ソフトウェアからFPGAを容易に扱うための分散オブジェクトプラットフォーム

大川 猛[†] 高野創司[†] 植竹大地[†]
横田隆史[†] 大津金光[†] 馬場敬信[†]

FPGA(Field Programmable Gate Array) を用いたハードウェアによる効率的な並列処理は，高性能計算やリアルタイムシステム構築などの様々な場面で用いられている．しかし，FPGA を扱うには一般的にハードウェア記述言語 (HDL: Hardware Description Language) による設計を必要とし，ソフトウェア開発者からは敷居が高い．

本稿では，CORBA(Common Object Request Broker Architecture) を用いて FPGA にプログラムするハードウェアを抽象化する手法を紹介する．FPGA 上のハードウェアを抽象化することにより，ソフトウェア開発者は FPGA 上の回路・計算リソースを抽象的な部品として容易に使用することが可能になる．更にホストシステムから FPGA を操作する実例を示すことで，FPGA を含むシステム開発への CORBA 適用の有効性を示す．Android 端末上の Java プログラムから FPGA を操作する分散システムの構築事例を解説し，性能評価結果を紹介する．

A Distributed Object Platform for Easy Handling FPGA from Software

Takeshi OHKAWA ,[†] Soshi TAKANO,[†] Daichi UETAKE,[†]
Takashi YOKOTA,[†] Kanemitsu OOTSU[†] and Takanobu Baba[†]

FPGA(Field Programmable Gate Array) is widely used for an optimiezed parallel processing in many situations, such as high performance computing and constructing realtime systems. However, there are barriers to introduce FPGA as a computing resouce for a general software developer because it is necessary to write a design in Hardware Description Language(HDL) to use FPGA.

This report introduces an abstraction method of a hardware programmed on an FPGA using CORBA(Common Object Request Broker Architecture). By the abstraction of the hardware on FPGA, a software developper can use the cirtuit or computing resource on FPGA very easily. And the adaptability of CORBA platform for a development of an FPGA system is exhibited by a case study of operating a digital circuit on FPGA from host system through CORBA. A Java program on Android terminal is used to operates FPGA and the performance evaluation results are shown.

1. はじめに

1.1 研究背景

FPGA（Field Programmable Gate Array）は，大容量化・高速化・低消費電力化が進んでおり，高性能計算やリアルタイムシステム構築などの様々な場面で用いられている．しかし，FPGA を扱うには一般的にハードウェア記述言語 (HDL: Hardware Description Language) による設計を必要とするため設計生産性が低いという問題があった．そのため，近年，FPGA 向けの設計手法として，C 言語や Java 言語からの動作合成を行うツールが多く製品化されている．ただし処理の並列性を活かして性能を向上するためには十分な習熟が必要であり，一般のソフトウェア開発者にとって必ずしも使いやすい開発環境ではない．様々な分野の応用で FPGA を有効に活用するためには，FPGA 開発に多くのソフトウェア・システム開発者を招き入れ，設計生産性を向上する必要がある．そのために，どのような技術を研究開発し，どのような設計環境を用意すればよいだろうか．

システム設計生産性の向上は様々な要因により成し

† 宇都宮大学大学院工学研究科
〒 321-8585 栃木県宇都宮市陽東 7-1-2
Graduate School of Engineering, Utsunomiya University, Yoto 7-1-2, Utunomiya, Tochigi, 321-8585 Japan
ohkawa@is.utsunomiya-u.ac.jp

遂げられる[1] ものであるが，その一つがコンポーネント指向による設計部品の再利用である．分散オブジェクト技術である CORBA (Common Object Request Broker Architecture)[2] はコンポーネント技術の筆頭として挙げられるものである．CORBA は，部品にインターフェイス定義 (Interface Definition Language :IDL 言語による) を行うことで，メソッド呼出しをメッセージ化して通信し部品を操作する枠組みである．ローカルであってもネットワーク上の遠隔オブジェクトであっても同じように操作する事が出来る点が優れている．更に，インターフェイスを定義してあれば内部の実装は任意の言語で開発できる点が設計生産性の向上につながる．CORBA は既に，多くのプログラミング言語に対応している (Java,C,C++,Python,Ruby,Perl など)[2]．この考え方で HW を部品化しようとする研究開発はいくつか例がある[3),4)]．

著者は，CORBA の考え方で FPGA 上のディジタル回路をオブジェクトとして扱うことを提案し，FPGA 上で動作する具体的な実装を行ってきた[15),16)]（ORB エンジン）．CORBA による遠隔メソッド呼出しはプログラミング言語に依存しないため，例えば Python や Ruby といったスクリプト言語からでも FPGA 上のハードウェアの処理を起動することが可能になる．FPGA による効率的な並列処理は，これまでソフトウェア開発者から距離があったが，ORB エンジンの導入により身近なものになると期待される．

1.2 研究目的

本研究は，FPGA 上のハードウェアをソフトウェアから扱うための抽象化手法について議論し，今後の FPGA システムの設計生産性向上のための設計手法を提案することを目的としている．本稿では，以上の背景を基に，分散オブジェクトを活用した FPGA のハードウェア抽象化によるプログラミング手法・設計開発方法について検討する．第 2 節は，従来のハードウェア抽象化手法について俯瞰的に解説したのちに，大規模・高性能 FPGA 時代に対応したハードウェア抽象化手法として，FPGA 用に開発された CORBA 準拠の分散オブジェクトである ORB エンジンについて紹介する．第 3 節は更に CORBA を用いてホストシステムから FPGA を操作する実例を示すことで，FPGA を含むシステム開発への CORBA 適用の有効性を示す．Android 端末上の Java プログラムから FPGA を操作する分散システムの開発事例を解説し，性能についての評価結果を紹介する．そして，開発過程を詳細に説明し設計生産性の向上について考察する．第 4 節で，本稿の内容をまとめる．

2. FPGA 向け設計環境の検討

2.1 大容量・高性能 FPGA 時代に対応したシステム設計手法

FPGA の大容量化・高速化が進み，2000 万論理ゲートがプログラム可能な FPGA が製品化されている[5)]．また，100MHz 以上のシステム周波数で動作する FPGA システムも珍しくは無く，モジュールレベルでは 400MHz 程度[6)] での周波数で動作している．一方，FPGA 上に最適な並列処理のための回路をアプリケーションに応じてプログラムし，ソフトウェアと組み合わせて対象のアプリケーションを高速化したいという要求が高まっている．

このような状況でありながらも，FPGA を設計する環境としては RTL(Register Transfer Leve) レベルでの回路設計環境がいまだに一般的である．RTL レベルとはすなわち，「1 つのクロックサイクルの間に，現在の回路中のレジスタの値と入力信号値から，次のサイクルにおける回路中のすべてのレジスタの値と出力信号値を決める論理を定義する」というプログラミングモデルである．

RTL でのシステム設計においては，全てのレジスタの値を決める論理を完全に並列に記述する必要がある．このために，一般的には VHDL や Verilog-HDL といった HDL(Hardware Description Language) プログラミング言語が用いられる．これらのプログラミング言語を用いて記述する際には，レジスタを書き換える論理を 1 行 1 行記述していくわけであるが，複数のレジスタに同時に書き込む論理を記述してしまうと期待通りに動作しないため，競合が起こらないよう，細心の注意を払う必要がある．また，逐次的な動作を記述する際には状態遷移と論理を両方記述する必要があるために記述量が多くなる．

すなわち，RTL の完全並列プログラミングモデルは一般的なソフトウェアの逐次的なプログラミングモデルとはかけ離れており，一般的なソフトウェア技術者が開発に参加することは難しい．この問題に対し，C 言語や Java 等の言語での記述から，HDL を生成する高位合成の研究が数多く行われており，設計環境は製品化され実用の域に達している．しかし，C 言語での設計で高性能なハードウェアを設計するには，ハードウェア化を想定した記述が必須であり，ハードウェア設計を熟知した技術者が，HDL 記述をする代わりに C 言語で記述する，という使い方が主である．

FPGA の設計環境と対照的なのが，GPU である．すなわち，GPU のハードウェアが CUDA という

C/C++言語を基にしたプログラミングモデルにより抽象化され，一般のソフトウェアエンジニアが GPGPU (General Purpose computing on Graphics Processing Unit) という概念で GPU のハードウェア資源を活かした並列プログラミングに多く参加している．GPGPU は，ソフトウェア開発者に広く受け入れられ，GPGPU に基づく研究開発・実問題への応用の進展は目覚ましい．同じ C 言語で設計できるはずの，GPGPU と FPGA の差とは何なのだろうか？

GPGPU においては，演算器が並列に配置されておりグループ毎にローカルメモリが存在するという計算モデルが既にあり，そこに合わせて C 言語プログラムを修正し実行して性能向上を即時に確かめることが出来る．一方，FPGA はビット単位で配線のプログラミングが可能であるため，計算資源を最大限に活用した並列処理を行う自由度を持っており，そこを HDL で記述することにより最適な並列処理を行うハードウェアを設計することで性能向上を行う．すなわち，FPGA 向けハードウェア合成は，想定する計算モデルがない点が GPGPU 向けの C 言語プログラミングと大きく違う．そのため，都度ハードウェアを合成するために時間がかかり，実行が即時に行えないために，プログラムの変更による性能向上を確認しづらいため，ソフトウェア開発者には扱いづらいものになっている．

以上をまとめると，FPGA は任意の回路を細かくプログラムして最適な並列処理を実現できることが特長であるが，自由度が高いことにより設計のターンアラウンド時間が長く設計生産性が悪いことが問題である．

また FPGA はチップ上に搭載されている演算器の量では GPU には劣る為，SIMD 的な並列処理を FPGA 上で行っても特長を活かせない．つまり等質なデータ並列処理を FPGA 上に展開することは，性能向上にはつながるが，FPGA のビットレベルのプログラマビリティを活かすことにならない．FPGA の特長を活かすためには，タスクレベルの並列処理もしくはデータパスでの進行方向の計算処理をまとめて行うという，ヘテロジニアスな並列処理が有効である．そのため，FPGA のプログラマビリティを最大限に生かすには C 言語を使うよりは，最適化されたハードウェア設計言語を使ったほうが良い．

ただし昨今のシステムは大規模化・複雑化しており，すべてをハードウェアとして設計することは現実的ではない．すなわち，比較的高速な処理を要求されないが種類と分量が多い処理に関してはソフトウェアで実装し，システムでの性能ボトルネックとなる部分についてはハードウェアモジュールとして実装する，「ソフトウェアとハードウェアの組み合わせ」が必要である．そのため，作成したハードウェアをソフトウェアから呼び出し可能なインターフェイスをつけて，抽象化・部品化（コンポーネント化）を行うことで，ハード部品を作成することが，ソフトウェアからハードウェアの呼び出しを容易にするために有効である．

2.2 提案する FPGA 向け設計環境の枠組み ～部品ごとにそれぞれ最適な言語での設計

以上を鑑み，筆者らは，大容量・高性能 FPGA 時代に対応したシステム設計手法は，「FPGA 上のハードウェアを，ソフトウェアから容易に操作可能にするために部品化すること」，と考える．

この手法のメリットは，部品ごとにそれぞれ最適な言語で設計できるようになるという事である．前述のように，単一の仕様記述言語（SystemC や ANSI S など）からのハードウェア・ソフトウェア合成は現在実用化の域に入りつつあるが，ハードウェア記述に特化した新しい言語（例えば Bluespec System Verilog）の方が，よりきめ細かいハードウェアの制御が可能であることは間違いない．一般に開発者は，本来最も得意な開発環境やプログラミング言語での開発が最も設計生産性が高い．そのため，大幅に設計生産性を向上する設計手法を切り拓くことが期待される．

さて，それでは FPGA 上のハードウェアの部品化をどのように行ったらよいのであろうか？

本稿では設計生産性の向上に関して，主に論理的なレベルで考える．現状，論理的なインターフェイスは，メモリマップド I/O によるレジスタアクセスが一般的である．FPGA を部品化するためには，FPGA を操作するための API(Application Programming Interface) を定義することが考えられ，いくつか提案もある．しかし，API を定義する時点で何らかのプログラミング言語や応用に特化してしまうことは避けられない．また，ホストシステムと FPGA の間の通信を陽にプログラム記述すると，境界が移動した際にプログラム記述を書きなおす必要がある．その為，特定の実装言語に依存せず，また特定の FPGA 操作用 API をプログラム中に埋め込むことなく，FPGA とソフトウェアを境目なく繋ぐことが出来るような環境が理想であると考える．

なお，電気信号のレベルについては，SoC(System on Chip) の時代になり，大規模な LSI チップ中の IP（ディジタル回路部品）をモジュール化する手法に関して，IP-XACT[7] というインターフェイス定義に基づく手法が一般的に用いられている．例えば，ARM プロセッサの標準的な AXI バスに接続可能な IP，と

段階	抽象化方式	アクセス単位	適応システム 例	通信路 例
第1段階	メモリマップドI/O ＋割り込み	メモリアクセス	・プロセッサ シングルスレッド	チップ内バス or ボード内バス
第2段階	デバイスドライバ インターフェイス	関数呼出し	・プロセッサ マルチスレッド	チップ内バス or ボード内バス
第3段階	分散オブジェクト指向 インターフェイス	メソッド呼出し	・組込み 分散プロセッサ システム	チップ内バス or ボード内バス or イーサネット・無線・USB等
第4段階	サービス指向 インターフェイス	サービス呼出し	・インターネット上の分散システム、クラウド	イントラネット・インターネット（IPネットワーク）

図 1 ハードウェア抽象化の方式と適応システム範囲

いう形で，電気信号的なインターフェイスの定義がなされている．電気信号レベルと論理レベルのマッピングについては，最適化の観点から今後検討の余地があると考える．

次節では，ハードウェアの抽象化方法という基本原理に立ち還り，FPGA 向けの設計環境に必要な部品化の手法について検討する．

2.3 ハードウェアの抽象化方法

FPGA 上のハードウェアを，ソフトウェアから容易に操作可能な部品として抽象化するためには，どのような手法が適しているのであろうか．

図 1 に，本稿で検討の対象とするハードウェア抽象化の方式と適応システム範囲例を示した．ハードウェアをソフトウェアから扱うためのインターフェイス (I/F) は，低い抽象度から高い抽象度まで目的に応じて多くの選択肢がある．本稿では，(1) 直接 I/O もしくはメモリマップド I/O ＋割り込み，(2) デバイスドライバ I/F，(3) オブジェクト指向 I/F，(4) サービス指向 I/F，の 4 つに分類して議論する．現在，ハードウェアの抽象化に関して一般的に用いられる考え方は (1) もしくは (2) であるが，ソフトウェアのコンポーネント化技術に分類される (3) もしくは (4) の考え方をハードウェア抽象化に拡張する事で，どのような利点があるか，また解決すべき課題は何であるかを明らかにする．

(1) 第 1 段階：メモリマップド I/O ＋割り込み

プロセッサ命令を用いてハードウェアのレジスタメモリを読み書きすることで，ソフトウェアからハードウェアを扱う最も原始的なインターフェイスである．通常はソフトウェアが動作の主体となり，ハードウェアが追従して動作する．ハードウェアからソフトウェアに対して動作を要求する際には，一般に割り込みが使われる．割り込みに対応した動作は，割り込みハンドラソフトウェアとして記述する☆．メモリ読み書きレベルのプログラミングが必要であるため，主にアセンブラもしくは C 言語で記述される．また，OS 環境としては，メモリ保護のない組込み用 OS や OS レスの環境で使われるモデルである．この段階の抽象化は，「ソフトウェアからの，メモリへの読み書きで操作するハードウェア」というモデルである．この抽象度における問題点は，以下の様に (a) 操作の粒度，(b) 操作の意味，(c) 排他的使用という観点で考えることが出来る．

- **(a) 操作の粒度** メモリへの読み書きはバイト単位・ワード単位の粒度である．ハードウェアに対して何らかのアクションをさせる，もしくはハードウェアからデータを受け取る際に，バイト単位・ワード単位だけでは完結しないことが殆どである．例えば，画像データの転送等は 1 回のメモリアクセスで完了することは無く，複数回のアクセスが必須である．そのため，一連の操作の手順はソフトウェアで逐次的に記述し，手順が守られるように管理する必要がある．
- **(b) 操作の意味** どのメモリアドレスに何の機能が割り当てられているか，各アドレスの意味は「ハードウェア IP 仕様書」などの別のドキュメントで与えられる．誤ったメモリアドレスにアクセスすると，期待通りには動作しない．そのため，操作の意味はソフトウェア側で管理する必要がある．
- **(c) 排他的使用** 複数のプログラムがハードウェアを同時に使用すると，一連の動作が阻害され，期待通りに動作しない．そのため，あるハードウェアを使用する権利を一つのプログラムに限定して，他のプログラムからの使用を禁止するのが排他的使用という考え方である．これは，(a) 操作の粒度，とも関連があるが，複数回のメモリアクセスで一連の動作を行うために，一連の動作中に別のプログラムが割り込んで使用しないように配慮する必要があるということである．
- **(d) 再利用可能性** あるハードウェアを操作するためのプログラムは，その特定のハードウェア専用となり，たとえ同じ様な機能を持つハードウェアであっても，一般にプログラムはそのまま再利用することは出来ない．

以上から，第 1 段階の「メモリマップド I/O ＋割り込み」という抽象化では，(a), (b), (c) の観点で，意図しない動作を引き起こすバグが入り込む可能性があり，設計生産性向上の妨げになっていると言える．また，(d) の観点で，ハードウェアのメモリアドレスに何の機能が割り当てられているかは，ハードウェア固有の

☆ I/O 専用命令を持つプロセッサの場合は，メモリマップド I/O の代わりに直接 I/O と呼ぶ必要があるが，本質的には同じである．

ものである．これは (b) 操作の意味とも関連するが，ハードウェアを開発するベンダーが，各社そのハードウェアの制約条件（コスト・開発期間・性能・消費電力）によって，様々な機能をメモリアドレスに割り当てるためであり，予め申し合わせて統一することが不可能だからである．

また，現状の FPGA 上の回路モジュールは，メモリマップド I/O としてプロセッサ上のソフトウェアからバスを通じて操作されることが一般的である．

(2) **第 2 段階：デバイスドライバ I/F**

第 1 段階の一連の問題に対応するため，OS 環境では，HAL(Hardware Abstraction Layer) という抽象化層を持たせて，特にアプリケーションプログラムの移植性 (ポータビリティ) を確保することが一般的である．多くの OS では，アプリケーション層と明確に分離した，デバイスドライバという層での抽象化を行っている．例えば Linux システムにおいては，アプリケーションプログラムから見て，全ての資源はファイルである，という抽象化を行い，「ファイルへの読み書き」を行うことでハードウェア (デバイス) を操作する意味付けをしている．デバイスドライバは，ファイルに対する操作に対応する動作（open, close, read, write 等）に対応する一連の関数の集まりである．また，デバイスドライバに含まれる各関数は特権モードで動作することで，(c) 排他的使用を行うための排他制御が可能となり，(a) 操作の粒度を大きくすることが出来る．また，ファイルに対する操作に抽象化することで，(b) 操作の意味をファイル操作の意味でカプセル化できるようになった．すなわち，アプリケーションは，各メモリアドレスを読み書きする操作の意味をいちいち気にすることは無く，ハードウェアをファイルとして扱うことが出来るようになったわけである．また，ハードウェアからのソフトウェア呼出しに関しても，特権モードでの割り込み処理プログラムをデバイスドライバ内に用意することで対応可能となっている．一方，Microsoft 社の商用 OS である Windows においても，デバイスの種類ごとに操作のための関数によるインターフェイスが用意されている．しかし，例えば Linux システム用に作られたデバイスドライバは，そのまま Windows で使用することは出来ない．一般に，デバイスドライバの抽象化の度合いやインターフェイス形式は OS 環境により様々である．すなわち，ハードウェアをある OS で使用するためには，その OS 向けのデバイスドライバを開発する必要がある，ということである．その為，(d) 再利用可能性については，単一の OS 内に限ってみれば大幅に向上するが，OS 環境が異なると途端に再利用不可能になる．

ここで，抽象化を進めることにより顕在化する問題点について，新たな観点を加える．

(e) 性能オーバーヘッド 操作する側とされる側の間での何らかの意味変換が必要となる為に，変換のための遅延時間が発生する．抽象度が高くなると，一般に遅延時間は大きくなる傾向にある．

例えば，カメラからの画像取り込みで，あるメモリアドレスにコマンド書き込みを行った後に，別のメモリアドレスから画像を読み出すケースでは，ファイルに抽象化すると「ファイルを open し，write, read を行った後に close する」という手順が必要である．これが性能のオーバーヘッドになる．更に，デバイスドライバは特権モードで動くため，ユーザモードとの切替に時間がかかる．また，第 1 段階の単純なメモリアクセスと比べると，関数呼び出しの時間がかかる，といったオーバーヘッドが随所に発生する．

すなわち，抽象化を進める際には，その際に追加される性能オーバーヘッドが十分許容範囲内であることを保証しながら進める必要がある．

(3) **第 3 段階：オブジェクト I/F**

第 2 段階までの抽象化は現在広く一般的に行われているハードウェア抽象化である．FPGA 上のハードウェアをどのように抽象化するとソフトウェアから扱い易くなるのであろうか．そして，ソフトウェア開発者に受け入れられ，設計生産性の向上につながるのであろうか．

ソフトウェアが動作しているプロセッサから見て，ハードウェアはペリフェラル（周辺機器）と呼ばれ，プロセッサの外部に存在するシステムである．離れたシステムのメソッド呼び出しを行う枠組みとして，旧来，分散ソフトウェアの分野では分散オブジェクト技術が研究されてきた．

分散オブジェクトとは，オブジェクトを複数のコンピュータに分散して配置し，各オブジェクトがメッセージをやり取りすることで目的の処理を行う枠組みのことである．分散オブジェクトを用いると，オブジェクト指向言語（C/C++, Objective-C，Java，Python, Perl, Ruby など）で記述して分散配置したオブジェクト間で，メソッド呼び出しを記述するだけで，システム間の連携動作が可能になる．この際メソッド呼出しはメッセージ通信により行われる遠隔メソッド呼出しとなるが，プロトコル処理を行う通信プログラム（Stub, Skeleton）は分散オブジェクトの枠組みにより自動生成されるため，通信に起因したバグに悩まされることなく，分散システムの開発が行える．

ORB（Object Request Broker）は，分散オブジェクトの概念を実現するミドルウェアである．ORB の代表格といえるのが CORBA (Common Object Request Broker Architecture)[2] である．CORBA においては，オブジェクト指向のインターフェイスを定義するために IDL (Interface Definition Language) という言語を用いる．IDL は C や Java といったプログラミング言語からは中立の記述であり，各言語へのマッピングが規定されている．IDL ファイルからは，IDL コンパイラを用いて，C や Java といったプログラミング言語で記述された通信プロトコル処理プログラム（Skeleton, Stub）を，アプリケーション毎に生成する．プロトコル処理をその場で行うのではなく，予めコンパイルしたコードで行うため，インタプリタ式のプロトコル処理より性能面で優れている．通信に用いられるプロトコルは GIOP(General Inter-ORB Protocol) と呼ばれ，予めデータ構造の配列を IDL に基づいて定めるバイナリデータのメッセージでの通信が行われる．ハードウェアをオブジェクト I/F により抽象化することに，どのようなメリットがあるのだろうか．(a) 操作の粒度・(b) 操作の意味の観点では，操作そのものをオブジェクト指向のメソッドとして定義できるため，デバイスドライバ I/F におけるファイルとしての抽象化よりも更にハードウェアの使用者にとって扱いやすいものとなる．(c) 排他的使用については，CORBA の枠組みとして提供されるものであり，メソッド呼び出し中に他の呼び出し要求があっても再入を禁止できる．また CORBA オブジェクト化による，(d) 再利用可能性の向上効果も大きい．IDL により定義されるオブジェクト指向のインターフェイスが変わらない限りは，性能向上やハードウェア量削減のためにのために内部の実装をいくら変えても良いし，また呼び出し側のソフトウェアも自由に変更可能である．まさに，ソフトウェアから使用しやすいハードウェアの部品化が可能になるといえる．

一方，(e) 性能オーバーヘッドについては問題となる場合が生じてくる．メソッド呼び出しの遅延時間のために，応用によってはオブジェクト指向のインターフェイスを実現する手段として CORBA が使用不可能な場合が生じる．これは次節で詳しく論ずる．

CORBA は 1990 年代に標準化が進んだが，1995 年ごろからインターネットが全世界で本格的に普及しだすと，CORBA の抱える問題が顕在化しだした．最も致命的だったのは，CORBA の IIOP プロトコルはサーバオブジェクト毎に任意の 1024 番以降の TCP/IP ポートを使用することであった．セキュリティが重要となった現在では，TCP/IP のポートをむやみに解放することは不可能で，インターネット上に CORBA のオブジェクトを分散配置して自由に使用する，という分散計算環境の実現は難しくなった．そのため必然的に，CORBA の活躍場所は，閉じたネットワーク（LAN : Local Area Network）で高速・高信頼な通信が必要なシステムとなった．例えば，ロボットの内部ネットワークでは，ビジョンシステム・センサ・アクチュエータを統合するために，CORBA を基にした分散ミドルウェア（RT ミドルウェア[11]）が積極的に検討されている．

(4) **第 4 段階：サービス I/F**

世界中の分散システムをインターネットを通じて連携するための仕組みは，分散オブジェクトよりも更に結合を疎にする方向で発展を続けている．WWW(World Wide Web) の実現に用いられている HTTP は，ステートレスなサーバを前提としたプロトコルである．ステートレスとは，サーバが状態を持たないという意味で，1 回 1 回のリクエストが完全に独立したものになる．1 回のサービス依頼で処理に必要な情報を全てリクエストメッセージ内に埋め込むことで，処理を担当するサーバを変更しても柔軟に対応可能となっている．クラウドコンピューティングは，インターネット上に存在する計算資源を，その資源のありかや状態を気にせずに，使用者はサービス要求を行い，提供するサーバがサービスを提供する，という枠組みである．こうした HTTP プロトコルを使用したサービスの提供は Web サービスと呼ばれる．Web サービスで用いられる，SOAP[13](もともとは Simple Object Access Protocol の略語) は，遠隔オブジェクトの操作を XML メッセージにして．XML によりオブジェクトの構造や値を表現しインタプリタ式のプロトコル処理を行うことが特徴である．すなわち，遠隔のサービスに対するリクエストの必要が生じた時点でメッセージを作成し，受信してメッセージを解釈するため，プロトコル処理の時間は多くかかる．

ハードウェア抽象化をサービスの I/F で行うことは，(a)・(b)・(c)・(d) の観点ではオブジェクト指向 I/F を更に柔軟にしたものであり，ソフトウェアからハードウェアを容易に扱う手段としては申し分ない．しかし，(e) 性能オーバーヘッドに関しては，オブジェクト指向 I/F 以上に大きい．これは次節で詳しく論ずる．今後，FPGA 上のハードウェアをサービス I/F レベルで抽象化することで，世界中からインターネットを用いて直接操作することが可能になると考えられる．

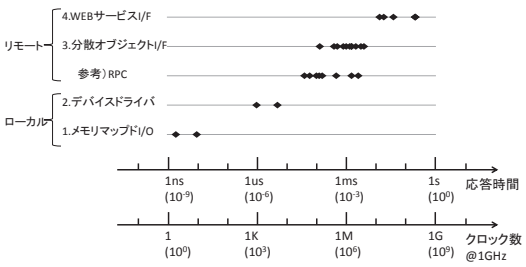

図 2　各段階の抽象化方式と応答時間

ローカル／リモートの別	分類	内容	呼出し時間（秒）
ローカル	メモリマップドI/O	Load/Store @ 50MHz	2.0E-09
		Load/Store @ 100MHz	1.0E-08
	デバイスドライバ	Write	1.0E-06
		Open/Write/Read/Close	5.0E-06
リモート	RPC	SunRPC(TCP)	1.5E-03
		Direct VRPC	1.0E-04
		Reduced VRPC	6.0E-05
		Optimized VRPC	4.0E-05
		LRPC	1.3E-04
		LRPC/MP	1.6E-04
		TAOS	4.6E-04
		Firefly RPC	2.5E-03
	分散オブジェクト指向	openORB (CORBA, Java)	3.1E-03
		JacORB (CORBA, Java)	2.1E-03
		JavaIDL (CORBA, Java)	3.9E-03
		RMI (Java)	1.5E-03
		HORB (Java)	1.0E-03
		JavaIDL (CORBA, Java)	1.3E-03
		RMI (Java)	8.0E-04
		OmniORB (C++)	1.3E-04
		MICO (C++)	3.9E-04
		OrbEngine (Spartan3E, C)	1.5E-03
		OrbEngine (Virtex6, C)	5.0E-04
	サービス指向	Apache AXIS (Java)	1.8E-02
		Apache AXIS (Java)	1.3E-02
		MS SOAP Toolkit (Visual Basic)	2.0E-01
		SoapRMI (Java)	3.8E-02
		SOAP::Lite(Perl)	2.0E-01
		Apache SOAP (Java)	2.1E-01

図 3　応答時間の詳細

2.4　抽象化方式の比較

これまで説明した 4 段階のハードウェア抽象化に対して,「ソフトウェアからハードウェアを 1 回操作するためにかかる時間」という観点でまとめたのが，図 2 である．もとにしたデータは，図 3 に示している．

想定しているシステムもしくは測定対象のシステムの動作周波数は様々であるが，大まかな傾向を示すために，同じ抽象化方式についてはまとめて図中にプロットを行った．

抽象化第 1 段階：メモリマップド I/O については，50MHz と 100MHz での 1 クロックでのメモリアクセス (Load/Store) を想定した算出値 (理論値) がプロットしてある．また第 2 段階：デバイスドライバ I/F については，一般的な PC（Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz）上で LinuxOS（CentOS6.3）を動作させ，デバイスドライバの単純な機能（open/close/read/write）にかかる時間を計測した結果を示してある．

また，第 3 段階：オブジェクト I/F については，これまで報告されている ORB の呼出し時間[12),14),16),18)]をプロットした．これらの主な通信路は 100Mbps もしくは 1Gbps のイーサネットである．更に，遠隔オブジェクト呼び出しの類似技術として RPC(Remote Procedure Call) の呼び出し時間を参考としてプロットした[8)~10)]．RPC はオブジェクト指向ではないが，C 言語や FORTRAN の関数呼び出しをメッセージ化して，遠隔呼び出しを可能にする環境である．主に HPC 分野で用いられ，オブジェクト指向の柔軟性よりも，高速性に主眼が置かれているため，より高速な通信環境を用いたシステムを構築することで 100us を下回る呼び出し時間を示す例もある．

更に，第 4 段階：サービス I/F として WEB サービス（SOAP）の呼び出し時間をプロットした[12),14)]．WEB サービスの呼び出し時間は，要求に応じて XML の解釈が必要であるため，性能オーバーヘッドは第 3 段階よりもかなり大きくなることが分かる．

2.5　FPGA 向けの分散オブジェクト環境 ～ORB エンジン

著者は，現状のデバイスドライバを経由したハードウェアの操作手法 (第 2 段階) に対して，第 3 段階のオブジェクト I/F での抽象化を行うことで設計生産性に関して大幅な向上を行うことが可能であると考えている．図 2 によると，第 3 段階の応答時間は 0.1ms～10ms に分布している．一方，第 2 段階の応答時間は最大でも 10 マイクロ秒程度である．リモートとローカルの違いはあるが，ハードウェアを扱うためのインターフェイスとしては，デバイスドライバの 10 マイクロ秒が目標になる．10 マイクロ秒は 1GHz システムクロックを想定した際に，10K サイクルに相当するため，少なくとも数 10 バイトある CORBA メッセージのプロトコル処理を行うには，少ないサイクル数であると考えられる．また，現在一般的に入手可能な ORB は Linux 等の環境で動作することを想定しおり大がかりであり，数 MB のフットプリントを要する[11)]．

その為，FPGA で使用可能な軽量･省資源な分散オブジェクト環境として著者が提案したのが ORB エンジンである[15),16)]．ORB エンジン実装は，FPGA 向けのソフトコアプロセッサ MicroBlaze 上で 16KB 程度のフットプリントで動作可能な C 言語ソフトウェアを中心にして，FPGA ハードウェア設計データおよびツール群で構成されており，SourceForge で公開されているオープンソースソフトウェアである[17)]．ORB エンジンの現状の呼び出し時間は，図 4 に示すとおり，0.5ms 程

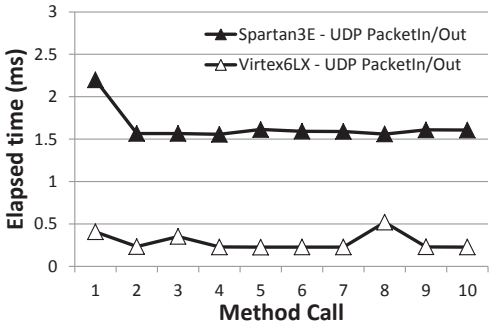

図 4　ORB エンジンの応答時間

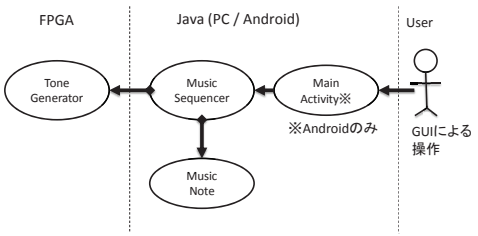

図 5　開発事例の役割分担

```
interface ToneGenerator {
  void keyOn (in long channel, in long frequency, in long velocity);
  void keyOff(in long channel);
};
```

図 6　ToneGenerator のインターフェイス定義（IDL）

度（FPGA システム周波数 100MHz，100Mbps イーサネットを通信として使用）である．但し，今後は FPGA のハードウェア資源を生かした並列処理により，更に応答時間を短縮することを計画している．

3. ORB エンジンを用いた FPGA システム開発事例

本節では ORB エンジンを用いたシステム開発事例を示す．ORB エンジン導入の有効性を示すために，システムの開発目標を「音色シンセサイザのオーディオ処理を FPGA 上に実現し Android 端末から GUI で操作すること」とした．これは，Android 画面のタッチから発音までには数 10ms 程度の即時応答性が要求され，その場での波形合成が必要なアプリだからである．本開発では FPGA 上の音色合成ハードウェアをオブジェクト指向のハードウェアインターフェイスで抽象化し，Android と FPGA を分散オブジェクトのプロトコルによる通信で連携することを主眼に置いた．

システム開発環境は Digilent 社製の Atlys(TM) Spartan-6 FPGA Development Board を用いた．同ボードには AC ’ 97 (Audio Codec 97) 音声入出力のデモが付随して提供されている．これをベースとし ORB エンジンを用いて，FPGA を遠隔メソッド呼出し可能なオブジェクト化する．なお，システム構築の詳細は，別の報告[18] で詳細に示しているため，本稿ではハードウェアの抽象化とプログラムからの扱いに着目し解説する．

3.1 FPGA の役割分担

FPGA と Android の連携システムを設計する際には，2 者の明確かつ適切な役割分担が必要である．本システムにおいては，図 5 に示すように FPGA 側と PC/Android 端末側の役割を分担した．

ToneGenerator は，FPGA 側において発音を担当するオブジェクトであり，図 6 に示すインターフェイス定義（ファイル名 : ToneGenerator.idl）を持つこととした．keyOn は発音を行う操作であり，パラメータとしては同時に発音可能な 3 つのチャンネルから一つを指定 (channel)，周波数を設定 (frequency)，発音の大きさ (打鍵速さ:velocity) を与える．また，keyOff は発音を停止する操作であり，channel のみを指定する．PC/Android 側のオブジェクトは，MusicSequencer, MusicNote, MainActivity の 3 つからなる．MusicNote は自動演奏に必要な音符情報を保持する役割である．また，FPGA 上オブジェクトの操作を行うのは，MusicSequencer であり，Android アプリの中核となる MainActivity からの指示で，ToneGenerator のメソッドを呼び出して発音を制御する．

3.2 システム構成

図 7 は，本開発事例におけるソフトウェア構成図である．前述の IDL ファイルから，FPGA 上サーバおよび Android 側クライアントに使用する CORBA プロトコル処理用の Skeleton ファイル（C 言語）を自動生成し使用した．FPGA 上の ORB エンジンは省リソースの観点から TCP/IP は使用せず，UDP/IP を使用している．そのため，Android 側において TCP パケットを UDP パケットに変換するブリッジを設けた．このブリッジは ORB エンジンのパッケージとともに配布されている．

Android 端末側で GUI によるタッチ入力に応じた遠隔オブジェクト呼び出しを行うために，Android SDK (バージョン 4.0.3) を用いた Java による Android ネイティブアプリケーション開発を行った．図 8 に，開発した Android アプリケーションの画面写真を示す．中央水平に SeekBar を配置し，その上に発音 ON/OFF 切り替えボタン，下には楽曲を自動演奏開始するボタンを配置した．ユーザは SeekBar をタッチすると

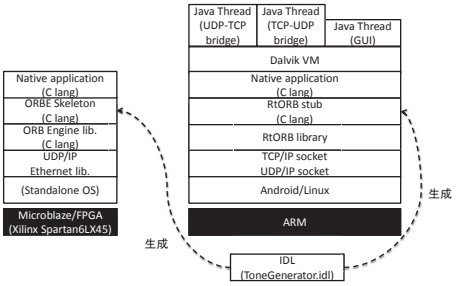

図 7　ソフトウェア構成図

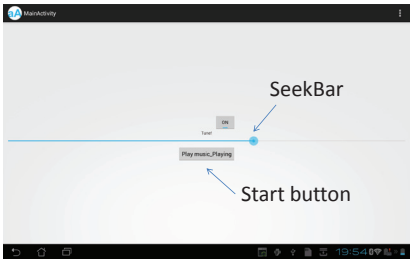

図 8　画面写真

FPGA から矩形波の音声が出力される．画面から指を離すと，音声の出力が止まる．また，左右に動かすとあらかじめ設定した周波数範囲で，矩形波の周波数が変更される．また自動演奏開始ボタンをタップした際に演奏される楽曲は予め Java プログラム内に埋め込んだデータ（MusicNote のリスト）とした．

FPGA 基板側で，CORBA の遠隔メソッド呼出しに応じた音声出力を行うためのハードウェア構成を図 9 に示す．基板は，FPGA の他にメモリおよび入出力用 LSI を複数搭載している．FPGA 内部において，それらの LSI のインターフェイス回路を作成して使用する．ハードウェア開発に使用したツールは，Xilinx 社製 ISE Design Suite Embedded Edition (13.4) である．付属のデモシステムにおいて既に音声入出力が可能となっていたため，本開発においての変更点は，Ethernet 通信用 IP(Xilinx 社製) の追加，割り込み信号の接続，内蔵 BlockRAM の容量変更 (16KB → 64KB) のみであった．

3.3　ハードウェアへのアクセス時間分析結果

開発した Android/FPGA 連携システムにおいて，GUI のタッチ操作からハードウェアへのアクセスにかかる時間を分析した結果を図 10 に示す．分析対象は SeekBar をタッチした際に呼び出される，ToneGenerator オブジェクトの keyOn メソッドの遠隔メソッド呼出しとした．分析結果より keyOn メソッドの呼び出しから 31.5ms 経過後に FPGA 側でのメソッド

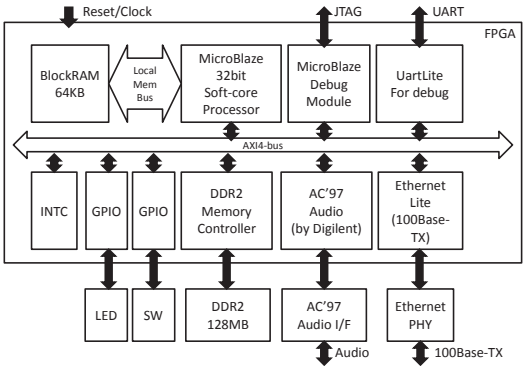

図 9　ハードウェア構成

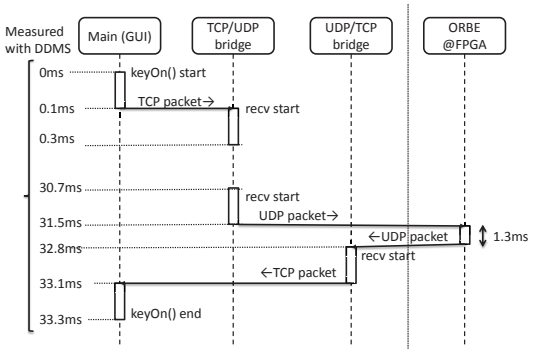

図 10　遠隔メソッド呼び出し時の時間内訳

呼出しがなされ，FPGA 側は 1.3ms で応答を返してから 0.5ms 経過後に keyOn メソッドが終了している事が分かる．

ハードウェアへのアクセス時間という観点でみると UDP パケットを FPGA に対して送信してから応答を受信するまでの 1.3ms が応答時間であるが，そこにたどり着く前の，TCP メッセージを UDP に変換する TCP/UDP ブリッジで時間を消費している．この原因を調査した結果，クライアント TCP パケットが 2 つに分かれて送信されている事が判明した．応答時間の改善のためには，CORBA の実装に手を入れ UDP で直接通信するか，ORB エンジン自体の TCP/IP 通信機能が必要である．

3.4　開発事例についてのまとめ

本開発事例においては，サーバとクライアントの間のインターフェイスを，IDL ファイルを通じて定義することにより，サーバ側の FPGA と，クライアント側の Android アプリケーションを，完全に独立して開発できることが確認できた．

但し性能面でまだ問題を残しており，今後，システム全体で見てハードウェアへのアクセス時間を最適化するような検討が必要であると考える．

4. おわりに

FPGA 上のハードウェアをソフトウェアから扱うためのインターフェイス手法について議論し，今後の FPGA システムの設計生産性向上のための設計手法を提案した．従来のハードウェア抽象化手法について俯瞰的に解説したのちに，大規模・高性能 FPGA 時代に対応したハードウェア抽象化手法として，FPGA 用の CORBA 準拠分散オブジェクトである ORB エンジンについて紹介した．

更に Android アプリケーションから FPGA を操作する実例を示すことで，FPGA を含むシステム開発への CORBA 適用の有効性を示した．

本事例を通じて，サーバ側開発とクライアント側開発を完全に独立して行えることが確認できた．これは CORBA の枠組みで FPGA にオブジェクト指向のインターフェイス定義を行う事が寄与している．すなわち，FPGA の扱いを知らないソフトウェア技術者であっても，ORB エンジンを用いる事で FPGA と密に連携したシステムを構築可能である事を示唆している．今後は，ORB エンジンの性能・機能改善，および他の開発手法との定量的な比較を進める予定である．

参考文献

1) William B. Frakes and Kyo Kang : ”Software Reuse Research: Status and Future,” IEEE Transactions on Software Engineering, vol. 31, no. 7, pp. 529-536, (2005).
2) Object Management Group: ”Common Object Broker Architecture Specification,” CORBA 3.2, http://www.omg.org/spec/CORBA/3.2/, (2011).
3) J. Barba et al., ”OOCE: Object-Oriented Communication Engine for SoC Design,” 10th EUROMICRO Conference on Digital System Design (DSD ’07), (2007).
4) Martin Schoeberl, Stephan Korsholm, Tomas Kalibera and Anders P. Ravn, ”A Hardware Abstraction Layer in Java,” ACM Transactions on Embedded Computing Systems, Vol. V, No. N, Pages 1-42, (2009).
5) P. Dorsey, Xilinx White Paper: Virtex-7 FPGAs, Xilinx, WP380, (2010).
6) Muhammad H. Rais and Mohammed H. Al Mijalli, ”Reconfigurable Implementation of S-Box Using Virtex-5, Virtex-6 and Virtex-7 Based Reduced Residue of Prime Numbers,” World Applied Sciences Journal, 18 (10), pp. 1355-1358, (2012).
7) Wido Kruijtzer et al., ”Industrial IP Integration Flows based on IP-XACT(TM) Standards,” 2008 Design, Automation and Test in Europe, (2008).
8) Michael D. Schroeder and Michael Burrows, ”Performance of Firefly RPC,” SRC RESEARCH REPORT 43, (1989).
9) Angelos Bilas and Edward W. Felten, ”Fast RPC on the SHRIMP virtual memory mapped network interface,” Journal of Parallel and Distributed Computing, Volume 40, Issue 1, 10 January 1997, Pages 138-146, (1997).
10) Brian N. Bershad, Thomas E. Anderson, Edward D. Lazowska and Henry M. Levy, ”Lightweight remote procedure call,” ACM Transactions on Computer Systems (TOCS), Volume 8 Issue 1, Feb. 1990, Pages 37 - 55, (1990).
11) 原功，安藤慶昭，神徳徹雄，末広尚士,“軽量 CORBA RtORB による OpenRTM の実装と評価,”ロボティクス・メカトロニクス講演会講演概要集 2009, ”2A2-D05(1)”-”2A2-D05(3)”, (2009).
12) Dan Davis and Manish Parashar, ”Latency Performance of SOAP Implementations,” 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 407-412, (2002).
13) W3C (MIT, ERCIM, Keio) Recommendation, ”SOAP Version 1.2,” http://www.w3.org/TR/soap12-part0/, (2007).
14) Qu Runtao, Satoshi Hirano, Takeshi Ohkawa, ”Memory Utilization Analysis of Java Middleware for Distributed Real-time and Embedded Systems,” the 18th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2006), pp. 134-140, (2006.11).
15) 大川猛, 平野聡, 久保田貴也,“オブジェクト要求ブローカのハードウェア化によりオブジェクトレベル通信を加速する「ORB エンジン」の提案,”情報処理学会研究報告, 2008-EMB-7(7), pp. 35-40, (2008).
16) 大川猛，戸田賢二,“CORBA/GIOP を用いた FPGA 向けオブジェクト指向プログラミング・テスト環境,”信学技報 IEICE Technical Report，ICD2008-137 (2009-1), pp. 45-50，(2009).
17) ORB Engine Sorceforge Project, https://sourceforge.net/projects/orbe/, (2012).
18) 大川 猛，高野 創司，植竹 大地，横田 隆史，大津 金光，馬場 敬信,“分散オブジェクト ORB エンジンの導入による FPGA 搭載システム連携の短期間開発事例”，デザインガイア 2012 —VLSI 設計の新しい大地— RECONF 研究会にて発表予定, 信学技報, (2012).