デザインシミュレーションモデル
フローインテグレーションガイド

ARM®

ARM DUI0219AJ-00

デザインシミュレーションモデル
フローインテグレーションガイド

リリース情報

以下の表は、本書のリリース情報および改定履歴を示しています。

改訂履歴

日付	発行	改訂
2003 年 4 月 17 日	A	初版

著作権

守秘義務

本書は誰でも閲覧することができます。配布に関する制限はありません。

製品ステータス

本書には最終情報（完成製品に関する情報）が記載されています。

ARM ホームページ

`http://www.arm.com`

目次
デザインシミュレーションモデル
フローインテグレーションガイド

図一覧
デザインシミュレーションモデル
フローインテグレーションガイド

序章

本章では、*デザインシミュレーションモデル*（DSM）について概説します。本章は以下のセクションから構成されています。

- *本書について*：P. viii
- *ご意見*：P. xi

本書について

本ユーザガイドには、*ARM デザインシミュレーションモデル*（DSM）に関する以下の情報が記載されています。

- DSM の用途
- 特徴
- シミュレータとのインタフェースと関係
- タイミングに関する注意点
- ネイティブの HDL コードを使用した場合と比較したときの DSM の制約

対象となる読者

本書は、Verilog または VHDL を熟知し、ARM 製品を使用する目的で ARM DSM を設計・シミュレーションフローに取り入れようとするハードウェア / ソフトウェアエンジニアおよびチップ設計者を対象に書かれています。

本書の構成

本書は以下の章から構成されています。

第 1 章*「はじめに」*

DSM の概要、主な特徴、DSM パッケージの内容について説明します。

第 2 章*「インタフェースと DSM のビヘイビアレベルでの相違点」*

シミュレータとのインタフェース、モデルマネージャ、イベントおよびインタフェースのセマンティクス、コンパイルされたモデルと RTL との違いとその意味、ならびにプログラマモデルについて説明します。

第 3 章*「タイミングに関する注意点」*

DSM のタイミングに関する注意点について説明します。SDF アノテーション、テンプレート、SDFremap についてはこの章を参照して下さい。また、ピンからピンへのタイミング、相互接続遅延、ネガティブタイミングチェック、スタティックタイミング分析、タイミングモデルの制約についても説明しています。

第 4 章*「制約事項」*

DSM の制約について説明します。再起動、保存、復元の手順、スキャンチェインのモデリング、キャッシュ、ならびにゼロ遅延シミュレーションについて説明しています。

表記規則

本書では、以下の表記規則を用いています。

italic 重要事項、重要用語、相互参照、引用箇所を斜体で記載しています。

bold メニュー名などのインタフェース要素を太字で記載しています。ARM プロセッサシグナルの名前にも使用しています。また、必要に応じて表中の説明文に含まれる専門用語にも太字を用いています。

`monospace` コマンド、ファイル名、プログラム名、ソースコードなどの、キーボードから入力可能なテキストを示しています。

`mono`space コマンドまたはオプションに使用可能な略語を示しています。コマンドやオプションの名前を全部入力する代わりに、下線部分のテキストだけを入力してこれらを指定できます。

`monospace italic`

コマンドまたは関数への引数で、特定の値に置き換えられることが可能なものをタイプライター書体の太字で記載しています。

`monospace bold` サンプルコード以外で使用されている場合は、言語のキーワードをタイプライター書体の太字で記載しています。

参考資料

このセクションでは、ARM モデルの開発コードに関する補足情報を提供している ARM 社および各社の出版物を紹介します。

ARM は資料の定期的な更新・修正を行っています。最新の正誤表と追補情報、ならびによく寄せられる質問とその回答については、ARM ホームページ http://www.arm.com をご覧下さい。

ARM の出版物

以下の資料には、DSM フローインテグレーションガイドを使用するエンジニアに有用な情報が記載されています。

- *ARM 設計サインオフモデル：タイミングアノテーション*（EDA QPRO 0001 A）

他の出版物

その他、以下の関連資料を参照して下さい。

- IEEE 規格 1076.4 - 1995 IEEE *Standard for VITAL Application-Specific Integrated Circuit* (ASIC) *Modeling Specification*（VITAL-95 としても知られています。）
- IEEE 規格 1364 - 1995 IEEE *Standard Hardware Description Language based on the Verilog Hardware Description Language*

- OVI SDF2.1 *Open Verilog International Standard Delay Format Specification Version 2.1*（1994 年 2 月、7 月発行）
- SDFremap Manpage（UNIX コマンド：man sdfremap）

ご意見

ARM 社では、DSM および本書に関するご意見等をお待ちしています。

DSM に関するご意見

DSM に関するご意見等がございましたら、製品購入元までご連絡下さい。その際、迅速かつ適切な対応をさせて頂くために、以下の情報をご用意下さい。

- 使用している製品のリリース情報
- ハードウェアプラットフォーム、オペレーティングシステムのタイプ、バージョンなど、ご使用になっているプラットフォームについての詳しい情報
- 当該の問題が繰り返し発生する、スタンドアロンの小さなサンプルコード
- 期待した結果と、実際に起こった結果に関する詳しい説明
- 使用したすべてのコマンドとコマンドラインオプション
- 問題がわかるサンプル出力
- バージョン番号、日付を含め、使用しているツールのバージョン情報

本書に関するご意見

本書に関するご意見等がございましたら、電子メールに以下の情報をご記入の上、`errata@arm.com` までお寄せ下さい。

- 資料名
- 資料番号
- ご意見のあるページ番号
- 問題点の詳しい説明

補足すべき点または改善すべき点についてのご提案もお待ちしています。

第 1 章
はじめに

本章では DSM について概説します。本章は以下のセクションから構成されています。

1.1 DSM について

DSM は、ターゲットの HDL シミュレータに組み込むことが可能なバックアノテーション機能に対応した（タイミングに厳密な）シミュレーションモデルです。シミュレータとホストプラットフォームごとに固有の DSM を使用します。DSM は ARM コア設計のアーキテクチャおよび機能と完全に整合しています。ARM では、各 ARM コアに関し *Architecture Validation Suite*（AVS）と *Device Validation Suite*（DVS）で DSM を検証することによって整合性を確認しています。

ARM コアの RTL 設計をベースとする DSM は、業界標準の各種 Verilog および VHDL シミュレータに対応しており、シミュレータの SDF アノテーション機能によりタイミングデータを処理することができます。DSM の実行速度は、シミュレータのインタフェース効率と、モデルを構築する設計の複雑度により、5 〜 500 サイクル / 秒となります。

DSM は以下の 2 つの部分から構成されます。

- 機能コアブロック
- タイミングシェルが組み込まれた Verilog または VHDL ラッパ

  このラッパではホストシミュレータの別の言語インタフェースが使用され、その機能モデルのインスタンス化が行われます。

  DSM は通常、Synopsys 社の *Verilog Model Compiler*（VMC）などのコンパイラを使用して、ARM 用に設計された RTL ソースから作成されます。これに命令の逆アセンブリといった特別な機能が ARM によって付加されています。

  ARM が開発したテクノロジを使用して DSM がラッパに接続されるので、1 つのコンパイル済みモデルを様々なロジックシミュレータに使用することができます。ARM によって開発されたこのテクノロジにより、タイミングシェルも容易に追加することができます。

一般的に DSM には、RTL モデルでの実現が難しい命令の逆アセンブラなどの、ビヘイビアレベルのデバッグ機能が組み込まれています。そのため、コンパイル済みのモデルを使用する場合に、知的財産を守りながらモデルを配布することができます。

注

合成可能コアの場合、DSM は実装前のモデルであり、サインオフモデルではありません。

1.1.1 シミュレーションの不確実性

コンパイル済みのモデルの特性と、ピン間でタイミングラッパを使用することから、同様の条件で使用される RTL コードと比較してこれらのモデルの使用方法と動作が異なってくる場合があります。これらの違いのいくつかは、同時イベントがモデルによっ

て処理される順序が変わるといった微細なものですが、タイミングを取る目的でコアを 1 つのブロックとして特性付ける必要があるなど、大きな違いもあります。

1.1.2 ARM DSM の特徴

ARM DSM の特徴は以下の通りです。

フルデバイス機能

DSM は ARM コア設計のアーキテクチャおよび機能と完全に整合しています。

フェーズに厳密

DSM のサイクル間タイミングは、使用する ARM コアのタイミングと同じとみなすことができます。フェーズに厳密であることは、ARM コア向け DVS によってほぼ証明されています。

レジスタの可視性

DSM によってコア内のレジスタのデバッグプロセスを可視化することができます。アーキテクチャ上表現されるすべてのモードのコアのレジスタセットを、DSM 内の特殊な VHDL 階層または Verilog 階層で可視化できます。これらのレジスタは、シミュレーションのトレースに使用することができます。

キャッシュとメモリのサイズ設定

特定の DSM インスタンスごとに、キャッシュまたは TCM のサイズを必要に応じて設定することができます。

タイミングとバックアノテーション

ARM は、VITAL 95 または Verilog SDF 標準を使用した SDF からのバックアノテーションをサポートしています。

注

場合によっては DSM に含まれるツールを使用して SDF のポストプロセシングを行う必要があります。

逆アセンブラ

一部の DSM には逆アセンブラが組み込まれています。逆アセンブラを使用できるかどうかは、DSM の対象となるコアの RTL に適切な実行トレーサが含まれているかにより、コアごとに異なります。

1.2 DSM パッケージの内容

各 DSM は以下のコンポーネントで構成されています。

- テンプレートタイミングファイル「.sdft」
- ARM *Condensed Reference Format*（CRF）プロダクションテストベクタを実行するためのテストモデル
- モデル関数を検証するための CRF テストベクタ群「.crf」と「.ctrm」
- ドキュメンテーション
- SDF からタイミングシェルへのバックアノテーションが正しいことを検証するためのユーティリティ「sdfremap」
- モデルマネージャとなる 1 つ以上の共有ライブラリ「.so」または「.sl」ファイル
- DSM 自体は以下のコンポーネントで構成されています。
  - コンパイル済みコア（共有ライブラリ）「.so」または「.sl」ファイル
  - 必要に応じて他社製のコンパイラを使用して生成された 1 つ以上の SWIFT または *Open Model Interface*（OMI）モジュール
  - HDL（Verilog または VHDL）ラッパ「.v」または「.vhd」
  - 各シミュレータに使用される固有のファイル

第 2 章
インタフェースと DSM のビヘイビアレベルでの相違点

本章では DSM の使用方法について説明します。本章は以下のセクションから構成されています。

2.1 シミュレータのインタフェース

VHDL および Verilog シミュレータは、他言語で記述されたコードをシミュレーションに組み込むためのインタフェースを備えています。Verilog シミュレータの場合、通常はプログラミング言語インタフェース（PLI）が使用されます。VHDL シミュレータの場合はシミュレータ固有のインタフェースが使用されますが、一般的には同様の機能を備えています。通常これらのインタフェースによって、シミュレータの制御やシミュレータとのインタラクションを低レベルで実現できます。インタフェースを適切に使用することで、HDL で記述されたコンポーネントを他言語で実装することができます。ARM DSM ではこのメカニズムを利用して、コンパイル済みのモデルを HDL/RTL シミュレーションのコンポーネントとして動作させることができます。

以下のセクションでは、シミュレータのインタフェースについて詳しく説明します。

- *モデルとシミュレータのリンケージ*
- *DSM のイベントとインタフェースのセマンティクス*：P. 2-4

2.1.1 モデルとシミュレータのリンケージ

ARM DSM のコアは、コンパイルされるホストプラットフォーム（Linux や Solaris など）ごとに異なりますが、シミュレータと言語には依存しません。モデルは DSM に含まれているシミュレータ固有のモデルマネージャによってシミュレータに接続されます。モデルマネージャによってモデルとシミュレータとの間のイベントトランザクションが処理されます。シミュレータは HDL ラッパを介してモデルマネージャを呼び出します。HDL ラッパも DSM に含まれています。HDL ラッパは、コアをロジックシミュレータに組み込む場合の外部モジュール / エンティティ（適切な TRM の定義によってコアへ接続される）として機能します

モデルマネージャと DSM は、オペレーティングシステム（OS）固有の共有オブジェクトファイルとして配布されます。HDL シミュレータがモデルマネージャをロードし、モデルマネージャが DSM をシステムにロードします（P. 2-3 図 2-1 参照）。

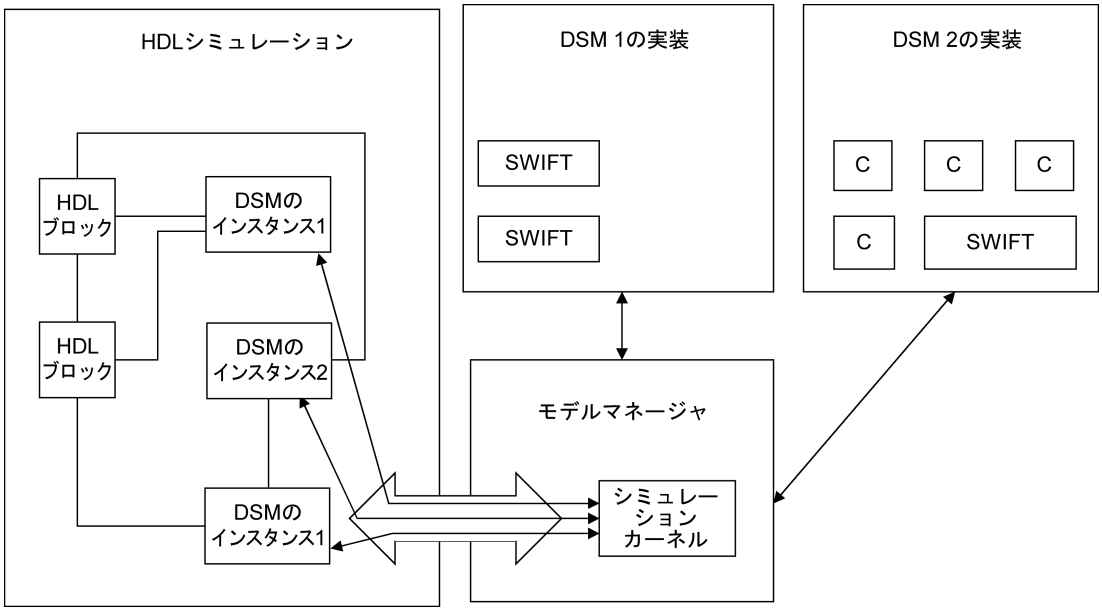

図 2-1 シミュレーションの構造

VMC でコンパイルされたコンポーネントは、SWIFT と呼ばれるシノプシスインタフェース標準を使用することでシミュレーションに組み込まれます。SWIFT によってシミュレータは VMC とその他のモデルタイプを統合することができます。ARM モデルマネージャでは、シミュレータの SWIFT インタフェースは使用されません。ARM モデルマネージャには、ホストシミュレータによって提供される専用の SWIFT インタフェースが使用されます。

DSM SWIFT オブジェクトを呼び出す場合に、シミュレータ内の SWIFT インタフェースを使用しないで下さい。

ホストシミュレータにVMCモデルを登録するために特別なことをする必要はありませんが、ARM モデルマネージャがこれらのモデルを参照できるように環境を正しく設定する必要があります。

DSM 共有オブジェクトはモデルマネージャによって処理されるため、シミュレータが DSM を直接処理することはありません。シミュレーションで複数の異なる DSM が使用されている場合でも、1 回のシミュレーションにおけるすべての DSM が 1 つのモデルマネージャによって管理され、各 DSM の特性がシミュレータに通知されます。シミュレータの観点から見ると、モデルマネージャは外部のモデルとなります。

モデルマネージャは、外部でコンパイルされたコードとしてホストシミュレータに登録する必要があります。この登録方法はシミュレータによって異なります。シミュレータは特別なスイッチが指定されて起動されるか、コンフィギュレーションファイルが使用されます。詳細については、DSM に付属しているインストールガイドを参照して下さい。その他の関連情報については、使用するシミュレータのユーザガイドを参照して下さい。

注

複数の DSM とのインタフェースに 1 つのモデルマネージャが使用されるため、通常はモデルマネージャをホストシミュレータに何度も登録する必要はありません。シミュレーションで、別のベンダーでコンパイルされたコードを使用する場合には、それらのコードを共存させるために、シミュレータの設定と使用方法がやや複雑になる場合があります。

詳細については、DSM リリースの DOCS ディレクトリ内にあるモデルマネージャのアプリケーションノートと、ベンダーから提供されているシミュレータのマニュアルを参照して下さい。

2.1.2 DSM のイベントとインタフェースのセマンティクス

DSM はタイミングのバックアノテーションをサポートしていますが、コンパイル済みのコアから生成されるシグナルの遅延は、コンパイル済みのコアがすでにドライブしたイベントを遅延させることで実際にタイミングシェルに適用されます。モデルマネージャとホストシミュレータとの間のインタフェースのセマンティクスにより、コアそのものは厳密に遅延がゼロとなる必要があります。

入力イベント（入力シグナルの値への変更）は、ホストシミュレータによってモデルマネージャに登録されます。DSM には実質的に 1 つの動作プロセスしかありません。起動された DSM は、その入力値に基づいて出力値の変化を計算します。

モデルマネージャはホストシミュレータによって起動されますが、DSM からの新しい出力がシミュレーションに読み込まれるまでシミュレータからの制御はありません。DSM は、新しい入力が与えられるとすぐに新しい出力を生成する反応モデルです。生成される出力と与えられる入力イベントは、ホストシミュレータのドメイン内で動作するタイミングシェルによって遅延させることができます。

HDL シミュレータは様々なメカニズムを利用して、ハードウェアの同時性をシミュレートします。状況によっては、この同時性がいわゆる競合の原因となる場合があり、同時イベントが評価される順序がシミュレーションの結果に影響を及ぼす可能性があります。

競合状態は、指定されたイベントキューのセマンティクスを使用することにより、HDL シミュレーションから排除されます。ほぼ同時に発生するイベントの処理に関するシミュレータの動作は、ハードウェア記述言語（VHDL または Verilog）で記述されます。たとえば VHDL では、ロジックの評価を、連続するシミュレーションデルタ内で行う

デルタ掃引により、潜在的な問題が処理されます。動作の一貫性を保つため、ネットリスト内のノードはデルタ間でのみ変更することが許可されます。Verilog ではこのような厳密なメカニズムは使用されませんが、その効果を、いわゆるノンブロッキングアサインメントを使用することでエミュレートできます。

2.2 コンパイル済みモデルの使用に起因する相違点

コンパイル済みモデルを使用する場合は、ネイティブの HDL コードを使用する場合には起こらないような問題に直面する可能性があります。これらの問題は、DSM と HDL の境界のイベントセマンティクスに関連した難しい問題に加え、コンパイル済みの他言語コードを組み込むシミュレーションの複雑さに起因しています。

以下のセクションでは、この相違点について説明します。

- *モデルとシミュレータのリンケージに関する問題点*
- *DSM のイベントとインタフェースのセマンティクスに関する問題点*：P. 2-7
- *コンパイル済みモデルのサポート*：P. 2-8

2.2.1 モデルとシミュレータのリンケージに関する問題点

複数種類のコンパイル済みコードを使用する場合に直面する可能性のある問題の一つは、それらのコードが共存できず、他のコンパイル済みコードの動作に干渉することです。通常、干渉は以下の 2 つのケースで発生します。

- コンパイル済みオブジェクトがそれ自体は正常に機能していても、コンパイル済みの別のオブジェクトと干渉する。

  この傾向は Verilog PLI オブジェクトによく見られます。なぜなら、PLI では一般的な VHDL インタフェースほどにはコンテキストの分離が行われないためです。ARM モデルマネージャは、起動時にシミュレータを既知の状態に設定するように記述されていますが、すべての PLI モデルがこのアプローチで正確に動作するとは限りません。これが原因で問題が生じる場合、時間のスケールに問題がある傾向にあります。

  この種の問題は、DSM に VMC でコンパイルされたコンポーネントが含まれている場合に、シミュレータのネイティブ SWIFT インタフェースを介して他の VMC コンポーネントがシミュレーションに取り込まれている場合にも発生する可能性があります。特に、ARM が提供する VMC コンポーネントが、DSM を介してインスタンス化され、ホストシミュレータ内でも直接インスタンス化されると、このコンポーネントは動作不可能となります。
- 1 つのコンパイル済みオブジェクト内のバグが別のオブジェクトのスタックまたはヒープを破損し、不正な動作やクラッシュの原因となる可能性があります。

  一般的にこの問題を診断したりデバッグすることは困難です。なぜなら、現在サポートされているすべてのシミュレータにおいて、使用されるシミュレータ、モデルマネージャ、DSM、VMC モデル、その他の他言語コードのすべてが同じアドレス空間で動作するためです。いずれか 1 つで発生したバグが、他のコンポーネントを破損する可能性があります。また、ヒープが管理される方法により、シミュレーションにさらにコードが追加されるまでこのような問題が検出されない可能性もあります。

DSMと他言語コードが含まれるシミュレーションが正しく機能しない場合には、可能な限りその他言語コードを除いてシミュレーションを試行して下さい。この方法で、他言語オブジェクトが不正な動作の原因かどうかを見極められる場合があります。問題が引き続き発生する場合は、DSM プロバイダーのテクニカルサポートに問い合わせて下さい。

2.2.2 DSM のイベントとインタフェースのセマンティクスに関する問題点

DSM 内部でのイベントの処理が外部のホストシミュレータに似たセマンティクスによって制御されるとき、他言語インタフェースのレベルで問題が発生する可能性があります。他言語のオブジェクトとのイベントの受け渡しを処理する場合、シミュレータが自らのイベントセマンティクスをどこまでサポートして保存するかは、シミュレータによって異なります。一般に VHDL シミュレータは、VHDL コードの場合と同じデルタ掃引セマンティクスを他言語オブジェクトに適用する傾向にありますが、VHDL シミュレータには多くの種類の他言語インタフェースがあるため、必ずしもそうなるとは限りません。

Verilog 言語では、他言語オブジェクトインタフェース（PLI）がその標準の一部として規定されていますが、Verilog シミュレータが異なれば他言語オブジェクトにイベントを渡す方法も異なるため、その両方が名目上は同じコードを実行する場合でも、一貫性が保証されるとは限りません。このことは、クロックとサンプリングされるデータシグナルの間にシミュレーション遅延を挿入する代わりにクロックとデータシグナルを分離するシミュレータのイベントキューのセマンティクスのみに依存するテストハーネスは、DSM と組み合わせて使用したときには正しく機能しない場合があることを意味します。なぜなら、シミュレータによってはノンブロッキングアサインメント前のイベントと、ノンブロッキングアサインメント後のイベントを、1 つの変化群として DSM に通知するためです。例：

```
module top ();

reg CLK;
reg I;

initial
begin
#10;
CLK <= 0;
#10;
CLK <= 1;
end

always @(posedge CLK)
begin
I <= 1;
end

ARMDSM theDSM (CLK, I);
```



```
endmodule
```

一部の Verilog シミュレータは、ノンブロッキングアサインメントを使用していても、20 ピコ秒の単一イベントとして **CLK** と **I** に生じた変化を渡します。このような場合には、ノンブロッキングアサインメントの変わりに、たとえば 1 ピコ秒の小さなシミュレーション遅延を挿入して、イベントを正しく分離する必要があります。これによって動作テストベンチの設計方法も変わってきます。

2.2.3 コンパイル済みモデルのサポート

DSM は、シノプシス VMC などのモデルコンパイラを使用してモデリングするデバイスの RTL ソースから作成されるのが一般的です。コンパイルされた RTL を DSM 内でどのように使用するかにより、以下のサポートのレベルが異なってきます。

- コンフィギュレーション
- 内部ノードの可視性

一部の ARM コア、特に合成可能な ARM コアは、パラメータまたは Verilog `'define` ディレクティブの使用により、コンパイル中あるいは合成中に構成することができます。コンパイル完了後、この方法で行われたコンフィギュレーションは固定され、その後に変更することはできません。このため、RTL コードを他の目的で構成する場合と同じ方法で、DSM 内で RTL コードを構成することは通常は不可能です。DSM によっては、C 言語の機能を DSM に追加することでコンフィギュレーションができるものもあります。サポートされているコンフィギュレーションオプションの詳細については、使用するモデルの DSM リリースのドキュメンテーションを参照して下さい。

モデルがコンパイルされると、ホストシミュレータはそれ自身のシミュレーション構造をプローブすることができません。したがって、デバッグ可視のために、モデル内のノードを検査することはできません。しかし DSM には、デバイス上のピンとしてではなくシミュレーションラッパにエクスポートされる特定の内部ノードが組み込まれています。通常これらのノードには、レジスタセットとその他の直接関係する情報が含まれています。これらのレジスタは HDL ラッパ内部で Verilog レジスタまたは VHDL シグナルにマッピングされ、ロジックシミュレータによって検査することができます。

注

DSM は実設計を表すため、エクスポートされるこれらのシグナルは、シミュレーション中も常に実際にレジスタ内に存在する値を表しているのが普通です。この点は、実装が最適化されるため、抽象的なプログラマモデルには当てはまらないかもしれません。フォワーディングパスを使用して他のブロックからアクセスされたとき、レジスタバンクの更新のために新しいレジスタ値がパイプラインに存在することがあります。したがって、プログラマモデルが必ずコアの真のステータスを反映するとは限りません。

第 3 章
タイミングに関する注意点

ARM シミュレーションモデルとサインオフモデルは、VHDL シミュレータおよびVerilogシミュレータのSDFタイミングアノテーション機能を使用することにより、バックアノテーションフローに統合できます。これらのモデルは、ゼロ遅延実行モデルの、大規模な単一コンポーネントです。タイミングフローにこれらのモデルを正しく統合するのは難しく、実現できるタイミング精度の程度にも影響します。本章ではこれらの問題点と、アノテーションプロセスの支援機能のいくつかについて説明します。本章は以下のセクションから構成されています。

3.1 Pin-to-Pin タイミングモデル

シミュレータの観点からは、ARM モデルコアはシミュレーション内の大規模な単一コンポーネントです。ARM モデルコアはその入力のすべてを検知し、論理上は、そのいずれの入力の変化に対しても、出力値を生成できます（つまり、大規模な組み合わせブロックとして扱われます）。このモデルとシミュレータでは、クロックシグナルを特別扱いしません。クロックシグナルは、他のシグナルと同様に、出力イベントを生成させるための入力シグナルとみなされます。

タイミングアノテーションは、HDL タイミングラッパをこのモデル周辺に配置することによって実現されます。このモデルはゼロ遅延モデルとして規定されているため、タイミング動作は完全に SDF ファイルからのバックアノテーションによって決定されます。タイミングシェルの動作を例示する以下の 3 つのシナリオが記述されています。

- *ゼロ遅延モデルコア*

  最初に、タイミング機能を使用しない単純な同期ブロックの動作が記述されています。
- *タイミングシェルの追加* : P. 3-3

  このモデルは周辺に外部 Pin-to-Pin タイミングシェルが追加されており、出力の遅延の他に、セットアップ / ホールドチェックが実装されています。ここでは、タイミングシェルの追加がモデルの機能に影響しないことを示しています。
- *ネガティブタイミング制約* : P. 3-6

  負のセットアップ / ホールド時間など、ネガティブタイミング制約があるかどうかを検査します。このシナリオでは、ゼロ遅延コアと組み合わせて使用される単純な Pin-to-Pin タイミングシェルは、ネガティブタイミング制約のモデリングには使用できないことを示しています。ネガティブタイミング制約をサポートするための基本実装の修正については、P. 3-19*「ネガティブタイミングチェックの使用」*を参照して下さい。

3.1.1 ゼロ遅延モデルコア

最初の例では、以下のシグナルを使用する単純な同期ブロックを示します。

- クロック **CLK**
- 入力シグナル **IN**。クロックの立ち上がりエッジでサンプリングされます。
- 出力シグナル **OUT**。クロックの立ち下がりエッジでドライブされます。

図 3-1 は、クロックを使用する単純な同期ブロックを示しています。

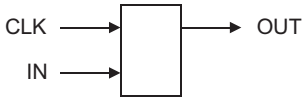

図 3-1 クロックを使用する単純な同期ブロック

タイミングシェルがないモデルの観点から見たイベントシーケンスは以下の通りです。

1. すべてのシグナルがゼロにセットされます。
2. 10ps で入力 **IN** が 1 に変化します。ブロックは **IN** の変化を検知するためモデルが評価されますが、ブロックの動作としては何のアクションも取られません。
3. 20ps で、クロック **CLK** が 1 に変化します。モデルが評価され、**IN** の値が内部でラッチされます。
4. 25ps で、**IN** が 0 に降下します。モデルが評価されますが、何のアクションも取られません。
5. 30ps で、**CLK** が 0 に降下します。モデルが評価され、**OUT** へのドライブが必要であることが決定されます。
6. 同様に 30ps で、ただし次のシミュレーション評価で、ブロックモデルからのドライブの結果として、出力 **OUT** が 1 に上昇します。

上記のイベントシーケンスは、タイミングシェルを使用しない、ゼロ遅延のビヘイビアモデル（DSM）です。このモデルはすべての入力を検知し、すぐに出力を生成します。DSM のシミュレーションセマンティクスにより、このシミュレーションで他に DSM 評価と並行して評価されるものはありません。つまり、シミュレーション内の他の機能ブロックは、同じシミュレーション時間で入力をドライブするように設定されている場合でも、個々の DSM 評価ステップの間に値を挿入することはできません。

3.1.2 タイミングシェルの追加

この例では、Pin-to-Pin タイミングシェルがモデルの周辺に配置されています。タイミングシェルはモデルと同様にビヘイビアコードです。タイミングシェルは、入力イベントの受け渡しに影響を及ぼすこと（単純なケースのみ）なくこれらのイベントを監視し（P. 3-19 *「ネガティブタイミングチェックの使用」*および P. 3-17 *「相互接続遅延」*参照）、出力イベントを出力する際の遅延バッファを挿入します。ここではタイミングシェルが以下の SDF フラグメントからアノテーションされると仮定します。

```
(SETUP (POSEDGE CLK) IN (5))
(HOLD (POSEDGE CLK) IN (6))
(IOPATH (NEGEDGE CLK) OUT (3))
```

タイミングシェルには、ラップされるデバイスの動作に関する情報は保持されません。また、シグナルの使用目的（クロックに同期してサンプリングされる入力など）に関する情報も保持されません。

この場合のイベントシーケンスとその結果は以下のように変更されます。

1. 最初にすべてのシグナルがゼロにセットされます。
2. 10ps で入力 **IN** が 1 に上昇します。タイミングシェルは、カレントのシミュレーション時間である 10ps を、**IN** が変化するときの時間として記録し、その変化を基礎モデルに渡します。この時点では以前と同様に動作し、何のアクションも取られません。

3. 20ps でクロック **CLK** が 1 に上昇します。タイミングシェルは、カレントのシミュレーション時間である 20ps を、クロックの立ち上がり時間として記録します。ここで、カレントのシミュレーション時間である 20ps から、**IN** が変化した時間として記録された 10ps を減算することで、クロックのポジティブエッジに対する **IN** のセットアップチェックが実行されます。記録された 10ps はセットアップ時間の 5ps よりも長いため、警告は生成されません。タイミングシェルは従来どおり、クロックの変化イベントを、**IN** の値をラッチする基礎モデルに渡します。
4. 25ps で **IN** が 0 に下降します。タイミングシェルは、**IN** が変化したカレントのシミュレーション時間を記録し、その前の値である 10ps と置き換えます。次に、カレントのシミュレーション時間である 25ps からクロック立ち上がり時間の 20ps を引くことによってホールドチェックを実行し、クロックのポジティブエッジに対する **IN** の実際のホールド時間が 5ps であったとして記録します。この時間は SDF ファイル内に記述されている 6ps よりも短いため、ホールド違反メッセージがシミュレーションログに出力されます。**IN** で発生したイベントが前述のようにモデルに渡されるため、ホールド違反によって基礎モデルの動作が影響されることはありません。タイミング違反が原因で、出力が X に変化することはありません。
5. 30ps で **CLK** が 0 に下降します。タイミングシェルは、この時間をクロックの下降時間として記録します。これ以上のアクションは必要ありません。モデルが評価され、**OUT** へのドライブが必要であることが決定されます。
6. 同様に 30ps で、ただし次のシミュレーション評価で、ブロックモデルからのドライブの結果として、出力 **OUT** が 1 に上昇します。タイミングシェルはモデルからのこのドライブを検知し、クロックのネガティブエッジが前のシミュレーション評価から生成された場合でも、カレントのシミュレーション時間で発生したものとして記録します。このため、タイミングシェルの外側のエッジに **OUT** の伝達遅延 3ps が挿入されます。
7. 33ps で、遅延された **OUT** がタイミングシェルの外側のエッジからシミュレーションの他の部分に伝達されます（P. 3-5 図 3-2 参照）。

注

実際にはモデルコアがこの出力を 30ps シミュレーション時間でドライブし、その後の遅延はタイミングシェルによって実装されます。

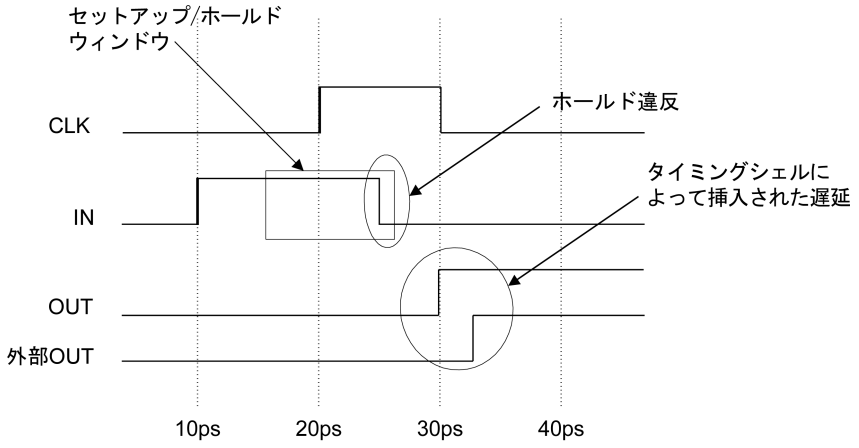

図 3-2 修正イベントシーケンス

イベントシミュレーション時、上記の動作は物理ハードウェアによって異なります。まず、デバイスによる **IN** のサンプリングのホールドウィンドウ違反がどの時点で発生したのかを検査します。

- 物理ハードウェアでは、**IN** の正しい値が実際にサンプリングされ、出力の評価で使用されることを保証できないため、ホールドウィンドウ違反が起こるとその後のデバイスの動作が予測不能となります。
- イベントシミュレーションでは、同期入力のサンプリングがクロックエッジで個別のイベントとして発生します。その結果発生するホールド違反はシミュレーションのその後の動作に影響しない表面的な警告であり、それ以後のシミュレーション結果は、実際と同じだとは仮定できないと考える必要があります。

注

ARM DSM では、タイミング違反が発生しても出力は X にドライブされません。

次にこの例では、モデルがすべての関連情報を受け取るとすぐに出力を評価します。これはクロックのネガティブエッジイベントが受信されたときに発生します。

- 物理ハードウェアの場合、出力 **OUT** の 3ps の遅延は、デバイス内の内部パスの遅延と、出力によってドライブされるロードに起因します。

- このイベントシミュレーションでは、新しい値がすぐに計算され、タイミングシェルがイベント発生後の累積遅延をシミュレートします。この動作は、出力値の計算が、出力が生成されたクロックエッジで実際に行われることを前提としたシミュレーションを想定しています。このケースが必ずしも該当するとは限りません。詳細については次のセクションで説明します。

3.1.3 ネガティブタイミング制約

この例では、高性能デバイスでサンプリングされる入力がネガティブセットアップ時間を示し、その入力がサンプリングされるクロックエッジと同じクロックエッジで出力がドライブされます。図 3-3 はこのタイミングを示しています。

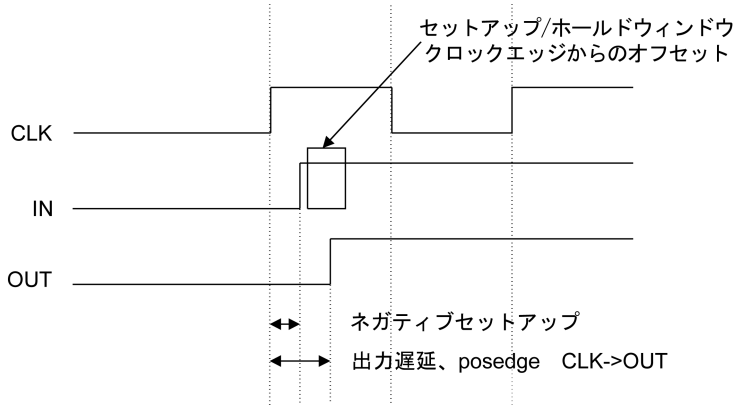

図 3-3 ネガティブセットアップ時間の高性能デバイス

この場合、**IN** の値がクロックエッジ後の特定の時間だけ変化し、生成される出力の値に影響を及ぼす可能性があります。たとえば、大規模デバイス内部のクロックの伝播遅延が原因の場合もあります。実際には、デバイス内部の伝播遅延により、入力クロックエッジが発生したと考えられる時点からしばらく経って出力**OUT**がドライブされます。DSM 方式のモデルでは、これらの遅延はモデリングされません。出力はタイミングシェルによってすぐにドライブされ、遅延が挿入されます。**IN** のその後の変化は、タイミングシェルによる **OUT** の最後のドライブよりも前に発生する場合でも、モデルによってその結果がすでにドライブされ、これを取り消すことができないため、シミュレーションに影響を及ぼすことはありません。

したがって、Pin-to-Pin タイミングシェルでこの種の動作をシミュレートすることは非常に難しく、前述のシナリオに適用してもうまくいきません。このような状況に対処するため、タイミングシェルの動作がどのようにシミュレータによって修正されるのかについては、P. 3-19「*ネガティブタイミングチェックの使用*」を参照して下さい。

3.2 複数のタイミングパス

P. 3-3「*タイミングシェルの追加*」では、タイミングシェルによって、クロックのポジティブエッジが出力遅延の原因となるイベントと判断されることが想定されています。もちろん、タイミングシェルはモデル内のロジックと状態に関する情報を保持しないため、タイミングシェルだけでこれを判断することはできません。しかし、出力を変化させ得る（原因となる）入力イベントが 1 つしかない場合は、大きな問題になりません。

大規模なデバイスでは、複数の入力イベントが発生したことが原因で出力が変化することがよくあります。（ある）例には、メモリクロックやテストクロックなどの複数のクロック、あるいは非同期リセットシグナルが含まれています。ARM がタイミングシェルを構成する場合には、各出力の原因となるイベントのリストがあります。タイミングシェルにはモデルの実際の動作に関する情報は保持されておらず、どのイベントが（出力の）原因なのかを判断できないため、複数のイベントが同時に発生した場合には、タイミングシェルによって間違ったイベントが選択される可能性があります。この問題は以下のどちらかの方法で解消することができます。

- ARM では、モデルによってエクスポートされる追加情報を使用し、モデルの内部状態に基づいて特定のタイミングアークのみを動的に有効にできるようにタイミングシェルを構成できます。これらのタイミングアークは、SDF 内にある条件付きタイミングアーク機能（`COND` コンストラクト）を使用して有効または無効にされます。
- モデルのタイミング制約を、問題の入力イベントが同時に発生する場合は不正とするように修正される場合があります（セットアップ / ホールドチェックによって行われます）。

特殊なケースについては「*状態依存タイミング*」を参照して下さい。

3.2.1 状態依存タイミング

問題のある複数のタイミングアークに関連した特殊なケースは、指定された出力遅延（`SDF IOPATH` コンストラクト。P. 3-9「*SDF アノテーション*」参照）の 2 つ以上のタイミングアークに同じ入力イベント（出力変化の原因となる）（negedge **CLK** など）が指定され、`SDF COND` 機能の使用方法のみが異なる場合です。これらは状態依存遅延と呼ばれ、SDF タイミングフローに特定の問題を引き起こします。アノテーションされるSDFデータを生成するベンダソフトウェアが複数のアークを区別できないかぎり、1 つの遅延時間群がそのすべてにアノテーションされます。その結果、状態に関係なく遅延がすべて同じになります。

通常、スタティックタイミング解析では、遅延値の計算時にデバイスがどの状態でモデリングされていたかを特定することはできません。状態依存タイミングは一部の ARM デバイス（ARM7TDMI など）に見られます。このようなデバイスには STA ベースのタイミングフローを使用できますが、シミュレーションでは状態依存はモデリン

グされません。状態依存タイミング動作をモデリングできる他のフローがサポートされていますが、このフローについては本書では説明しません。詳細については、*ARM Design Signoff Models : Timing Annotation* を参照して下さい。

注

SDF の COND 機能を使用しても、状態依存タイミングが有効なことを示すわけではありません。真の状態依存が発生するのは、COND 機能の使用によってのみ同一のタイミングアークが区別される場合です。個々のタイミングアークが正しくない場合は、COND をタイミングシェルで使用してそれらのタイミングアークを無効にすることもできます。

3.3 SDF アノテーション

VHDL および Verilog ロジックシミュレータは同じような方法で SDF アノテーションを実行しますが、この 2 つの言語では、一部に相違があります。

図 3-4 は、単純なタイミングシェルの構造を示しています。

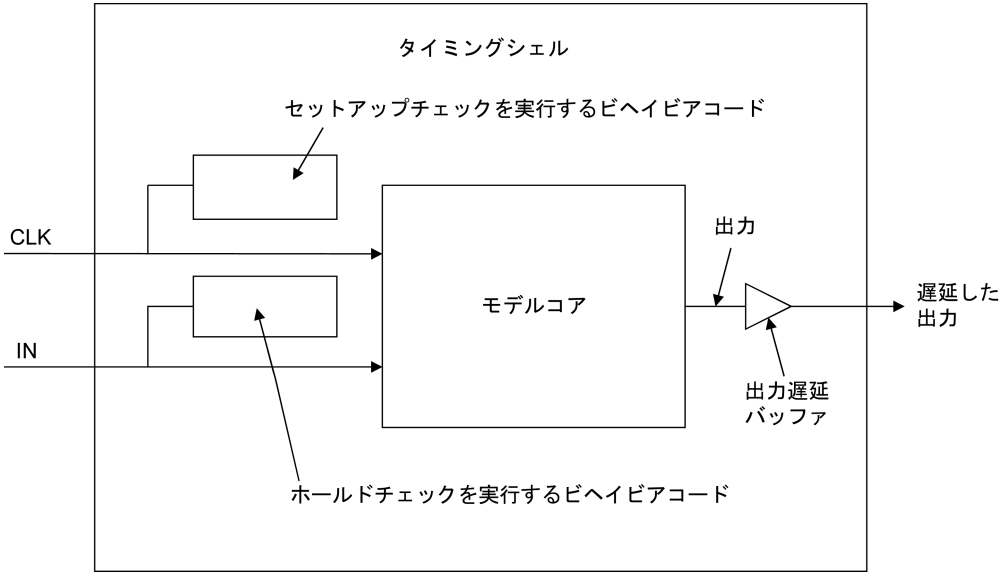

図 3-4 単純なタイミングシェルの構造

タイミングシェルはビヘイビア VHDL コードまたは Verilog コードのブロックですが、実際の動作のほとんどは、タイミングコンストラクトが言語機能として提供される Verilog 実装で推論されます。このコードは設計内の各シグナルのルーチンにまとめられ、そのシグナルでイベントが発生したときにこのルーチンが呼び出されます。

入力シグナルの場合、このコードはイベント発生時間が正しいかどうか、つまり対象のシグナルのタイミングチェックに違反していないかどうかをチェックします。入力シグナルの受け渡しが、タイミングチェックコードによって妨害されることはありません。タイミングチェックコードは監視だけを実行します。しかし、他の理由で入力シグナルの遅延が生じることがあります。詳細については、P. 3-17 *「相互接続遅延」*および P. 3-19 *「ネガティブタイミングチェックの使用」*を参照して下さい。

出力シグナルは、イベントを外部に伝播するときに遅延を挿入するバッファによって応答が返されます。

どちらのケースでも、タイミングシェルが機能するには時間の値が必要となります。入力チェックには、イベントと比較されるセットアップ時間とホールド時間が必要です。出力遅延では、ドライブをどれくらい遅延させるのかを知る必要があります。通常の HDL コードとは異なりこれらの値はコード内に存在せず、代わりにプレースホルダ値がタイミングシェルが挿入される各タイミングアークに使用されます。これらのプレースホルダには、HDL ソース内の標準値（通常はゼロ）が設定され、シミュレーション開始時に SDF ファイルから自動的に値が挿入されます。

SDF ファイルからタイミングシェルに値が挿入されるとき、シミュレータはどのプレースホルダを使用してその値を保持する必要があるのかを知る必要があります。この動作は、SDF ファイル内のタイミングアークと同じ定義が Verilog または VHDL ソースにあるため、標準的なマッピングによって行われます。VHDL では、タイミングアークを VHDL ジェネリック文の名前にマッピングすることで行われます（IEEE1076.4 参照）。この場合は、特定の条件に加え、タイミングアークに関連するシグナルとエッジが名前にエンコードされます。Verilog 言語はこれと若干異なり、エッジと条件情報を仕様に含むタイミングアークのための特殊なシステムタスクを保持します。シミュレータ内の SDF アノテータは、SDF ファイル内で検出されるアークと一致するタイミングアークのデフォルト値（ソースコード内で指定されている値）をオーバライドします。

SDF ファイル内の値がタイミングシェル内のプレースホルダと異なる場合には、SDF アノテーションが難しくなります。SDF ファイルは、通常はタイミングシェルの供給者が関知しないソースから作成されるため、不一致が発生する可能性があります。SDF は構造が非常に複雑なため、タイミングシェル内の特定のコンストラクトと明らかに一致するタイミングアークが、SDF アノテータによって一致として検出されない可能性があります。たとえば、シグナル **IN** とクロック **CLK** のポジティブエッジとの間でセットアップチェックが行われるようにタイミングシェルが構成される場合、以下のような SDF コードが予想されます。

```
(SETUP (POSEDGE CLK) IN ...)
```

しかし、SDF ファイル内で **CLK** と **IN** の間で 1 回のセットアップチェックしか行われない場合は以下のようになります。

```
(SETUP CLK IN ...)
```

指定されていないクロックエッジでのセットアップチェックはポジティブエッジでのセットアップチェックのスーパーセットです。そのため、理論上はタイミングシェル内での適切なチェックにアノテートされる有効なコンストラクトもスーパーセットとなります。OVI SDF2.1 標準では、この場合もアノテーションを続行する必要があると提案しています。しかし現実には、シミュレータが異なればこの状況の扱い方も異なります。通常、Verilog シミュレータは、特定のタイミングアークをこのようなやや曖昧な SDF コンストラクトからアノテートします。一方の VHDL は柔軟性に欠け、完全にマッチすることが要求されます。また、タイミングシェルに存在しないタイミングアークが SDF ファイル内に存在する場合、VHDL アノテータは一般的にエラーを生成してシミュレーションを停止します。

通常、SDF アノテーションを行う場合には SDF ファイルとタイミングシェルが一致している必要がありますが、Verilog シミュレータはこの点に関してより柔軟性があります。この要件は以下のテンプレートとツールを使用することで満たせます。

- *テンプレート*
- *SDFremap ツール*：P. 3-13

3.3.1 テンプレート

P. 3-9「*SDF アノテーション*」に記載されている通り、アノテートされる SDF ファイルは、一般的にはタイミングシェル内で指定されているタイミングコンストラクトとマッチすることが前提です。ARM モデルには、このマッチを確実にするためのテンプレートファイルが付属しています。`<device>.sdft` という名前のテンプレートファイルは構造上は SDF ファイルですが、数値情報は含まれていません。代わりに、デバイスに使用されるタイミングパラメータの名前が含まれています。テンプレートの主な目的は、タイミングシェル内のアークに関する明確な参照情報を提供することです。ただし、アノテートされる SDF ファイルがテンプレートファイルと厳密に一致していなければならないことを意味するわけではありません。通常はタイミングアークの順序は重要ではなく、テンプレートファイルをアノテートされる SDF から区別する方法は他にも数多くあります。

- SDF は、セットアップ時間とホールド時間のチェックを 2 つの個別のタイミングアークとして扱います。通常、これらのタイミングアークは SDF ファイル内で SETUP アークおよび HOLD アークとして表現され、シミュレーションでは 2 つのプレースホルダが使用されます。しかし、SDF では `SETUPHOLD` コンストラクトを使用することもできます。通常 ARM モデルは、`SETUPHOLD` を `SETUP` アークと `HOLD` アークの簡略形として何の副作用もなく使用できるように記述されています。

  注

  1 つのクロックエッジでセットアップチェックが実行され、次のエッジでホールドチェックが実行されるというタイミングチェックがあり、データシグナルはこのチェック間のクロックフェーズの際に安定状態で保持される必要があるという場合があります。これを不変タイミングチェックとも呼びます。一部の ARM モデルにはこの種類のチェックが含まれており、SDF ファイルでは対になっていない `SETUP` アークまたは `HOLD` アークによってこれらを表現することができます。対になっていないアークを `SETUPHOLD` コンストラクトで置き換えることはできません。もし置き換えると、シミュレータはタイミングシェル内に存在しないタイミングチェックをアノテートしようと試み、エラーが発生します。

- SDF では、1 つの値、あるいは 3 つの値（最小、標準、最大時間）を指定することができます。テンプレートのプレースホルダには常に 3 つの値の形式が使用されますが、必ずしもそうする必要はありません。また、テンプレート内の `IOPATH` エントリは最高 12 個までのプレースホルダ群を保持することができますが、アノテーションに 12 個の値の SDF を使用する必要はありません。SDF テンプレート内のプレースホルダは、ファイルの構造に関する重要度としては副次的なもの

で、アノテートされる実際の SDF は 6 個までの値を保持することができます。これらの値は、3 つの論理値 **1**、**0**、**Z** で 1 つのシグナルが形成できる 6 つの遷移を表しています。テンプレートファイル SDFremap および SDFgen から SDF を生成する ARM ツールでは、テンプレートの内容に関わらず、IOPATH に対して 6 個を超える値は書き込まれません。また、アノテートされる SDF が保持する IOPATH アークの値が 6 個より少なくても問題ありません。

- ベクタシグナルに適用されるタイミングアークの場合、テンプレートは以下のようにそのアークに対するビット範囲をインクルードします。

```
(IOPATH (NEGEDGE CLK) A[31:0] ...
```

注

. . . はラインデータの続きを示していますが、ここでは関係ありません。

実際、この簡略形で 32 の異なるタイミングアークを表すことができます。

```
(IOPATH (NEGEDGE CLK) A[31] ...
(IOPATH (NEGEDGE CLK) A[30] ...
(IOPATH (NEGEDGE CLK) A[29] ...
(IOPATH (NEGEDGE CLK) A[28] ...
- - -
```

注

--- は SDF の続きのラインを示しています。

アノテーションの目的で SDF ファイルを生成する場合、ビット範囲の塊の他に、ベクタシグナルの各ビットに個別のアークを使用することもできます。また、これらを混在させることも可能です。

```
(IOPATH (NEGEDGE CLK) A[31:24] ...
(IOPATH (NEGEDGE CLK) A[23] ...
(IOPATH (NEGEDGE CLK) A[22] ...
- - -
```

SDF ファイルから特定のアークを省略することはできても、これらのアークをアノテートしない場合には、VHDL ではベクタに関連するタイミングアークがアノテートされる場合に、ベクタの両端が SDF ファイル内に記述されている必要があり、そうでなければアノテーションに失敗します。Verilog シミュレータには通常はこのような条件はありません。

まとめると、モデルにアノテートされる SDF をタイミングシェルとマッチさせる必要があります。付属の SDF テンプレートファイルには、タイミングアーク群とそれらの形式の明確な定義が記述されています。アノテーションプロセスでは以下のガイドラインを参考にして下さい。

- テンプレート内のアークの汎用バージョンのタイミングアークは、Verilog を使用する場合は通常はどのようにもアノテートできますが、VHDL では不可能です。つまり、テンプレート内のエッジ指定子と COND 修飾子を省略しても、Verilog では SDF が正しくアノテートされます。
- テンプレート内のすべてのアークが、アノテートされる SDF ファイル内に存在していなければならないわけではありません。SDF ファイル内には存在せず、テンプレートには存在するアークには、そのモデルのゼロ遅延動作がデフォルトで適用されます。これらのアークにアノテートされる値を持たないタイミングチェックのパラメータにも、デフォルトでゼロが設定されます。
- モデルに関連する SDF ファイルの部分に、テンプレート内に存在しないアークが存在していてはなりません。この条件には INTERCONNECT または PORT 遅延は含まれません。詳細については、*「SDFremap ツール」*を参照して下さい。

3.3.2 SDFremap ツール

生成された SDF ファイルにテンプレートとマッチするものがないためにアノテーションできない場合には、本モデルに付属している SDFremap ツールを使用することができます。SDFremap は、システム全体のデータを保持している SDF ファイルとテンプレートファイルを読み出し、構造上テンプレートとマッチするようにモデルに適合するセルを書き直します。

このツールは、テンプレートをライン単位で処理します。テンプレート内の各アークに対し、SDFremap はそのアークの種類とマッチし、同じシグナルを使用するアークを SDF ファイル内で検索します。SDFremap は SDF アノテータよりも柔軟なため、本来ならば拒否されるアークについても検討が行われます。

1 つのアークに対してマッチする可能性のあるものが複数検出された場合、SDFremap はこれらのマッチを比較して最も近いものを選択します。ただし、このツールによって必ず状況が解決されるとは限りません。テンプレートに以下のコードが含まれている状況を考えてみましょう。

```
(SETUPHOLD (POSEDGE CLK) IN (Ts:Ts:Ts) (Th:Th:Th))
```

一方、SDF ファイルには以下の 2 つのエントリが保持されています。

```
(SETUPHOLD (POSEDGE CLK) (POSEDGE IN) (5) (2))
(SETUPHOLD (POSEDGE CLK) (NEGEDGE IN) (4) (2))
```

この場合、SDF を生成したタイミングソフトウェアは、シグナル **IN** の上昇と **IN** の下降を個別に捉えることによって、1 つのセットアップ / ホールドチェックを 2 つのアークに分割します。テンプレートがあるためにこのことはアノテートされず、したがってタイミングシェルは与えられた SDF よりも曖昧になります。この状況を逆にすると、

Verilog シミュレータではアノテーションが機能しても、VHDL シミュレータでは機能しません。なぜなら、タイミングシェルは **IN** の遷移を考慮しない 1 回のセットアップ / ホールドチェックを前提に設計されているためです。

この場合、SDFremap はアークの種類、シグナル、エッジ情報に基づいて、両方のアークが等しくテンプレート内のアークとマッチしているとみなします。次に、SDF ファイルから最もペシミスティックなアークを選択します。このアークに対して選択される出力は以下のようになります。

```
(SETUPHOLD (POSEDGE CLK) IN (5) (2))
```

監査履歴を残すため、SDFremap はテンプレート内の各アークについて以下の情報をログファイルに書き込みます。

- アークが存在する場合は、どのアークが検討されたか。
- どのような決定がなされたか。

エッジ仕様の修正に加え、SDFremap はテンプレート内に存在する COND エントリも追加し（Verilog では必要ないかもしれませんが、これによる有害な影響はありません）、アノテーションに不要なコンストラクトを削除します。

SDF ファイルは、たとえば以下のような 2 つのベクタ間のタイミングアークを指定できます。

```
(IOPATH IN[31:0] OUT[31:0]...
```

注

上記の 31 の値は単なる例です。

通常、これによって **IN[0]** が **OUT[0]** をドライブし、**IN[1]** が **OUT[1]** をドライブするような結合パスが参照されます。P. 3-11「テンプレート」に記載されている通り、SDF 内のベクタ範囲は、その範囲内の各ビットに対するタイミングアーク群の簡略形です。上記の IOPATH を展開すると、最初に以下が生成されます。

```
(IOPATH IN[31:0] OUT[0]...
(IOPATH IN[31:0] OUT[1]...
- - -

(IOPATH IN[31:0] OUT[30]...
(IOPATH IN[31:0] OUT[31]...
```

しかし、IOPATH の半分の入力部分もベクタであるため、上記のアークのそれぞれも 32 の更なるアークの簡略形となります。

```
(IOPATH IN[0] OUT[0]...
(IOPATH IN[1] OUT[0]...
- - -

(IOPATH IN[31] OUT[0]...
```



```
(IOPATH IN[0] OUT[1]...
(IOPATH IN[1] OUT[1]...
- - -

(IOPATH IN[31] OUT[1] ...
```

このプロシージャは、アークの合計数が 1024（32 の二乗）になるまで継続されます。出力ベクタ上の各ビットが、入力ベクタ上の任意の 32 ビットによってドライブされると仮定されています。しかし通常はこのようにはならず、DSM モデルは 1024 ではなく 32 のアークしか実装しません。したがって、上記のアークに対するテンプレートは以下のようになります。

```
(IOPATH IN OUT[31:0]...
```

ここでは **IN** が 1 つのシグナルとして表現されているため、シミュレータは 32 のアークのみを推論して必要な動作が指定されます。よく誤解されるのは、この表現がベクタ全体 **IN** を一時シグナルとして指定するために、何らかの異常な動作を引き起こすと考えられていることです。しかし、タイミングシェルが指定するのはタイミングのみではあり、動作は指定しません（P. 3-3 *「タイミングシェルの追加」*および P. 3-7 *「複数のタイミングパス」*参照）。たとえば、`IN[5]` が変化したためにモデルによって `OUT[5]` がドライブされると、タイミングシェルはこれを `IN -> OUT[5]` のパスとして認識します。なぜなら、**IN** の 1 ビットが変化すると、そのベクタシグナルも変化し、タイミングシェルが必要に応じて機能するためです。

注

VHDL シミュレーションにおける vector->vector IOPATH の問題については、P. 3-27 *「Vector-to − Vector の IOPATH アーク」*を参照して下さい。

SDFremap の厳密な動作は、そのプリファレンスファイルから設定されます。入力によっては、ツールのデフォルトの動作では有用な結果が生成されない場合があるので注意して下さい。ツールを使用する前に、SDFremap の UNIX マンページをよくお読み下さい。よくある問題の原因には以下のようなものがあります。

- ツールによって再マップするセルが検出されない。SDFremap は、SDF ファイル内のセルタイプのエントリに対してマッチングを行います。入力 SDF 内のセルタイプが、テンプレートファイル内で使用されているセルタイプとマッチしない場合、SDFremap はそのセルが目的のセルであると認識することができません。この場合、プリファレンスファイル内の `<celleq>` 機能を使用して、SDFremap に正しいセルタイプを認識させることができます。プリファレンスファイルの使用方法については、SDFremap のマンページ（UNIX）を参照して下さい。
- 再マップされた SDF が間違った階層レベルを参照している。ARM モデルを記述する場合は、タイミングシェルをラッパのトップレベルよりも 1 レベル低いところに配置します。ほとんどの SDF ファイルは、ARM モデルがインスタンス化されているレベルにタイミングシェルが存在するという想定の下に生成されます。将来の ARM モデルがこのレベルに配置されたタイミングシェルを使用して生成されるようになれば、この問題は解決されます。現在は、SDFremap プリファレンスファイル内の `<pathtrails>` 機能を使用することでこの問題を解決できます。

注

<pathtrails>機能を使用しても、相互接続遅延内で階層が固定されることはありません。なぜなら、相互接続遅延は参照するセルの外部で指定されるため、SDFremap に干渉されないためです。現在、相互接続遅延に関する階層の問題は手動で解決するか、UNIX sed ユーティリティなどのテキスト処理ツールを使用して解決する必要があります。

- SDFremap が最もペシミスティックなタイミングアークを正しく選択しない。この問題にはいくつもの原因が考えられます。入力チェックでは、チェックのタイプに基づくマッチングが、ペシミスティック時間のマッチングよりも優先されるため、SDFremap はペシミスティックなチェックよりも、テンプレートのフォーマットによりマッチしたオプティミスティックなタイミングチェックを選択する場合があります。
  最高 6 個の値群を保持できる IOPATHs の場合、SDFremap は最もペシミスティックな遅延を、指定された値の平均のうち最も高い値として認識します。
  トリプル値の SDF では、SDFremap はデフォルトでタイミング標準値（トリプル値の真ん中にある値）を考慮することによって、どのアークが最もペシミスティックであるかを計算します。問題があるのは、以下に示すように、SDF ファイルがすべてのトリプル値に対して空の標準値を保持している場合です。

  ```
  (SETUPHOLD (POSEDGE CLK) IN (1::2) (2::2))
  ```

  この場合、SDFremap はすべてのタイミングアークで標準値にゼロが指定されているとみなすため、デフォルトではペシミスティック値のマッチングが無効にされます。プリファレンスファイルを使用して、ペシミスティックな遅延を計算する目的で SDFremap がトリプル値の任意の値を使用するように設定することができます。詳細については、SDFremap マンページ（UNIX）を参照して下さい。

3.4 相互接続遅延

タイミングチェックと遅延はタイミングシェル内に明示的に組み込まれて SDF テンプレートファイル内で参照され、モデルの境界で適用されます。設計内の他のセルと直接やり取りされることはありません。

タイミングシェルには、アノテートされる隣接セルからの相互接続遅延に関しても規定があります。単純なケース（セル間の相互接続線に 1 つのドライバと 1 つのリーダーしかない、単一ソース相互接続遅延）では、遅延バッファを線の入力側に配置することで、相互接続遅延が HDL シミュレータによってモデリングされます。タイミングシェルは、モデルへの各入力にバッファを配置します。IOPATH 遅延と同様に、このバッファには最初にゼロが設定されるプレースホルダ遅延があります。

SDF アノテーション時、プレースホルダは SDF の INTERCONNECT または PORT タイミングアークによってオーバライドされます（単純なケースでは、PORT と INTERCONNECT が SDF アノテータによって同じように処理されます）。相互接続遅延に関連するすべての入力シグナルは、適切な時間だけバッファに保持されます。この遅延後、タイミングチェックとモデル自身によってイベントが登録されます。したがって相互接続遅延バッファは、入力イベントが通常のタイミングシェルに渡される前に遅延を挿入する、タイミングシェル周辺の特別なシェルとして考えることができます。セットアップ / ホールドチェックとその他のタイミングチェックは、遅延前の入力シグナルではなく、遅延後の入力シグナルの値に対して行われます。IOPATH 遅延内の基準シグナルとして使用されるシグナルが相互接続遅延の対象である場合、そのシグナルはイベントの遅延挿入後のシグナルとなります。

モデルによってドライブされる出力イベントは、モデルのタイミングシェルによって追加される IOPATH 遅延の他に、相互接続遅延の追加も選択できる、より広範な設計内の他のコンポーネントに伝播されます。これらの遅延には、イベントを使用するものによってダウンストリームが適用されるため、タイミングシェルへの影響はありません。

3.4.1 複数ソースの相互接続遅延

場合によっては、モデルへの入力に複数のドライバを使用できることがあります（一般的な例として、メモリまたは DMA ペリフェラルによってドライブできるデータバスがあります）。このような場合、ドライバによって相互接続遅延が異なる可能性があるため、シグナルが読み出される側でアノテートされる相互接続遅延に対して 1 つの値では不十分です（OVI SDF2.1 の相互接続遅延のセクションを参照）。詳細については、P. ix *「参考資料」*を参照して下さい。

複数ソースの相互接続遅延が発生する場合、正しく呼び出された Verilog SDF アノテータ（詳細についてはシミュレータのマニュアルを参照）はその遅延の展開を試みます。Verilog SDF アノテータは、シグナルの入力部分の遅延と、シグナルがドライブされた出力バッファ上の遅延をアノテートします。ARM Verilog タイミングシェルは、これらをドライブするモデルに適切な出力バッファがある限り、複数ソースの相互接続遅延

を正しく処理できます。また ARM Verilog タイミングシェルは、出力に適切なバッファコンストラクトを与え、複数ソースの相互接続遅延のアノテーションサイトとして使用できるようにします。

ARM VHDL タイミングシェルは、単一ソースの相互接続遅延のみをサポートしています。

3.5 ネガティブタイミングチェックの使用

典型的な事例のシミュレーションには、通常は P. 3-2*「ゼロ遅延モデルコア」*で説明されている単純なゼロ遅延動作が適しています。同期入力はクロックエッジよりも前のある時点、すなわちセットアップ時間で安定している必要があり、その後のホールド時間でしばらく安定状態で保持される必要があります。図 3-5 は、セットアップ時間がネガティブになるケースを示しています。

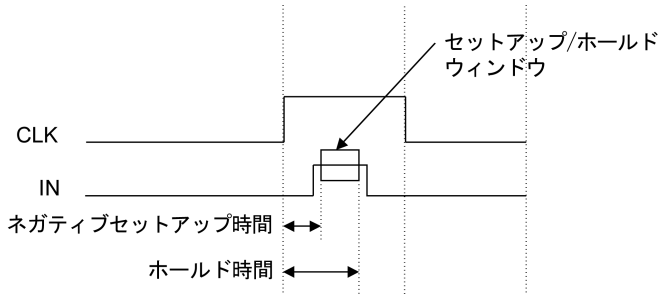

図 3-5 ネガティブセットアップ時間

このケースでは、入力シグナル **IN** が、このシグナルの名前が指定されてサンプリングされるクロックエッジの後にしばらくの間変化することが許可されます。これはデバイス内部のクロックツリー分散遅延によるものです。クロックはハードウェアの一部に、他の部分よりも先に到達します。これらの入力を使用するデバイス内部のロジックは、実際には遅延が挿入されたクロックを使用しています。SDF ファイル内で指定されているとおり、デバイスのタイミング動作は、そのデバイスの境界に関連しています。デバイス境界で見られるクロックに対して、入力 **IN** のセットアップ時間はネガティブになります。

境界タイミングシェルを使用するゼロ遅延モデルは、デバイス内部での分散遅延を直接使用しないため（直接使用する場合は Pin-to-Pin タイミングアノテーションが不可能となります）、この状態を直接処理することはできませんが、ドライブされる出力の生成に関連するすべての遅延を使用してこの出力を遅延させることにより、分散遅延をモデリングします。この累積遅延は、SDF ファイル内の `IOPATH` ディレクティブで使用される値となります。

したがって、サンプリングされる **IN** の値に依存するものの、クロックエッジに対する応答として定義されるモデル内の動作には、**IN** の誤った値をサンプリングして、実デバイスの動作から逸脱する危険性があります。

この問題の解決策としては、特定のイベントがモデルに与えられる前に、これらのイベントをタイミングシェル内で遅延させる方法があります。上記のケースでは、クロックシグナルに遅延が挿入されることで新しいシグナル **CLK'** が生成されます。P. 3-20 図 3-6 はこのプロセスを示しています。

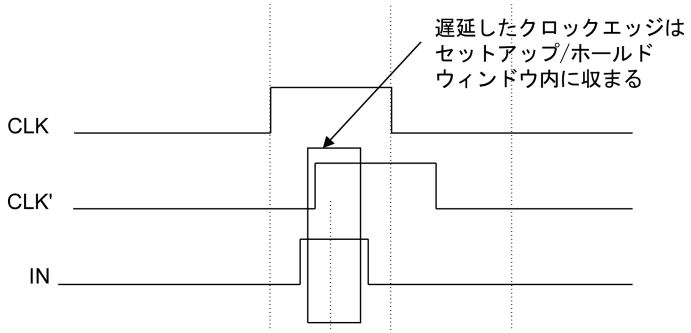

図 3-6 クロックシグナルの遅延による新しいシグナルの生成

どのタイミングチェックも元のクロックに対して実行されますが、モデルは遅延が挿入されたクロックを認識するため、**IN** が安定状態になってから（セットアップ / ホールドウィンドウ内で）サンプリングします。

単独では、必要とされることはこれだけです。しかし、モデルに認識されるクロックには遅延が挿入されているため、そのクロックからドライブされる出力にも遅延が挿入されます。クロックに挿入される遅延時間が、クロックから出力へのすべての IOPATH 時間から引かれても、タイミングシェルの外部は正しく動作します。また、クロックの遅延により、他の入力がホールド時間の終わりよりも後でサンプリングされる可能性があります。図 3-7 は、このシーケンスを示しています。

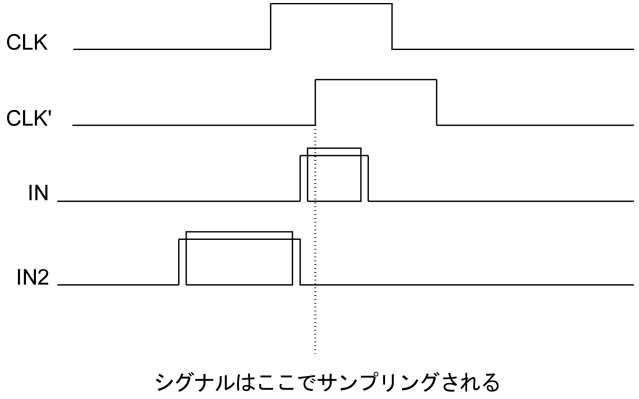

図 3-7 ホールド時間の後にサンプリングされる入力シグナル

これを解決するには、クロックに遅延が挿入された場合には、その分だけ IOPATH の遅延を短縮する必要があります。また、他の入力シグナルもセットアップ / ホールドウィンドウ内でサンプリングされるように、クロックの遅延と同様にこれらのシグナルにも遅延を適用しなくてはならない場合があります。

Verilog SDF アノテータには、適切な長さの遅延を自動的に入力シグナルに追加し、それを補正するために出力遅延を調整する機能があります。通常、Verilog シミュレータの起動時に、シミュレータへの適切なスイッチを使用することでこの機能が有効にされます（詳細についてはシミュレータのマニュアルを参照して下さい）。このスイッチが指定されていない場合、一般的にネガティブタイミングチェックにはゼロが設定されます。

Verilog シミュレータ向けの ARM タイミングシェルは、シミュレータがネガティブタイミング制約に基づいてこの調整を行えるように構成されています。このため、Verilog モデルでネガティブタイミングチェックを使用できます。しかし、ネガティブセットアップ時間に合わせてクロックを遅延させる時間よりも IOPATH の遅延が短い場合があるため、すべての IOPATH およびタイミングチェックが定義通りに機能するよう動作を調整することが不可能な場合もあります。この結果、一部のタイミングチェックが依然として全体的にペシミスティックになる可能性があります。

ARM VHDL モデルでは、ネガティブタイミングチェックはサポートされていません。

3.6 スタティックタイミング解析と HDL でアノテートされるシミュレーションとの相違点

このセクションでは、ARM コアを使用する設計のスタティックタイミング解析（STA）の結果と、その設計の動的 HDL シミュレーションにおける DSM のタイミング動作との潜在的な相違点について説明します。コアの STA モデルは、そのコアのタイミングリファレンスモデルを表し、DSM のタイミングシェルはこのモデルとできるだけマッチしている必要があります。

ARM コアの STA タイミングモデル（シノプシスライブラリ）は、Pin-to-Pin のブラックボックスタイミングモデルです。これらのモデルには、システム内でのコアの使用方法に基づいて、タイミングアークおよびタイミングアークの値の計算に使用されるルックアップテーブルが保持されます。したがって、DSM に含まれる HDL タイミングシェルの場合と同様に、STA ツールにもコアの内部構造に関する情報がありません。このため、タイミングシェルが STA モデルとまったく同じタイミングアーク群を保持することが理論的には可能となりますが、そうではないモデルもあります。

2 つの種類のモデルでタイミングアークが異なる理由はいくつもあり、これらの理由については P. 3-22「アークの不在」で説明されています。また、STA に関する HDL シミュレータの制約により、タイミングの結果が異なる可能性もあります（P. 3-23「HDL シミュレーションの制約」参照）。STA と HDL シミュレーションとのミスマッチによって、シミュレーションタイミング動作が STA よりもオプティミスティックになる場合もあれば、ペシミスティックになる場合もあります。オプティミスティックなケースの、設計の HDL シミュレーションでは STA によって検出されるタイミング違反は通知されません（シミュレーション時に、フェイルするアークが実行されると仮定）。ペシミスティックなケースでは、HDL シミュレーションで False タイミングエラーが発行されるため、エラーが発生しないフルスピードのシミュレーションは不可能です。

3.6.1 アークの不在

このケースでは、コアの STA モデルに存在するタイミングアークが、DSM タイミングシェルには存在しません。このため、DSM は STA モデルよりもオプティミスティックになります。この状態が発生する理由としては以下のような理由が考えられます。

- バージョンの不適合
- 出力間の遅延
- 出力のセットアップ / ホールドチェック
- コアのライセンス取得者による合成可能コアの実装と DSM との不適合（-S 接尾文字の付いたコアと ETM があります）。

バージョンの不適合

コアの設計変更によりタイミングアークが STA モデルに追加されているが、同じ変更が DSM タイミングシェルには追加されていないという場合があります。この問題、DSM のマイナーアップデートによって容易に解決することができます。

出力間の遅延

合成可能デバイスの実装によっては、内部で直接フィードバックされる出力シグナルが、別の出力ピンをドライブする何らかのロジックに渡されることがあります。現時点でこれらのアークは DSM には実装されていません。コアのライセンス取得者は、シノプシス製ツールによる合成時に `set_fix_multiple_port_nets` コマンドを使用してデバイスを再合成することで、この問題を解決できます。

出力のセットアップ / ホールドチェック

この問題は出力間遅延と同様に合成可能デバイスに関連する問題です。この問題は、出力が内部で直接フリップフロップ入力にフィードバックされる場合に発生します。出力間遅延の解決方法と同じ方法でこの問題を解決できます。

コアのライセンス取得者によるコアの実装と DSM との不適合

これは合成可能デバイスのみに関する問題です。これらのコアの DSM で使用されるタイミングアークは、関連コアのテストチップの実装に基づいて開発されます。合成可能デバイスはコアのライセンス取得者によって実装され、使用されるテクノロジライブラリ、合成方法、ツールのバージョンによってモデリングが異なるため、アークは実装ごとに若干異なります。合成可能デバイスに関しては、シリコンベンダーは自社のコアの実装に適合する STA ビューを作成しており、ARM はこれを関知しません。コアのライセンス取得者は、自社のコアの実装に適合した DSM タイミングシェルを生成することでこの問題を解決できます。

3.6.2 HDL シミュレーションの制約

HDL シミュレーションの制約は、シミュレーションのバックアノテーションフローにおける SDF データの処理方法に関連します。これらの制約により、STA よりもペシミスティックなシミュレーションが行われます。これらの制約は、ネガティブタイミングチェック、立ち上がりおよび立ち下がりセットアップ / ホールドチェック、単一のネガティブセットアップチェックまたはネガティブホールドチェックで発生する可能性があります。

ネガティブタイミングチェックに関する注意点

P. 3-19「*ネガティブタイミングチェックの使用*」では、特定のポートの出力遅延が、クロックの遅延時間分だけ短くなる場合について説明されています。HDL シミュレータは出力遅延要件に基づいて、矛盾するネガティブセットアップチェックをすべてゼロにします。STA にはこのような制約がないため、アークを個別に扱うことができますが、シミュレーションでは出力遅延とセットアップ / ホールドチェックとの関係を考慮する必要があります。

立ち上がりおよび立ち下がりセットアップ / ホールドチェック

P. 3-13「*SDFremap ツール*」で説明されているとおり、SDFremap ツールは SDF ファイル内のエッジ固有のセットアップ / ホールド文を、エッジに限定しない単一の文に結合します。この結合を行うときに、SDFremap ツールは元のファイルから最もペシミスティックな値を選択します。合成コアおよび STA には、それぞれ個別の立ち上がりおよび立ち下がりの値が使用されます。実装済みの設計では、入力の立ち上がりエッジに対するセットアップマージンが立ち下がりエッジに対するセットアップマージンより大きい、またその逆のケースという事実に依存してしまうことがあります。これが原因で、シミュレーションでセットアップ違反またはホールド違反が間違って通知される可能性があります。

注

この問題は、データシグナルに対するエッジ指定子のみに発生します。タイミングシェルにはクロックエッジに固有のタイミングチェックが実装されています。

単一ネガティブセットアップチェックまたはホールドチェック

滅多にありませんが、対応するホールドチェックがない単一のセットアップチェック、あるいは対応するセットアップチェックがない単一のホールドチェックの処理が STA モデルに含まれていることがあります。チェック値がポジティブであれば問題ありませんが、ネガティブの場合はそのチェックにシミュレーションではゼロが設定されます。これは、ネガティブの値でアノテートできるのは複合された Verilog $setuphold チェックのみであり、対となる値の一方が存在しない場合には、そのタイミングでのデフォルト値であるゼロが使用されるためです。2 つの値の合計がポジティブになる必要があるため、チェックの半分はネガティブでもう半分がゼロとなるチェックは無効です。Verilog シミュレータは SDF ファイル内の値をオーバライドしてゼロを設定します。また、タイミングチェックの値にゼロを設定したことを通知する警告メッセージを発行します。

3.7 タイミングモデルの制約

以下のセクションではタイミングモデルの制約について説明します。

- *IOPATH の保持*
- *Vector-to − Vector の IOPATH アーク*：P. 3-27

3.7.1 IOPATH の保持

これまで、遅延が発生する出力上の遷移は、特定のシミュレーション時刻の古い値から新しい値への 1 つの遷移として表現されていました。現実には、古い値が有効であることが保証される最後の時刻と、新しい値が有効であることが保証される最初の時刻は、その値が未知である不確実な時間によって分離されます（図 3-8 参照）。

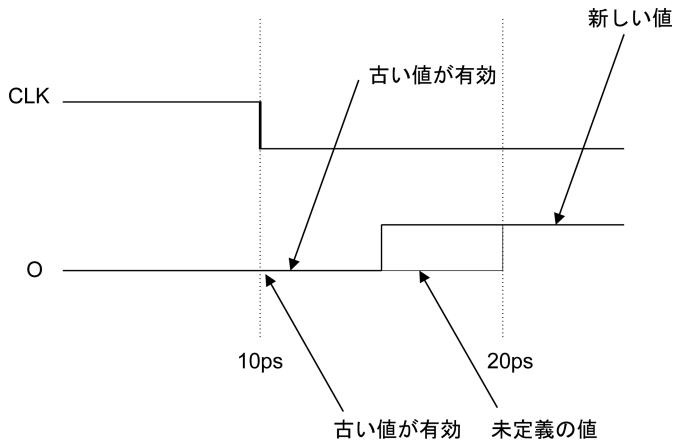

図 3-8 値の不確実性

シミュレーションでは、15ps で出力を X にドライブし、20ps で 1 にドライブすることによって、上記の動作がモデリングされます。

DSM 内の遅延はタイミングシェルの制約の範囲内で発生する必要があるため、この動作を直接モデリングすることはできません。すべての遅延はタイミングシェル内のコードによって適用され、モデルは起動された時点で出力を新しい値に直接ドライブします。未知の期間のモデリングは、タイミングシェル内で実行される必要があります。

SDF ファイルでは、出力の前の値が有効のまま保持される時間を、新しい値が有効になる時間と区別して指定することができます。この機能は IOPATH RETAIN と呼ばれ、この機能を使用することで保持時間（前の値が有効であると認識される時間）を最大 3 回まで指定することができます。これらの 3 つの時間は、それぞれ HIGH、LOW、ならびにハイインピーダンスからの遷移を表しています。

記述時に、ARM 社が対応している Verilog シミュレータで SDF アノテーション中に IOPATH RETAIN 機能を使用するものがないことが分かっているため、この機能は無視されます。その結果、Verilog モデルでは出力遅延上の保持時間がサポートされません。

VHDL は最近になって独自の SDF アノテータ内における IOPATH RETAIN のいくつかのサポートを追加しましたが、このサポートを実現するには、SDF ファイル内の対応する IOPATH エントリが、タイミングシェルの生成時にどれだけの数の遅延（最大 6）を挿入するかということを、タイミングシェルが知っている必要があります。この制約により、ARM VHDL タイミングシェルは現時点では IOPATH RETAIN をサポートしていません。

シミュレーションの不確実性

IOPATH RETAIN がサポートされていないということは、古い値が有効でなくなった時点で、あるいは SDF バックアノテーションの実行時に新しい値が有効になることが分かる時点で、新しい値に遷移するかどうかを選択する必要があることを意味します。

DSM からの出力値を使用するデバイスが、その値が定義されていない時間内に値をサンプリングすると、2 つの時間のうちどちらが遷移に使用されているかにより、古い値と新しい値のどちらか一方が取得されます。実際には、定義されていない時間内でこれらの値がサンプリングされてはなりません。IOPATH RETAIN がサポートされていれば、この時間内にサンプリングされた値はすべて X として読み出されます。

IOPATH RETAIN がサポートされていない場合は、シミュレーションと物理デバイスの動作が異なる可能性があります。さらに、この動作の違いは警告メッセージで通知されず、シミュレーションに誤った値が暗黙に使用されることになります。この状態を検出する 1 つの方法としては、シミュレーションを 2 回行って、1 回目は最小値を含む SDF ファイルを使用し、2 回目は最大値を含む SDF ファイルを使用する方法があります。これらの SDF ファイルは、それぞれデバイスの最小タイミングビューと最大タイミングビューを使用して STA ツールによって生成される必要があります。未定義時間内で出力がサンプリングされないことを前提に、この 2 回のシミュレーションの結果は同じになることが予想されます。

3.7.2 Vector-to ー Vector の IOPATH アーク

1995 VITAL 標準に基づく VHDL では、2 つのシグナルの両方がベクタとなる IOPATH のアノテーション方式が規定されており、2 つのベクタのデカルト積にアノテーションプレースホルダを使用することが要求されています。たとえば、32 ビットの 2 つのベクタには、32 * 32 (1024) アノテーションポイントが必要となります。P. 3-13「*SDFremap ツール*」で説明しているとおり、この方式は非効率なため、ARM SDF テンプレートファイルではこれらのシグナルの一方をスカラで表現しています。

最近の VITAL 2000 標準では、入力ベクタ上のエレメントと、出力ベクタ上の同一エレメントが以下のように一対一の関係となる状況では、先の方法が不適切であると認識されています。

IN[0] が **OUT[0]** をドライブする。

IN[1] が **OUT[1]** をドライブする。

– – –

このような場合、新しい VHDL シミュレータでは、両方の入力ベクタが同じサイズとなる特殊なケースが許可されています。

注

一部のデバイスでは、2 つのベクタのサイズが異なるベクタからベクタへの結合パスがあり、VITAL 2000 に準拠したシミュレータではこれが問題になる可能性があります。

ARM 社では、この問題の解決策を現在調査しています。VITAL 2000 準拠シミュレータ上で Vector-to-Vector IOPATH のアノテーションに問題が生じる場合、現時点で推奨される解決策は VITAL 95 のみを実装している旧バージョンのシミュレータを使用することです。

この問題が Verilog シミュレータに影響を及ぼすことはありません。

第 4 章
制約事項

本章では DSM の制約事項について説明します。本章は以下のセクションから構成されています。

4.1 DSM の制約事項

DSM は ARM コア設計のアーキテクチャおよび機能と完全に適合していますが、以下のような制約があります。

シミュレータ機能はサポートされていません。

以下の機能はサポートされていません。

再開 シミュレーションを終了させずにシミュレーションをゼロから再開することはできません。

保存と復元（チェックポインティングとも呼ばれます。）

ある時点でシミュレーションを保存し（スナップショット）、シミュレーションをその時点に復元することはできません。

多くの DSM に含まれている SWIFT コンポーネントを再開することはできず、それらのシミュレーション状態を保存・復元することはできません。SWFIT コンポーネントは DSM の大半に含まれているため、DSM ではこれらの機能はサポートされていません。

内部スキャンチェインのモデリング

DSM は、モデリングするコアの RTL の記述から作成されます。コアの最終的なネットリストには、合成時に追加された内部スキャンチェインが含まれている可能性があります。デバイス RTL には内部スキャンチェインが存在しないため、DSM でこれらのスキャンチェインをモデリングすることはできません。ただし、スキャンチェインはデバイスのサインオフモデル（SOM）によってモデリングされます。

キャッシュとレジスタ

DSM シミュレーション内で使用されるレジスタ値を見ることは可能ですが、エンジニアや設計者がキャッシュまたはレジスタにテストデータを直接挿入することはできません。なぜなら、DSM の基となる RTL でこれを行うことができないからです。

ゼロ遅延シミュレーション

この制約については、P. 2-7 *「DSM のイベントとインタフェースのセマンティクスに関する問題点」*を参照して下さい。

タイミングモデルの制約

この制約については、P. 3-25 *「タイミングモデルの制約」*を参照して下さい。

用語集

この用語集では、本書で使用されている一部の専門用語について説明します。複数の意味をもつ用語については、この用語集にある意味で使用されているものとします。

バックアノテーション
タイミング特性を実装プロセスからモデルに適用するプロセス。

バウンダリスキャンチェイン
バウンダリスキャンチェインは、標準的な JTAG TAP インタフェースを使用してバウンダリスキャンチェインテクノロジを実装する、シリアル接続されたデバイスで構成されます。各デバイスには少なくとも 1 つの TAP コントローラがあり、TAP コントローラに保持されるシフトレジスタによって、テストデータがシフトされる **TDI** と **TDO** を接続するチェインが形成されます。プロセッサには、選択されたデバイスの部分にアクセスできるように複数のシフトレジスタが組み込まれている場合があります。

クロックゲーティング
マクロセルのクロックシグナルを制御シグナルとゲーティングし、結果としてマクロセルの動作状態を制御する修正クロックを使用すること。

CRF Condensed Reference Format *参照*。

Condensed Reference Format（CRF）
テストベクタを指定するための ARM 独自のファイル形式。

デルタサイクル サイクル開始時のシミュレーション時間が、そのサイクルの終わりと同じになるシミュレーションサイクル。つまり、デルタサイクルではシミュレーション時間が進行しません。

デザインシミュレーションモデル（DSM）
ターゲット HDL シミュレータに組み込むことが可能な、バックアノテーション対応のシミュレーションモデル。機能ブロックと、Verilog ラッパまたは VHDL ラッパで構成されます。

デルタ掃引 VHDL シミュレータがデルタサイクルを通過するプロセス。1 回の掃引で多くのデルタサイクルが処理されます。

ダイレクトメモリアクセス
プロセッサが関連するデータにアクセスすることなく、メインメモリに直接アクセスする動作。

DVS Device Validation Suite *参照*。

Device Validation Suite
テクニカルリファレンスマニュアルで定義されている機能に対し、デバイスの機能をチェックするためのテスト群。バスインタフェースユニット（BIU）、低レベルメモリサブシステム、パイプライン、キャッシュおよび密結合メモリ（TCM）の動作のストレステストにも使用されます。

内部スキャンチェイン
デバイス内にパスを形成するように結合された一連のレジスタ。プロダクションテスト時にテストパタンをデバイスの内部ノードにインポートし、結果の値をエクスポートするために使用されます。

モデルマネージャ モデルとシミュレータ間のイベントトランザクションを処理するソフトウェア制御マネージャ。

プログラミング言語インタフェース（PLI）
Verilog シミュレータでは、別の言語で記述されたコードをシミュレーションに組み込めるようにするためのインタフェースを意味します。

SDF Standard Delay Format *参照*。

Standard Delay Format（SDF）
バスの個々のビットにいたるまでのタイミング情報を含んだファイルの形式であり、SDF バックアノテーションで使用されます。SDF ファイルは様々な方法で生成できますが、一般的には遅延計算機によって生成されます。

TAP テストアクセスポート*参照*。

テストアクセスポート（TAP）
JTAG バウンダリスキャンアーキテクチャへの I/O インタフェースおよび制御インタフェースとして機能する、4 本の端子（必須）と 1 本のオプション端子の集合。必須端子は **TDI**、**TDO**、**TMS**、**TCK** です。オプション端子は **TRST** です。

索引

本索引では項目をアルファベット順に並べ、記号と数字を最後に記載しています。参照符はページ番号を示しています。