Xilinx 社製 FPGA を搭載した PCI ボードのシミュレーション

小野　雅晃

筑波大学　電子・情報工学系

1. はじめに

私は Xilinx 社の FPGA(Field Programmable Gate Array)を使用して、論理回路を作製している。規模の比較的大きい回路を開発する際、まず機能モジュールに回路を分割し、機能モジュールごとに回路を作成する。次に機能モジュールを組み合わせて全体の回路を完成させるという手順を取る。仕様通りに動作する回路を作製するためには、各手順ごとに回路が動くかどうかをテストする工程が必要となる。まずは、分割された機能モジュール単位でシミュレーションを行う。このような、機能モジュール単位のシミュレーションは、簡単な入力信号に対して、仕様どおりに出力が出てくるかを確認する。この時に、入出力信号を規定するのがテストベンチと呼ばれるファイルである。テストベンチは通常ハードウェア記述言語(HDL)で書かれている。直接 HDL でテストベンチを書くのは難しいので、グラフィカルにテストベンチを生成するツールが存在する。後でそのツールの 1 つである Xilinx 社の HDL Bencher を紹介する。

次に機能検証の終わった機能モジュールを組み合わせて、目的の回路を作り上げる。その回路に対してシミュレーションで動作を確認する。これを全体シミュレーションと呼ぶことにする。全体シミュレーションでも先ほどのツールを使用することは出来るが、複雑な手順を必要とするシミュレーションには不適である。

複雑なシミュレーションを実現する場合には、シミュレーション用モデルを作りこんで、検証すると便利である。その場合、シミュレーション用モデルの複雑さのレベルが問題になる。余り複雑なモデルであると、検証する回路よりも複雑になってしまうことがある。検証スピードは、複雑なモデルの方が簡単なモデルよりも遅くなる。本稿では、比較的実行スピードの速い簡単なモデルを用いた回路のシミュレーションについて述べる。

2. 対象回路

今回検証に使用する回路は、並列コンピュータ用のネットワークカード(NI)の FPGA に実装した回路である。これを以降 NI FPGA 回路と呼ぶことにする。NI はホストコンピュータの 64 ビット 66MHzPCI バスに挿して使用する PCI カードである。NI はホストコンピュータのデータを PCI バス越しに吸い上げて、LVDS (Low Voltage Differential Interface)信号で、他のコンピュータに転送する。NI は PowerPC プロセッサ、LVDS、64 ビット 66MHzPCI バス、FLASH ROM、SDRAM、NI FPGA 回路、CPLD で構成されている。図 1 に NI のブロック図を示す。NI FPGA 回路は、ほとんどすべてのデバイスを制御している。よって、NI FPGA 回路をシミュレーションする場合には、それらのモデルを接続しなければならない。今回は回路、モデルとも VHDL(VHSIC Hardware Description Language) で実装した。

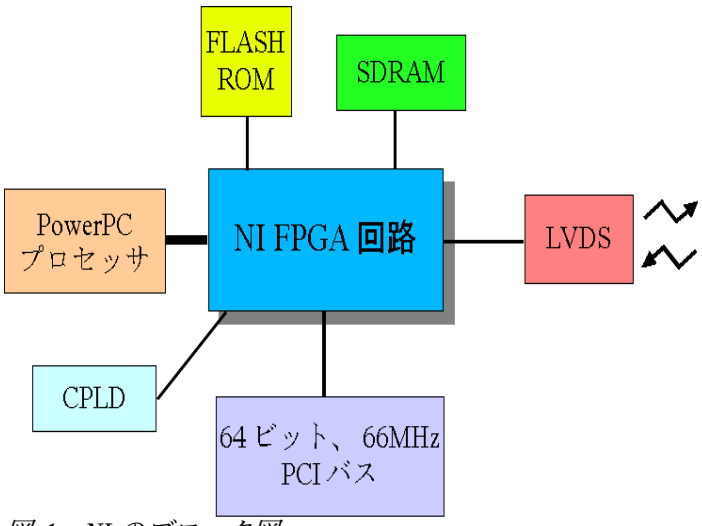

図 1　NI のブロック図

3. HDL Bencher を使用した機能モジュールの回路シミュレーション

Xilinx のツール ISE6.1i に統合されている HDL Bencher を使用した SDRAM 制御回路の回路検証例を示す。SDRAM 制御回路はすべて VHDL で書かれている。HDL Bencher は入力信号をグラフィカルに編集して、VHDL のテストベンチを生成するツールである。このツールを使用すると、簡単にテストベンチを生成することが出来る。HDL Bencher の起動時にクロックがどの信号なのかを指定する。指定したクロックに対して入力信号のセットアップ時間と、出力信号の出力時間を設定する。次に入出力波形がタイミングチャートで表示され、波形入力モードとなる。図 2 が HDL Bencher の起動画面である。

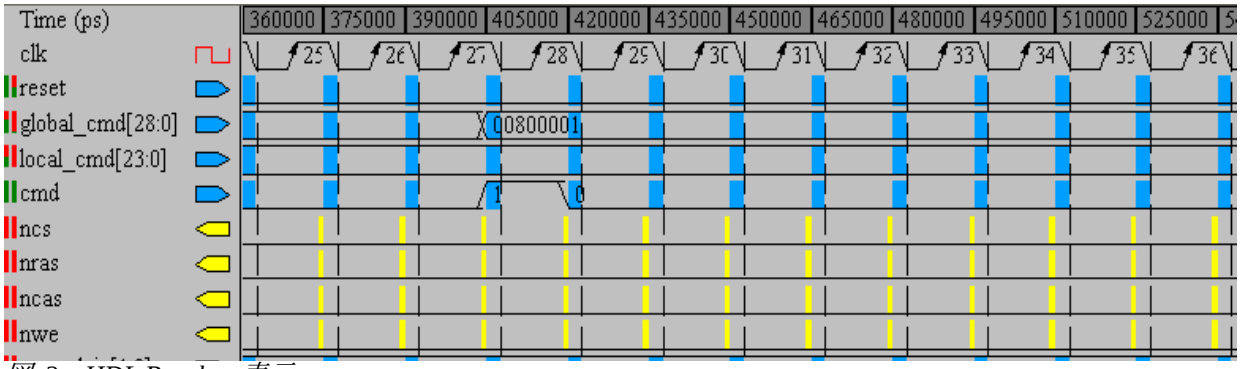

図 2　HDL Bencher 表示

図 2 で上から 4 つまでの右向きの箱は入力、下から 4 つまでの左向きの箱は出力を表す。図 2 のように、入力波形をタイミングチャートでグラフィカルに編集して、VHDL のテストベンチ記述を生成する。図 2 では、405000psec で global_cmd に 00800001、cmd に 1 を入力している。その後、420000psec で cmd を 0 に戻している。これは SDRAM 制御回路に対する 1 ロングワードのリード要求である。

テストベンチは入力波形を記述した VHDL ファイルである。HDL Bencher から出力されたテストベンチは、入力の 1 から 0、0 から 1 の変位が時間を追って書かれている。テストベンチと実際の回路（ここでは SDRAM 制御回路）を波形シミュレータ ModelSim でコンパイル、シミュレーションして、入力波形に対する出力波形の様子を確認する。出力波形を図 3 に示す。HDL Bencher で入力した入力信号に対する出力信号をタイミングチャートで見ることが出来る。

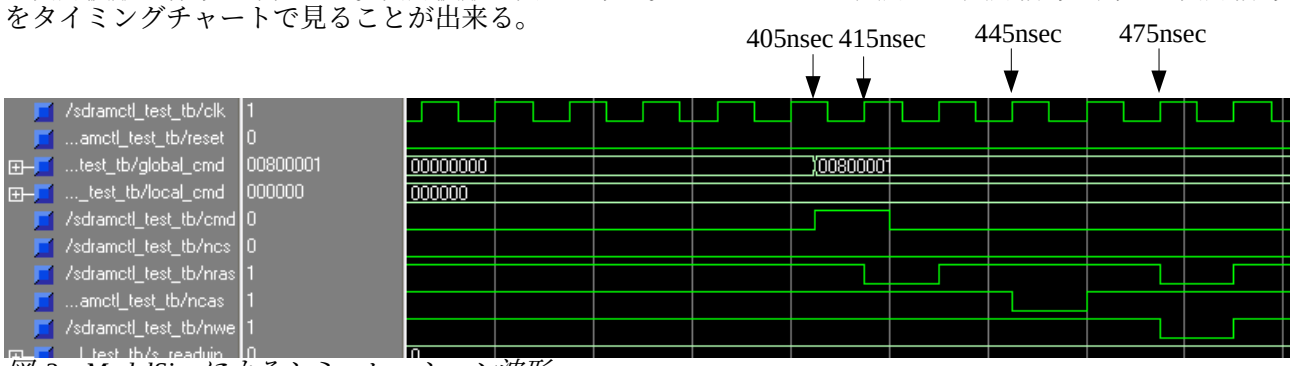

図 3　ModelSim によるシミュレーション波形

図 3 の時刻 405nsec のときに cmd が 1 になって、SDRAM 制御回路へのコマンド入力となる。global_cmd と local_cmd は SDRAM 制御回路へアドレスや動作モードを通知するために使用される。その後、SDRAM 制御回路は 415nsec で ACTIVATE コマンド、445nsec で READ コマンドを SDRAM に対して出力し、SDRAM リードを行う。その後、プリチャージコマンドを 475nsec に発行して、SDRAM リードサイクルを終了する。

このように、実際に FPGA に回路をダウンロードして、ロジックアナライザで確認することなく、回路の動作状況を確認できる。図 3 の段階でのシミュレーションは論理シミュレーションと呼ばれる。

FPGA の開発手順は、VHDL などの HDL ファイルや回路図を元に、論理合成、配置配線を行って FPGA の構造に合わせた回路に変換し、それを FPGA にダウンロードする。その際に、HDL ファイルや回路図の回路がそのまま最終回路になるわけではない。ツールが FPGA 独自の回路方式に合わせるために、中間の論理を変更する。そのために、自分で定義したノードが消されていることがある。論理合成後にシミュレーションを行うと、前述した理由でノードが消されている場合があるため回路の動作の解析が難しい。

論理シミュレーションでは、論理合成する前のソースファイルに対してシミュレーションを行うので、論理合成によって消されたノードはなく、自分で定義したすべての信号を見ることができる。これは、バグの発見をより容易にする。

4. NI FPGA 回路全体のシミュレーション

機能シミュレーションが終了したら、今度は全体の機能モジュールを接続して、NI FPGA 回路を完成させる。今度は NI FPGA 回路全体のシミュレーションをすることになる。ここでの全体のシミュレーションは、すべてシミュレーション・ツール(ModelSim)上で仮想的に行い、実際の電気的な回路は全く使用しない。

全体シミュレーションの最初の段階では、HDL Bencher を使用して、基本的な機能を確認する。シミュレーションが進んでいくと、機能を確認する際に比較的長いシークエンスを必要とするものが出てくる。例えば、PCI バスの初期化処理などである。その初期化処理をしないと NI FPGA 回路の機能を確認することが出来ない。よって、HDL Bencher で確認するのは困難である。そこで、各周辺デバイスのモデルを作製して、シミュレーションすることにした。

2 つの NI を LVDS チップを省いた LVDS インターフェース同士で対向接続して、全体シミュレーションをする。そのために、各デバイスの動作をシミュレーションするモデルを NI FPGA 回路に接続した。各モデルは VHDL で書かれている。FLASH ROM モデルは、チップの製造会社である富士通（株）のモデルを

使用した。SDRAM モデルは自分で作製したが、64Mbyte の容量では、シミュレータが異常終了してしまったので、256byte まで容量を減らした。PowerPC モデルと PCI バスモデルは、それぞれ複雑なシーケンスを必要とするので、どう実装するか迷ったが、実行させる 1 動作をテキスト 1 行で指定する方式にした。それぞれのモデルはテキストに書かれたコマンドを順次実行する。CPLD は余り全体の動作に関係ないので省いた。ブロック図を図 4 に示す。

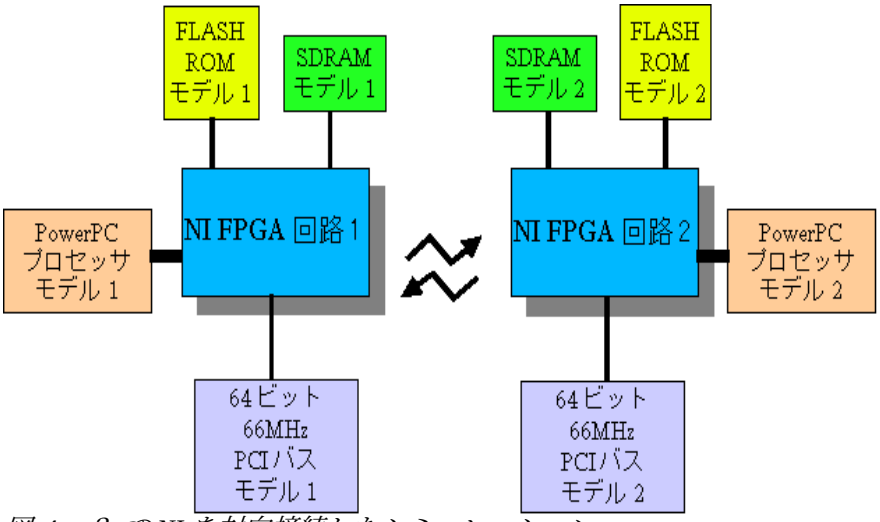

図 4　2 つの NI を対向接続したシミュレーション

4.1　PCI バスモデル

PCI バスモデルは、PCI バスをシミュレートする PCI バス・プロトコルモデルと、PCI ターゲット・コマンドプロセッサに分けられる。PCI バスモデルのブロック図を図 5 に示す。

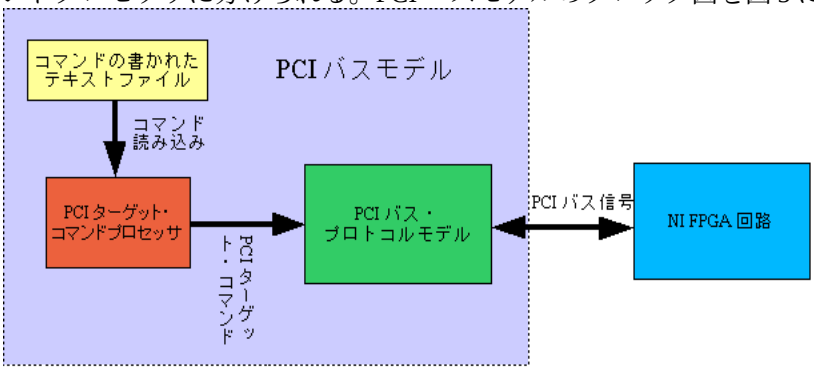

図 5　PCI バスモデル　ブロック図

PCI バス・プロトコルモデルには PCI のプロトコルが記述してある。また、簡単な PCI バス・アービタを持っている。その PCI バス・アービタは NI FPGA 回路の PCI マスタアクセスとホストパーソナルコンピュータ(ホスト PC)からの PCI バスアクセス（NI FPGA 回路からは PCI ターゲットアクセス）を切り分ける。通常は、ホスト PC のグラント(バスの使用許可)がアサートされていて、ホスト PC からの PCI ターゲットアクセスに PCI バス・プロトコルモデルがすぐに応答できる状態にある。NI FPGA 回路の PCI マスタアクセスの場合は、PCI バス・アービタに要求を送り、許可された場合に PCI マスタアクセスを行う。PCI ターゲット・コマンドプロセッサは、ホスト PC の PCI ターゲットアクセス用のモデルである。それは、テキストファイルのコマンドを VHDL の TEXTIO ライブラリのルーチンで読み込んで、PCI ターゲットアクセスを発生させる。コマンドの一例を下に示す。

```
0 ns 00000010 0 11 3000000020000000 1 0 # SDRAM - 0x20000000, NB – 0x30000000
```

最初のフィールドの 0 ns は遅延時間を表す。このコマンドを実行する時間間隔を表す。2 番目の 00000010 は 16 進数で表されたアドレスを表す。3 番目の 0 はデータの書き込みを表す。1 のときは読み込みである。4 番目の 11 はデータ幅を表し、この値はロングワード（64bit データ）を表す。その他に、バイト(8bit)、ショートワード(16bit)、ワード(32bit)の指定が出来る。5 番目の 3000000020000000 は書き込むデータを表す。データの読み込みの場合はこのフィールドは無視される。6 番目の 1 は PCI のコンフィギュレーション・アクセスである。0 の場合は通常のメモリアクセスとなる。尚、NI は IO アクセスをサポートしないので、IO アクセスに関するフィールドは無い。7 番目の 0 は、STOP 信号のアサートに関するフィールドで、1 にすると、次のクロックで STOP 信号をアサートして、PCI バスのトランザクションが停止する。な

ぜこのようなフィールドを設けたかというと、実際のホストコンピュータは頻繁に STOP 信号をアサートして、PCI バストランザクションを停止している。そのシミュレーションのために STOP 信号アサート用のフィールドを設けている。
このようにコマンドを順次実行することで、ホストコンピュータと NI 間のやり取りをシミュレーションすることが出来る。

4.2　PowerPC プロセッサモデル

PowerPC プロセッサは、アドレス転送とデータ転送が分かれている。データ転送にかかわらずにアドレス転送は 2 つの転送を完了できる。データ転送はアドレス転送の順番を守って、転送を行う。例えば、アドレス 4 への読み込みのアドレスを送ってデータを読みだす前に、次のアドレス転送、アドレス 8 への書き込みのアドレスを転送できる。その後、データ転送はアドレス 4 への読み込み、アドレス 8 への書き込みを順序を守って実行される。

PowerPC プロセッサモデルのブロック図を図 6 に示す。

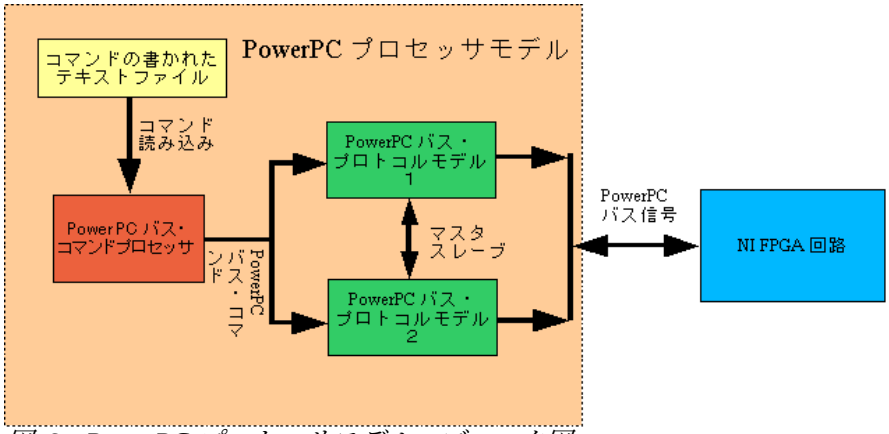

図 6　PowerPC プロセッサモデル　ブロック図

PCI バスモデルと同様に、テキストファイルのコマンドを 1 行ずつ PowerPC コマンドプロセッサが解析し、PowerPC バスコマンドとして、2 つの PowerPC バス・プロトコルモデルに送る。2 つの PowerPC バス・プロトコルモデルは、あらかじめマスタデバイスとスレーブデバイスが決められている。PowerPC バス・プロトコルモデルは 1 つのアドレス転送とそれに続くデータ転送を担当する。実際の動作は、読み込みの場合はデータ転送のアクノリッジを返すだけだが、書き込みの場合はデータ転送時にコマンドで指定されたデータを用意する。マスタの PowerPC バス・プロトコルモデルが動作中の時は、スレーブが動作を開始する。
下に PowerPC のコマンドの例を示す。

```
0 ns 40000050 0 8 1000000000000000
```

最初の 0 ns は PCI バスの場合と同様に、遅延時間を表す。2 番目の 40000050 は 16 進数で表された番地を示す。3 番目の 0 は書き込みを表す。1 の時は読み込みとなる。4 番目の 8 はデータ転送のサイズを表し、ロングワード(64bit)を示す。9 はバイト、A はショートワード(16bit)、C はワード(32bit)、2 はバースト転送(4beat)を表す。最後の 5 番目は書き込むときのデータを表す。このデータは読み込みの時には無視される。

5. まとめ

私の場合、回路を作製しただけで、動作することは 100%ありえない。よって、回路のシミュレーションは欠かせない。比較的大規模な回路を作成する場合には、機能モジュール単位でシミュレーションし動作を確認してから、機能モジュールを相互に接続し全体シミュレーションを行う。

Xilinx 社のデバイスを使用して機能モジュール単位でシミュレーションする場合には、HDL Bencher を使用して VHDL のテストベンチを作製し、ModelSim でコンパイル後シミュレーションを行う。HDL Bencher を使用した全体シミュレーションでは、出力信号と入力信号に依存関係のある場合は、HDL Bencher で入力波形を作成するのが難しい。よって、各デバイスのモデルを使用して全体シミュレーションを行うことにした。モデルのうち、マスタとなるデバイスはテキストファイルに書かれたコマンドで動作を記述する方式にした。これが、一番複雑な動作を表現するのが容易であると考えたからである。現在、実機の動作を解析しておかしいタイミングを発見した時に、今回作製した全体シミュレーションを使用し、バグの発見に役立てている。

欠点としては、PCI バスモデル、つまりホストコンピュータのモデルが簡単すぎる点である。実際のホストコンピュータは、都合によって頻繁に PCI バスの転送を止めたり、NI の転送要求に対して応答が遅い時がある。実際の回路動作のすべてをシミュレーションでモデル化できないので、すべてのバグをシミュレーションによって発見することはできない。

参考文献

[1] Xilinx Inc. , "Xilinx ISE 6 Software Manuals", 2003
[2] Model Tecnology Inc. , "ModelSim 5.6 リファレンスマニュアル", 1999-2002