システムＬＳＩ設計技術　104－16
(2002. 1. 24)

組込みシステム向け Java 実行環境の構築

木村　基† 三木 裕介† 尾上 孝雄†† 白川　功†

† 大阪大学大学院工学研究科情報システム工学専攻
†† 京都大学大学院情報学研究科通信情報システム工学専攻

あらまし 組込みシステム向けの Java 実行環境の構築において，処理速度，消費電力，あるいはメモリ消費量に関して大きな技術的課題が残っている．本文では，これらの課題を解決するために，ハードウェアエンジンおよびソフトウェアカーネルを含めた組込みシステム向け Java 実行環境の構築を行う．考案するハードウェアエンジンは 6 段のパイプライン構成をとり，ソフトウェアカーネルを構築するため新たに 39 の追加命令を実装している．ハードウェアエンジンを VLSI 化した結果，ゲート数約 3 万，最大動作周波数 96 MHz となり，ホストプロセッサと共に 1 チップ内に集積することが可能である．CaffeineMark によるベンチマーク評価では，J2ME ソフトウェア実装に対して約 6 倍の高速化を実現している．

キーワード 組込みシステム，Java，Java 仮想マシン，命令畳み込み，Java 2 ME

Implementation of Java Execution Environment for Embedded Systems

Motoki KIMURA†, Morgan Hirosuke MIKI†, Takao ONOYE††, and Isao SHIRAKAWA†

† Department of Information Systems Engineering, Osaka University,
2-1 Yamada-Oka, Suita, Osaka, 565-0871 Japan
†† Department of Communications & Computer Engineering, Kyoto University,
Yoshida-Honmachi, Sakyo-ku, Kyoto, 606-8501 Japan

Abstract The implementation of the Java execution environment for embedded systems suffers from a great amount of momory and slow execution speed. In order to solve these technical issues, this paper devises a hardware/software codesign approach for enhancing the performance of an existing embedded system, which consists mainly of a hardware engine and a software kernel. The hardware engine is constructed of a 6-stage pipeline, and 39 additional instructions are provided for the software kernel implementation. The proposed hardware architecture has been synthesized with 30K gates, which can operate at 96MHz of clock rate, enabling a single chip implementation of the whole system together with Host Processor. The proposed codesign approach is 6 times as fast as J2ME software implementation with the use of the CaffeineMark benchmark.

Key words embedded systems, Java, JVM, instruction folding, Java 2 ME

1. はじめに

近年，携帯電話，PDA，ページャなどの組込みシステムの高機能化，多機能化に対する要求が増大している．Java[1] は (1) Java 仮想マシン (JVM)[2] によるマルチプラットフォーム性，(2) 命令レベルのセキュリティの保証，(3) オブジェクト指向言語，という特徴を持ち，組込みシステムに広く用いられている．

一般的に，JVM はインタープリタ方式，JIT(Just-In-Time) コンパイラ方式などのソフトウェアで実装される．ソフトウェアで JVM を実装する場合，Java の命令をネイティブコードに変換する必要があるため，メモリ消費量および消費電力が増大し実行速度が低下する．ハードウェアによる実装 [3-8] ではこれらの問題を解決できるが，既存インターフェース，OS などの再構築が必要であるため，開発コストならびに設計期間の面で課題が残る．

このような背景から，本文では組込みシステム向け Java 実行環境を提案し，そのシステム設計を行なう．本実行環境はホストプロセッサと並列に動作する Java ハードウェアエンジン，ソフトウェアカーネル，およびホストプロセッサインターフェースより構成される．本アクセラレータはホストプロセッサから呼び出され非入出力関連の Java プロセスを実行し，プロセス終了後ホストプロセッサに制御を返す．

提案するハードウェアエンジンは 6 段のパイプライン構成を採る．JVM はスタックベースで動作するため，パイプラインにはスタックステージが含まれる．また Java アプリケーションの実行速度を向上するため，複数の命令を 1 サイクルで実行する命令畳み込みを実装する．さらに，ソフトウェアカーネルを効率的に構築するため，ダイレクトメモリアクセスあるいは命令の書き換えなどを行なう 39 の追加命令を新たに実装する．ハードウェアエンジンを Synopsys Design Compiler を用いて Virtual Silicon Technology 0.18 μm ライブラリで VLSI 化した結果，ゲート数約 3 万，最大動作周波数 96 MHz となった。

本実行環境は J2ME[9] と比較して約 6 倍の高速化を実現している．

2. 提案 Java システム

図 1 に Java 実行環境の全体構成を示す．Java システムは Java ハードウェアエンジン，Java メモリおよびホストプロセッサインターフェースから構成され，既存のシステムはホストプロセッサ，システムメモリおよび入出力インターフェースから構成される．

JVM には 32 bit および 64 bit の命令が含まれる．しかしながら，64 bit の命令は 32 bit の命令と比較し使用頻度が少ないため，本文では小面積化という観点から Java ハードウェアエンジンを 32 bit で構成する．また，提案 Java システムは，ホストプロセッサインターフェースを修正することにより，さまざまなシステムに適用可能である．

Java プロセスの実行手順を図 2 に示す．ホストプロセッサにより Java プロセスが呼び出されると，Java システムはバイトコードを実行する．入出力プロセスを実行する場合は，Java システムはホストプロセッサに制御を移し，実行終了後ホストプロセッサは Java システムに制御を返す．Java システムがバイトコードを実行している間，ホストプロセッサは別のプロセスを並列実行可能である．

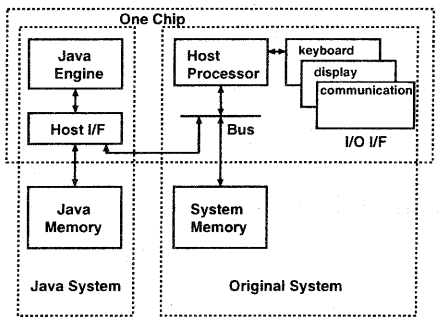

図 1 全体構成

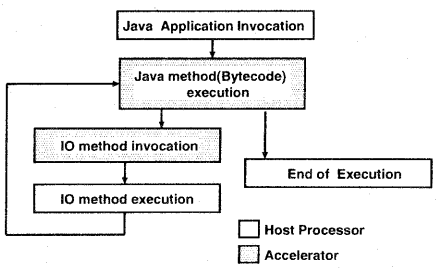

図 2 Java プロセスの実行手順

2.1 設計フロー

図 3 にシステムの設計フローを示す．

まず，SpecJVM98[10] および JavaCaffeineMark[11] により命令出現頻度を調査し，ハードウェアエンジンにおける命令畳み込みの検討を行なう．その

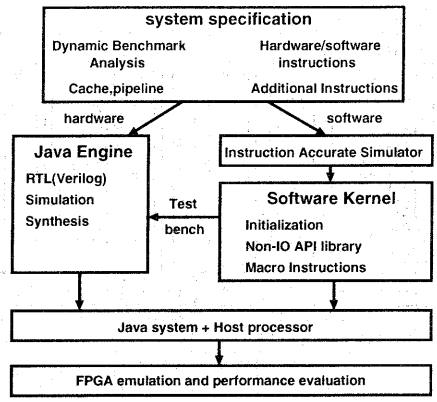

図 3　システム設計フロー

後命令セットをハードウェア実装命令，ソフトウェア実装命令に分類し，追加命令，キャッシュ，パイプラインの仕様を決定する．

仕様決定後，ハードウェア設計およびソフトウェア設計を並行して行う．ソフトウェアカーネルを効率的に設計するため，ハードウェア実装命令およびキャッシュをシミュレート可能な命令セットシミュレータを作成する．また，ハードウェアにおけるシミュレーションはソフトウェア設計で生成したテストパターンを使用する．ハードウェアおよびソフトウェアを設計後，FPGA による動作検証および性能評価を行なう．

2.2　命令セット

JVM には 203 の命令が含まれるが，その実装方法については規定されていない．本システムでは，32 bit データ転送命令，32 bit 演算命令，32 bit 分岐などの単純な命令はハードウェアのみで実装する．一方，メソッドの呼び出しやオブジェクトの確保などの複雑な命令はハードウェアのみでは実行困難となるため，ソフトウェアカーネルを構築し，これらの命令をソフトウェアマクロ命令を用いて実装する．また比較的出現頻度の高い 64 bit の整数演算命令，データ転送命令をマルチサイクル命令としてハードウェアのみで実装する．

浮動小数点演算命令の実装方法は，それぞれのアプリケーションに応じて取捨選択を可能とする．浮動小数点演算命令の出現頻度が高い場合，命令を高速に実行するため浮動小数点演算ユニットを追加する．一方，浮動小数点演算命令の出現頻度がそれほど高くない場合，回路規模を考慮し浮動小数点演算命令をソフトウェアマクロ命令を用いて実装する．

またソフトウェアカーネルを効率的に構築するため 39 の追加命令を実装する．これらは直接メモリアドレスを指定したメモリアクセス，オペランドスタックとレジスタ群とのデータ転送，メソッド呼び出しなどの複雑な命令の書き換えを行う命令である．

表 1 に本システムの命令セットについて示す．

表 1　命令セット

実装方法	命令の種類	
JVM 命令 (203 命令)		
ハードウェア	32 bit データ転送命令	iload, iconst_0, ..
	32 bit 演算命令	iadd, ior, ishl, ..
	分岐命令	ifeq, goto, ..
	64 bit データ転送命令	lload, dload, ..
	64 bit 整数演算命令	ladd, lor
ソフトウェア	メソッド呼び出し	invokevirtual, return, ..
	オブジェクト確保	new, newarray, ..
	スタック操作	swap, dup, ..
	浮動小数点演算命令	fadd, dadd, ..
追加命令 (39 命令)		
ハードウェア	ダイレクトメモリアクセス	dataload, datastore, ..
	レジスタ群およびスタック間のデータ転送	burstload, burststore, ..
ソフトウェア	命令の書き換え	invokevirtual_fast, ..

2.3　命令畳み込み

命令畳み込みとは，特定のパターンで命令が連続している場合，1 サイクルで演算を行なうことによりスタックの操作を省略し，Java バイトコード実行の高速化を計ることである．畳み込みを行なえる命令組み合わせ多数存在するが，全ての考えられる畳み込みパターンを実装するのは効率的でないため，畳み込みパターンの出現頻度の解析結果に基づき，実装する命令畳み込みを選定する．表 2 に出現頻度の高い畳み込みパターンを示す．また表 3 にこれら 6 種類の命令畳み込みを実装した場合の SpecJVM98 および CaffeineMark それぞれのベンチマークにおける性能向上率を示す．

表 3 から，大量のデータを単純な演算によって処理するベンチマークを使用した場合性能向上率が高く，全てのベンチマークにおいて，命令畳み込みにより性能が平均 15 % 向上することが分かる．

2.4　ハードウェアエンジンアーキテクチャ

図 4 に Java ハードウェアエンジンのアーキテクチャを示す．本ハードウェアエンジンは 6 段のパイプライン構成を採る．JVM はスタックベースで動作するため，命令デコードステージ (ID) とメモリリードステージ (MR) の間にスタックステージ (STK) を置く．また命令畳み込みを実行するため，メモリリードステージ (MR) を実行ステージ (EX) の前に置く．さらにデータ依存によるパイプラインハザードを低

表 2　出現頻度の高い畳み込みパターン

pattern	comment
LC LV OP	メモリアドレスを計算し，データをメモリから読みだし，演算を行なう
LV LC OP	メモリアドレスを計算し，データをメモリから読みだし，演算を行なう
LC LC OP	スタックに積むデータを判別し演算を実行する
LV OP	メモリアドレスを計算し，データをメモリ，スタックから読みだし，演算を行なう
LC OP	スタックに積むデータを判別し演算を実行する
LC MEM	スタックに積むデータを判別し，その値をメモリに書き込む

LC: 定数をスタックに積む命令
LV: メモリからデータを読み出す命令
OP: 演算命令
MEM: メモリへデータを書き込む命令

表 3　命令畳み込みによる性能向上率

ベンチマーク	性能向上率
JVM98.check	25.4%
JVM98.compress	20.1%
JVM98.jess	5.8%
JVM98.db	12.0%
JVM98.javac	6.2%
JVM98.mpegaudio	26.2%
JVM98.mtrt	4.4%
CaffeineMark	16.3%
平均	14.6%

減するため，各ステージにデータバイパス機構を備える．メモリアクセスは命令キャッシュおよびデータキャッシュを介して行う．ハードウェアエンジンがバイトコードを実行中でない時のみ，ホストプロセッサはハードウェアエンジンあるいは Java メモリへのアクセスを可能とする．

命令フェッチステージ (IF)

命令フェッチ部では，命令コードを命令メモリから取得しプログラムカウンタを更新する．Java バイトコードは可変命令長であるため 13 バイトのシフトレジスタを用意する．シフトレジスタの先頭 7 バイトを命令デコード部へ出力し，命令デコード部から出力されるシフト数を元にシフトレジスタの内容およびプログラムカウンタの更新を行う．

命令デコードステージ (ID)

命令デコード部は命令畳み込みユニット，マルチサイクル命令用ステートマシンおよびコマンド生成部の 3 つのユニットに分割される．命令畳み込みユニットでは，命令コード 7 バイトが命令フェッチ部から入力されると，単一の命令あるいは命令畳み込み可能な命令パターンを解析し，畳み込みパターン，オペコードおよびオペランドを出力する．

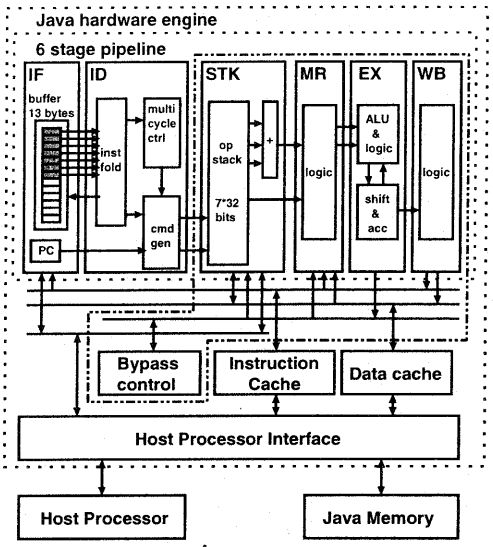

図 4　ハードウェアエンジンアーキテクチャ

命令畳み込みユニットがマルチサイクル命令であることを検出した場合，ステートマシンに制御を移す．ステートマシンはコマンド生成部に対して制御信号を出力し，以降のパイプラインステージに出力する命令，データを制御する．マルチサイクル命令のうち，64 bit の整数演算命令あるいはデータ転送命令を 2 サイクル，imul を 16 サイクル，idiv を 32 サイクルでそれぞれ実行する．

コマンド生成部では，命令畳み込みユニットおよびステートマシンの出力をもとに，以降のパイプラインステージに必要な命令およびデータを生成し出力する．

スタックステージ (STK)

スタック部は，スタックレジスタ，加算器および各種レジスタ群から構成される．

Java で使用するオペランドスタックのうち，先頭の 7 オペランド分はハードウェアエンジン内のスタックレジスタにロードされ，残りの部分はメモリに格納される．スタックレジスタがオーバーフロー，アンダーフローした場合，スタック - メモリ間で優先的にデータを連続転送する．

加算器では命令デコード部から出力されるデータおよびスタックレジスタ内のデータを用い，メモリアクセスに使用するアドレス計算および条件分岐に使用するデータの比較演算を行う．

メモリリードステージ (MR)

メモリリード部では，メモリから 8bit, 16bit, 32bit 単位でのデータを読み出す．

実行ステージ (EX)

実行部では算術演算，論理演算およびシフトを行う．実行部は ALU とシフト部から構成される．ALU では，算術演算および論理演算を行う．シフト部はシフトレジスタであり，シフト演算を行うほかマルチサイクル演算命令の中間結果を保存する．

ライトバックステージ (WB)

ライトバック部では，メモリ，スタックおよびプログラムカウンタに 8bit，16bit，32bit 単位でデータを書き込む．

バイパス制御

本ハードウェアエンジンはスタックベースのアーキテクチャであるため，データハザードが頻繁に発生する．パイプラインにおけるデータハザードを低減するため，ハードウェアエンジンはデータバイパス機構を備える．実行部あるいはメモリリード部からの出力を実行部，メモリリード部，あるいはスタック部に任意の演算結果をバイパスする．

キャッシュ

命令データはアドレス順に読み出されることが多いため，実装の容易さおよび回路規模を考慮し，命令キャッシュは直接マッピング方式を採る．一方，データはランダムなアドレスから読み出されることが多く，ヒット率の低下を防ぐため，データキャッシュは 2-way のセット連想方式を採る．各ブロックのサイズは，命令キャッシュ，データキャッシュ共に 16 バイトである．

また，キャッシュサイズは命令キャッシュ，データキャッシュとも，1K，2K，4K，8K，16K に設定可能である．

3. 実装結果

提案するシステムはハードウェアエンジンおよびソフトウェアカーネルより構成され，JVM の全 203 命令に加え 39 の拡張命令を実装している．表 4 に実装命令数の内訳を示す．

表 4　実装命令数

命令の内訳	実装命令数	詳細
JVM 命令	ハードウェア 140 命令	32 bit データ転送，整数演算，分岐
	ソフトウェア 63 命令	メソッド呼び出し，浮動小数点演算
追加命令	ハードウェア 24 命令	ダイレクトメモリアクセス，レジスタ転送
	ソフトウェア 15 命令	fast 命令

考案したハードウェアエンジンを Verilog-HDL を用いて記述し，FPGA に Altera APEX EP20KE400 を用い，ホストプロセッサに Tensilica Xtensa XT1000 を用いて動作検証を行なった．図 5 に評価用ボードを示す．本ハードウェアエンジンを Synopsys Design Compiler により，Virtual Silicon Technology 0.18 μm ライブラリで VLSI 化を行なったところ，ゲート数が約 3 万，最大動作周波数が 96MHz となった．表 5 にこれらの諸元を示す．

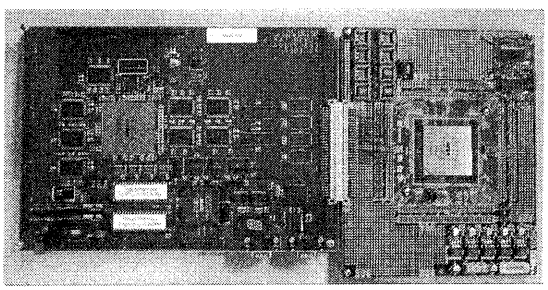

図 5　評価ボード: Xtensa XT 1000 (左)，ハードウェアエンジン (右)

表 5　ハードウェアエンジン諸元

FPGA 実装結果	
FPGA device	Altera EP20KE400
使用セル数	12,000
FPGA 動作周波数	10MHz
Cache	30,000 bit
ASIC 実装結果	
テクノロジ	0.18μm VST Inc.
ゲート数 (FPU 有/無)	52,000/30,000
最大動作周波数	96MHz
Cache	30,000 bit

表 6　CaffeineMark による性能評価

ベンチマーク	J2ME Ultra Sparc (*cm*/MHz)	提案システム (*cm*/MHz) (FPU 有/無)
CaffeineMark.Sieve	0.52	9.65
CaffeineMark.Loop	0.50	14.65
CaffeineMark.Logic	0.49	8.35
CaffeineMark.String	1.45	4.60
CaffeineMark.Float	0.46	7.30/0.85
CaffeineMark.Method	0.53	0.75
全体	0.60	5.60/3.85

表 7　他の Java システムとの比較

	ゲート数	*cm*/MHz
ホストプロセッサをハードウェア拡張		
JStar (Nazomi Corp.)	30,000	2.9
JVX (Insilicon Inc.)	15,000	5.0
Jazelle (ARM)	12,000	5.5
ホストプロセッサと並列動作		
aJ-100 (aJile)	25,000	2.9
提案システム (FPU 有/無)	52,000/30,000	5.6/3.9

4. 性能評価

Java CaffeineMark を用いて，提案するシステムの性能評価を行なった．表 6 に Sun Ultra Sparc (450MHz) 上での J2ME を用いた場合と提案システムのベンチマーク結果を示す．浮動小数点演算器搭載の有無により，J2ME の結果と比較してそれぞれ 9 倍，6 倍高速である．

表 7 に他の Java システムとの性能およびゲート数の比較結果を示す．JStar [13]，JVX [14] および Jazelle [15] はホストプロセッサをハードウェア拡張したものである．aJ-100 [6] と提案するシステムは Java バイトコードを独立に実行でき，ホストプロセッサとの並列動作を実現することが可能となる．並列動作可能なシステムの中では，提案するシステムは aJ-100 よりも高速である．

5. 結　　論

本文では組込みシステム向け Java 実行環境を提案し，そのシステム設計を行なった．本システムは JVM の全 203 命令を実装し，ソフトウェアカーネルを効率よく構築するため，拡張命令として 39 命令を実装した．

考案した Java ハードウェアエンジンを Synopsys Design Compiler を用いて，Virtual Silicon Technology 0.18 μm により論理合成を行なったところ，ゲート数が約 3 万，最大動作周波数が 96 MHz となった．これにより，Java システム，ホストプロセッサ，入出力インターフェースを 1 チップに集積し，高速な Java 実行環境を容易に構築することが可能となる．

今後の課題としては，性能向上のため，メソッド呼び出し命令の最適化が挙げられる．

文　　献

[1] J. Gosling, B. Joy, G. Steele, and G. Bracha, *The Java Language Specification Second Edition*, Addison Wesley, Los Altos, California, April, 2000.

[2] T. Lindholm and F. Yellin, *The Java Virtual Machine Specification Second Edition*, Addison Wesley, Palo Alto, California, April, 1999.

[3] J.M. O ' Connor and M. Tremblay, "picoJava-I: The Vitrual Machine in Hardware", *IEEE MICRO*, Vol. 17, No. 2, Mar./Apr. 1997, pp. 45-53.

[4] Sun Microsystems Inc., *picoJava-II Microarchitecture Guide*, Mar. 1999.

[5] Advancel Logic Corporation Inc., *TinyJ Processor Core Datasheet*, May 1999.

[6] aJile Systems Inc., *Real-time Low-power Java Processor aJ-100 Datasheet*, Sept. 2000.

[7] Patriot Scientific Corporation Inc., *PSC1000 Microprocessor*, July 1997.

[8] S. Kimura, H. Kida, K. Takagi, T. Abematsu, and K. Watanabe, "An application specific Java processor with reconfigurablities", in *Proc. Asia and South Pacific Design Automation Conference 2000 (ASP-DAC 2000)*, Jan. 2000.

[9] Sun Microsystems Inc., *Java 2 Platform, Micro Edition*, June 1999.

[10] The Standard Performance Evaluation Corporation, *SpecJVM98 VERSION 1.03*, 1998.

[11] Pentagon Software Corporation, *Java CaffeineMark 3.0*, 1999.

[12] Motorola Inc., *MC68EZ328 - DragonBall EZ Product Brief*

[13] Nazomi Communications, *JSTAR Product Brief*

[14] Insilicon Inc., *JVX Accelerator*

[15] ARM, *Jazelle - ARM Architecture Extensions for Java Applications*