情報処理学会論文誌 Vol.53 No.12 2740–2751 (Dec. 2012)

Androidにおける Javaアプリケーションの FPGAアクセラレーション

小池 恵介[1,a)] 太田 淳[1] 大島 浩太[1] 藤波 香織[1] 郡 信幸[2] 竹本 正志[2] 中條 拓伯[1]

受付日 2012年3月5日, 採録日 2012年9月10日

概要：Android 端末における Java 実行の高速化とともに，さまざまなネットワークプロトコルへの対応や，種々のセンサへの柔軟な接続が求められている．そのために，FPGA を活用した Reconfigurable Android を提案し，その有効性，可能性を検証することを目指す．本稿では，Reconfigurable Android の概念について述べ，FPGA ボードにプロセッサカードを搭載したシステムに Android を移植し，FPGA によるアクセラレータを実装し，ハードウェア実行可能な Java のソース部分を FPGA 上で実行することにより全体の性能向上を実現する．Dalvik VM が稼働する Intel Atom プロセッサと FPGA との間において，PCI Express インタフェースの実装を完了し，DMA 転送により 150 MByte/sec の通信性能を確認した．さらに，FPGA アクセラレータによる Android の高速化手法について述べ，実験環境および，プロセッサと FPGA 間の通信性能について報告し，アクセラレーションによる性能評価を示す．

キーワード：Android，FPGA，リコンフィギュラブルシステム，ハードウェアアクセラレーション

FPGA Acceleration of Java Applications in an Android

Keisuke Koike[1,a)] Atsushi Ohta[1] Kohta Ohshima[1] Kaori Fujinami[1]
Nobuyuki Kohri[2] Masashi Takemoto[2] Hironori Nakajo[1]

Received: March 5, 2012, Accepted: September 10, 2012

Abstract: In an Android terminal, as well as fast execution of Java programs, coping with many kinds of network protocol and flexible connection of various sensors are needed. For that purpose, Reconfigurable Android with an FPGA is proposed to explore its availability and possibility. In this paper, a concept of Reconfigurable Android is introduced. We have implemented an FPGA accelerator, which realizes higher performance by executing a part of a Java source code executable in hardware, to accelerate Java execution in an Android mobile terminal. Between an Intel Atom processor which executes a Dalvik virtual machine and an FPGA accelerator, we have implemented PCI Express interface which performs high speed communication of 150 MByte/sec with DMA transfer in our experimental environment. Moreover, Acceleration of Android with an FPGA accelerator is described. Communication performance between a processor and an FPGA has been measured and the performance with the acceleration is evaluated.

Keywords: Android, FPGA, Reconfigurable System, Hardware Acceleration

1. はじめに

1.1 背景：携帯機器と FPGA

近年，携帯機器は高機能化が進み，通話や電子メールに限定されずさまざまな情報処理を行うようになった．なかでも，スマートフォンは，これまでの携帯電話と比べ処理能力が大きく向上している．また，各 OS ベンダからスマートフォン向けの SDK が公開されており，手軽にスマート

[1] 東京農工大学
Tokyo University of Agriculture and Technology, Koganei, Tokyo 184–8588, Japan
[2] 株式会社ビート・クラフト
BeatCraft, Inc., Sumida, Tokyo 130–0013, Japan
[a)] koike@nj.cs.tuat.ac.jp

フォン向けアプリケーションを開発することが可能となった．特に，Google 社による Android は，Linux カーネルをベースとしたプラットフォームであり，ソースコードも一般公開されているため，アプリケーションの開発だけでなく，各携帯キャリアや国内メーカが独自のスマートフォンを開発し販売することが可能となっている．さらに，最近では組み込み Android として，組み込み機器への利用が広まりつつある．

Android アプリケーションは Java 言語で記述されるが，その実行方法は通常の Java と異なる．一般に Java 言語は，Java バイトコードと呼ばれる中間言語にコンパイルされた後，Java VM によって実行される．しかし，Android 環境では，Dalvik バイトコードと呼ばれる中間言語を独自の仮想マシン（Dalvik VM）によって実行する．この 2 つの仮想マシンは，基本となるアーキテクチャが異なっており，Java VM がスタックマシンであるのに対し，Dalvik VM はレジスタベースのマシンである．一般に VM を介しての実行は，機械語への逐次翻訳をともなうため，機械語の直接実行に比べオーバヘッドが大きい．したがって，Android アプリケーションの実行速度改善は重要な課題である．現在 JIT コンパイルや Android NDK（NativeDevelopment Kit）[1] の利用など，ソフトウェアによる高速化が主に行われているが，これらの手法はプロセッサの処理能力に依存し，CPU の利用率が実行時間に大きく影響する．

従来から，書き換え可能 LSI である FPGA（Field Programmable Gate Array）を活用して，ターゲットとなる LSI をエミュレーションにより動作確認する方策がとられることが多い．さらに，その FPGA も大容量化し，高速化が進められ，従来の LSI エミュレーションだけでなく，ネットワーク機器やメディア処理機器など，さまざまなプロトコルに対応したり，内部仕様が頻繁に更新されるような機器などにおいて利用されたりするようになってきた．

今後の携帯機器においては，さまざまなネットワーク環境への対応が要求され，それぞれに応じたプロトコルに対応する必要がある．現在，携帯機器の通信処理は，主に SoC（System-on-Chip）内部のハードウェア IP（Intellectual Property core）によって行われているため，複数のプロトコルへの対応は，搭載するハードウェアの増加を招く．しかしながら，対応する通信プロトコルごとに携帯機器シリーズを多種準備することは，メーカ側にとってコストに見合わない．そういった中で，組み込み機能をハードウェア的に再構成可能となるような携帯機器が望まれている．

2010 年 11 月に，Intel 社は，モバイル系のプロセッサと FPGA を同一のパッケージに搭載した新たなプロセッサを発表した [2]．現状このプロセッサと FPGA は PCI Express x2 で接続されており，アプリケーションによっては通信遅延が問題になると考えられる．しかし，今後高速化が求められ，また LSI 実装技術が進めば，プロセッサと FPGA はより密に接続されるようになり，この 2 つの間の通信遅延は短縮されていくものと期待される．

1.2 今後の携帯/組み込み機器の方向性

以上から，今後の Android 携帯機器および Android 組み込み機器への FPGA 搭載という動きは 1 つの方向性をなすものと予測される．現状で問題となっているのは，FPGA の消費電力であり，電池駆動で利用するには，長時間の利用に耐えられないということがあげられる．しかしながら，ACTEL 社などの FPGA ベンダにより FPGA の低消費電力化は進められており，今後の携帯機器への応用も進められている [3]．

前述のように，Android 機器では，Java をもとにした Dalvik バイトコードを実行することで，さまざまな携帯アプリケーションに対応している．しかしながら，Dalvik VM のオーバヘッドから，一部のアプリケーションについては，実行速度が問題となっている．単純な高速化の方向性として，プロセッサの周波数やコア数を増加させることが考えられるが，消費電力や回路面積の問題など，多くのハードルが存在する．

以上の背景から，今後さらに増加していくさまざまな Android アプリケーションの実行速度改善は必要不可欠であり，ネットワークの発展に柔軟に対応することも要求され，さらにさまざまなセンサが携帯機器に搭載されるようになり，そのデータ処理にも高速性が要求される．そのために FPGA アクセラレータを 1 つの解決策であると判断し，ここではその高速化手法を提案する．そこで，今後のスマートフォンや Android 組み込み機器の 1 つの形態として，Reconfigurable Android を提唱し，FPGA 搭載の Android 機器を開発することで，画像，動画，音声などのマルチメディア処理の高速化とともに，Android 機器の高機能化を目指す．さらに，種々のセンサを搭載し，さまざまなネットワーク環境に対応できるように，再構成可能な Android プラットフォームの一形態を作りあげていく．

本稿では，現在推進している Reconfigurable Android の概念について述べ，その性能見積りを行うためのプラットフォームとして，FPGA を搭載して再構成可能とする機構を組み込んだ市販のボードに Android を移植し，その上での高速化手法について説明する．現在の実験環境および，プロセッサと FPGA 間の通信性能について報告し，動画処理とハッシュ関数の SHA-1 を例として，アクセラレーションによる高速化の性能評価結果を示し，今後の発展について述べる．

2. Reconfigurable Android

Reconfigurable Android は，FPGA を搭載した Android 機器である．FPGA は従来 LSI 開発用テストベッドとして使用されていたが，大容量化や高速化により SoC として

の利用が進んでいる．また，SoC 構築のための IP コアや ARM などの汎用 CPU および FPGA ベンダ独自の CPU コアが各種 FPGA ベンダから豊富に提供されている．したがって，Android 機器に FPGA を搭載することでさまざまな IP コアを利用したアプリケーションが開発，提供できるようになり，活用の幅が広がると考えられる．

以上の状況をふまえ，次世代の Android 機器の可能性を探るべく，Android のオープンソース・ガジェット [4] を開発している株式会社ビート・クラフトとともに，Reconfigurable Android のハードウェア構成を中心に設計を進めている．本研究で目指す全体構造イメージを**図 1** に示す．CPU と FPGA を備えた携帯端末において，利用者のニーズに応じて，CPU だけでは実現が難しい高性能なサービスを FPGA でオンデマンドに提供する．

2.1 Reconfigurable Android が提供する機能

Reconfigurable Android が提供する主な機能は，以下の 3 つである．

(1) アプリケーションの高速処理
(2) さまざまなプロトコルへの柔軟な対応
(3) 多種多様なセンサに対するインタフェースの提供および，センサデータの高速処理

本稿では，上記 3 項目のうち，アプリケーションの高速処理について実験および評価を行った．以下で，これらの機能の詳細について述べる．

2.1.1 アプリケーションの高速処理

1.1 節で述べたように，Android アプリケーションは Dalvik VM 上で実行されるため，実行時のオーバヘッドが大きい．Android NDK によって高速化は可能であるが，その効果はプロセッサの利用率に大きく依存する．

そこで，アプリケーションの処理の一部を FPGA にオフロードすることで，高速化におけるプロセッサの利用率の影響を減らすことができる．

一般に，プロセッサの動作周波数が数 GHz であるのに対し，FPGA の動作周波数は数十～数百 MHz である．しかし，FPGA は CPU による逐次実行とは異なり，処理の大規模な並列化や，余計なメモリ入出力の削減などにより，特定の処理に対し CPU の処理速度を上回ることが可能である．また，2 章で述べたさまざまな IP コアはアプリケーションの高速処理においても有効利用可能であると考えられる．

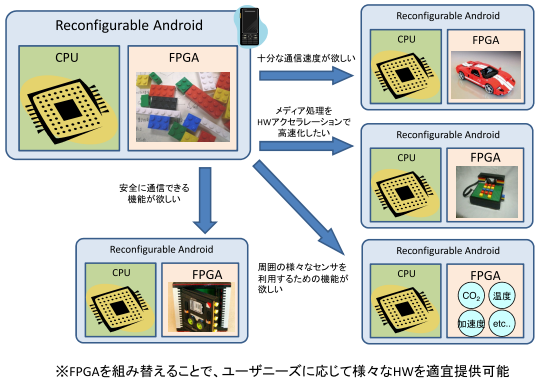


図 1 リコンフィギャラブル Android 携帯機器プロジェクトの全体イメージ
Fig. 1 Overall project of Reconfigurable Android.

2.1.2 さまざまなプロトコルへの柔軟な対応

ここ数年，スマートフォンによるインターネット利用は増加しており，それにともない従来よりも高速な通信方式が検討されている．さらに，今後スマートフォンはインターネット利用だけにとどまらず，ホームネットワークの一部となり，家電や車両などと通信を行うことが予想される．したがって，今後の携帯機器には，多種多様な通信プロトコルに柔軟に対応していくことが求められる．

そこで，使用時に必要な通信回路のみを FPGA 上に実装することにより，搭載するハードウェアの削減が可能となる．また，従来ソフトウェアで行っていたプロトコルの処理を FPGA 上で実行することによって，通信の高速化とともにプロセッサの負担軽減が可能であると考えられる．

2.1.3 多種多様なセンサに対するインタフェースの提供および，センサデータの高速処理

携帯機器は通常，ポケットやかばんなどに携行する場合が多く，またネットワークに接続されているため，環境に関するさまざまな情報の収集や伝達が可能となる [5]．また，周囲の状況だけでなく，ホームエレクトロニクスなどから得られる家庭内の状況に対するきめ細かい環境分析といったことも可能となる [6]．

こうした機能を実現するためには，携帯機器にさまざまなセンサを取り付けるためのインタフェースが必要になる．そこで，そのインタフェースにも FPGA が活用でき，さらに FPGA 上のセンサインタフェースからセンサデータを直接取り込むことで，FPGA による高速処理も可能となる．

3. 現状の Reconfigurable Android 評価環境と高速化方式

3.1 評価環境

Reconfigurable Android の評価環境を構築するにあたって，**図 2** に示す東京エレクトロンデバイス社製の市販の Application Reference Platform for Image Processing（ARPIP）[7] を使用した．ARPIP は FPGA に Xilinx Spartan-6 LXT（6822 Slices）と汎用の Intel Atom プロセッサを ComExpress インタフェースにより接続したカードを搭載しており，PCI Express バスによる CPU-FPGA 間通信が可能である．この Atom プロセッサ搭載カード上に Android を移植し，各種ドライバを開発し，FPGA 上にアクセラレーション回路を実装して，Reconfigurable Android

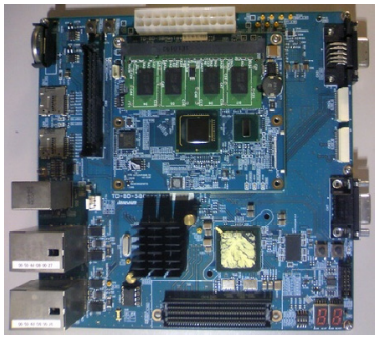


図 2 Application Reference Platform for Image Processing
Fig. 2 Application Reference Platform for Image Processing.

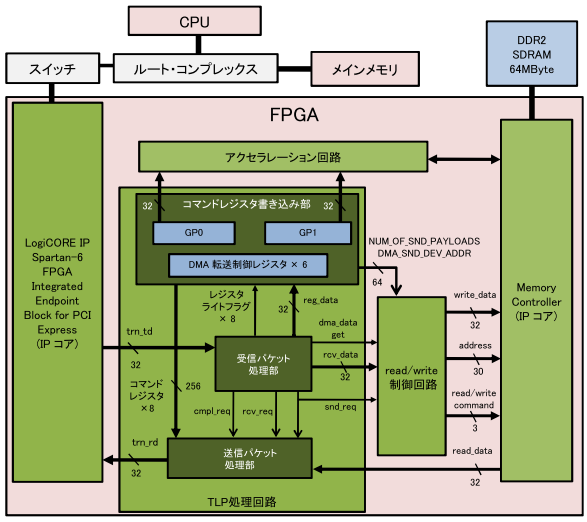


図 3 Reconfigurable Android 評価環境
Fig. 3 Evaluation environment of Reconfigurable Android.

の性能を見積もるためのデータを取得することとする．しかし，ARPIP は本来画像処理用の評価ボードであるため，ARPIP 付属の画像処理向け通信回路では Reconfigurable Android のための柔軟な設計が困難である．また，付属のデバイスドライバは Windows 用のものしか提供されていないため，Linux カーネル用のものを作成する必要があった．そこで，多様なアプリケーションの高速化を目的とした双方向の DMA 通信をサポートする PCI Express 通信機構を FPGA 上に実装し，そのデバイスドライバを Linux 上において作成した．

3.2 ARPIP モジュール構成

Reconfigurable Android の評価環境における，通信回路のブロック図を**図 3** に示す．各モジュールの機能であるが，まず，CPU は次節で述べる Android アプリケーション高速化において Dalvik VM を実行する．メインメモリは，Android アプリケーションの実行に必要なデータを格納するほか，DMA 転送に必要な DMA バッファが確保される．

FPGA の各回路について説明する．LogiCORE IP Spartan-6 FPGA Integrated Endpoint Block for PCI Express は TLP（Transaction Layer Packet）の入出力ポートを持っており，PCI Express のデータリンク層および，物理層の処理を行う．

TLP 処理回路は，コマンドレジスタへのアクセスと DMA 転送を行う．受信パケット処理部は，受信した TLP のヘッダを基にパケットの種類を判別し，各リクエストに応じた処理を行う．TLP がコマンドレジスタへの書き込みを要求する場合，対応するレジスタライトフラグを立て，コマンドレジスタ書き込み部に reg_data の書き込みを命令する．一方，コマンドレジスタの読み出しを要求する場合は，cmpl_req をアサートし，送信パケット処理部にコンプリーション送信を命令する．また，DMA 受信データの場合は，dma_data_get をアサートし read/write 制御回路に受信データ（rcv_data）の書き込みを命令する．送信パケット処理部は，snd_req，rcv_req，cmpl_req がアサートされると，それぞれの制御信号に応じた TLP を送信する．snd_req がアサートされた場合，FPGA→CPU 方向の DMA 転送を行う．一方，rcv_req がアサートされた場合，CPU→FPGA 方向の DMA 転送を行う．また，cmpl_req がアサートされるとコマンドレジスタのデータをコンプリーションとして送信する．

DDR2 SDRAM は DMA 転送で使用されるデバイスメモリで，アクセラレーション回路の処理に必要なデータが格納される．コマンドレジスタは，メモリマップドレジスタとなっており，TLP 処理回路やアクセラレーション回路などの制御に使用される．コマンドレジスタの詳細を**表 1** に示す．アクセラレーション回路は，GP0，GP1 レジスタによって制御され，Android アプリケーションの処理の一部を CPU に代わり高速に実行する．

3.3 デバイスドライバが提供する機能

本システムのデバイスドライバは，***init_module***，***delete_module***，***open***，***close***，***read***，***write***，***ioctl***の 7 種類のシステムコール，および割込みハンドラによって構成されており，PCI Express バスを通じて FPGA を制御し，DMA 転送をサポートする．それぞれのシステムコールとその主な機能を**表 2** に記す．

3.4 Android アプリケーションによる FPGA の制御

Android アプリケーションは，**図 4** に示すプラットフォーム内部のさまざまなコンポーネントを呼び出しながら実行される．図 4 の矢印はコンポーネント間の呼び出し関係を表している．一般に，Android アプリケーションは，**アプリケーションフレームワーク → ネイティブコードライブラリ → Linux カーネル**という順番でコードを呼び出し，最終的にハードウェアを制御する．さらに，Android NDK を用いると，C/C++で記述したファイルをもとに

表 1 コマンドレジスタ一覧（下り：CPU→FPGA 上り：FPGA→CPU）
Table 1 List of command registers.

レジスタ名	アドレス	説明
DMA_RCV_CPU_ADDR	0x00	DMA 転送（下り）のメインメモリアドレス 書き込まれると DMA 転送（下り）を開始する
DMA_SND_CPU_ADDR	0x04	DMA 転送（上り）のメインメモリアドレス 書き込まれると DMA 転送（上り）を開始する
NUM_OF_RCV_PAYLOADS	0x08	DMA 転送（下り）の受信データ数
NUM_OF_SND_PAYLOADS	0x0C	DMA 転送（上り）の送信データ数
DMA_RCV_DEV_ADDR	0x10	DMA 転送（下り）のデバイスメモリアドレス
DMA_SND_DEV_ADDR	0x14	DMA 転送（上り）のデバイスメモリアドレス
GP0	0x18	汎用レジスタ 1 アクセラレーション回路の制御などに使用
GP1	0x1C	汎用レジスタ 2 アクセラレーション回路の制御などに使用

表 2 システムコールと機能
Table 2 System calls and the functions.

システムコール	機能
init_module	・キャラクタデバイスの登録 ・PCI デバイスの有効化 ・割込みハンドラの登録
delete_module	・キャラクタデバイスの削除 ・割込みハンドラ登録解除
open	・DMA バッファの確保
close	・DMA バッファの解放
read	・DMA バッファからユーザ空間へコピー
write	・ユーザ空間から DMA バッファへコピー
ioctl	・DMA バッファからデバイスメモリへ転送するデータ数をデバイスに登録 ・送信元 DMA バッファアドレスをデバイスに登録 ・送信先デバイスメモリアドレスをデバイスに登録 ・DMA バッファからデバイスメモリへの転送を開始 ・デバイスメモリから DMA バッファへ転送するデータ数をデバイスに登録 ・受信元デバイスメモリアドレスをデバイスに登録 ・受信先 DMA バッファアドレスをデバイスに登録 ・デバイスメモリから DMA バッファへの転送を開始 ・任意のコマンドレジスタに対する読み書き

ネイティブコードライブラリを作成することができ，アプリケーションフレームワークを介さずに Android アプリケーションから直接呼び出すことが可能である．本稿では Android NDK を用いて作成したネイティブコードライブラリを NDK ライブラリと呼ぶ．

本システムにおける Android アプリケーションの FPGA

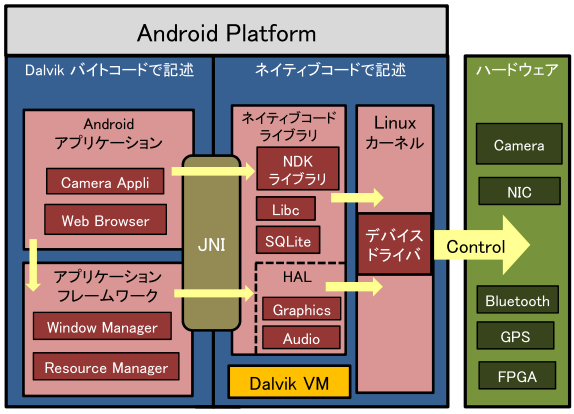


図 4 Android プラットフォームとアプリケーション，ハードウェアの関係
Fig. 4 Relationship among an Android platform, an application and hardware.

アクセラレーションは，アプリケーションの処理の一部を FPGA 上で実行し，高速化を行うものである．FPGA の制御は Java ソースコード中に記述する．したがって，open や read などのファイル操作システムコールを Java ソース内に記述する必要があるが，Java には ioctl など一部のファイル操作システムコールが存在しない．そこで，C 言語ソースファイルにファイル操作システムコールを記述し，Android NDK を用いて NDK ライブラリ化したものを Java から呼び出す．したがって，Android アプリケーションは **NDK ライブラリ → デバイスドライバ**という順番でコードを呼び出し，FPGA を制御する．そのため，本システムでは FPGA の制御を行うために，

(1) アプリケーションとなる Java ソースファイル
(2) NDK ライブラリとなる C ソースファイル
(3) デバイスドライバとなる C ソースファイル

の 3 つのファイルを用意する．

3.5 FPGA アクセラレーションを用いた実行方式

図 5 に Android アプリケーションの処理の流れについて記す．本手法では，あらかじめ Java ソースコード中で高速化を行う部分を Verilog-HDL で記述し，論理合成・配置配線して Configuration ファイルを作成する．Configuration ファイルとは，FPGA の回路書き換えに必要なファイルであり，このファイルを用いて FPGA をコンフィギュレートした後，アプリケーションを実行する．Java アプリケーションは FPGA の処理に必要なデータをすべて FPGA へと転送した後，FPGA のアクセラレーション回路に処理開始命令を出し，FPGA と並列に処理を進める．FPGA は処理が完了すると CPU に割込みをかけ，デバイスドライバに処理の終了を通知する．デバイスドライバには，FPGA の状態を管理する変数が用意されており，Java アプリケーションは適当なタイミングでこの変数を読み取り，処理が完了していれば結果を受け取る．

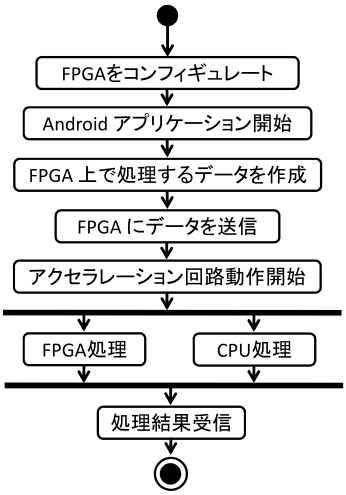


図 5 Android アプリケーションの処理の流れ
Fig. 5 Processing flow of an Android application.

3.6 アプリケーションの記述例

図 6 にアプリケーションの記述例を示す．Fpga クラスの関数はすべて，JNI を通じて NDK ライブラリの関数を呼び出している．NDK ライブラリ関数は，関数名に対応

```
/*データ送信*/
Fpga.devOpen(); //デバイスオープン
Fpga.devWrite(data_array, snd_data_num); //DMA バッファにデータコピー
//FPGA に送信するデータ数を指定
Fpga.devIOcntl(Fpga.SET_SND_NUM, snd_data_num);
//デバイスメモリの送信先アドレスを指定
Fpga.devIOcntl(Fpga.SET_SND_DEV_ADDR, snd_dev_mem_addr);
//DMA バッファの送信元アドレスを指定 → 送信開始
Fpga.devIOcntl(Fpga.SET_SND_CPU_ADDR, buf_addr_offset);
//送信完了待ち
do condition = Fpga.devIOcntl(Fpga.CONDITION_GET , 0);
while(condition!=CONDITION_FREE);

/*アクセラレーション制御*/
// デバイスドライバの condition 変数を書き換え
Fpga.devIOcntl(Fpga.SET_CONDITION, Fpga.CONDITION_BUSY);
Fpga.devIOcntl(Fpga.MMAP_REG_SET, processing_data_num);
//処理するデータ数をコマンドレジスタに書き込む → アクセラレーション開始
Fpga.devIOcntl(Fpga.MMAP_REG_WRITE, Fpga.GP0_ADDR);
//アクセラレーション終了待ち
do condition = Fpga.devIOcntl(Fpga.CONDITION_GET , 0);
while(condition!=CONDITION_FREE);

/*データ受信*/
//FPGA から受信するデータ数を指定
Fpga.devIOcntl(Fpga.SET_RCV_NUM, rcv_data_num);
//デバイスメモリの受信元アドレスを指定
Fpga.devIOcntl(Fpga.SET_RCV_DEV_ADDR, rcv_dev_mem_addr);
//DMA バッファの受信先アドレスを指定 → 受信開始
Fpga.devIOcntl(Fpga.SET_RCV_CPU_ADDR, buf_addr_offset);
//受信完了待ち
do condition = Fpga.devIOcntl(Fpga.CONDITION_GET , 0);
while(condition!=CONDITION_FREE);
Fpga.devRead(result_data_array , rcv_data_num); //ユーザ空間にコピー
Fpga.devClose(); //デバイスクローズ
```

図 6 アプリケーション記述例
Fig. 6 Description example of an application.

するファイルシステムコールを行っており，これによりデバイスドライバの関数を呼び出し，表 2 に示した機能を実現している．

アプリケーションの記述例について説明する．まず，devOpen 関数を用いて FPGA に対応するデバイスファイルを開いた後，devWrite 関数を用いて data_array 配列から DMA バッファへと snd_data_num 分データをコピーする．次に，devIOcntl 関数で，FPGA に送信するデータ数と送信先である FPGA 上のメモリアドレスを指定する．このとき，表 1 の NUM_OF_RCV_PAYLOADS レジスタと DMA_RCV_DEV_ADDR レジスタに，送信するデータ数と送信先のアドレスが書き込まれる．そして，SET_SND_CPU_ADDR コマンドで送信元である DMA バッファアドレスを指定する．このとき表 1 の DMA_RCV_CPU_ADDR レジスタに送信元アドレスが書き込まれ，DMA バッファから FPGA 上のメモリへと転送が開始される．送信の完了は，引数に CONDITION_GET を指定した devIOcntl の戻り値によって確認可能である．

アクセラレーションの制御について説明する．まず，SET_CONDITION コマンドでデバイスドライバ内部の condition 変数を書き換える．condition 変数は，FPGA の状態を表しており FPGA が DMA 転送などの処理を行っているとき，CONDITION_BUSY となる．また，SET_CONDITION コマンド以外にも SET_SND_CPU_ADDR，SET_RCV_CPU_ADDR コマンドによっても自動的に CONDITION_BUSY へと書き換わり，DMA 転送終了時やアクセラレーション終了時に呼び出される割込みハンドラによって，CONDITION_FREE へと書き換えられる．したがって，CONDITION_GET コマンドによって condition 変数を読み取ることで処理の完了が確認できる．次に，処理するデータ数など，アクセラレーション回路が必要とするデータを汎用レジスタに書き込む．左記の例では，MMAP_REG_SET コマンドでレジスタに書き込む値（処理するデータ数）をセットしたあと，MMAP_REG_WRITE コマンドによって GP0 レジスタへと書き込んでいる．また，汎用レジスタをどのように使うかは，アクセラレーション回路作成者が自由に決めることができる．

データの受信について説明する．まず，devIOcntl 関数で，FPGA から受信するデータ数と受信元である FPGA 上のメモリアドレスを指定する．このとき，表 1 の NUM_OF_SND_PAYLOADS レジスタと DMA_SND_DEV_ADDR レジスタに，受信するデータ数と受信元アドレスが書き込まれる．そして，SET_RCV_CPU_ADDR コマンドで受信先である DMA バッファのアドレスを指定する．このとき，表 1 の DMA_SND_DEV_ADDR レジスタに受信先アドレスが書き込まれ，FPGA 上のメモリから DMA バッファへと転送が開始される．最後に，受信完了を確認した後，devRead 関数で result_data_array 配列へと受信データがコピーされる．

4. Reconfigurable Android 評価環境の性能評価

4.1 プロセッサ–FPGA 間の通信性能

メインメモリ–FPGA 間の DMA 転送のバンド幅を計測し，**図 7**，**図 8** のグラフを作成した．バンド幅は Android アプリケーション上で計測した．図 7，図 8 は横軸が転送バイト数を表し，縦軸がそれに対するバンド幅を示している．転送バイト数が多いほどバンド幅が増加している理由は，転送にともなうオーバヘッドの割合が，転送時間に対して小さくなるからである．

ここで，PCI Express 1.1 の 1 レーンあたりのバンド幅は，送信受信それぞれ 250 MByte/sec である．しかし，ARPIP では 1 回の転送で送信可能なデータが 64 Byte に制限されていることや，スイッチによるデータの蓄積交換，ルート・コンプレックスの処理能力などの影響により，実際のバンド幅は 150 MByte/sec 程度となった．

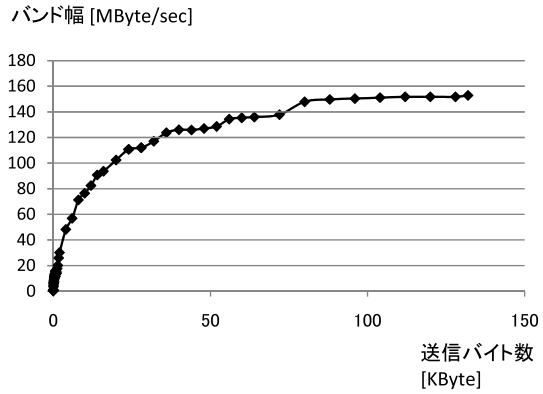


図 7 転送バイト数とバンド幅の関係（メインメモリ → FPGA）
Fig. 7 Bandwidth vs. the number of transferred bytes (Main memory → FPGA).

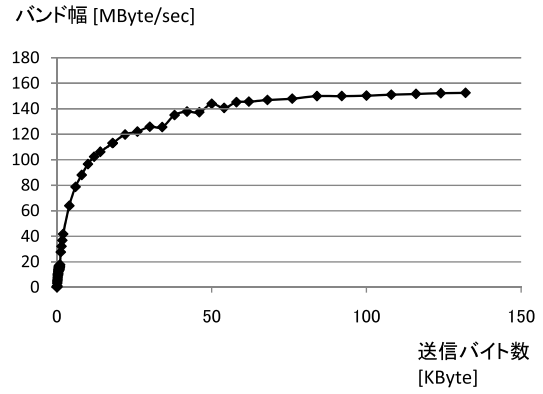


図 8 転送バイト数とバンド幅の関係（FPGA → メインメモリ）
Fig. 8 Bandwidth vs. the number of transferred bytes (FPGA → Main memory).

4.2 PCI Express 通信機構の回路規模

PCI Express による DMA 転送に必要な部分の回路規模を**表 3** に示す．これは，図 3 のブロック図からアクセラレーション回路を除いた部分であり，メインメモリ-FPGA 間のデータ転送機能のみを有する．

表 3 より，PCI Express 通信回路規模は FPGA 全体に対して 1 割以下と十分小さく，アクセラレーション回路は FPGA 上のリソースの大部分を使用可能であることが分かる．

4.3 Android アプリケーション高速化結果

Android アプリケーションの高速化において，(i) Java による記述のみの実行時間，(ii) Android NDK による高速化を行った実行時間，(iii) FPGA による高速化を行った実行時間の 3 種類を測定した．さらに，CPU の利用率が異なる場合の実行時間も計測した．なお，Hyper-Threading Technology により 2 つの論理コアが存在するため，1 プロセスあたりの CPU 利用率は最大 50%である．

使用した Android のバージョンは，Android 2.2 を x86 向けにポーティングしたものであり，JIT コンパイルは実装されていない．そこで，JIT に代わる比較対象として，(ii) を計測する．

4.3.1 画像処理による性能評価

1 画素あたり 3 Byte の 256 × 256 の画像に対し，ラプラシアンフィルタを適用したエッジ検出の実行時間とその内訳を以下に，回路規模を**表 4** に示す．なお，エッジ検出回路の動作周波数は，62.5 MHz である．

表 3 PCI Express 通信回路規模
Table 3 Size of the PCI Express communication circuit.

Logic Resources	Used	Utilization Rate
Slice	475	6%
FF	748	1%
6 入力 LUT	1,331	4%

表 4 エッジ検出回路規模
Table 4 Size of the edge detection circuit.

リソース	使用量	使用率	アクセラレーション回路の使用率
Slice	3,417	50%	44%
FF	13,334	24%	23%
6 入力 LUT	8,466	31%	27%

FPGA による実行時間の内訳

CPU の利用率が 48.9%のとき，FPGA による高速化を行った処理は Java による記述のみの場合に対し，約 2.45 倍高速である．しかし，FPGA による処理時間の内訳を見ると，画像データの送受信が 2/3 以上を占め，送信に 1,250 μs，受信に 1,364 μs 要している．画像のデータサイズは，192 KByte なので送信のバンド幅が 153 MByte/sec，受信のバンド幅が 141 MByte/sec となり，図 7，図 8 に示したバンド幅の最大値とほぼ一致する．一方，実際にエッジ検出回路が動作していた時間は，310 μs であるので今後 CPU-FPGA 間の PCI Express レーン数が増加すれば，さらに高速化が可能であるといえる．また，エッジ検出回路の動作時間と各処理時間を比較すると，Java による記述のみの場合に対し約 28.7 倍の高速化が可能であり，NDK による高速化を行った場合に対しても約 11.8 倍の高速化が可能となっている．

作成したエッジ検出回路は，パイプライン化されており，表 4 の FF の大部分はパイプラインレジスタとして使用されている．また，処理する画像の幅によってパイプラインレジスタの大きさが変わるため，384 × 256 の画像を処理する回路を作成した場合，FF の使用率は 35% ($\fallingdotseq$ 23% × 378 ÷ 256) となった．

表 5 の CPU 利用率は A と B で約 1.6 倍の差がある．ここで，(i) と (ii) の処理時間がそれぞれ約 1.9 倍，1.7 倍に増加しているのに対し，(iii) の処理時間は約 1.3 倍の増加となっている．したがって，FPGA を使用した処理では，CPU の利用率による実行速度への影響が小さくなっているといえる．ここで，**図 9** のグラフ A と B を比較すると，グラフ B の FPGA へのデータ送信時間時間が A に対し大きく増加していることが分かる．これは，エッジ検出アプリケーションと同時に実行していた別のプログラムのメモリアクセスにより，バスの使用権が分散したためであると考えられる．

4.3.2 SHA-1 による性能評価

SHA-1 は 2^{64} ビット以下の元データから 160 ビットのハッシュ値を生成するハッシュ関数であり，SSL 通信などに利用されている．1 MByte のデータに対し，SHA-1 を適用しハッシュ値を計算する時間とその内訳，および回路規模を以下に示す．なお，SHA-1 処理回路の動作周波数は，100 MHz である．

表 5 エッジ検出処理時間
Table 5 Processing time of edge detection.

	Java による記述のみ	Android NDK 使用	FPGA 使用
A：CPU 利用率 48.9%	8,900 μs	3,644 μs	3,640 μs
B：CPU 利用率 30.1%	16,700 μs	6,195 μs	4,732 μs

FPGA による実行時間の内訳

FPGA による高速化を行った処理は Java による記述のみの場合に対し，約 157 倍高速である．FPGA による処理のうち，1 MByte のデータ送信に要した時間が 6,605 μs，20 Byte のハッシュ値受信に要した時間が，14 μs である．したがって，送信バンド幅は 151 MByte/sec，受信バンド幅は 1.43 MByte/sec となり，それぞれ 図 7 に示したバンド幅の最大値と図 8 に示した原点付近のバンド幅にほぼ一致する．実際に SHA-1 処理回路が動作していた時間と各処理時間の比較では，Java による記述のみの場合に対し約 241 倍の高速化が可能であり，NDK による高速化を行った場合に対しても約 3.8 倍の高速化が可能となっている．

表 6 は SHA-1 処理回路の回路規模である．SHA-1 の回路規模は処理するデータサイズによらず一定である．また，SHA-1 はデータの先頭から逐次的に処理していく必要があり，並列処理が困難であるため並列化による回路規模の大幅な増加は発生しないと考えられる．

表 7 の CPU 利用率は A と B で約 2.0 倍の差がある．ここで，(i) と (ii) の処理時間がそれぞれ約 2.4 倍，2.9 に倍増加しているのに対し，(iii) の処理時間は約 1.3 倍の増加となっている．したがって，FPGA を使用した処理では，CPU の利用率による実行速度への影響が小さくなっているといえる．また，**図 10** よりグラフ B の FPGA へのデータ送信時間が A と比べ増加していることが分かる．これは SHA-1 と同時に実行していた別のプログラムのメモリアクセスにより，バスの使用権が分散したためであると考えられる．

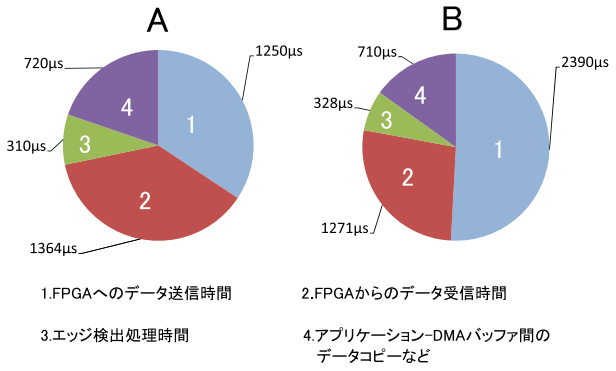


図 9 エッジ検出アプリケーション実行時間内訳
Fig. 9 Breakdown of the execution time in edge detection.

表 6 SHA-1 回路規模
Table 6 Size of the SHA-1 circuit.

リソース	使用量	使用率	アクセラレーション回路の使用率
FF	5,147	9%	8%
6 入力 LUT	6,835	25%	20%

4.3.3 実用的な問題の処理時間の見積り

今後，携帯端末で動画像を処理する場合や，現在注目されているビッグデータを処理する場合を考える．しかしながら，使用した評価ボードの CPU-FPGA 間通信速度が十分ではなく，さらに性能向上の可能性があることや，今回利用したボード上の FPGA のリソース不足により，実際のアプリケーションで扱われるようなデータサイズの実験は困難であった．そこで，CPU-FPGA 間の通信速度と FPGA のリソースが十分あるものと想定し，本実験のデータをもとに，より実用的な問題の処理時間について考察する．

4.3.1 項のような画像処理を適用可能なアプリケーションの例として，動画処理が考えられる．480 × 800 の大きさで，1 秒あたり 60 フレームの動画が 60 秒あるとすれば，処理しなければならない総ピクセル数は約 1.38 ギガピクセルである．エッジ検出の処理時間は画素数に比例するため，Android NDK を用いた CPU の処理時間は約 77 秒 ($\fallingdotseq 1.38\times10^9/(256\times256)\times3644/10^6$) となる．一方 FPGA の処理時間は，PCI Express 4 レーンで接続された FPGA で処理した場合，転送時間が 1/4 になるため，約 35 秒 ($\fallingdotseq 1.38\times10^9/(256\times256)\times(720+310+(1250+1364)/4)/10^6$) となる．以上から，全体の処理時間は FPGA による処理を行った場合，NDK に比べて，2.2 倍の性能差が期待できる．

4.3.2 項のようなハッシュ値計算を用いる例として，データ転送後に送信元と送信先でハッシュ値を照合し，転送データに誤りがないかを確認する処理が考えられる．SHA-1 の処理時間はデータ量に比例するため，100 MByte のデー

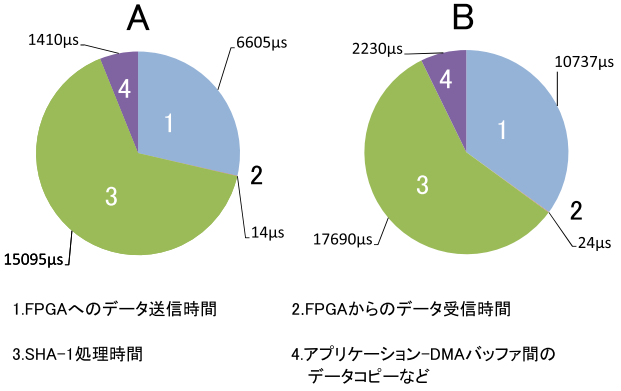


図 10 SHA-1 アプリケーション実行時間内訳
Fig. 10 Breakdown of the execution time in SHA-1.

表 7 SHA-1 処理時間
Table 7 Processing time of SHA-1.

	Java による記述のみ	Android NDK 使用	FPGA 使用
A：CPU 利用率 49.5%	3,619 ms	57 ms	23 ms
B：CPU 利用率 24.7%	8,777 ms	167 ms	30 ms

タを処理すると，Android NDK を用いた CPU の処理時間は約 5.7（$= 57 \times 100/1000$）秒となる．一方 FPGA の処理時間は，PCI Express 4 レーンで接続された FPGA で処理した場合，転送時間が 1/4 になるため，約 1.8 秒（$\fallingdotseq (15095 + 1410 + (10737 + 14)/4) \times 100/1000$）となる．このことから，全体の処理時間は FPGA による処理を行った場合，NDK に比べて，約 3.2 倍の性能差が出るものと思われる．

次に，FPGA を用いて処理する場合のコンフィギュレーション時間について考える．本実験では，利用したボードの制約上，コンフィギュレーションに JTAG を利用しているが，実際の Reconfigu Android では，より高速なコンフィギュレーション方法を想定しており，現状の JTAG によるコンフィギュレーション時間よりも，データシートでコンフィギュレーションデータが公開されている PROM を用いたコンフィギュレーション時間を考える．PCI Express IP コアのデータシート [8] によれば，実験に使用した FPGA（XC6SLX45T）を，PROM を用いてコンフィギュレーションするのにかかる時間は 45 ms である．これは，上記の実際のアプリケーションの処理時間と比較すると十分小さく，コンフィギュレーションにかかる時間を含めた場合でも FPGA による Android アプリケーションの高速化が有効であるといえる．

5. 関連研究

Java の高速化に関するさまざまな研究は，ソフトウェアによる手法，ハードウェアによる手法の両面から行われてきた．ソフトウェアによる高速化手法で一般的なものは，Java VM によるバイトコードの逐次翻訳に代わり，バイトコードから実行するプロセッサに対応する機械語を生成し，それを実行するといった手法である [9]．一方，ハードウェアによる手法では，特定のメソッドを Java VM の代わりにハードウェアで実行する方法や Java バイトコードをハードウェアによって解釈・実行する方法などが存在する．

特定のメソッドを Java VM に代わり FPGA で実行する手法として，Dynamic Method Migration on FPGAs [10] が存在する．処理の一部を FPGA に移すことで，Java VM によるオーバヘッドが減少し，CPU と FPGA の並列処理も可能となる．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している．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される．問題点は，メソッドごとのハードウェア設計を必要とすることと，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高速化の効果が得られなくなることである．

我々は，これまで Jazelle 方式 [11] 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 [12], [13], [14]．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，アクセラレーション機構を組み込むためには，既存のプロセッサに追加することは困難であり，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，既存のプロセッサには手を加えない手法に着目し，その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．

6. まとめ

本稿では，Reconfigurable Android の評価環境を構築し，実際の Java アプリケーションの高速化を行った．JNI を通じて FPGA を制御する方法はすでに提案されている手法 [15] であるが，Android において実現された例はなく，使用した評価環境で即座に利用できる通信機構が存在しない．したがって，FPGA–プロセッサ間の通信機構と，ハードウェアアクセラレーションを実行するための API を提供する必要があり，本稿ではそれらをハードウェア，ソフトウェアの両面から実装を行い，FPGA アクセラレーションの可能性を Android 上において切り拓いた．

Reconfigurable Android が掲げる目標には，アプリケーションの高速化のほかにも，さまざまなプロトコルへの柔軟な対応とセンサデータの高速処理 [6] があげられる．したがって，今後これらの目標達成も目指して研究を進める予定である．

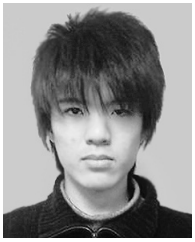
また，今回の評価実験においては，JTAG によるコンフィギュレーションに要した時間を含めておらず，公平な評価とはいい難いが，現在開発を進めている Reconfigurable Android においては，SRAM やフラッシュメモリを用いた高速なコンフィギュレーションが可能なものを想定している．そこで，PROM によるコンフィギュレーション時間をもとに，性能見積りを行った．さらに，今後コンフィギュレーション時間が問題とならないような計算時間の長い処理について実験を行うとともに，ARPIP を用いた評価環境において問題となったプロセッサ–FPGA 間の通信速度やコンフィギュレーション機構を改善した開発環境を構築し，Reconfigurable Android の優位性を示す予定である．

また，パーシャルリコンフィギュレーションを取り入れ，アクセラレーション回路のみを再構成することで，通信回路を動作させたままアプリケーションを実行しつつ，データ転送とコンフィギュレーションをオーバラップさせてコンフィギュレーション時間を隠蔽するような機構を考案していく予定である．

謝辞 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 2 国間交流事業（共同研究）による支援を得た．

参考文献

[1] available from ⟨http://developer.android.com/sdk/ndk/index.html⟩.
[2] available from ⟨http://www.intel.com/jp/intel/pr/press2010/101124.htm⟩.
[3] available from ⟨http://www.actel.com/intl/japan/products/IGLOOseries/⟩.
[4] available from ⟨http://labs.beatcraft.com/ja/index.php?Open%20Source%20Gadgets⟩.
[5] 河内智志，薛 媛，藤波香織：携帯端末の身体上格納場所判定機能のスマートフォンへの実装，インタラクション 2011，pp.531–534 (2011).
[6] Vu, T.T., Sokan, A., Nakajo, H., Fujinami, K., Suutala, J., Siirtola, P., Alasalmi, T., Pitkanen, A. and Roning, J.: Detecting water waste activities for water-efficient living, *Proc. 13th International Conference on Ubiquitous Computing*, pp.579–580 (2011).
[7] available from ⟨http://www.teldevice.co.jp/product/intel/focus.html#3⟩.
[8] available from ⟨http://www.xilinx.com/support/documentation/user_guides/s6_pcie_ug654.pdf⟩.
[9] Arnold, M., Fink, S.J., Grove, D., Hind, M. and Sweeney, P.F.: A Survey of Adaptive Optimization in Virtual Machines, *Proc. IEEE*, Vol.93, No.2, pp.449–466 (2005).
[10] Lattanzi, E., Gayasen, A., Kandemir, M., Narayanan, V., Benini, L. and Bogliolo, A.: Improving java performance using dynamic method migration on fpgas, *Proc. 18th International Parallel and Distributed Processing Symposium*, p.134 (Apr. 2004).
[11] available from ⟨http://www.arm.com/ja/products/processors/technologies/jazelle.php⟩.
[12] 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).
[13] 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).
[14] 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).
[15] Christiaens, M. and Stroob, D.: Interfacing java with reconfigurable hardware (2004).



小池 恵介 （学生会員）

1989 年生．2008 年東京都立戸山高等学校卒業．2012 年東京農工大学工学部情報工学科卒業．現在，同大学大学院工学府博士前期課程情報工学専攻に在籍．

太田 淳 （正会員）

1984 年生．2005 年育英工業高等専門学校情報工学科卒業．2007 年東京農工大学工学部情報コミュニケーション工学科卒業．2007 年より 4 年間，サレジオ工業高等専門学校情報工学科非常勤講師．2008 年東京農工大学院工学府博士前期課程情報工学専攻修了．2008 年より 3 年間，同大学院工学府博士後期課程電子情報工学専攻に在籍．組み込みシステム，スマートデバイスに関する研究に興味を持つ．組み込みシステムシンポジウム 2010 優秀論文賞を受賞．

大島 浩太 （正会員）

2003 年東京農工大学大学院工学研究科電子情報工学専攻博士前期課程修了．2006 年同大学院工学教育部電子情報工学専攻博士後期課程修了．博士（工学）．現在，東京農工大学大学院工学研究院助教．無線センサネットワーク，異種ネットワーク連携，オーバレイネットワーク等の研究に従事．電子情報通信学会会員．

藤波 香織 （正会員）

1970 年生．1995 年早稲田大学大学院理工学研究科電気工学専攻修士課程修了．同年日本電信電話（株）入社．1997～2003 年 NTT コムウェア勤務．2005 年早稲田大学大学院理工学研究科情報科学専攻博士課程修了．早稲田大学理工学術院客員講師，オウル大学訪問研究員を経て，現在，東京農工大学大学院工学研究院准教授．ユビキタスコンピューティング，HCI，分散システムに関する研究に従事．ヒューマンインタフェース学会会員．博士（情報科学）．

郡 信幸

1978 年生．2002 年東京商船大学商船学部流通情報工学課程卒業．2004 年同大学大学院商船学研究科交通電子機械工学専攻修了．同年株式会社ビート・クラフト入社，2005 年同社退社他社勤務後，2007 年より株式会社ビート・クラフトに復帰，現在に至る．修士（工学）．

竹本 正志

1971 年生．2000 年に株式会社ビート・クラフトを設立し，代表取締役に就任．以降，同社にてコンシューマエレクトロニクス用ソフトウェアを開発．2006 年からはハードウェアの設計も手がけ，ソフトウェア・エンジニアのためのハードウェアプラットフォームの開発を続けている．

中條 拓伯 （正会員）

1961 年生．1985 年神戸大学工学部電気工学科卒業．1987 年同大学大学院工学研究科修了．1989 年同大学工学部助手の後，1998 年より 1 年間 Illinois 大学 Urbana-Champaign 校 Center for Supercomputing Research and Development（CSRD）にて Visiting Research Assistant Professor を経て，現在，東京農工大学大学院工学研究院准教授．プロセッサアーキテクチャ，並列処理，リコンフィギャラブルコンピューティングに関する研究に従事．電子情報通信学会，IEEE CS，ACM 各会員．博士（工学）．