

第5章

バス・ファンクション・モデルによる

テスト・ベンチ・コーディング法

Daniel Notestein

はじめに

HDLでの設計でもっともめんどろで時間がかかるのは、設計の動作確認のためのHDLテスト・ベンチのコーディングです。

テスト・ベンチは設計対象と相互に作用する外部システムをモデル化したものです。設計モデルMUT (Model Under Test)に対してスティミュラスを与え、またMUTからの出力と期待値との照合を実行することもできます。

テスト・ベンチとしてモデル化される外部システムの機能は、設計の対象物以上に複雑になることがしばしばあります。例えば、設計対象がメモリ・マネージメント・チップの場合を考えてみましょう。この場合、マイクロプロセッサやビデオ・コプロセッサを含むシステムが外部システムになるといえばわかりやすいでしょう。

マイクロプロセッサなどの複雑なデバイスのモデルは、つねに用意されているとは限りません。仮に用意されていたとしても高価であったり、モデルが大きいために実行が遅くなってしまいます。

テスト・ベンチを どのように準備するか

●テスト・ベンチ作成のアプローチ

複雑なデバイスを含むテスト・ベンチを用意する方法として、以下の三つが考

えられます。

- (1) デバイスのモデルを購入する。あるいは、自分で内部機能の完全なモデルを書く
- (2) デバイスの内部動作をモデル化するのではなく、インターフェース・レベル(バス・トランザクション・レベル)の動作を記述するバス機能モデル(BFM: Bus Functional Model)を書く
- (3) 単にMUTの入力へのテスト・ベクタ(スティミュラス)のみ用意し、MUTからデジタル・データの出力を得てそれを観察するにとどめる

●完全な機能モデル作成時の問題

複雑なデバイスの完全な機能モデルを書こうとすると、三つの問題に直面します。

- (1) デバイス内部の詳細な動作は、通常はデバイス・ベンダから公開されていない
- (2) 複雑なデバイスの完全な動作モデルは規模が大きくなるため、コーディングだけでなくデバッグにも膨大な時間がかかる
- (3) 規模が大きくなると実行スピードが遅くなる

●バス機能モデルを使うメリット

バス機能モデル(BFM: Bus Functional Model)は、完全な動作モデル作成時ににおける三つの問題を解決します。

- (1) BFMで要求されるインターフェース情報はデバイス・ベンダより公開されている
- (2) デバイスのすべてをエミュレートするのではなく、必要なインターフェースのみなので、BFMは完全なモデルにくらべてたいへん小さくなる
- (3) BFMはたいへん少ない信号の計算だけなので、高速に動作する

●BFMのモデリング

BFMは、モデリングに以下の高度な機能表現の記述能力を要求するため、従来のゲート・レベルのシミュレータではむずかしく、HDLの高度な記述能力を利用できるHDLシミュレータの環境でのみ実現されます。

- (1) 単純なテスト・ベクトルにくらべて、より精密なモデリング
- (2) MUTの複雑な動作の機能確認が可能
- (3) MUTの出力にตอบสนองして、出力ごとに異なるスティミュラスをMUTへ与えられる

●BFMの記述はむずかしい

BFMには大きな利点があるにもかかわらず、現在はあまり使われていません。荒っぽいテストしかできない、単純なテスト・ベクタ方式に止まっているのは、BFMの記述がむずかしく、しかもシミュレーション時にしかテスト・ベンチのデバッグができないためです。

BFMのデバッグは、BFMの応答的な

動作のため、可能なすべての動作をチェックすること、MUT 自身の問題との分離の困難さにより、テスト・ベンチによるテスト・ベンチのデバッグよりもきわめて困難です。

BFM の典型的な使用例は、マイクロプロセッサと I/O デバイスまたはメモリ・サブシステムとのインターフェース部です。このタイプの BFM は、マイクロプロセッサの各バス・トランザクション(リード・サイクル、ライト・サイクル、割り込み処理など)のための制御/データ信号を正確なタイミングで生成しなければなりません。また MUT の出力に応答し、MUT の動作を検証しなければならないため、さらに複雑になります。マイクロプロセッサのある BFM は、リード・サイクルを終了する前に、メモリ・サブシステムからのデータ・バリッド信号を待たなければならないという場合などです。

●BFM を効率的に作成するツール

多くの BFM は、現在、インターフェース信号とデバイス・ベンダからのタイミング情報(タイミング・ダイアグラム)をもとに、HDL によってコーディングされています。このような HDL コードの作成とそのメンテナンスは、視覚化がむずかしい HDL コードから生成するウェブ・フォームと、デバイス・ベンダからのタイミング・ダイアグラムの比較を必要とするため困難を極めます。

最近では、グラフィカルに仕様を定義するツールが出現してきています。それは、設計者が入力したバス・トランザクションを記述したタイミング・チャートにより、HDL テスト・ベンチ・コードを生成するものです。

■ BFM ベースのテスト・ベンチの設計手順

BFM は、デバイスの内部動作はまったく必要ありません。また、デバイスの完全機能のうちのほんの一部分の機能しか必要ありません。

例えば、メモリ・インターフェースをテストするために使うマイクロプロセッ

サ用の BFM では、マイクロプロセッサのリード/ライト・バス・トランザクションのモデル化だけが必要です。

●トランザクションの仕様を得る

BFM モデルには、テストを行うバス・トランザクションに関連する信号の波形の変化の生成と検証のみが必要です。ですから、BFM の設計の最初のステップとして、どのトランザクションを MUT のテストするのにモデル化する必要があるかを決定し、それらの仕様を得ます。

この情報は通常、タイミング・ダイアグラムのかたちでデバイス・メーカより供給されています。

●HDL コードの準備

モデル化するトランザクションを決定したら、次には各トランザクションに対応する HDL コードを用意します。

トランザクション・モデルは、異なるデータ値に対応できる、パラメタライズされたテスト・ベンチの部品として、再利用可能になります。

例えば、メモリ・サブシステムのテスト・ベンチは、いろいろなデータ・バス値をメモリ内のいろいろなアドレス・ロケーションとの間で読み書きします。

●シーケンサのモデル化

各トランザクションのモデル化ができたら、これらのモデルを駆動し制御するシーケンサをモデル化します。

最後に、各トランザクション・モデル、シーケンサそして MUT を結合して、最上位の閉じたシステムとなり、テスト・ベンチの完成です。

■ BFM ベースのテスト・ベンチの設計例

これから、簡単な BFM ベースのテスト・ベンチの設計例を見てみましょう。ここでは、HDL としては VHDL を選択しますが、多くの部分は Verilog-HDL にも共通しています。

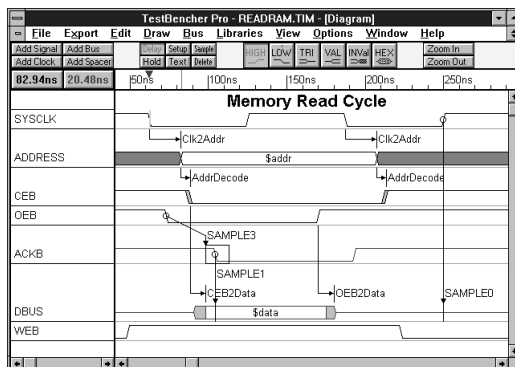
MUT :

8ビット・バスの RAM を二つつなげて 16ビットにしたシステム

考えるトランザクション :

CPU のメモリ・リード・サイクル(リード・トランザクション)とメモリ・ライト・サイクル(ライト・トランザクション)

TestBench Pro で記述した、リード・トランザクションのタイミング・ダイアグラムを **図1**に、ライト・トランザ



グラフィカルにバス・トランザクションを記述したタイミング・チャートから HDL テスト・ベンチを自動生成するツールの例

HDLコード生成

```

Desktop, v1.0 - [C:\PROJECT\SAMPLE.VHD]
File Edit Options Window Help
new open save as print help
CEB <= '1';
OEB <= '1';
WEB <= '0';
wait for 48640 ps;
WEB <= '1';
wait for 24576 ps;
OEB <= '0';
wait for 9284 ps;
ADDRESS <= addr;
wait for 4000 ps;
CEB <= 'X';
wait for 2000 ps;
CEB <= '0';
wait for 11500 ps;
if (ACKB /= '1') then --sample3-1
  assert FALSE
  report "Bad state: ACKB /= '1'"
  severity WARNING;
end if;
wait for 5000 ps;
if (DBUS /= data) then --sample1
  assert FALSE
  report "Bad state: DBUS /= data"
  severity WARNING;
end if;
wait for 10000 ps;
if (ACKB /= '0') then --sample3-2
  assert FALSE
  report "Bad state: ACKB /= '0'"
  severity WARNING;
end if;
wait for 66000 ps;
  
```