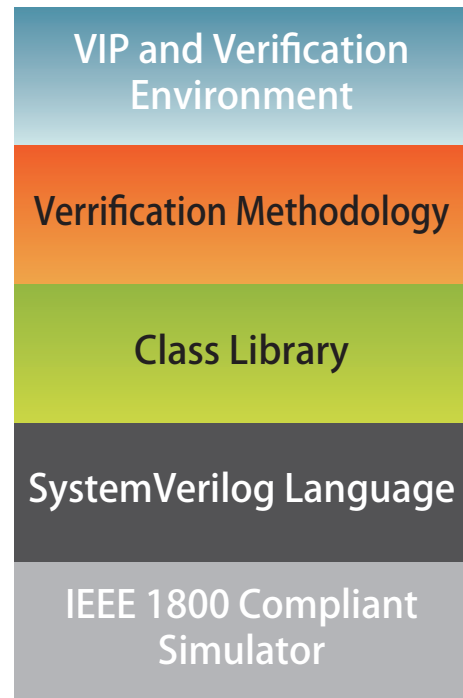


図1. SystemVerilog検証階層図

これらのメソッドロジは、メソッドロジ内部での一貫性はありますが、メソッドロジを超えた互換性がなく、そのため、互換性や再利用性などに挙げられる標準言語としてのSystemVerilogへの期待が危ういものとなっていました。さらにシミュレータごとにSystemVerilogに対する言語サポートの範囲に相違があるという事実と、メソッドロジによっては制約の多い独自仕様があるという事実とが相まって、検証IPや検証コードが複数のシミュレータ上で動作しないという状態になっていました。多くのシミュレータにおいて、SystemVerilogの言語サポート範囲が将来的に同じになるとしても、複数のクラス・ライブラリとメソッドロジが存在することにより、検証IPの互換性・再利用性は大きく阻まれてしまいます。メソッドロジごとに検証IPとテストベンチ間の通信方式が異なるため、異なるベンダーから提供される検証IPを組み合わせて使用するということは困難であり、すでに検証済みの検証IPを利用できる利点すら打ち消してしまいかねません。このような状況では、設計者や検証者が期待していた形でSystemVerilogを利用することはままならず、EDAベンダーが早急にこの課題に取り組む必要がありました。

## 真にオープンなSystemVerilogメソッドロジ

このような状況を打破するために、ケイデンスとメンター・グラフィックスは両社のシミュレータで動作する共通のメソッドロジとクラス・ライブラリを開発しました。2007年8月16日の発表の通り、SystemVerilogの検証階層において、OVMはクラス・ライブラリとメソッドロジのレイヤーを対象としています（図2参照）。クラス・ライブラリは、メンター・グラフィックスのQuestaおよびケイデンスのIncisive検証プラットフォームでサポートしています。従って、このライブラリを使用して作成した検証IPやテストベンチは、記述の書換えなど余分な工数をかけることなく、相互のプラットフォームで動作します。さらに、OVMはオープン・ソースで公開されていることから、このスタンダードに準拠するコードはSystemVerilogをサポートするどのシミュレータでも実行

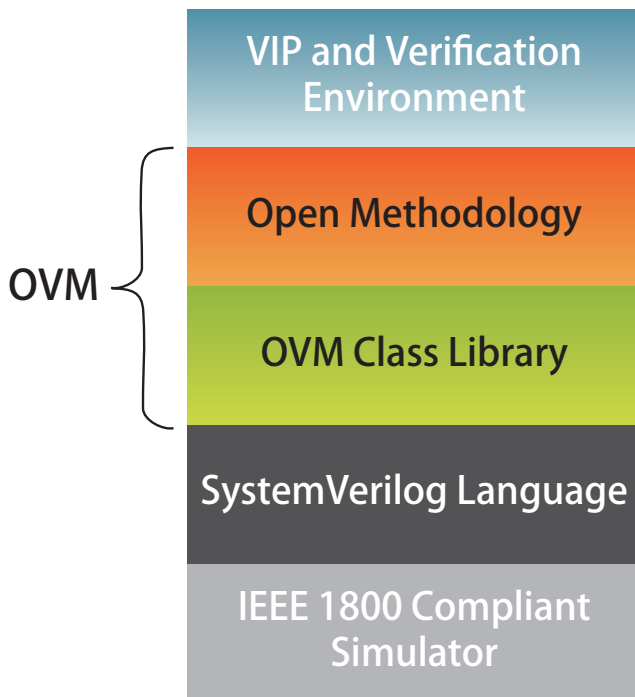


図2. OVM検証階層図

可能です。

前述の通り、検証の相互運用性には、共通の言語サポート以上のことが必要とされます。ケイデンスとメンター・グラフィックス、およびサードパーティから提供されるOVM準拠の検証IPは、OVM準拠のテストベンチにシームレスに組み込むことが可能となります。これは、検証IPプロバイダーおよびテストベンチを構築する検証エンジニア双方にとって大きな利点です。OVMにおける検証IPとテストベンチの間の通信は、OSCI (Open SystemC Initiative) によって定義され、広く使用されているTLM標準のSystemVerilog実装で行っています。このTLMの採用により、SystemVerilogで記述されたコンポーネント間の接続だけでなく、SystemCや他の言語で書かれた検証コンポーネントとの接続が容易になり、またブロック・レベルからシステム・レベルまで、複数の抽象度で記述された検証コンポーネント、モデル、環境の接続が可能になります。

SystemVerilog OVMクラス・ライブラリのソース・コード、使用例、補助ドキュメントは、OVM World (www.ovmworld.org) ウェブサイトで既に公開されています。このサイトは広く一般に公開するものであり、利用に際しての費用はかかりません。OVMクラス・ライブラリのソース・コードは、著作権表示以上に多くを求めないことからオープン・ソース契約で広く使用されているApache2.0ライセンス使用規約に従って公開されています。これを独自構築の検証環境で使用したい方、検証環境をサービスや検証IPのような商品として販売したい方、標準団体、またメンター・グラフィックスやケイデンスの競合EDA企業であっても、OVMのソース・コードをダウンロードして使用することが可能で、ケイデンスやメンター・グラフィックスへの金銭の支払いも一切発生しません。OVMとそのクラス・ライブラリのオブジェクト指向性により、ソース・コードを修正することなく機能拡張を実現することが可能ですが、Apache2.0ライセンスは、ソース・コードの改変も必要であれば認めています。さらに、OVMウェブサイトのユーザー・コミュニティでの要望や提案により、将来的にOVMがそのような拡張に取り

込むことも考えられます。

## 実証済みの検証テクノロジーに基づく

OVMは、アナウンスされたばかりと言う意味では「新しい」といえませんが、10年以上に渡る業界のベスト・プラクティスを反映した、確立された検証テクノロジーおよびメソッドロジを基にして構築されていることから、既に高い実績を持っています。具体的には、OVMはメンター・グラフィックスのAVM3.0およびケイデンスURM6.2が基になっており、これらに対して後方互換性を持っています。

AVMは、2004年にメンター・グラフィックスによって発表されて以降、多くの検証チームに広く採用されてきました。2006年5月より入手可能となったAVM3.0は、新機能を追加し、ユーザー・コミュニティからの多くのフィードバックを反映した第三世代のメソッドロジです。AVM3.0を使用する検証エンジニアは、迅速かつ簡単に移植性と相互運用性をもたらすOVMへと移行することができます。

URMは、ケイデンスにより2006年に紹介され、2002年から導入されているオブジェクト指向検証技術eRMを、SystemVerilogによって包含するべく、そのサポートを拡張してきたものです。URM6.2リリースを使用する検証エンジニアおよびVIPパートナーは、スムーズにOVM準拠への移行が可能です。

OVMの素早い提供とその堅牢性は、同じTLM通信をベースにする前出の2つのメソッドロジが補完的であり、メソッドロジと機能において密接に連携することができたことに所以します。OVMの実装は両社の真の協業によるもので、最先端メソッドロジに要求される機能を実現しました。この協業は、公平な相互協力、優れた機能、有用性、SystemVerilogに関する長年の開発経験による知識の融合によりもたらされました。

## OVMライブラリ

図3は、Unified Modeling Language (UML) ダイアグラムによりOVMライブラリを表したものです。ライブラリとメソッドロジは、再利用可能な制約付きランダムによるカバレッジ・ドリブン・テストベンチを構築するのに必要なすべてのテクノロジーを提供し、先進の柔軟性、カスタマイズ性、再利用性を可能にします。

OVMは、洗練されたトランザクション・レベル・インターフェースで通信を行う、モジュール化され再利用可能な検証環境を構築するための、TLMベースの構造基盤を提供します。ユーザーはこのクラス・ライブラリの使用により、制約付きランダムのシーケンス・スティミラス生成、機能カバレッジ・データの集計・解析、コンフィギュレーション可能なテストベンチ環境へのアサーション統合を行うことができます。具体的には次の項目が挙げられます。

- モジュール化と再利用を促進する、検証コンポーネント間を接続する下層基盤としてのTLM通信
- 検証環境内の全コンポーネントを協調動作させるための、ユーザーによる拡張が可能な共通フェーズ
- テストベンチ環境の動的変更が可能。最小コード変更で同一ベース環境を基に、異なるテストを記述することが可能

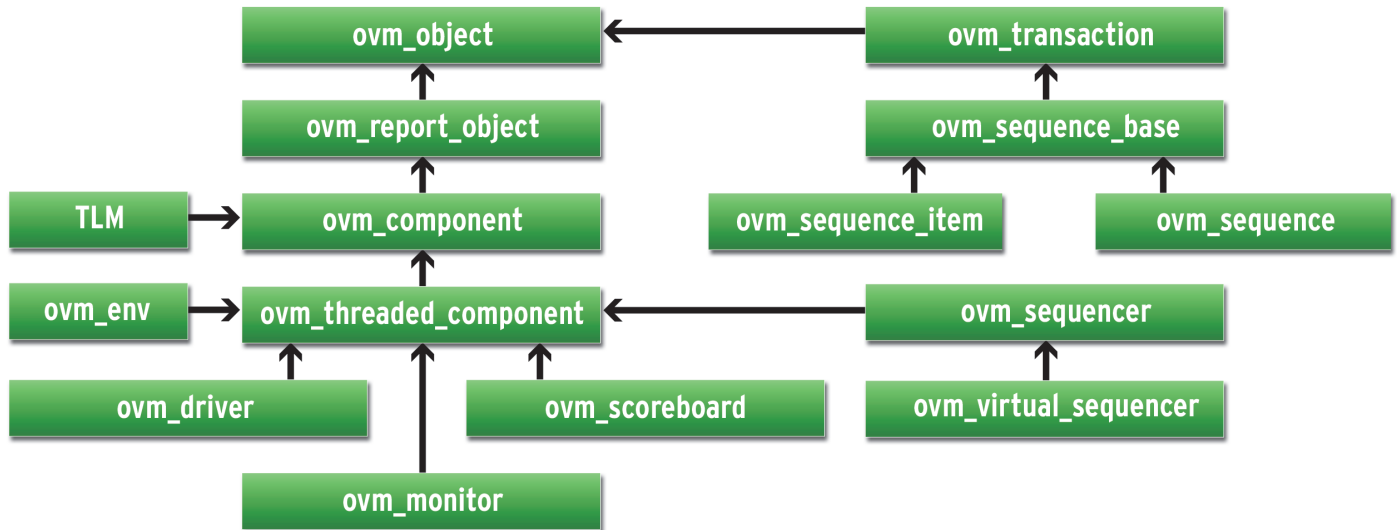


図3. OVMクラス・ライブラリ

- 簡単なテスト作成者用インターフェースにより、下層コードを変更することなく、全コンポーネントを型およびインスタンスごとにカスタマイズ可能
- テストベンチのコンフィギュレーションや制約付き階層型シーケンシャル・スティミュラスの定義のための直感的なテスト記述インターフェース
- 共通のメッセージ・レポートおよび書式インターフェース

## 機能概要

### TLM通信

OVMコンポーネントは、標準TLMインターフェースを介して通信することにより高い再利用性を実現します。TLMはインターフェース・メソッドの標準セットを定義する通信仕様であり、インターフェースとその実装とを切り離します。TLMを用いることにより、コンポーネントはTLMインターフェースを通して、そのインターフェースの実装を持つコンポーネントと通信することができます。これにより、あるコンポーネントを、異なる抽象レベルを持つ別のコンポーネントと接続することが可能です。TLM通信の共通仕様により、コンポーネントごとの交換を、他の環境に影響を与えることなく行うことができます。OVMでは、通信パスの定義と正常な接続を確実にするための組込みチェックを用意しています。

### フェーズと実行管理

複数提供者からのVIPを1つの環境に組み込むために、OVMではシミュレーション実行時に全検証コンポーネントが経由する、一連のフェーズを定義しています（図4参照）。最上位環境がnew()フェーズで生成されると、続くpost\_new()フェーズで下層コンポーネントが階層的に、インスタンス化、コンストラクト、コンフィギュレーションされます。コンポーネント間の接続はelaboration()フェーズで定義され、post\_elaboration()フェーズで接続チェックと接続解決が行われます。このpost\_elaboration()の最後で、検証環境内の全コンポーネントが配置・接続され、動作可能な状態になります。

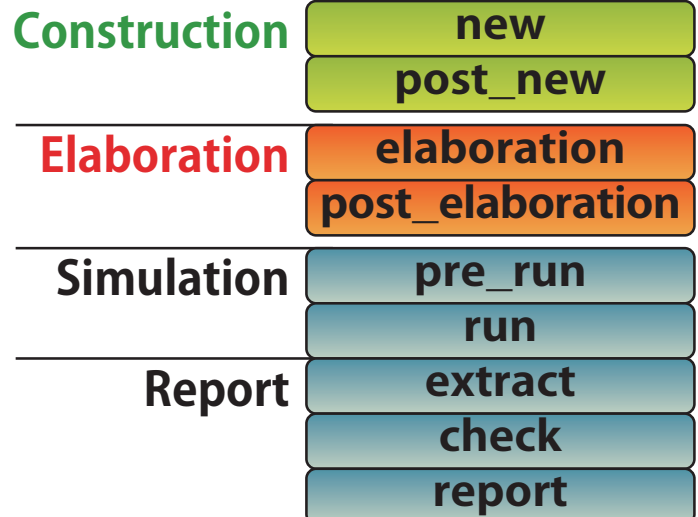


図4. OVMのシミュレーション・フェーズ

次に、pre\_run()フェーズにより、run()フェーズでのテスト実行前に、そのテストに必要な追加のカスタマイズと検証コンポーネントおよびDesign Under Test (DUT) のコンフィギュレーションを行います。taskプロセスであるrun()フェーズの終了後、次の3つのレポート・フェーズが実行されます。extract()フェーズは各コンポーネントの動作結果収集に、check()フェーズはその結果からテストのパス/フェイル状態評価に、report()フェーズは各コンポーネントがメッセージのセバリティおよびフォーマット・ルーチンを用い、テスト結果とステータスをログファイルもしくはディスプレイに表示するために使用されます。

ユーザー/プロジェクト/会社特有のフェーズを定義し、デフォルトのフェーズ・リスト内に挿入することも可能です。これにより、独自フェーズをシミュレーション時の適切なタイミングで全コンポーネントに対して実行することができます。このフェーズ実行は、各階層レベルの検証環境すべてに渡り、自動で管理されます。

## テストベンチ／環境のカスタマイズ

OVMでは、次に示されるような各コンポーネントのインスタンス化は不要です。

```
red my_a;
my_a=new("a",this);
//hard-coded instantiation/allocation
```

その代わりOVMでは、次のようにオブジェクト指向の標準技術であるクラス・ファクトリを用いて、中央集権的にコンポーネントのインスタンス化や初期化を動的に行い、再利用とカスタマイズを容易にします。

```
red my_a;
ovm_component cmp;
cmp=create_component("red","a");
//flexible instantiation
$cast(my_a,cmp);
```

OVMの全コンポーネントは、ovm\_componentクラスを継承しており、これはクラス・ファクトリのcreate\_component()メソッドの返り値の型になります。このため、その返り値の型を、特定のコンポーネント(my\_a)に対しキャストする必要があります。クラス・ファクトリは返り値を渡す前に、コンポーネントのアロケーションと初期化も行います。このクラス・ファクトリを用いる利点は、検証環境をカスタマイズするための上書きができることです。

```
ovm_factory::set_type_override("red","blue");
ovm_factory::set_inst_override("top.a","red","green");
```

set\_type\_override()メソッドは、クラス・ファクトリに対してred型が要求されるすべての場所で、blue型のコンポーネントを返すように伝えます。上記のように記述された検証環境は柔軟性を持ち、ここではredクラス型のmy\_aやその他のredクラス・インスタンスに対し、blueクラス型のコンポーネントを持たせることで、検証環境の記述を変えることなく、異なる動作をさせることができます。コンポーネント間でTLMインターフェースを用い通信のカプセル化を強化することで、この機能の実現を容易にします。blueクラス型がredクラス型と同じインターフェースを持つ限り、その他の環境コンポーネントに対する互換性が完全に保たれます。同じように、クラス・ファクトリから返されるクラス型は、set\_inst\_override()メソッドを使うことにより、インスタンス別に上書きすることも可能です。

## コンフィギュレーション

再利用に必要な要素の1つとして、コンポーネントの内容に基づいて、カスタマイズやコンフィギュレーションが可能なが挙げられます。OVMはこのプロセスの管理を、コンポーネントがそのサブコンポーネントに対しコンフィギュレーションを指定させることで実現しています。post\_new()フェーズの実行中、各コンポーネントは、その内部の各プロパティに対するコンフィギュレーション設定値の有無をチェックし、そのプロパティにその設定値をセットします。このbuildプロセスは、上位階層から下位階層へと順に行われ、各コンポーネントがコン

フィギュレーションやサブコンポーネントのインスタンス化の変更を可能にしています。

```
class my_env extends ovm_env;
block b;
...
function void build(); // called from post_new()
    set_config_int("b","is_active",0);
    ...
    b.build();
endfunction
endclass

class block extends ovm_component;
driver d;
bit is_active = 1;
function void build();
    ovm_component cmp;
    if(get_config_int("is_active",is_active))
        if(is_active)begin
            cmp=create_component("driver","d");
            $cast(d,cmp);
            d.build();
        end
endfunction
endclass
```

上記の例では、検証環境はblockクラス型に対し、is\_activeビットを0に設定するようにコンフィギュレーションを与えています。全OVMコンポーネントは、各々に対するコンフィギュレーション情報を受け取る必要があるため、blockコンポーネントのbuildプロセス中に、グローバルのデータ・テーブルからis\_activeの値をチェックします。値が見つかった場合にはその値が使用され、見つからない場合はデフォルトの値が使用されます。コンフィギュレーション値はintやstring、ovm\_object型が取り得ます。そのため、このテキスト・ベースのインターフェース（名前のワイルド・カードも可能）により、使用している各コンポーネントに応じて、どのような情報に対してもコンフィギュレーション値の設定を行うことができます。

この例では、blockはインアクティブ (is\_active==0) に設定されています。blockコンポーネントはbuildプロセスを継続し、このコンフィギュレーション情報を用いてそのサブコンポーネントのコンフィギュレーションを制御します。この場合、is\_activeの値を用いてdriverコンポーネントをインスタンス化するかどうかを制御します。

これでこの1つのblockコンポーネントは、多くの異なる環境において再利用し制御することが可能になり、各コンポーネントがdriverとしてアクティブ化するかどうかを選択することができます。コンポーネントがこのような柔軟性を持つように設計することで、その内部の詳細を変更せずに再利用することが可能になり、検証コンポーネントのライブラリ作成（または購入）がしやすくなります。

## 階層化されたスティミラス・シーケンス

ある特定のテストのカスタマイズを容易にするOVMの第3の方法は、実行されるスティミュラスそのものを設定することです。OVMは、検証環境インフラストラクチャの詳細な知識を持っていなくても、意図するトランザクションのスティミラス・パターンを迅速に作成することができます。この機能により、検証のエキスパートでなくとも、様々なテストや検証環境トポロジおよびプロジェクトで再利用可能なテスト・シナリオを素早く作成することが可能です。スティミラス・シーケンスは、純粋なダイレクト・テスト手法から、前述したクラス・ファクトリによる、制約の階層化を可能にする制約付きランダム手法まで、幅広く対応することができます。

定義されたスティミラス・シーケンスは、より大きく複雑なテスト・シナリオを構成するためのサブセットとして再利用可能です。スティミラス・シーケンスの組み合わせにより、デザインの検証するための様々なシナリオを実行することができます。複雑なプロトコルは、シーケンスを階層的にレイヤ化し、各階層における完全な抽象度を持たせることでモデル化が可能です。より大きな検証環境に対しては、バーチャル・シーケンスと呼ばれる中央制御メカニズムで、複数のインターフェースを協調制御することが出来ます。

## まとめ

検証プロジェクトの成功には、標準言語以上のものが要求されます。最先端のテストベンチを構築するために相互運用性を確実にし、検証の再利用を促進させるためには、高度化されたメソドロジが必須です。既に広く使われている複数の非互換な検証メソドロジに対し、業界が「メソドロジ戦争」を終息させるためのEDAベンダーの相互協力を求めていることは明らかです。

このメンター・グラフィックスとケイデンスの共同開発と共同承認は、業界の懸念に対する答えであり、OVMに信頼性と実用性をもたらしました。OVMはまさに、相互運用性を持ち、オープンで、実績のある唯一のメソドロジなのです。OVMのリリースとともに、もうこれからは、メソドロジ選択について悩む必要はなくなります。再利用性、相互運用性を考えたとき、OVMこそがベスト・ソリューションであることは明白です。

Copyright © 2008 Mentor Graphics Corporation and Cadence Design Systems, Inc.  
All company or product names are the registered trademarks or trademarks of their respective owners.

**cadence™**

日本ケイデンス・デザイン・システムズ社  
〒222-0033  
神奈川県横浜市港北区新横浜3-17-6  
営業本部・ペリフィケーション営業本部  
TEL: (045)-475-8410, (045)-474-9407  
<http://www.cadence.co.jp>

**Mentor  
Graphics®**

メンター・グラフィックス・ジャパン株式会社  
〒140-0001  
東京都品川区北品川4丁目7番35号  
御殿山ガーデン  
TEL: (03)-5488-3030 (代表)  
<http://www.mentorg.co.jp>