Reconfigurable Androidにおける JavaRockによるハードウェア・アクセラレーション

榎戸健二† 三好健文†† 小池恵介†
船田悟史†† 藤波香織† 中條拓伯 †

FPGA 搭載の Android 機器の一形態として Reconfigurable Android を開発し，Android における FPGA によるアプリケーションの高速化を試みている．そのハードウェア・アクセラレーションを設計・開発する上で，Java 言語で記述されたソースコードを直接 VHDL コードに変換する JavaRock に着目し，その利用可能性の検討を進めている．本研究では JavaRock により合成された回路をさまざまな視点で分析することによって JavaRock の持つ特徴，性能の評価を行い，今後 Reconfigurable Android 上において開発環境として利用する上での課題について検討を行う．

Towards Hardware Acceleration with JavaRock on Reconfigurable Android

Kenji Enokido,† Takefumi Miyoshi,†† Keisuke Koike,† Satoshi Funada,†† Kaori Fujinami† and Hironori Nakajo †

We have been developing Reconfigurable Android which is Android with an FPGA for the purpose of accelerating Android Application with FPGA acceleration. In the case that we design and develop the hardware acceleration, we focus on JavaRock which is directly converts a Java source code into a VHDL code. With the JavaRock, we have been evaluating its availability from the many point of view. In this research, we have investigated feature and efficiency of several VHDL codes which are generated by JavaRock. From the results we have examined subjects of JavaRock as a development environment for Reconfigurable Android by analyzing a synthesized circuit.

1. はじめに

1.1 Android を取り巻く状況

現在，Google 社によって開発された Android を搭載した端末が普及しつつあり，スマートフォンやタブレットといった携帯機器だけでなく，組み込み機器への搭載が実用的なものとなりつつある．

Android は Dalvik VM と呼ばれる独自の VM を搭載している．Java プログラムを Dalvik バイトコードと呼ばれる中間コードに変換し，Dalvik VM 上において実行する．しかしながら，VM を介して実行するため，そのオーバヘッドから高速実行が阻害されることとなり，その高速化が望まれている．さらに，独自の VM を使用しているため，既存の Java VM の高速化手法を適用することが難しい．この Dalvik VM のオーバヘッドは，画像や動画，音声などといったメディアデータを処理するアプリケーションにおいて，計算量が膨大となる場合があり，計算速度の面で問題となる．計算処理能力を向上させるためにプロセッサの周波数を上げるアプローチでは消費電力の増加が生じ，また，コア数を増やすことによる解決策はハードウェア，ソフトウェア両者の観点から，組込み機器においてはさまざまなハードルがある．

携帯機器においては，急速に進化する情報通信技術への即時対応や様々なネットワーク環境への対応が要求され，今後現れてくる新たなプロトコルへの対応が必要となる．こういった処理をソフトウェアで対応するにはプロセッサパワーが不足し，通信技術の進化や環境の変化に応じて機器を作り変えるにはコストがかかりすぎて現実的ではない．

また，今後の利用形態として，種々のセンサを携帯機器や組み込み機器に接続し，温度，湿度といった環

† 東京農工大学
Tokyo University of Agriculture and Technology
†† 株式会社イーツリーズ・ジャパン
e-trees.Japan,Inc.

ⓒ 2012 Information Processing Society of Japan

境情報とともに，人体における体温，脈拍などの健康面でのセンシング技術が注目を集めている．こういったセンサから得られる情報をリアルタイムに処理したり，機器内に蓄積された膨大なセンサデータを圧縮・展開といった処理を施したりするにはプロセッサパワーの観点から困難である．

以上の問題点を克服するような機構が Android において，今後ますます必要となってくるものと考えられる．

1.2 Reconfigurable Android による問題解決

以上に述べたアプリケーションの高速化，情報通信技術への柔軟な対応，多種多様なセンサから得られるデータの処理といった問題点を解決するための方策として，FPGA により柔軟にハードウェアで対応することを考える．この FPGA を搭載した Android の形態を Reconfigurable Android として提唱する．我々の提唱する Reconfigurable Android とは，OS 自体が Reconfigurable なわけではなく，Reconfigurable 要素を含んだ Android の枠組みで活用可能なプラットフォームである．FPGA により実装するハードウェアによる処理能力の高さとともに，再構成可能な柔軟性を Android に搭載することにより，高機能な Android 機器の開発環境の実現を目指している．

1.3 Reconfigurable Android におけるアクセラレーション機構の開発

これまでの研究において，Android 上の Java アプリケーションは，FPGA によりアクセラレーションを行うことができたが，FPGA 内のアクセラレーション回路はハードウェア記述言語（HDL）により設計・記述しなければならない．すなわち，開発者は本来のソフトウェアとしての Java アプリケーションの開発に加え，ハードウェアとしてのアクセラレーション回路の開発を行うこととなる．しかしながら，アプリケーション開発者にとって HDL の習得が困難であり，アクセラレーション回路を効率よく構築するために，ある程度のハードウェアの知識が必要になることから，開発者にとって大きな負担となる．このことは Reconfigurable Android を利用する上において大きな問題となる．

この問題を解決するために本研究では JavaRock[1] と呼ばれる高位合成器に着目し，Reconfigurable Android における開発環境としての可能性を検証することとした．JavaRock は Java 言語で書かれたコードをハードウェア記述言語である VHDL コードへと変換するものである．これにより，Java アプリケーションとアクセラレーション回路を同一の言語により開発できるようになり，プログラマの負担の軽減および開発効率の向上を促すことができると考えられる．

本研究では JavaRock により合成された回路を評価し，さまざまな視点で分析することによって JavaRock の持つ特徴，性能の評価を行い，今後 Android 上において開発環境として利用する上での課題について検討を行った．

本論文では，まず Reconfigurable Android における現状のアプリケーションの高速化手順について述べ，高位合成器の現状と Java の FPGA 実装について述べる．4 章では，JavaRock の詳細について説明し，現在の開発環境と JavaRock におけるハードウェア・アクセラレーションの評価に用いたアプリケーションについては 5 章で述べる．6 章では評価結果について示し，最後にまとめと今後について述べる．

2. Reconfigurable Android におけるアプリケーションの高速化手順

これまで，我々は Reconfigurable Android の研究・開発環境として，インテル Atom プロセッサと Xilinx Spartan-6 を PCI Express インタフェースにより，高速に通信を行う環境を構築し，そのシステム上において Android を実装した．その上で，Java アプリケーションの計算処理を FPGA によりハードウェア・アクセラレーションを行なった場合の性能評価を行った[2]．

ハードウェア全体の構成を**図 1** に示す．FPGA 内部は PCI Express による DMA 転送を行うモジュール，DDR2 メモリ，メモリコントローラ，アクセラレーション回路，コマンドレジスタから構成される．DMA 転送はメインメモリと DDR2 メモリ間で行われ，アクセラレーション回路にデータが送られる．アクセラレーション回路はコマンドレジスタにより制御され，Java アプリケーション内の特定の処理を CPU の代わりに行う．

本システム上において，Java アプリケーションは以下の手順により処理が行われる（図 2）．

(1) Java アプリケーションから JNI を通じて C 言語の処理をネイティブコードライブラリとして呼び出す．
(2) C ソースから open や write などのシステムコールを行い，デバイスドライバの関数を呼び出す．
(3) DMA 転送により FPGA 側へデータの転送を行った後，コマンドレジスタを介して処理開始命令を出す．
(4) FPGA は処理が完了すると CPU に割り込みをかけ，デバイスドライバに処理の終了を通知

ⓒ 2012 Information Processing Society of Japan

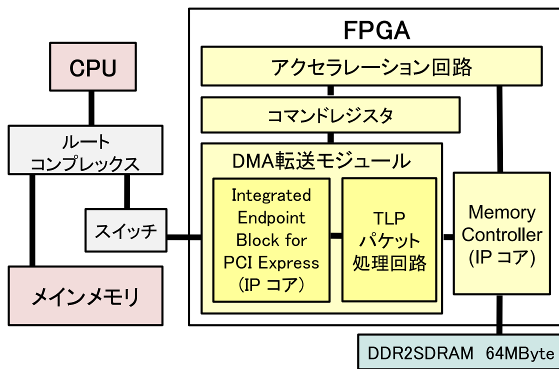

図 1 ハードウェア構成

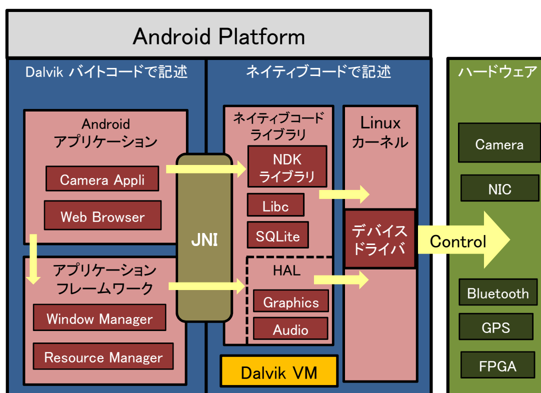

図 2 Reconfigurable Android における Java アプリケーションの高速実行

する．

(5) DMA 転送により CPU 側へデータの転送が行われ，Java アプリケーションが処理の結果を取得する．

CPU と FPGA は PCI Express の 1 レーンにより通信が行われるためデータ転送に時間を要するが，実際の処理時間のみを比較した場合，高速化が期待できる結果を先の研究では得ることができた[2]．

3. 高位合成器の現状と Java の FPGA 実装

アルゴリズムとして複雑な処理を RTL において記述することは煩雑な手順を要し，デバッグが困難となる問題がある．さらに，ハードウェア化を施したい処理が高度化・詳細化するにつれて，ハードウェア設計者がその手順，アルゴリズムを理解した上で，実装するには開発期間を余分に要することとなる．

逆にソフトウェア・アルゴリズムの設計者が，考案したアルゴリズムやプログラムをハードウェア化するのにもハードルが高い．そこで，その解決手段として，高水準言語を用い，回路記述言語よりも高い抽象度によるハードウェア設計を可能にする様々な高位合成器について，これまでに研究・開発が行われてきた．高位合成器が出現し始めた 2000 年代前半は，性能，合成品質については，まだまだ実用的なものではなく，開発フローも確立していなかった．近年，こうしたことが徐々に解決されることとなり，実際の製品開発へ利用される機会も増えてきている．まだ熟練者の経験に頼る必要があるものの，回路記述言語よりも少ない工数で開発できるケースもある．

現在では，既存の言語にハードウェア設計のために特別な型やクラスライブラリ，文法などの拡張を加えることにより，高位合成器として利用可能にしたものがある．SystemC[3]，ImpulseC[4]，Handel-C[5] といった C 言語をベースにしたものをはじめ，JHDL[6]，MaxCompiler[7]，Lime[8] など Java 言語をベースにしたものなどがその例である．いくつかは商用として販売もされている．

しかしながら，ハードウェアとして最適化するように指示した箇所のコードはソフトウェアとして実行できず，専用のシミュレータか RTL で検証する必要があり，デバッグの困難さは解決されていない．また，C や Java をベースにしているが，独自の拡張により，ハードウェア的な記述をする必要が生じ，その言語の経験者であっても簡単にはコードを実装することができない問題点がある．

他のアプローチとして，言語そのものの記述からハードウェア化するものがあり，CyberWorkBench[9] や LegUp[10] といったものが挙げられる．これらは C 言語をベースに開発されたものであり，標準的な C コンパイラによりコンパイルすればソフトウェアとして動作可能である．しかし，C 言語自体に並列計算を行う仕組みがないため，独自のサポートが必要となる．

上記のように普及している高級言語をベースに用いず，新たに設計された高位合成用の言語として，Bluespec System Verilog (BSV)[11] がある．これは，回路記述言語よりも高い抽象度と厳密な型チェックを導入している．しかしながら，新たな言語の習得や開発手法の修練にかかる時間，人的コストは無視できない．

Java の実行を FPGA によりアクセラレーションする研究の 1 つとして，特定のメソッドを Java VM に代わり FPGA で実行する，Dynamic Method Migration on FPGAs[12] がある．処理の一部を FPGA に計算させることで，Java VM を介すオーバーヘッドが減少し，CPU と FPGA の並列処理が可能となる．この手法では，FPGA で実行する処理を増やすために，メソッドの実行率に応じて FPGA を動的にコンフィ

ギュレーションする．FPGA のコンフィギュレーションに必要な配置配線ファイルは，プログラムの実行前に，実行率が高くなると予想されるメソッドを元に作成する．メソッドごとのハードウェア設計が必要であり，FPGA 上で処理するメソッドが頻繁に入れ替わる場合，コンフィギュレーションが多発し，アクセラレーション効果が得られなくなる問題がある．

また，我々はこれまで Jazelle 方式[13] に基づいたハードウェア・アクセラレーションについて研究を進めてきた[14)~16)]．Jazelle DBX は ARM アーキテクチャ向けの Java アクセラレータであり，実際の ARM プロセッサに実装されている．しかしながら，Jazelle DBX はプロセッサ内部に組み込まれるため，既存のプロセッサに追加することは困難である．つまり，新たなプロセッサを設計・実装することとなり，今後もバージョンアップされていく Android の Dalvik コードに対しては，開発コストが問題となる．そこで，既存のプロセッサの修正が不必要な手法に着目し，アクセラレータの実装に FPGA を活用する方式の研究を進めている．

4. JavaRock による Java 言語からのハードウェア開発

JavaRock は，Java プログラムからハードウェアへの合成を実現することを目指している高位合成処理系である．所望の処理をプログラムとして一度記述したら，CPU 上でソフトウェアとして実行するだけではなく，高速化や消費電力削減などの必要に応じて，手間なく専用ハードウェア化したいという欲求が満たされる．ここでは，JavaRock の設計開発の基本方針，Java プログラムをハードウェアに変換する規則，並列処理機構，備えている最適化手法および，ハードウェアとの連携について説明する．

4.1 基本方針

JavaRock は特別なクラスや型，文法を導入せず，ソフトウェアとして実行可能な Java プログラムをそのまま VHDL に変換することを目指している[17]．これにより，Java プログラムをソフトウェアとしてコンピュータ上で実行し，アルゴリズムレベルのデバッグを行い，それをハードウェア化することができる．また，ソフトウェアとしてアルゴリズムレベルのデバッグが可能になることで，開発効率が向上する．

さらに，新たな文法や規則を覚えることが必要なく，開発環境に関しても慣れ親しんだ環境でそのまま開発が可能なため，プログラマの負担軽減につながる．Java 言語を用いることにより，クラスによるオブジェクト指向設計はハードウェアへのモジュール化がしやすく，スレッドの仕組みをハードウェアの並列処理に利用可能という利点がある．

以上の JavaRock の基本方針は，Reconfigurable Android を利用する Java プログラマにとっても，導入しやすいものと考えられる．

4.2 変換規則

JavaRock が Java プログラムをどのような規則で VHDL コードに変換するかを述べる．通常 Java プログラムは逐次的に実行されるので，JavaRock ではこの逐次処理をステートマシンに変換することにより実現する．Java 言語の 1 行が 1 つのステートに変換されることになる．例えば，**図 3** に示す Java プログラムは**図 4** のように変換される．

```
A = 0;
A = A + 1;
```

図 3 代入文

```
case conv_integer(F_method_state) is
  when 0 =>
    A <= conv_std_logic_vector(0, A'length);
    F_method_state <= F_method_state + 1;
  when 1 =>
    A <= conv_std_logic_vector(conv_integer(A) + 1,
                               A'length);
    F_method_state <= F_method_state + 1;
```

図 4 代入文の VHDL 変換結果

JavaRock は Java の if 文，while 文，for 文の制御文をサポートしており，これらはメソッド全体のステートマシンに内包されるサブステートマシンに変換される．

Java プログラム中に記述された boolean，int，char などの変数は VHDL のシグナル変数に変換されるが，配列に関しては BlockRAM に変換される．配列 W を宣言すると JavaRock は**図 5** のようにその配列に対応する BlockRAM のインスタンスを自動的に生成する．また，配列へのアクセスはアドレス信号，データ信号および制御信号の操作に対応づけられる．

Java のクラスは 1 つのモジュールに，メソッドは VHDL の process 文に変換される．つまり，**図 6** に示す簡単な Java のクラスとメソッドは，**図 7** のような回路に変換される．各メソッドにはリクエストとビジー信号があり，引数はワイヤ線により接続される．

具体的にメソッドの呼び出しは，以下の手順により実現している（**図 9** 左）．

```
U_W : simpledualportram
generic map( DEPTH => W_DEPTH, WIDTH => W_WIDTH )
port map(
    clk => clk,
    we => W_we,
    raddr => W_raddr,
    rdata => W_rdata,
    waddr => W_waddr,
    wdata => W_wdata
);
```

図 5　配列の VHDL 変換結果

```
public class Test{
  public int sum(int a, int b){
    return a + b;
  }
}
```

図 6　Java によるクラスとメソッドの記述

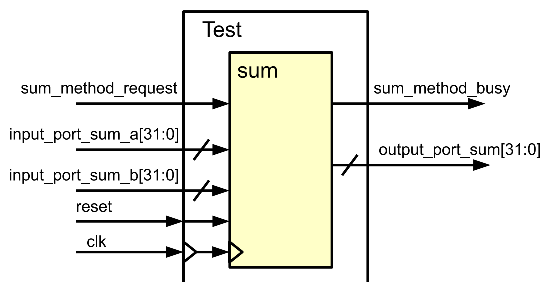

図 7　JavaRock により生成される回路

(1)　呼び出し側は呼び出すメソッドの request 信号をアサート
(2)　呼び出されたメソッドは request 信号のアサートにより，busy 信号をアサート
(3)　呼び出し側は引数用の信号に引数をセットし，request 信号をディアサート
(4)　呼び出された側は request 信号のディアサートにより実行開始し，終了時にビジー信号をディアサート
(5)　呼び出し側は busy 信号のディアサートにより，呼び出し結果を取得

以上述べたように JavaRock では Java 言語の基本的な構文をサポートをしている．

4.3　並 列 処 理

FPGA の処理能力を生かすには並列に処理を行えることが必要不可欠である．JavaRock では並列処理を表現するために Java の Thread を利用する．Java では，Thread クラスを継承したクラスの start メソッドを呼び出すことによりスレッド処理が行われる（**図 8**）．

```
//スレッド呼び出し元クラス
public class Test{
  ThreadTest threadTest = new ThreadTest();
  public void Main(){
    threadTest.start(); //スレッド処理開始
  }

}

//Threadクラスを継承したスレッド処理を行うクラス
public class ThreadTest extends Thread{
  //呼び出し元からstart()メソッド呼び出し時に実行される
  public void run(){
    //スレッド処理
  }
}
```

図 8　Java によるスレッドの書き方

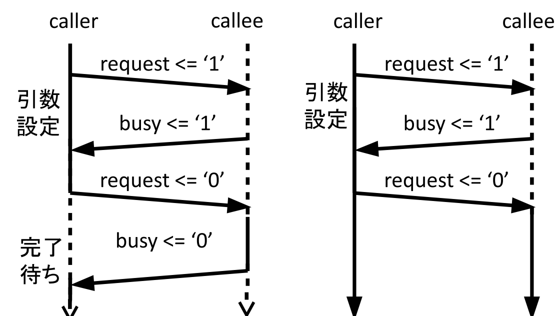

図 9　JavaRock におけるメソッド呼び出し（左）と Thread による並列処理（右）．破線は処理停止状態を示す．

4.2 で述べた通り，JavaRock ではメソッド呼び出しの際には呼び出し側は呼び出したメソッドが実行完了するまで待つことになる．一方，Thread を継承したクラスの start メソッドを呼び出した場合は，実行完了を待たずに処理を継続する（図 9 右）．これにより，Java のプログラミング手法同様に並列処理を行うことが可能である．

4.4　最　適　化

4.2 で述べたように JavaRock では 1 行を 1 ステートメントへ変換する．しかし，全てをこの方法で変換していては並列性が上がらず，性能向上を見込むことができない．そこで，JavaRock では演算の依存関係を調べることにより，並列に演算可能なものを 1 つのステートにまとめる最適化を行っている．

図 10 に示すコードは JavaRock で VHDL に変換すると**図 11** に変換される．変数 a，b，c の演算に関しては 1 クロックで演算可能であるが，d，e に関しては前の演算に依存関係があるため，同一クロックで演算できない．JavaRock ではこういった依存関係を解決し，最適化を行っている．

```
a = 10;
b = 20;
c = 30;
d = a + b;
e = c + d;
```

図 10　依存関係のある演算処理

```
case conv_integer(F_method_state) is
  when 0 =>
    a <= conv_std_logic_vector(10, a'length);
    b <= conv_std_logic_vector(20, b'length);
    c <= conv_std_logic_vector(30, c'length);
    F_method_state <= F_method_state + 1;
  when 1 =>
    d <= conv_std_logic_vector(conv_integer(a) +
              conv_integer(b), d'length);
    F_method_state <= F_method_state + 1;
  when 2 =>
    e <= conv_std_logic_vector(conv_integer(c) +
              conv_integer(d), e'length);
    F_method_state <= F_method_state + 1;
```

図 11　最適化結果

4.5　ハードウェアとの連携

Java 言語の仕様の範囲内ではハードウェア設計に必要なクロックや信号のビット幅といった部分の記述をすることができない．そういった処理は無理に Java 言語で設計するより，RTL で設計した方が簡単でわかりやすい．

また，IP コアや既存の HDL 資源を有効に活用したいケースも考えられる．JavaRock では JavaRock Hardware Interface（JRHI）が用意されており，VHDL で書かれたモジュールの信号線を Java プログラムの変数に置き換えることにより，Java プログラムにハードウェアに対するインタフェースを提供している[18]．これにより，PCI Express や DDR メモリなどのメモリコントローラを信頼のある IP コアを用い，それを利用したアプリケーションを Java 言語で記述することができる．

5.　JavaRock による Java アプリケーション開発における評価環境

5.1　評価実験環境

評価に使用したボードは東京エレクトロンデバイス社の Application Reference Platform for Image Processing（ARPIP）（**図 12**）である．ARPIP には，インテル Atom プロセッサ Z530 1.6GHz と Xilinx Spartan-6（XC6SLX45T）（**表 1**）が搭載されている．

論理合成には，XilinxISE14.1 を使用し，JavaRock のバージョンは r193 を使用した．

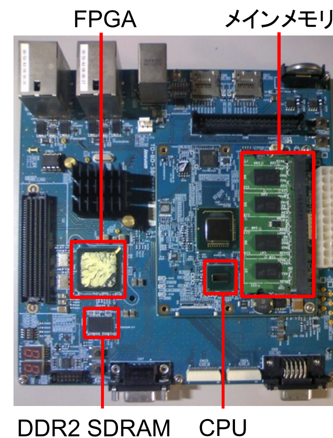

図 12　Application Reference Platform for Image Processing（ARPIP）

表 1　XC6SLX45T のリソース量

	個数
レジスタ数	54576
LUT 数	27288
スライス数	6822
Block RAM 数（18KB）	116

実験に使用する主な回路構成は図 1 に示した通りである．

5.2　評価アプリケーション

JavaRock によるアクセラレーション性能を評価するために，実アプリケーションとして SHA-1 と画像のエッジ検出回路の 2 つのアプリケーションを用い，JavaRock により変換した結果と，アルゴリズムから直接 HDL により記述したものについて，回路規模と性能について比較を行った．この 2 つのアプリケーションの詳細について以下に示す．

5.2.1　SHA-1

SHA-1 はハッシュ関数の 1 つであり，2^{64} 長以下のメッセージから 160 ビットのハッシュ値を計算するものである．SHA-1 の計算の特徴としては，メッセージを 512bit のブロックごとに区切り，そのブロックを順番に計算していく．メッセージの末尾にはメッセージ長を付与し，最終ブロックが 512bit に満たない場合はパディングを行う．各ブロックは順番に計算しなければならず，ブロック内の計算についても逐次的な計算を必要とするため，並列処理には適さないアルゴリズムとなっている．

JavaRock において SHA-1 を実装する場合，入出力に関しては，ハードウェアをある程度意識する必要がある．ソフトウェアとしての Java プログラムであれば，byte 配列にメッセージを格納し，メソッドに一

© 2012 Information Processing Society of Japan

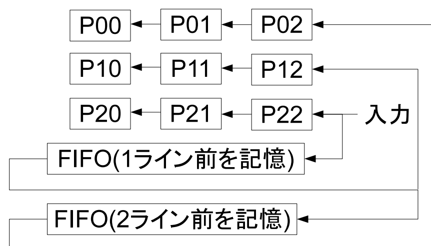

図 13　エッジ検出回路の概要

表 2　SHA-1 の回路規模

	レジスタ数	LUT 数	Block RAM
JavaRock	3276	10634	1
手書き	996	4458	1

度に渡すことができるが，ハードウェアではそれが困難であるため，数バイトずつ転送することとなる．しかしながら，SHA-1 の主要なブロックごとの計算については，逐次的な計算が多数占めるため，Java プログラミングによるアルゴリズム実装の感覚でハードウェア実装を行うことができた．

5.2.2　画像のエッジ検出

画像のエッジ検出は画像処理の中でも極めて重要な処理の 1 つである．ハードウェアによる高い演算能力と並列処理が，画像データという大量のデータに対し高速に処理することができる．今回はモノクロ画像に対して 3 × 3 のラプラシアンフィルタを施す処理を実装した．簡単のため端の画素については考慮しないこととした．ラスタスキャンに 1 画素ずつ入力するような仕様としているため，3 × 3 のフィルタ処理を行うには，上部 2 列分の画素を保存しておく必要がある．これには FIFO を 2 つ用いて実装を行った（**図 13**）．

RTL で記述する場合，画素の入力から各 FIFO への書き込み，読み込み，フィルタ処理の演算を並列に行える．Java プログラムで逐次的にこの処理を記述していては，FPGA の持つ並列処理を生かすことができない．そこで，FIFO をスレッドとして記述した．4.3 で述べたとおり，JavaRock において，スレッドによる記述はハードウェアの並列処理に変換される．エッジ検出ではこの機構を使用し，FIFO への書き込みとフィルタ処理の演算に並列性を与えることとした（**図 14**）．

6.　評価結果および考察

論理合成後の回路規模について，**表 2** に SHA-1 の回路規模を，**表 3** にエッジ検出の回路規模を示す．

動作性能の比較には，動作周波数と演算に必要なクロック数について比較実験を行った．**表 4** に SHA-1 の結果を，**表 5** にエッジ検出の結果を示す．SHA-1 はメッセージを 512bit のブロックに分けてハッシュ値の計算を行うため，この 1 つのブロックの計算にかかるクロック数をカウントした．エッジ検出に関しては 64 × 64 の画像の各画素の入力から出力までにかかるクロック数となっている．

さらに，実機上で CPU と FPGA で通信を行い，実際の処理にかかった時間を測定した．SHA-1 は 1Mbyte のデータを，エッジ検出には 512 × 512 のモノクロ画像を使用した．比較として，Android 実機上での Java による記述のみの実行時間と，NDK による C 言語による記述を利用した場合の実行時間を測定した．SHA-1 の各実行時間を**表 6** に，エッジ検出の各実行時間を**表 7** に示す．このとき，計算以外の CPU と FPGA 間のデータの転送に生じた時間は SHA-1 のときは約 8ms であり，エッジ検出のときは約 4ms であった．

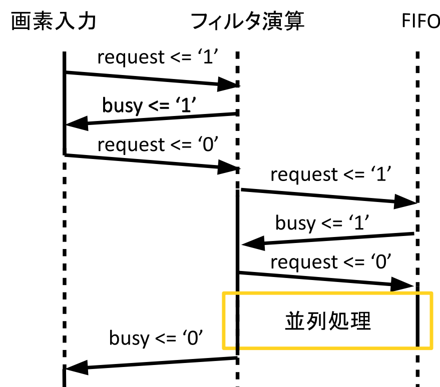

図 14　エッジ検出回路の並列処理

表 3　エッジ検出の回路規模

	レジスタ数	LUT 数	Block RAM
JavaRock	331	387	2
手書き	172	205	2

表 4　SHA-1 の性能比較

	動作周波数 [MHz]	演算に必要なクロック数
JavaRock	82.5	616
手書き	115	118

表 5　エッジ検出の性能比較

	動作周波数 [MHz]	演算に必要なクロック数
JavaRock	178	61439
手書き	153	4096

ⓒ 2012 Information Processing Society of Japan

表 6　SHA-1 の実行時間

	実行時間 [ms]
Java による記述のみ	3360
Andoird NDK を使用	51.7
FPGA 使用（手書き）	41.3
FPGA 使用（JavaRock）	209

表 7　エッジ検出の実行時間

	実行時間 [ms]
Java による記述のみ	392
Andoird NDK を使用	8.85
FPGA 使用（手書き）	8.83
FPGA 使用（JavaRock）	105

以上の開発，実験を通して Reconfigurable Android に JavaRock をアクセラレータとして利用することについて，以下のように考察した．

6.1　導入コスト

JavaRock は現在フリーで公開されており[1)]，Java の実行環境と JavaRock の jar ファイルを用意しさえすれば開発が開始できるという点で導入しやすいといえる．様々なライブラリなどの煩雑なインストール手順を踏むことなく，開発を始められる．さらに，元のソースのままの Java 言語で開発できるため，既存の開発環境をそのまま使えるのも利点である．Java プログラムを書く上で普及している Eclipse などの統合開発環境で記述できるため，プログラマが慣れ親しんだ環境を利用できる．これにより，新たなツールの習得にかかるコストが不要となる．以上のことから，Java アプリケーション開発者にとって導入コストを低く抑えられ，利用しやすい高位合成器といえる．

6.2　開 発 効 率

次に開発効率について評価を行う．Java 言語により，RTL よりも高い抽象度で開発できることから，開発効率が上がらなければならない．それには Java プログラマが Java プログラムを書くのと同じようにハードウェアを設計できるかどうかを評価する必要がある．

JavaRock は Java 言語を用いて回路作成を可能にするが，RTL の持つ特質からは完全に解放されない点は否めない．例えば，JavaRock ではメソッドを VHDL の process 文へと変換するが，VHDL では複数のプロセスから同一の信号へ書き込みはできない．そのため，図 15 のような記述は論理合成時にエラーとなる．同様のことから，1 つのメソッドを複数のメソッドから呼ぶことができない．こうしたことはアービタを導入することにより解決することが可能であるが，現在の JavaRock（r193）では導入されていない．このことから，Java プログラマが通常通りにプログラムを

```
boolean  A;
private void func0(){
    A = true;
}
private void func1(){
    A = false;
}
```

図 15　1 つの変数に対する複数メソッドからの書き込み

完全には書くことはできず，ある程度の修練と独自のテクニックが多少必要となる．そして，高速化を目指す場合は Java プログラミングのテクニックとは別のハードウェアを意識したテクニックが必要になる．

今回 SHA-1 の実装を行ったが，SHA-1 のメインの処理となる各ブロックの計算に関しては，Java プログラムで記述する感覚で開発を行うことができた．JavaRock は独自の拡張を行っていないため，記述した Java プログラムは Java プログラマであれば，そのソースコードから十分に処理過程を追うことができる．このことはコードの保守性の観点から望ましいものであり，仕様変更，機能の追加やプロジェクトメンバの変更などに柔軟に対応できるものと考える．

また，JavaRock にはアルゴリズム部をソフトウェアとしてデバッグを行えることのメリットがある．統合開発環境のデバッガを利用することにより，さらにデバッグ効率を向上できる．これは RTL とテストベンチを用いたシミュレーションによるデバッグと比べ，手軽でバグを発見しやすいといえる．

当然のことながら，ハードウェアに実装を行い，入出力を含めた全体的なテストとデバッグはハードウェア上で行わなければならない．しかしながら，アルゴリズムレベルでのテストを事前に行うことにより，バグを早期に発見できることは有用であると考えられる．

6.3　回路規模，実行性能

回路規模と実行性能評価において JavaRock を用いた場合では，プログラマが直接 HDL により記述した場合と比べ，現状においては数倍劣っているものとなっている．特にエッジ検出の場合の性能において大きな差が出る結果となった．エッジ検出はプログラマがクロックと並列処理を意識して記述した場合には，１クロック当たり１画素に対する計算が行え，高速な処理が可能である．一方，JavaRock で性能が伸びない原因として Java コードの１行をすべてステートマシンに置き換えていることが挙げられる．そのためクロックを意識した記述ができず，プログラマが HDL により記述するような綿密な並列処理を行うことはできない．

ⓒ 2012 Information Processing Society of Japan

4.4 で述べたように JavaRock は最適化を行っているが，まだ研究段階であり，最適化できていないケースが見受けられる．現状では，メソッド呼び出しの前後や if 文などの制御文の前後など，並列化可能な部分はまだ未対応となっている．そのため，最適化可能な範囲と精度が向上することにより，性能が向上する可能性は十分にある．

もう 1 つの要因としてメソッド実行時のリクエストとビジーのやり取りがある．4.2 で述べたようにメソッド呼び出し時にはリクエストとビジー信号を介して処理の発行を行う．エッジ検出では 1 画素ずつの入力をメソッド呼び出しとして実装しているため，これが性能低下の要因となっている．

現在，JavaRock で Java コードをコンパイルする際には指定できるオプションはない．回路規模を抑えたい場合や，ループ展開や BlockRAM のレジスタ化をすることにより回路規模を大きくして速度を上げたい場合など，ユーザの用途に応じたコンパイル方法も考えられる．

多くの高位合成器は特殊なクラスや文法を導入したり，アノテーションやコンパイル指示子などを用い，独自の拡張を行うことにより，回路構成を変え，性能向上を行える柔軟性を与えている．しかし，それらを用いることにより可読性が薄れ，言語本来の特徴が損なわれるため，JavaRock ではそういった解決策を取らない方針であり，どのように性能を向上を行うかは今後の課題である．

6.4 実用的な問題に対する処理時間の見積もり

膨大な計算が必要な動画像の処理やビッグデータの処理をすることを想定する．本来ならば，ボード上で実装し，評価を行うところだが，使用した評価ボードの CPU と FPGA 間の通信速度やリソースが十分ではないため実際のアプリケーションで扱うようなデータサイズの実験は困難であった．そこで，本実験のデータを基に，実用的な問題に対する処理時間を考察した．評価ボードでは PCI-Express 1 レーンで接続されていたが，本想定では PCI-Express 4 レーンで接続されているものとし，通信速度が十分ある環境を想定した．通信速度は PCI-Express 4 レーンであるため，4 倍になる．

スマートフォンの解像度の 1 つである 480 × 800 の大きさの動画処理を考える．1 秒当たり 60 フレームの動画が 60 秒あるとすると，処理する必要がある画像の枚数は 3600 枚となる．エッジ検出の処理時間は画素数に比例するため，今回測定した 512 × 512 の画素に対する実行速度から 1 ピクセル辺りの処理にかかる時間を算出し，計算を行った．その際，FPGA を用いる場合は CPU と並列に計算できるため，ヘテロジニアスな並列処理をする．さらに，FPGA 内にはエッジ検出回路を複数用意し，並列に画像を計算できる構造とした (**図 16**)．このとき FPGA 内にエッジ検出回路を 10 個配置したと仮定した場合の計算結果は**表 8** の通りである．

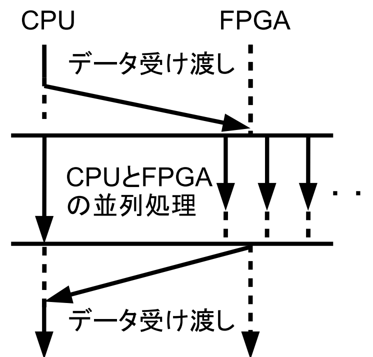

図 16 CPU と FPGA の並列処理

表 8 動画処理に対するエッジ処理時間の見積もり

	実行時間 [s]
Java による記述のみ	2060
Andoird NDK を使用	46.6
FPGA 使用（手書き）	26.5
FPGA 使用（JavaRock）	31.6

単体で実行した場合，現状の JavaRock では NDK を使用した場合の実行速度に及ばない．しかし，CPU と並列に処理したり，FPGA 内に同じ回路を複製し，並列度を上げることにより，アクセラレーション効果を得ることができる可能性がある．SHA-1 に関しても同様にサーバで複数のファイルのハッシュ値計算を継続的に行うような場合，同様な手法でアクセラレーション効果を期待できる．

また，JavaRock と Java のみによる記述と比べた場合においては JavaRock を利用することにより，数倍の速度向上がある．NDK を利用するためには C 言語の記述による実装が必要であるが，JavaRock を利用した場合は FPGA を利用するものの，Java 言語で記述することができる．このことは Android の Java アプリケーションをアクセラレーションする際に NDK を利用する選択肢の他に，JavaRock を利用してアクセラレーションを行う選択肢があるといえる．

7. まとめ

本研究では Reconfigurable Android における JavaRock によるアクセラレータについて，その利用可

ⓒ 2012 Information Processing Society of Japan

能性の評価を行い，今後の課題について検討を行った．Java プログラミングをハードウェア記述言語 VHDL へと変換する JavaRock を用いることにより，より簡単に Android アプリケーションが FPGA によるアクセラレーションを受けることができる可能性を拓いた．

JavaRock はまだ研究段階であり，最適化能力の向上やユーザへの機能の追加により回路規模や性能は大きく改善する可能性がある．さらに，論理合成時に発生するエラーを JavaRock で検出することにより，開発の出戻りを極力抑え，開発効率の向上が期待できる．Android アプリケーションの作成と同じ Java 言語でハードウェアが開発できるということはプログラマの負担の軽減につながり，ソフトウェア上でアルゴリズムレベルのデバッグができるということはデバッグの難しいハードウェアの設計において大きな強みである．導入コストの低さ，開発効率の向上という点において，JavaRock の可能性には期待を寄せられるものと考えられる．

今後の Reconfigurable Android の発展には，Reconfigurable Android を汎用的に利用可能にする必要がある．現状では 1 つのアプリケーションに対してのみのコンフィグレーションを行っているため，汎用的な利用を目指すためには，様々なアプリケーションに対応するように動的にコンフィグレーションする必要がある．そのためには Reconfigurable Android を構成する，Java プログラム，デバイスドライバを呼ぶ C プログラム，デバイスドライバ，FPGA の PCI Express インタフェース，DMA コントローラ，DDR2 メモリ，JavaRock で記述されたアクセラレーション回路といった全てのモジュールに対して柔軟なインタフェースを用意しなければならず，その開発について今後進めていく．

さらに，VHDL への変換を行う現状の JavaRock に加え，Verilog-HDL に変換する JavaRock の別のバージョンの開発も行なっていく予定である．

謝辞　Android のポーティングには，株式会社ビート・クラフトにご協力をいただいた．

本研究の一部は，独立行政法人日本学術振興会とフィンランドアカデミーとの二国間交流事業（共同研究）による支援を得た．

参　考　文　献

1) JavaRock. http://javarock.sourceforge.net/
2) 小池 恵介, 太田 淳, 大島 浩太, 藤波 香織, 郡 信幸, 竹本 正志, 中條 拓伯 : FPGA アクセラレータによる Android アプリケーションの高速化手法 , 組込みシステムシンポジウム (ESS2011),pp.10-1 - 10-8 (2011)
3) SystemC. http://www.systemcjapan.com/
4) Impulse C. http://www.impulseaccelerated.com/
5) Handel-C. http://www.mentor.com/products/fpga/handel-c/
6) JHDL. http://www.jhdl.org/
7) MaxCompiler. http://www.maxeler.com/products/software/maxcompiler/
8) J. Auerbach, D. Bacon, P. Cheng and R. Rabbah : Lime: a Java-compatible and synthesizable language for heterogeneous architectures, OOPSLA '10 Proceedings of the ACM international conference on Object oriented programming systems languages and applications, pp.89-108 (2010)
9) CyberWorkBench. http://www.nec.co.jp/soft/cwb/
10) A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. Anderson, S. Brown, and T. Czajkowski : legup: high-level synthesis for fpga-based processor/accelerator systems , ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA), pp. 33-36 (2011)
11) Bluespec. http://www.bluespec.com/
12) E. Lattanzi, A. Gayasen, M. Kandemir, N. Vijaykrishnan, L. Benini, A. Bogliolo: Improving Java performance using dynamic method migration on FPGAs, Int. Journal of Embedded Systems Vol.1, No.3-4, pp.228–236 (2005)
13) ARM Jazelle. http://www.arm.com/ja/products/processors/technologies/jazelle.php
14) 太田 淳, 茂手木 貴彦, 三輪 忍, 中條 拓伯: Dalvik アクセラレータのための MIPS シミュレータを用いた評価環境, 先進的計算基盤システムシンポジウム (SACSIS2010) ポスター・セッション, Vol.2010, No.5, pp.113-114 (2010)
15) 太田 淳, 三輪 忍, 中條 拓伯: Dalvik アクセラレータ : Android 端末における Java アプリケーションの高速実行機構, 組込みシステムシンポジウム (ESS2010), pp.13-22 (2010)
16) 太田淳，三輪 忍，中條 拓伯 : Android 端末におけるハードウェアによる Java の高速化手法の提案, 情報処理学会論文誌 コンピューティングシステム, Vol.4, No.3, pp.115-132 (2011)
17) 三好 健文, 船田 悟史 : FPGA 向け高位合成言語としての Java の活用手法の検討 , 情報処理学会第 53 回プログラミングシンポジウム,pp. 59-68 (2012)
18) 三好　健文, 船田　悟史　: JavaRock を用いた HW/SW 協調設計の検討，信学技報,vol. 112,no. 70, CPSY2012-21,pp. 119-124 (2012)