2004－SLDM－117（9）
2004／12／1

標準協調エミュレーションモデリングインターフェース SCE-MIのFPGAボードへの実装試行

大山 将城† 名野 響† 近藤 信行† 清水 尚彦† 星野 民夫††

† 東海大学 〒259–1292 神奈川県平塚市北金目 1117
†† 株式会社 アプリスター 〒243–0021 神奈川県厚木市岡田 3050 厚木アクストメインタワー 3F
E-mail: †{4aepm010,3aepm036,1adt2415,nshimizu}@keyaki.cc.u-tokai.ac.jp, ††hoshino@applistar.com

あらまし 昨今のSoC開発においてハードウェア, ソフトウェアの協調エミュレーションは盛んに行われている. しかし, エミュレーション環境はベンダツールへの依存度が高いケースが多く標準的といえるものはまだ無い. したがって, 構築したエミュレーション環境が使用しているベンダツールに縛られているのが現状である. これに対し accellera [1] は2003年に標準協調エミュレーションモデリングインターフェース SCE-MI を策定した. SCE-MI は披試験デバイス (以下 DUT) に対するテストベンチの設計/使用を容易にすることを目的にハードウェア/ソフトウェア間のインターフェース (以下 IF) 仕様を定義するものである. IF のソフトウェア側は C++の API として, ハードウェア側は Transactor と呼ばれるモジュールとして定義される, ただし, これらの実装仕様については定められておらず実装者に任されている. そこで標準的エミュレーション環境の実装試行として,SCE-MI 仕様にのっとった IF 開発と FPGA ボードへの実装を行った.

キーワード LSI開発環境, 協調エミュレーション, 標準インターフェース,SCE-MI

Implementation of Standard Co-Emulation Modeling Interface(SCE-MI) on an FPGA

MASASHIRO OHYAMA†, HIBIKI NANO†, NOBUYUKI KONDOH†, NAOHIKO SHIMIZU†, and TAMIO HOSHINO††

† Tokai University 1117 Kitakaname, Hiratsuka, Kanagawa, 259–1292 Japan
†† Applistar Corporation
3050 Okada, Atsugi, Kanagawa, 243–0021 Japan
E-mail: †{4aepm010,3aepm036,1adt2415,nshimizu}@keyaki.cc.u-tokai.ac.jp, ††hoshino@applistar.com

Abstract Recently at SoC development, designers often use hardware/software co-emulation. But, there is no environment which can be called standard emulation environment. *Accellera* set Standard Co-Emulation Modeling Interface(SCE-MI). SCE-MI defines an interface specification between hardware and software, for the purpose of ease creating test bench's design and use. Software side interface is defined as a C++ application interface, hardware side interface is defined as a module which is called Transactor. But, implementation specification is not defined. So we developed interface which is based on SCE-MI and implement it on an FPGA evaluation board.

Key words LSI Development Environment, Co-Emulation,Standard Interface,SCE-MI

1. はじめに

SoC開発が盛んに行われるようになり, 検証手法が色々と提案されている. 協調エミュレーション環境においては独自のアプリケーションインターフェース (以下 API) を持つものが多く, 移植性が低いのが現状である. そのため, 限られたツールでのみ使用できるエミュレーション環境を構築せざるをえない.

したがって, エミュレーション環境の標準化は移植性, 生産性の点から有用である. そこで協調エミュレーションの標準化を目的として2003年に accellera は SCE-MI を策定した. SCE-MI はハードウェアとソフトウェア間の IF 仕様としてソフトウェア側に C++API を, ハードウェア側に RTL 記述モジュールで

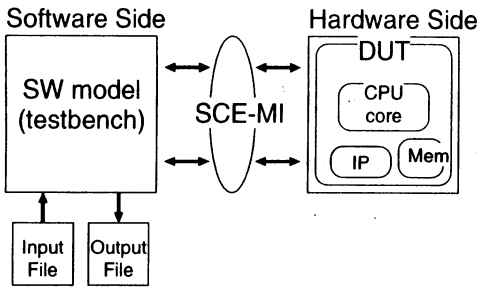

図 1 SCE-MI の概念図

ある Transactor を定める. ただし,SCE-MI は実装仕様までは定めておらず実装者が実装法を考える.

今回, 標準的な協調エミュレーション環境の実装試行として SCE-MI 仕様を満足する IF の開発を行い,FPGA 評価ボードへの実装を行った.

2. SCE-MI 仕様概要

まず,SCE-MI の仕様について説明する. SCE-MI はソフトウェアとハードウェアの間の IF であるので図 1 のイメージとなる.

仕様書にはソフトウェア側 (テストベンチの入力側. 以下ソフトウェアサイド) とハードウェア側 (DUT 側. 以下ハードウェアサイド) の 2 つの仕様が記載され, 以下のように定められている.

2.1 ソフトウェアサイドの仕様概要

C++で以下の 6 つのクラスが定められている.

(1) SceMiEC

エラーハンドリング用のクラスである.

(2) SceMi

SCE-MI の主要な処理を受け持つクラスである.

(3) SceMiMessageInPortProxy

アプリケーション側からハードウェア (Transactor) の入力ポートにアクセスするためのインターフェースを提供するクラスである.

(4) SceMiMessageOutPorProxy

アプリケーション側からハードウェア (Transactor) の出力ポートにアクセスするためのインターフェースを提供するクラスである.

(5) SceMiParameters

アプリケーション側からハードウェアにアクセスするために決められた形式のパラメータファイルが必要になる. 本クラスはパラメータファイルへのアクセスをするためのクラスである.

(6) SceMiMessageData

ソフトウェアサイドの SceMiMessageInPortProxy から結び付けられたハードウェアサイドの SceMiMessageInPort まで, またはハードウェアサイドの SceMiMessageOutPort から結び付けられたソフトウェアサイドの SceMiMessageOutPortProxy までのメッセージデータの受渡しをする機能を提供するクラス.

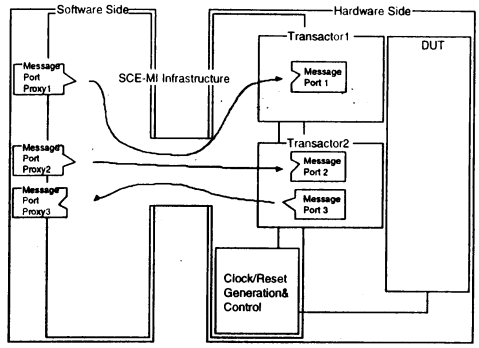

図 2 SCE-MI の概念図 (詳細版)

2.2 ハードウェアサイドの仕様概要

ハードウェアサイドは Transactor および Port 呼ばれるハードウェア論理で構成する. Transactor がソフトウェアサイドのメッセージポートプロキシーとの通信を行い, ソフトウェアと DUT との信号のやりとりが実現する. Port は Transactor 内に存在しデータの入力, 出力, クロックの供給といった機能を提供するモジュールでバッファ的な役割をする.Port の制御は全て Transactor により行われる.

以下,Port の種類とその説明を行う.

(1) SceMiMessageInPort

Transactor に対しソフトウェアサイドからメッセージが到着していることを通知する. メッセージの受渡しは Transactor とのハンドシェイクプロトコルで行う.

(2) SceMiMessageOutPort

Transactor からソフトウェアサイドにメッセージを渡す時に使用する. SceMiMessageInPort と同様にハンドシェイクプロトコルによりメッセージの受渡しを行う.

(3) SceMiClockPort

DUT に対して制御クロックを供給する. 周波数, デューティ比, 位相についてはパラメータで指定する. DUT に対する制御リセット信号についてもこの Port が供給する.

(4) SceMiClockControl

Transactor が DUT に対して制御クロックを供給するためにこの Port が必要になる.

Transactor,Port を含め図 1 をより詳細なモデルとして表現したのが図 2 である.

3. FPGA ボードへの実装

SCE-MI の仕様を踏まえ C++クラスファイル,Transactor の開発を行った.

PCI-IF [2] に DUT を接続する形をとり, 実装を行った. これにより, 図 3 のようにユーザーインターフェース (UI) 側から DUT へのアクセスをする.

次に Transactor の仕様について説明する.

3.1 Transactor の仕様概要

- SceMiMessageInPort[31:0], SceMiMessageOutPort[31:0]

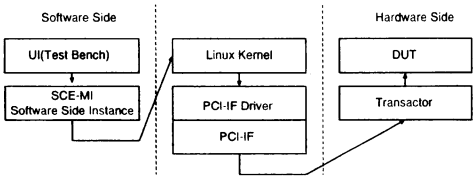

図 3　UI から DUT へのアクセス

表 1　ピン仕様

入出力バス・ライン仕様

端子名	方向	バス幅 [bit]	機能
pI1_Write	入力	1	SceMiMessageInPort への書き込み制御信号 Port のステータスレジスタへの書き込みは不可能
pI1_Read	入力	1	SceMiMessageInPort への読み込み制御信号
pI1_ms	入力	1	SceMiMessageInPort に対する制御ターゲット指定 (0→ Message レジスタ, 1→ステータスレジスタ)
pI1_in	入力	32	SceMiMessageInPort への入力バス
pI1_out	出力	32	SceMiMessageInPort からの出力バス
pO1_Write	入力	1	SceMiMessageOutPort への書き込み制御信号 Port のステータスレジスタへの書き込みは不可能
pO1_Read	入力	1	SceMiMessageOutPort への読み込み制御信号
pO1_ms	入力	1	SceMiMessageOutPort に対する制御ターゲット指定 (0→ Message レジスタ, 1→ステータスレジスタ)
pO1_in	入力	32	SceMiMessageOutPort への入力バス
pO1_out	出力	32	SceMiMessageOutPort からの出力バス
DUT_OutCtrl	出力	1	DUT への制御出力
DUT_Out	出力	32	DUT へのデータ出力
DUT_InCtrl	入力	1	DUT からの制御入力
DUT_In	入力	32	DUT からのデータ入力
Creset	出力	1	DUT のリセット
Cclock	出力	1	DUT へのクロック出力

を搭載した双方向 Transactor

- DUT との入出力は 32 ビットの入力バスと出力バスを用意
- 内部モジュールは PCI バスクロックに同期
- SceMiMessageInPort へデータをセットし, SceMiMessageOutPort からデータを受信する事で自動的に DUT に供給する Cclock が 1 クロック遷移
- クロック, リセットの制御用に SceMiClockControl,SceMiClockPort を 1 つずつ用意

実装時のハードウェア構成を図 4 に示す.

PCI ドライバを使用してのアクセス時には Port1 つに対してデータ, ステータス用のメモリ空間を 4 バイトずつ計 8 バイトのアドレスを図 5 の要領で割り振った.

また,DUT は 32 ビットの入力データを内部のレジスタでラッチし 1 加算した結果を出力するモジュールとした.

3.2　テストベンチ開発

動作の検証用にテストベンチの開発を行った. すでに開発している SCE-MI クラスファイルを使用し図 6 のソースコードでの検証を行った. 内容は PCI-IF の初期化,SceMi クラスのインスタンスを生成, メッセージデータのインスタンスを生成しインポートプロキシへデータの書き込み/確認の繰り返しである.

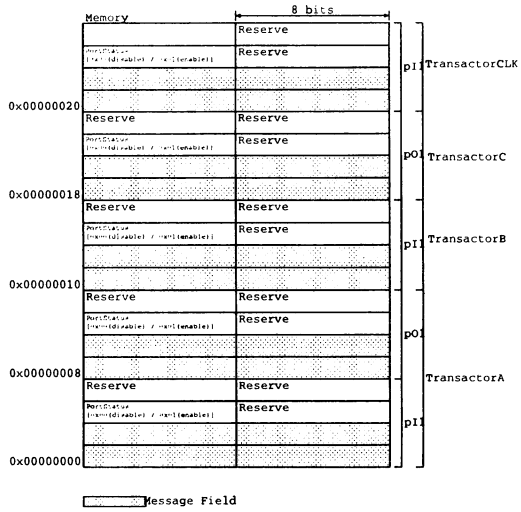

図 5　Transactor のデータ/ステータスレジスタのメモリマップ

```
//PCI-IF の初期化
ResetPCI();

//SceMi 初期化
SceMi *scemi = SceMi::Init(versionchk,scepam,NULL);

//メッセージデータインスタンス生成
SceMiMessageData *mdata =
new SceMiMessageData(*InP1 , NULL);

//送信するメッセージデータ
unsigned mDat = 2;
for(int bbb = 0 ; bbb < 10 ; bbb++) {
    cout << "input data = " << mDat << endl;
    //メッセージデータセット
    mdata->Set(0 , mDat , NULL);
    //データ送信
    InP1->Send(*mdata , NULL);
    //受信確認
    scemi->Serviceloop();
    //メッセージデータ更新
    mDat *=2;

    cout << "end 1cycle" << endl;
}

cout << "complite program!" << endl;
```

図 6　テストベンチソースコードの一部抜粋

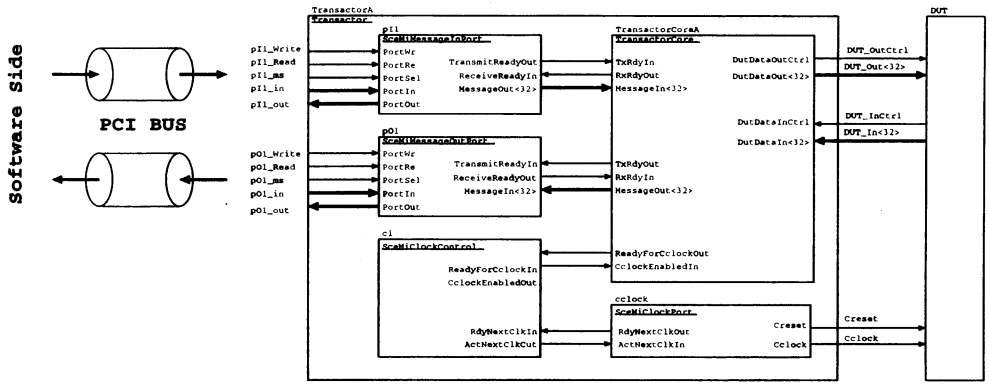

図 4　実装時のハードウェア構成

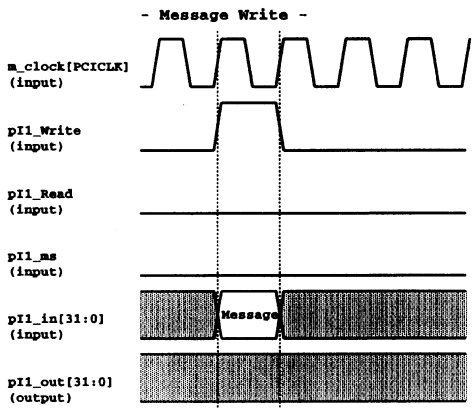

図 7　SceMiMessageInPort のメッセージ書き込み

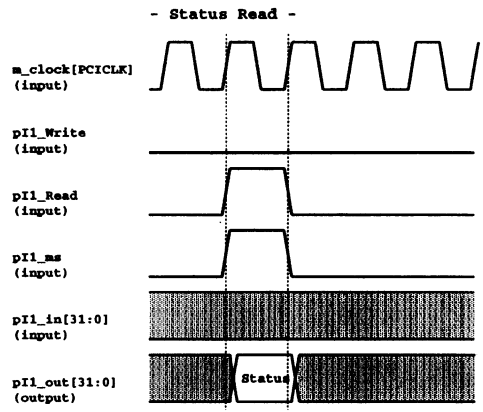

図 8　SceMiMessageInPort のステータス読み出し

4. エミュレーションの実施

ハードウェア論理とクラスファイルおよびテストベンチを用意したので実際にエミュレーションを行った．エミュレーション時は PCI バスを介してアクセスしタイミングは以下のとおりである．

4.1 SceMiMessageInPort へのアクセス

Transactor 内の SceMiMessageInPort への書き込みタイミングを図 7 に示す．SceMiMessageInPort は受信した Message を保存するレジスタ “MessageReg” と Port 内の状態を示すステータスレジスタ “StatReg” を持つ．Transactor と接続するモジュールは pI1_ms で書き込み対象レジスタを指定可能だが，本バージョンでは StatReg への書き込みを許可していない．

Transactor 内の SceMiMessageInPort からの読み出しタイミングを図 8 に示す．本バージョンは “StatReg” からの読み出しのみサポートする．

読み出した StatReg の最下位ビットが 0 の時は，前の処理でソフトウェアから書き込んだ MessageReg の値が TransactorCore に送信されていない事を示し，ソフトウェアサイドから Message の書き込みは不可能．最下位ビットが 1 の時は TransactorCore へ Message 送信が完了した事を示し，ソフトウェアサイドから新しい Message を送信可能．

4.2 SceMiMessageInPort へのアクセス

SceMiMessageOutPort も SceMiMessageInPort と同様に TransactorCore から受信した Message をセットする MessageReg と MessageReg のステータスを示す StatReg を持つ．本バージョンは SceMiMessageOutPort の MessageReg と StatReg への書き込み処理は許可していない．

Transactor 内の SceMiMessageOutPort からの読み出しタイミングを図 10 に示す．SceMiMessageOutPort については pO1_ms で “MessageReg” と “StatReg” のどちらから読み出すかを選択可能．

読み出した StatReg の最下位ビットが 0 の時は，MessageReg の値が更新されていない事を示し，最下位ビットが 1 の時は TransactorCore から Message を受信し，MessageReg が更新されている事を示す．

このタイミングチャートを踏まえ検証結果図 11 を見る．PCI-IF の初期化後,SCE-MI を通じて DUT にアクセスし一回のラ

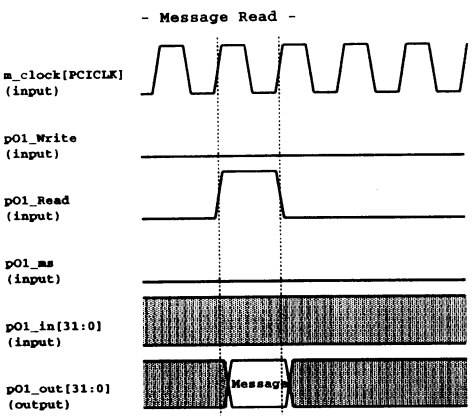

図 9 SceMiMessageOutPort のメッセージ読み出し

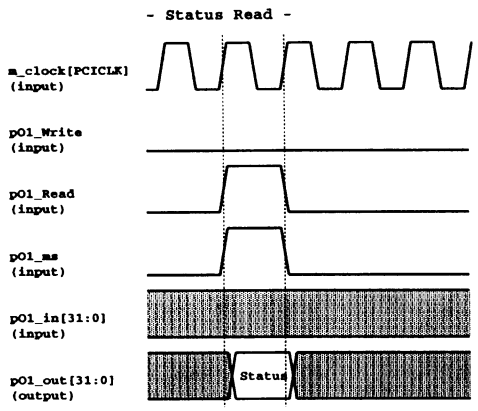

図 10 SceMiMessageOutPort のステータス読み出し

表 2 論理合成結果

モジュール名	ロジックセル数
PCI-IF	449
Transactor	216
DUT	32

イトリードサイクルを経て 1 加算した結果が正常に出力されていることが確認できた.

5. 合成結果

FPGA 実装時の結果について評価する. QuartusII ver3.0 を使用しターゲットデバイスを EP1S10F780C7ES とした時, 動作周波数は 88.46[MHz] であった. また, 論理の規模は表 2 のようになった. Transactor は 32 ビット入力 Port1 つ,32 ビット出力 Port1 つ, クロック供給 Port1 つの構成で 216 ロジックセルを使用した. 無視できるほど小さい値ではないが,Port が 10 個程度までなら実用的に実装できる数字と評価している.

6. まとめ

協調エミュレーションの標準インターフェース SCE-MI の仕様に沿ったハードウェア論理の開発, クラスライブラリの開発を行った. また, これらを使用して実際に FPGA でエミュレーションを試行し, 結果, 正常な動作を確認した.

現在は単純な仕様での実装にとどまり, 複雑な DUT の検証についても行っていない. 今後はより実装仕様の洗練をし,Transactor をハードウェアのソースコードから自動生成するソフトウェアの開発についても行う予定である.

```
in ResetPCI()
reset now!
reset finish

input data = 2
writting now!
receiveRdy called
ReceiveOutData = 1025
end 1cycle

input data = 4
writting now!
receiveRdy called
ReceiveOutData = 3
end 1cycle

～中略～

input data = 512
writting now!
receiveRdy called
ReceiveOutData = 257
end 1cycle

input data = 1024
writting now!
receiveRdy called
ReceiveOutData = 513
end 1cycle

complete program!
```

図 11 エミュレーション時のテキストログ

文献

[1] accellera:"SCE-MI Reference Manual DRAFT",2003
[2] 早坂, 志水, 横山, 孕石 : "第 9 回 ASIC デザインコンテスト 規定課題 B PCI バスインターフェース" 第 22 回パルテノン研究会,2003
[3] ALESSANDRO RUBINI : "LINUX デバイスドライバ" O'REILLY ジャパン,1998