システムＬＳＩ設計技術　108－1
(2003. 1. 28)

Javaを用いたハードウェア設計

大沼 孝至　金子 格　白井 克彦

†早稲田大学大学院理工学研究科 〒169-8555 東京都新宿区大久保 3-4-1

E-mail: {onuma, itaru-k, shirai}@shirai.info.waseda.ac.jp

あらまし　設計されるハードウェアが大規模化するなかで十分なシミュレーションを行うための設計手法として Java と VHDL を併用する方法を提案する．ハードウェアの機能シミュレータを Java で記述し，Java コンパイラの最適化機能を活かすことで，高速なシミュレーションが可能となる．また，オブジェクト指向やマルチプラットフォーム，ネットワーク分散処理など Java のシステム記述能力を生かすことで，ソフトウェアモデルとハードウエアモデルのシームレスな結合も可能である．将来のシステム記述言語としての Java の可能性を考察し，Java を用いたハードウェアシミュレーションの概要と設計方法およびその有効性について説明する．

キーワード　シミュレーション，Java，RMI，高速化

The hardware design using Java

Takashi ONUMA　Itaru KANEKO　and　Katsuhiko SHIRAI

†Graduate School of Science and Engineering, Waseda University　3-4-1 Okubo, Shinjuku-ku, Tokyo, 169-8555 Japan

E-mail: {onuma, itaru-k, shirai}@shirai.info.waseda.ac.jp

Abstract　The method using Java and VHDL as the design technique for sufficient simulation of the larger scale hardware design is proposed. Java describes the functional simulator of hardware and a high-speed simulation is possibly assisted by the optimization by the Java compiler. Moreover, the seamless combination of software models and hardware models is also possible by employing the system description capability of Java, such as object-oriented description, multiple platform support and network based distributed processing. The possibility of Java as a future system description language is considered, and the outline of a hardware simulation and the design method using Java, and its validity are explained.

Keyword　Simulation, Java, RMI, High-speed

1. はじめに

携帯電話，PDA，AV 機器などの電子機器のマルチモーダル化，ネットワーク化が進み，機器に搭載される電子回路も多機能化，複雑化，大規模化が進み，汎用マイクロプロセッサと専用プロセッサとを一つの半導体チップ上に搭載したシステム LSI と呼ばれる集積回路が主流になりつつある．システム LSI の搭載される電子機器の将来的な利用動向として，ヒューマンインターフェースのマルチモーダル化，ネットワーク接続などの機器の高機能化と，それに伴う情報リソースの拡大，通信の広帯域化などが予想される．電子機器においては限られた大きさの筐体に多くの機能が搭載されることが予想され，LSI チップの高機能，高集積という条件をクリアしなければならない．しかし，LSI の設計・製造のための初期コストが急速に上昇し，開発期間も長期化している．これは製造技術が進んで大規模な回路が LSI に入るようになったにもかかわらず，それだけ大きな LSI を設計する技術が追いついていないためである．

LSI 設計の検証工程では，主にソフトウェアによるさまざまなシミュレーションが行われていて，製品の品質を保証するためにかなりの重点がおかれている．しかし，設計規模の増大，機能の複雑化により検証時間は急激に増大し，適切な時間内に設計品質を確保することが困難になりつつある．この問題を解決するためには限られた時間内に，設定された検証項目を検証することが必要であるが，専用プロセッサの高速化によりリアルタームシミュレーションはますます困難になり，他方論理検証の結果，設計のやり直しということになれば莫大な費用がかかってしまう．

このような現状から克服すべき課題は，大規模ソフトウェアに匹敵するハードウェアをいかに簡単に設計するか，製品開発に必要なリアルタイムシミュレーションにどう対応するか，ソフトを含めたシステム全体の検証をどう行うか，急速にすすむ並列化処理に対応したハードウェアをどのように設計するかであると考えた．そしてこれらの課題を解決するための方法として Java をハードウェア設計に利用することを提案したい．

2. Java による設計システムの概要

Java による設計システムは，ハードウェアのシミュレータを含む Java クラスで構築した Java フレームワークである．このクラスにはシミュレータの他に VHDL 生成のメソッドを実装する．シミュレータは入力信号から値を受け取り回路の内部動作に相当する演算を施して出力信号に引き渡す．この動作は Java 演算子を使って簡潔に記述されるため非常に高速に処理される．VHDL 生成メソッドは，既に用意された VHDL コードを Java クラスに対応した VHDL ソースファイルとしてはきだす．

プリミティブな回路からそれらを組み合わせたやや複雑な回路まで，いくつか Java クラスを作成することにより，ハードウェアのライブラリを構成し，設計システムとして Java フレームワークの一部を構築することができる．

まず Java 併用設計の利点を説明し，それに沿った記述方法や設計方法さらに簡単な設計例を説明する．

2.1. Java 併用設計の利点

Java と VHDL を併用することで，当然一つのハードウェアについてはコードの記述量が増えることになるが，設計の全工程から考えると大きなメリットがある．

第一は，コンポーネントを Java で記述することで Java 処理系が持つ最適化コンパイラ能力によって単一で動作させても VHDL のシミュレータより十分高速にシミュレーションできることである．さらに，CORBA や Java の RMI によって大規模な回路をネットワーク上に分散させ超並列なシミュレーションが実現できることである．動画像処理を行うハードウェアフィルタやエンコーダなどは動作が重くシミュレーションを行うのが容易ではない．ハードウェアコンポーネントをネットワーク上に分散させることによって一つのシミュレータにかかる負荷を減らし，より高速にシミュレーションができるようになる．

第二に，Java の強力な開発環境を利用できることである．Java はマルチプラットフォームで，かなりよく普及したプログラミング言語であるため統合開発環境などの発達した開発環境がたくさんある．それらを大いに利用することによって開発者の負担を軽減することができるし，Java ソフトウェアモデルともシームレスな結合ができる．

第三に，Java がもともとオブジェクト指向言語であり，ハードウェア設計に向いていることである．Java には，C や C++にあるポインタなどフォンノイマン型構成を前提とした機能がなくプログラマは自然にコンポーネント(部品)化が可能なモジュールの独立性の高いプログラムを書くことができる．ハードウェア設計にとってソフトウェアが管理するメモリに直接アクセスするポインタは必要ではない．よって文法が C++に似ていながらポインタをなくしてしまった Java は C++よりもハードウェア設計によりふさわしいということができるだろう．

2.2. Java による記述方法

Java ではクラスが基礎になっている．クラスは，そのインスタンスの状態ならびに振舞いを記述する青写真である．クラスをインスタンス化すると，その同じクラスの他のインスタンス群と同じ状態と振舞いを備えるオブジェクトがひとつ生成される．クラスやオブジェクトの状態はメンバ変数に保持され，振舞いはメソッドで実装される．

回路の入出力信号線に相当するインターフェースの部分は public または private メンバ変数として宣言する．回路の機能や振舞いが記述されるのはシミュレーションメソッドである．ハードウェア Java クラスが持つメンバ変数はコンポーネントの入出力および内部信号線や変数である．出力信号は public 宣言し，入力信号や内部信号は private 宣言する．入力信号にはシミュレーションメソッドのパラメータが渡される．ハードウェア Java クラスにはその仕様をハードウェアで実現するための VHDL コードを生成するメソッドを実装する．

2.2.1. シミュレータの実装

最上位シミュレータはメインメソッドによって駆動される．下位コンポーネントのシミュレータはそれを呼び出す上位シミュレータによって駆動される．テストパターンはファイルから読み込む．入力信号値のシミュレータへの引渡しと出力信号値の照合を行う．シミュレータメソッドは入力信号を受け取り，出力信号をコンポーネントのアーキテクチャに従って生成するだけである．動作の状態を検証するアルゴリズムは main メソッドの中に記述する．アーキテクチャと検証アルゴリズムを分けることで必要に応じてプログラマが簡単にカスタマイズすることを可能にする．

2.2.2. VHDL 生成メソッドの実装

標準ファイル出力に対してすでに確定しているVHDLのコードを投げる．生成されるコードはクラスに対して固定である．クラスのコンパイル後に何らかのパラメータを与えて生成するVHDLコードを最適化するような機能はない．クラスメソッドとして static 修飾子を用いて定義する．単にコードを生成するだけである．

2.3. Java ライブラリによる設計方法

基本となるハードウェアコンポーネントのクラスがある場合は，そのクラスを継承したサブクラスを作り，VHDL 生成メソッドをオーバーライドして詳細化されたハードウェア仕様を実装する．基本となるハードウェアコンポーネントを組み合わせて別の回路を作る場合は，新たにクラスを定義して，シミュレーションメソッドと VHDL 生成メソッドを実装する．さらにテストパターンを生成し，シミュレーションを行って正しく動作するか検証する．

Java で具現化したクラスはシミュレーションメソッドで下位コンポーネントを呼び出したり，そのコンポーネント自身が他のコンポーネントから呼び出されたりということが可能である．

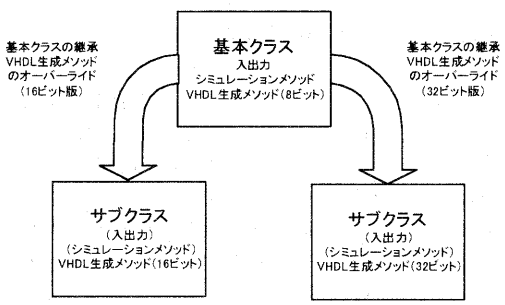


図 1　基本クラスとサブクラス

2.4. ハードウェア設計の簡単な例

ライブラリを使ってフルアダーを設計する場合について説明する．フルアダーは，ハーフアダーと論理和（OR）から作ることができるので，この 2 つを下位コンポーネントとして利用する．さらに，ハーフアダーは論理積（AND）と排他的論理和（XOR）から作ることができるのでこの 2 つを下位コンポーネントとして利用する．呼び出しはシミュレーションメソッドと VHDL 生成メソッドで下位コンポーネントのそれを呼び出すだけである．

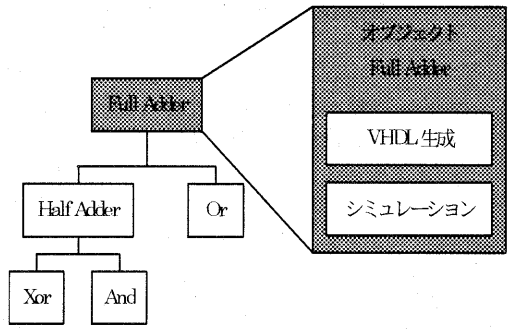


図 2　フルアダーの階層

3. シミュレーションの高速化

この章では，Java を併用することでシミュレーションを高速化することができるということについてさらに詳しく述べる．

Java ベースでシミュレーションを行うことだけでも十分高速化が見込めるが，さらにネットワーク対応の言語であることを生かした分散処理についても述べる．

3.1. RTL と Java シミュレーション

3.1.1. RTL シミュレーション

現在一般的に普及している RTL 用シミュレータはイベントドリブン方式と呼ばれ，伝播遅延を扱えるようなシミュレータである．しかし，RTL の設計段階でこれら遅延を設定してシミュレーションしているケースはどれくらいあるだろうか．シミュレーション速度をかせぐために全ての遅延をゼロとして扱っているのがほとんどである．

3.1.2. Java シミュレーション

Java シミュレーションでは RTL シミュレータを大幅に超えてシミュレーションを高速に行うために大変シンプルな記述になっている．このためタイミングや遅延などの情報は犠牲にされ，信号は 2 つの論理状態（つまり 0 または 1）に限定されている．システムの初期状態に登場するハイインピーダンス（Z）や不定（X）を扱うことはできない．現段階では Java シミュレーションだけでは不十分であり，RTL シミュレーションなどと組み合わせて検証を行う必要がある．しかし論理検証の大部分ではこの高速なシミュレーションを利用することが可能である．

3.2. Java による高速化原理

Java シミュレーションが高速である理由について詳しく説明すると，単にタイミング的な遅延情報をまったくといっていいほど保持する必要がなく，Java で簡単な機能シミュレーションを行うには入力信号に対し

て必要な演算を施し，出力信号に代入するだけでよいということである．

また，別の理由はJavaコンパイラによる最適化である．まず，Javaはバイトコードを中間コードとして実行するために低速になる場合もあるが，現在中間コードの実行はホットスポット技術を使うことによりほぼネイティブコードなみのスピードとなっている．また高速実行が必要な場合はネイティブコード，コンパイラも利用できる．

一方，Javaは最新の言語であり，多くの最適化コンパイル機能を実装することが可能である．今回評価に利用したJ2EEのコンパイラでは必ずしも多くの最適化コンパイル機能が実現されていないが，Java言語そのものは，C++言語同様高度な最適化が可能な言語仕様となっている．

3.3. 分散処理による高速化

分散処理はJavaのRMIによって実現できると考えられる．

RMIは2つの別のマシン上で動作するJavaプログラムの一方のオブジェクトのメソッドを他方のプログラムから呼び出す機能を実現するための仕組みである．通信部分のコードを作成することなく，クライアント・サーバ機能を実現することができ，他のマシン上のオブジェクト（リモートオブジェクト）に対して次のようなことが可能である．

◆メソッドの引数にプリミティブ変数およびオブジェクト変数を渡すことができる．

◆メソッドの戻り値をプリミティブ変数あるいはオブジェクト変数として受け取ることができる．

RMIはJava特有の機能であるためクライアントもサーバもJavaで記述する．

3.3.1. RMIの利用方法

まず，サーバとなるリモートオブジェクトのインターフェースを定義し，リモートオブジェクトクラスを作成する．それをサーバ側ローカルのネームサーバに登録する．次にクライアントのプログラムを作成し，ソースファイルをコンパイルする．さらにリモートオブジェクトをrmicによってコンパイルする．こうすることで，実行に必要なスタブクラスとスケルトンクラスが生成される．

実行にはまずサーバ側でネームサーバを起動し，サーバ側のリモートオブジェクトを起動する．クライアントオブジェクトをサーバ側のホスト名を加えて実行することでリモートメソッドを利用することができる．

3.3.2. RMIの動作原理

クライアントとリモートメソッドとの通信は，rmicにより生成されたスタブおよびスケルトンを通して行われる．RMIの実装とファイルの適切な配置を行うことでクライアントからはメソッドが遠隔にあることをほとんど意識することなく利用できる．

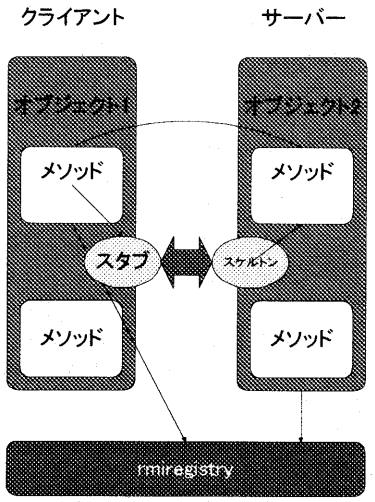


図3　RMIの動作概念

3.4. フィルタ回路によるシミュレーション例

3.4.1. フィルタ回路の概要

シミュレータにわざと負荷をかけるために作ったD-フリップフロップ（DFF）と加算器および乗算器を組み合わせた16ビットの簡単なフィルタについて説明する．このフィルタは，同期式のDFFを直列に並べて左から信号を次々と伝播させていくものである．各DFFの出力は加算器または乗算器につながっていて左から流れてきた値にその値を2倍もしくは4倍した値を足し合わせる．

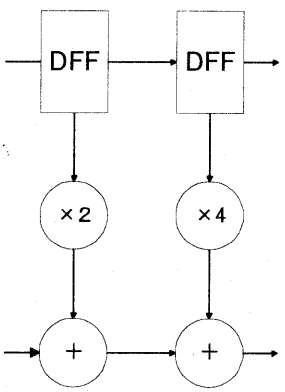
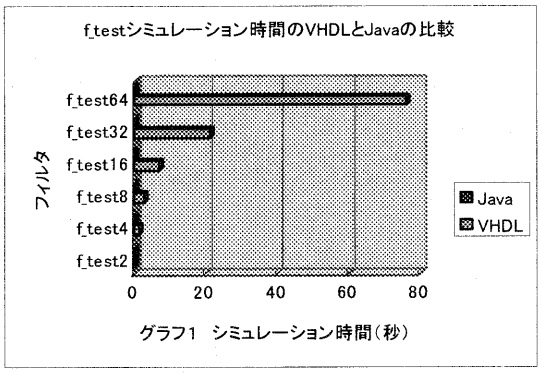


図4　フィルタの基本単位

次に，10μs 分の RTL と Java のシミュレーションにかかった時間の比較を示す．"f_test"の後の数字はフィルタに含まれる DFF の数を表している．RTL シミュレータには Altera の MAX+PLUSII を利用した．



3.4.2. 超並列シミュレーション

大規模なフィルタを分散させてシミュレーションを行う方法を説明する．

分散させるのはコンポーネントが単位となる．コンポーネントは Java のクラスに対応しているので，コンポーネントごとにシミュレーションを分散させる方法をとる．

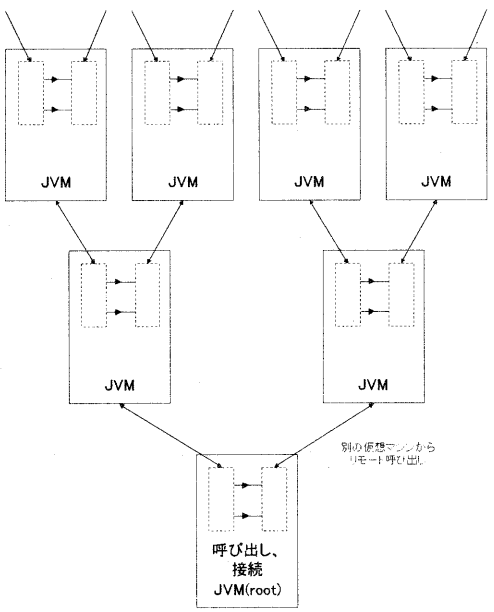


図 5 フィルタコンポーネントの分散

別の PC 上の呼び出される側のコンポーネント（シミュレータ)には RMI を実装しておく必要がある．RMI はただ遠隔にあるメソッドを呼び出すためのものであるので，通常分散させて高速なシミュレーションを行うには並列処理が必要となる．つまり各呼び出しをマルチスレッド化する必要があるわけである．

3.4.3. 分散コンポーネントの利用

RMI を用いた分散シミュレーションの方法をさらに発展させれば，ハードウェア化したデバイスがネットワーク上に分散していても有効に利用することが可能である．例えばあるクラスではその処理の一部を実際のハードウェアにやらせるようにしておく．そのオブジェクトを RMI によって呼び出せば，デジタル TV 用回路など，多数の信号処理ハードウェアが並列的に動作するシステムを容易にシミュレーションすることが可能となる．

4. 今後の課題

今後はより実用的な回路による検証を進める予定である．その第一段階として FFT を多数搭載したプロセッサや行列乗算などの複雑な回路を設計し，よりはっきりした効果を確かめる．開発ツールの利用効果について評価を行い，将来のシステム記述言語になるためには何が必要かを評価する．

文　献

[1] メアリ・カンピオーネ，キャシー・ウォルラス，アリソン・ハムル，“Java チュートリアル第 3 版” pp.181~252，283~327，ピアソン・エデュケーション，東京，2001.

[2] ブルース・エッケル，“Java プログラミングマスターコース（下)”，pp.327~335，ピアソン・エデュケーション，東京，2000.

[3] 中山茂，“Java 分散オブジェクト入門”，pp52~75，技報堂，東京，2000.

[4] 牧野淳一郎，“LSI 設計技術の問題”，http://grape.astron.s.u-tokyo.ac.jp/~makino/papers/gb99-version2/node3.html

[5] 清尾克彦，“システム LSI 設計 I －高位設計編－分冊 1”，pp.22~23，半導体理工学研究センター，横浜，2001.

[6] IDG Japan，“コンパイラとインタプリタの融合”，http://www.computerworld.jp/resource/keyword/back/200104sw.html

[7] Perter Hagger，“Practical Java からの抜粋：コンパイル時のコード最適化に頼るべからず”，http://www-6.ibm.com/jp/developerworks/java/000818/j_pr29.html