社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－ARC－163（1）
2005／5／31

メモリスロット装着型ハードウェアの評価検証環境の構築

伊澤　徹[†]　渡邊　幸之介[†]　北村　聡[†]
宮部保雄[†]　宮代具隆[†]　天野英晴[†]

PCI バスやメモリバス等を介してホスト CPU との間で協調動作を行うような FPGA 上のハードウェアの開発においては，ホスト CPU が実際に行うものと同様のアクセスをシミュレーションで実現する必要がある．そこで我々は Verilog PLI の機能を用いて，Verilog HDL シミュレータと，C++により記述されたホストプログラムの協調シミュレーション環境を構築した．現在，この環境を用いて，メモリスロット装着型ハードウェアである DIMMnet-2 上の FPGA 内部論理を開発中である．この環境では，使用するライブラリを切り替えてコンパイルし直すことで，同一のソースファイルを実機およびシミュレーションの両方で実行することが可能である．

Cooperative Simulation Environment of Hardware Plugged into a DIMM slot

TETSU IZAWA,[†] KONOSUKE WATANABE,[†] AKIRA KITAMURA,[†]
YASUO MIYABE,[†] TOMOTAKA MIYASHIRO[†] and HIDEHARU AMANO [†]

In order to bypass the overhead of standard I/O buses, memory slots are useful to connect attachment hardware modules including network interfaces and reconfigurable accelerators. In such systems, the total job is divided, and performed with the cooperation of attached hardware and software on the host processor. In order to simulate such systems easily for development of the attached hardware, a cooperative simulation environment of Verilog-HDL and software on a real machine is proposed and developed. In the environment, a source code can be executed both by the simulation and with the real machine by changing the library.

A memory slot connected network interface DIMMnet-2 is now designed using the proposed co-simulation system.

1. はじめに

FPGA 等で構成されたハードウェアを設計する場合，Verilog HDL 等のハードウェア記述言語 (HDL) を用いて論理を記述し，実機上で検証を行う前に，RTL シミュレーションにより動作を検証する必要がある．このようなシミュレーションでは，FPGA の内部論理の他に，接続される ASIC や，メモリ等のシミュレーションモデルが必要となる．

設計対象のハードウェアモジュールが，ほぼ独立した形で動作する場合，シミュレーションモデルは比較的容易に構築することが可能である．一方，ハードウェアモジュールがホスト PC に接続され，ホストとの間で協調動作を行うような場合，十分な動作検証を行うためには，実際にホスト CPU 上で協調処理プログラムを実行した場合と同様のアクセスをシミュレーション上で実現することが不可欠である．

しかし，ハードウェア記述言語で記述されたホスト CPU やチップセットのシミュレーションモデルは一般に入手が困難であり，また仮に入手できたとしても，このようなモデルを用いて，実際にホスト上で協調処理を行うプログラムを実行した場合と同様の挙動を記述するのは非常に繁雑でコストが高い作業となってしまう．

さらに，実機上での動作検証の段階になって，一度 HDL で記述したホスト上 CPU の挙動を再度ホスト上のプログラムとして実装し直すのは二度手間であり，二つのコードの記述間に機能的な不整合が発生し，検証用コードが適切であったとしても不具合の見落とし等の問題が起こる危険性が考えられる．

現在，PC のメモリスロットに装着するネットワークインタフェースである “DIMMnet-2” の開発が行われている．DIMMnet-2 のコントローラは FPGA 上に実装され，用途上，ホスト CPU から頻繁にアクセスが行われることになる．

先に述べたような理由から，我々は DIMMnet-2 のコントローラの設計を行うに当たり，十分な動作検証

† 慶應義塾大学
Keio University

を効率的に行えるよう，Verilog PLI の機能を用いて，Verilog HDL シミュレータと C++により記述したホストプログラムを協調実行させるシミュレーション環境を構築した．構築したシミュレーション環境では，使用するライブラリを切り替えてコンパイルし直すことで，同一のソースファイルを実機およびシミュレーションのいずれにおいても実行することが可能となる．

以下，本論文では，まず 2 章において実装に用いた Verilog PLI について概要を述べ，次に 3 章において，本シミュレーション環境の構築の動機となった DIMMnet-2 について述べる．4 章でシミュレーション環境の実装の詳細を示し，5 章でプログラムの実例を示した後に，最後に 6 章でまとめを述べる．

2. Verilog PLI

Verilog PLI(Programming Language Interface[3]) は Verilog HDL の提供するインタフェースであり，Verilog HDL 以外のプログラミング言語を用いて記述された外部のルーチン (PLI ルーチン) を定義することが出来る．

PLI の用途としては，以下に示すようなものが想定されている．

- Verilog コードの遅延の動的な変更
- テストベクタ等の外部ファイルの読み込み
- 波形表示やデバッガの提供
- コンパイル済みの Verilog コードの解析
- Verilog 以外で記述されたシミュレーションモデルの利用
- シミュレーションと協調動作する外部プログラムとのインタフェース

PLI ルーチンを記述したライブラリは，Verilog シミュレーション実行時にシミュレータに動的にリンクして呼び出される．

C 言語あるいは C++で記述された PLI ルーチンは，Verilog のシミュレーションモジュール内でシステムタスクの形で明示的に呼び出すか，予め，特定の信号の変化に反応するコールバック関数や，シミュレーションの初期化，後処理等の関数として定義しておくことで，利用することができる．よって，PLI ルーチンの起動の主導権は Verilog シミュレータ側が持つことになる．

3. DIMMnet-2

DIMMnet-2[1] は，PC に一般的に搭載されている DDR メモリスロットに装着するタイプのネットワークインタフェースである．CPU に近いメモリスロットにネットワークインタフェースを装着することで，PCI バスを用いた場合に比べて通信レイテンシを低減させることが可能となる．また，低速な PCI バスしか備えていないような安価な PC においても広帯域なメモリバスが提供されていることから，ハイエンドな PCI バスに匹敵する高いバンド幅を獲得できると考えられる．

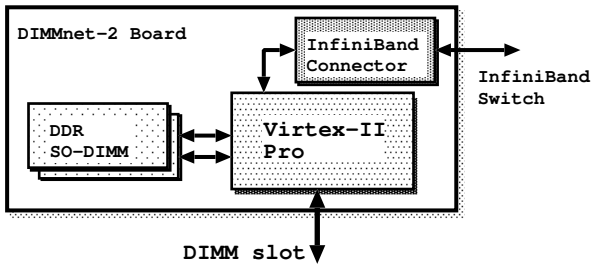

図 1 DIMMnet-2 試作基板のブロック図

また，メモリモジュールの代わりにネットワークインタフェースのコントローラが位置することになるため，コントローラに対して多様なメモリアクセス機構を実装することで，ネットワークインタフェース上に搭載されるメモリに対して，ホストからベクトルアクセスのような特殊なメモリアクセスを高速に行えるようになる[2]．

現在，DIMMnet-2 のプロトタイプである DIMMnet-2 試作基板を用いて，論理検証が行われている．DIMMnet-2 試作基板のブロック図を図 1 に示す．

試作基板は 184 ピンの DIMM スロットに装着され，PC1600 の DDR SDRAM として認識される．試作基板上には，コントローラを実装する Xilinx 社の FPGA(Virtex-II Pro XC2VP70) や，DIMMnet-2 がネットワークとして利用する InfiniBand[4] に接続するためのコネクタ，通信バッファやホストプログラムのデータ格納領域として用いられる DDR SO-DIMM 等が搭載されている．

3.1 コントローラの構成

コントローラは大きく分けて，以下のようなブロックから構成される．

- DDR ホストインタフェース部
  DDR メモリバスを介したホスト CPU とのトランザクション処理を行う．
- DDR SO-DIMM インタフェース部
  基板上の SO-DIMM へのアクセス制御を行う．
- コアロジック部
  送信パケットの生成や受信パケットの解析等の他，SO-DIMM インタフェース部を介して SO-DIMM へのアクセスを行う．

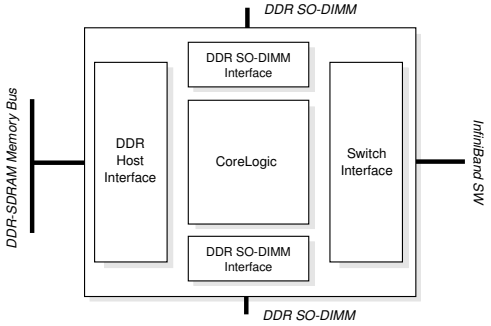

図 2 DIMMnet-2 コントローラのブロック図

- スイッチインタフェース部
  InfiniBand の低レベルなプロトコル処理や再送処理等を行う．

図 2 にコントローラのブロック図を示す．

3.2 DIMMnet-2 上の資源へのアクセス

DIMMnet-2 では，基板上の SO-DIMM はホスト PC の DDR メモリバスに直結していない．そのため，ホスト CPU は基板上の SO-DIMM に直接アクセスすることができず，コントローラを介して間接的に読み書きを行うことになる．ホスト CPU は，以下に示すコントローラ内の資源を介して，SO-DIMM へのアクセス等を行う．

- Write Window
  ホスト CPU から基板上の SO-DIMM やネットワークにデータを転送する際に中間バッファとして利用するメモリ領域である．この領域にデータを書き込んでから，後述する User Register に書き込み要求を書き込むことで，データが基板上の SO-DIMM に書き込まれる．また，User Register に通信要求を書き込むと，データはパケット化されてネットワークへ送出される．ホスト側からは書き込み専用，コントローラ側からは読み出し専用の領域である．
- Prefetch Window
  ホスト CPU による基板上の SO-DIMM からのデータ読み出しや，ネットワークからのデータ受信に，中間バッファとして利用されるメモリ領域である．User Register に読み出し要求を書き込むと，基板上の SO-DIMM からこの領域に読み出しが行われる．指定したサイズの読み出しが完了すると，完了フラグが User Register 内の特定の位置にセットされる．また，User Register に通信要求を書くことで，リモートの DIMMnet-2 のデータを読み出し，この領域に取得することも可能である．ホスト側からは読み出し専用，コントローラ側からは書き込み専用の領域である．
- LLCM(Low Latency Common Memory)
  ホスト CPU から通常のメモリと同様に汎用的に利用可能なメモリ領域である．受信したパケットの送信元等の情報を保存する用途にも用いられる．ホスト側とコントローラ側の両方から読み書きが可能である．
- User Register
  ホスト CPU からの通信や SO-DIMM へのアクセスの要求の発行，処理完了の通知等に用いるレジスタ群がマップされた領域である．この領域は，ユーザプロセスのメモリ空間にマップされ，ユーザレベルで直接アクセス可能となっている．
- System Register
  コントローラのリセットやネットワーク接続に必要な情報の設定等を行うためのレジスタである．この領域だけはカーネル空間にマッピングされ，ユーザは ioctl システムコールを呼び出してアクセスする．

system register 以外は，ユーザ空間に直接マップされユーザレベルでアクセスされる．

SO-DIMM へのデータ読み書きの手順を以下に示す．

- 書き込み時
  (1) Write Window へデータを書き込み
  (2) User Register の特定の番地にサイズやアドレスを書くことで，書き込み処理を要求
  (3) User Register 内のフラグにより，書き込み処理の完了を検出
- 読み出し時
  (1) User Register の特定の番地にサイズやアドレスを書くことで，読み出し処理を要求
  (2) User Register 内のフラグにより，読み出し処理の完了を検出
  (3) Prefetch Window からデータを読み出す

4. 実　　装

本章では，本研究において構築を行った協調シミュレーション環境の実装について述べる．

実機による動作検証と Verilog-HDL シミュレータ上での検証の双方において同一のテストコードを用いることを可能とするためには，実機で使用する C 言語や C++で記述されたテストコードを，シミュレータ上でも実機と同様に実行できる環境を整えればよい．

Verilog PLI は C 言語や C++のインタフェースを持つが，2 章で示したように，PLI ルーチンは Verilog コード内からの明示的な呼び出しや，信号線の変化によるコールバック等の PLI の仕様で定められたイベン

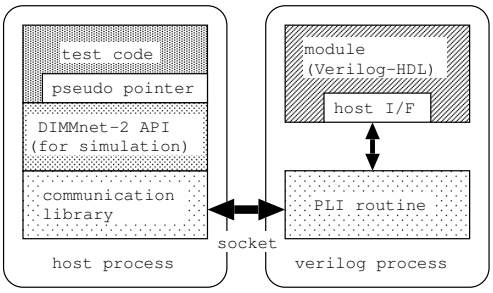

図 3 協調シミュレーション環境の構成

トに応じて起動される．

PLI ルーチンが実行されている間は，Verilog シミュレータ上のシミュレーション時間は進行しない．そのため，単純にテストコードを PLI ルーチンとして記述する場合，テストコード内に，必要に応じて関数から return して Verilog シミュレータに制御を戻し，適宜シミュレーション時間を進めるような記述をする必要が生じてしまう．このような理由から，実機と同様のテストコードを，そのまま PLI ルーチンとして用いて Verilog-HDL シミュレーション上での動作検証に用いることは難しい．

そこで本シミュレーション環境では，Verilog シミュレータのプロセス (以下 Verilog プロセスと表記) に対して，動作を要求し結果を受け取るだけのプロセス (以下ホストプロセスと表記) を導入する．これによりホストプロセスでは，前述のような Verilog シミュレータの制御を意識する必要がなくなり， C 言語や C++ で記述されたテストコードが主導のシミュレーションを行う事が可能となる．

ホストプロセスは，Verilog プロセスとソケットを用いたプロセス間通信を行うことで，シミュレータを制御する．シミュレーション環境の構成を図 3 に示す．

Verilog プロセスとホストプロセスの実装を以下に示す．

4.1 Verilog プロセス

Verilog プロセスは，Verilog-HDL によって記述されたモジュールと C++で記述された PLI ルーチンからなる．

検証対象である DIMMnet-2 試作基板は 3 章で述べたように，DIMM スロットに装着され DDR SDRAM として認識される．そのため，DIMMnet-2 コントローラのシミュレーションでは，Finish や Wait 等のシミュレーション自体を制御する命令の他，Write や Read, Memory Register Set(MRS)☆等の DDR SDRAM を制御する命令が必要となる．

```
integer fin, stopper;

for(fin=0; fin<1; stopper=0) begin
  $sv_wait();
  wait(stopper);
end
```

図 4 Veriog プロセス内のループ

また，命令によっては，Write ならば書き込むデータ，MRS ならばバースト長等の設定項目といったように，実行に付加情報が必要となる場合がある．そこで，ホストプロセスから Verilog プロセスに対しては命令の他に，その命令に必要な付加情報を合わせて送信するものとし，これをコマンドと称することとする．

ホストプログラムからシミュレーション時にコマンドがいくつ来るかを事前に知ることが出来ないため，図 4 に示すような無限ループ内でコマンド受信用の PLI ルーチン ($sv_wait()) を呼び出し続けることで，コマンドの受信を行う．

$sv_wait() では，ホストプロセスからコマンドが送られてくるのを待ち，受信後に以下の処理を行う．

- ソケットからのコマンドの受信
- 受信コマンドの処理
  - コマンドがメモリバスへのアクセスを表すものであった場合，DIMMnet-2 のホストインタフェースモジュールに対して DDR SDRAM のコマンドを発行
  - メッセージがシミュレーションの制御を行うものであった場合，直接制御処理を実行
- 遅延処理
  - 受信したコマンドに対応する動作が完了し，次のコマンドを受け付けられる時刻まで, シミュレーション時間を進行
    - コマンドに応じたシミュレーション時間後にアサートされる stopper 信号を，メインループ内で wait で待つことで遅延を付加

4.2 ホストプロセス

ホストプロセスは図 3 に示すように，通信ライブラリ (client ライブラリ)，疑似ポインタクラス (ptr クラス) およびシミュレーション用の DIMMnet-2 API で構成される．

通信ライブラリには，発行するコマンドに対応した

☆ バースト長や CAS Latency 等のメモリモジュールのパラメータを設定する命令

```
DATA d; // データとマスクが対になった構造体
d.d = 0x1;  d.m = 0;

client::write(0x50000, d);
long n = client::read(0x20000);

client::finish();
```

図 5 通信ライブラリ使用例

関数 (Read, Write, MRS, Finish 等) が用意されている．それぞれ Verilog プロセスに対してコマンドを送信し，必要なら結果の受信まで行う．

ただし，通信ライブラリの提供する関数をそのまま用いてテストコードを記述しようとした場合，図 5 の例のようにメモリアクセスを行う際に，プログラム中でアクセス対象領域の物理アドレスを，明示的に指定する必要が生じてしまう．

3 章で述べたように，User Register や各種 Window 等はユーザメモリ空間上にマップして利用されるため，ユーザプロセスからは物理アドレスを意識したアクセスは行われず，また明示的に関数を呼び出してのメモリアクセスを行う必要が無い．

そのため，実機用に記述されたテストコードを通信ライブラリ上で利用しようとした場合，ポインタを用いたメモリアクセスを，通信ライブラリの対応する関数呼び出しに置き換える必要がある．

これを実現するために，C++で提供されている演算子のオーバーロードを使用して，ポインタとほぼ同様な操作が可能なテンプレートクラスである疑似ポインタクラス (ptr クラス) の実装を行った．

疑似ポインタクラスは，内部にメンバ変数としてアドレスの値を持ち，ポインタ実体への代入なら Write，ポインタ実体の参照なら Read というように，アクセスに応じて通信ライブラリを呼び出し，アドレス値とあわせてシミュレータに対して対応するコマンドを発行する．インスタンスに対してポインタ演算が指定された場合には，このアドレス値に対して演算が行われる．

疑似ポインタクラスの実装の一部を図 6 に示す．この疑似ポインタクラスを使うことで，通信ライブラリのみの場合と異なり，Verilog シミュレーション上でも実機上でのメモリ操作と同様な透過的なメモリアクセスが可能となる．

テストコードを実機上で動作させる場合は，疑似ポインタクラスは不要となり，代わりに通常のポインタを使用することになる．この切り替えは，マクロ定義を用いて行う．予め，テストコードを記述する際に，

```
template <typename T> class ptr{
  …
  ptr<T>& operator++(){
        adr += sizeof(T);
        return *this;
  }
  ptr<T>& operator+=(const int &rhs){
    adr += rhs*sizeof(T);
    return *this;
  }
}
```

図 6 疑似ポインタクラスの実装の一部

```
#define  P_INT  ptr<int>
P_INT p;
...
*++p = 0x100;
int tmp = *(p+10);
```

図 7 疑似ポインタ使用例

int 型を用いずに P_INT 型のようなマクロ定義された型を用いるようにし，コンパイル時のヘッダファイルに応じて，その実体を通常のポインタと疑似ポインタクラスとで切り替える．

疑似ポインタクラスの使用例を図 7 に示す．この例では，値の代入，ポインタ演算，ポインタの参照先からの値の読み出しを行っている．

4.3 DIMMnet-2 API

DIMMnet-2 は，ユーザレベルで利用可能なネットワークインタフェースである．そのため，上位のユーザプログラムに対して DIMMnet-2 へのアクセスを提供する DIMMnet-2 API は，直接 DIMMnet-2 のハードウェアに対してアクセスを行うユーザレベルの関数と，DIMMnet-2 のデバイスドライバを介してアクセスを行う関数を提供している．

- setup, cleanup
  DIMMnet-2 のデバイスの open/close や，その他の初期化/終了処理を行う．
- sodimmInit, controllerReset 等
  内部でデバイスドライバを呼び出し，System Register を操作する．
- getControllerStatus 等
  User Register を操作する．
- getWriteWindow, getPrefetchWindow 等
  Window のポインタを得る．
- VL, VS 等
  User Register のコマンド領域にパラメータを書き込み，SO-DIMM へのアクセスや通信等，

DIMMnet-2 の持つ命令 (プリミティブ) を発行する.

- waitSodimmWrite, waitPrefetchFlag 等
  特定の番地のフラグが条件を満たすまでポーリングする.

これらの関数は元々実機用であるが，疑似ポインタクラスに対応させることで，シミュレーション上でも利用可能とした.

コンパイル時にどちらの API をリンクさせるかを選択することで，同じソースコードから実機用の実行ファイルと，シミュレーション用のホストプロセスとなる実行ファイルとを生成することが可能である.

2 種類の DIMMnet-2 API の内部実装の違いを次に示す.

- 実機用 API
  - ポインタを用いて読み書き
  - System Register へのアクセスは ioctl を呼び出し，ドライバが処理
- シミュレーション用 API
  - 通信ライブラリを呼び出して読み書き

5. ホストプログラム動作例

実際に本環境で動作するホストプログラムの例を図 8 に示す．このプログラムでは，以下に示す処理を行っている.

(1) 試作基板の初期化処理
(2) Write Window のポインタを取得
(3) データファイルを open
(4) Write Window へのポインタを使った書き込み
(5) SO-DIMM への連続書き込み命令
(6) 書き込み完了フラグのポーリング

本研究で実装を行った協調シミュレーション環境を利用することで，ホスト CPU と協調処理を行うハードウェアシミュレーションを，このような簡潔な記述で行うことができる．また，この記述はコンパイルし直すだけで実機上でもそのまま利用することができ，実機検証時の開発コストを削減することができる.

6. ま と め

本研究では Verilog PLI の機能を用いて，Verilog HDL シミュレータと C 言語により記述したホストプログラムを協調実行させるシミュレーション環境を構築した.

本研究で実装を行った協調シミュレーション環境を利用することにより，ホストプロセッサの挙動まで含めた複雑な動作を検証することが可能となった．また，実機とシミュレータで同一のコードを使用できることにより，検証にかかるコストが軽減された.

```
#include <fstream>
#include "dnet2_api.h"

int main(int argc, char **argv){
  using namespace std;
  using namespace dnet2;
  P_ULL ww0;

  goma();           // host port enable
  controllerReset();// controller reset
  sodimmInit();     // SO-DIMM Initialize

  ww0 =(unsigned long long*)getWriteWindow(0);
  ifstream infs("data");

  for(i=0, p = ww0; i<0x20; i++, p++){
    infs >> tmp;
    *p = tmp;
  }
  VS(0 ,0 ,0x100);  // vector store
  waitSodimmWrite();// wait complete flag
  ...
```

図 8 ホストプログラムの例

現在も DIMMnet-2 のコントローラの開発が進んでおり．InfiniBand を用いた通信機能の検証も行われている．今後は DIMMnet-2 のメモリスロット側の接続の他に，InfiniBand コネクタ側の入出力もシミュレーションが行えるよう，拡張を行う予定である.

謝辞 本研究は総務省戦略的情報通信研究開発推進制度の一環として行われたものです.

(株) 日立 IT の今城氏，岩田氏，上嶋氏，(株) 東芝 研究開発センターの田邊氏，東京農工大学の中條氏，浜田氏，慶應義塾大学の西助手，大塚氏をはじめ DIMMnet-2 の開発に関する議論、開発にご参加頂いている全ての方々に感謝いたします.

参 考 文 献

1) 田邊 昇, 濱田 芳博, 三橋 彰浩, 中條 拓伯, 天野 英晴: メモリスロット装着型ネットワークインタフェース DIMMnet-2 の構想, 情報処理学会アーキテクチャ研究会, Vol.2003-ARC-152, pp.61–66(2003).
2) 田邊 昇, 土肥 康孝, 中條 拓伯, 天野 英晴: メモリスロット装着型ネットワークインタフェース DIMMnet-2 の構想, 情報処理学会アーキテクチャ研究会, 2003-ARC-152, Mar.2003.
3) IEEE: IEEE Standard Hardware Description Language, IEEE std 1363-2001, 2001
4) InfiniBand Trade Association http://www.infinibandta.com/