社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2004－ＳＬＤＭ－113　（８）
2004／1／22

動的再構成プロセッサDRPを用いた関数オフローダの試作

犬尾 武† 西野 賢悟† 中谷 正吾† 梶原 信樹†
紙 弘和‡ 戸井 崇雄‡ 粟島 亨‡

†NEC シリコンシステム研究所 〒229-1198 神奈川県相模原市下九沢 1120
‡NEC マルチメディア研究所 〒211-8668 神奈川県川崎市中原区下沼部 1753
E-mail: †{inuo, nishino, nakaya, kajihara}@mel.cl.nec.co.jp, ‡{kami, toi, awash}@ccm.cl.nec.co.jp

あらまし　応用の一部を関数単位で切り出し、DRPへオフロードするためのツールを紹介する。DRPとソフトウェア間のインタフェースを自動生成し、実行時にDRPのコンテキストを仮想化して制御することで、DRPのリソースに制約されない関数のオフロード、及びその初期評価が可能となる。動作時のプロファイリング結果に基づいてオフロード関数のCソースコードを繰り返し最適化することが可能になり、アプリケーション開発を加速する。
キーワード　SW/HW協調検証、C言語ベース設計、仮想エミュレーション

Function Offloader for DRP

Takeshi INUO† Kengo NISHINO† Shogo NAKAYA† Nobuki Kajihara†
Hirokazu KAMI‡ Takao TOI‡ Toru AWASHIMA‡

†NEC Silicon Systems Research Laboratories, 1120, Shimokuzawa, Sagamihara, Kanagawa, 229-1198, Japan
‡NEC Multimedia Research Laboratories, 1753, Shimono Numabe, Nakahara, Kawasaki, 221-8668, Japan
E-mail: †{inuo, nishino, nakaya, kajihara}@mel.cl.nec.co.jp, ‡{kami, toi, awash}@ccm.cl.nec.co.jp

Abstract　We propose and discuss application development tools which offload functions of application program to DRP. Those tools generate interfaces between DRP (hardware) and CPU (software), and generate the virtual Context controller (software) of DRP. Therefore, it is possible to evaluate the availability of the offload function rapidly without the constraint of DRP resources. The application could be developed by optimizing C source code based on its profiling results.
Keyword　SW/HW co-verification, C-based design, virtual emulation

1. はじめに

我々は、NECエレクトロニクス社で開発された動的再構成プロセッサ(以下DRP)[1]とCPUとを結合したアーキテクチャ、及び弊社の動作合成ツールcyberをベースとしたDRP開発ツール(以下、DRPコンパイラ)を用いた、DRPの開発手法の研究を行っている。

近年、プロセスの微細化によりLSIに搭載可能なゲート数が増大しているにもかかわらず、LSIの設計生産性がそれに追いつかないという問題[2]があり、短TATなLSI設計には設計のレイヤを上げることが必須である。ソフトウェア設計者やアルゴリズム開発者が専ら使用するCベースの言語を用いて、システムやアプリケーションの設計を行い、プロファイル結果に基づいてSWとHWに分割してCPUと専用HWもしくはFPGAといったデバイスに実装するSW/HW協調設計が行われている[3][4][5]。特に従来より行われてきたVHDLやVerilog HDL等のハードウェア記述言語によるRTL設計は、動作合成ツールの登場により、より抽象度の高い言語で設計可能となり、その性能も十分に実用レベルに達している[6]。

一方、再構成可能なデバイスとCPUを組み合わせたアーキテクチャの研究は古くから行われており[7][8]、近年ではExcalibur (Altera社)[9]、Virtex-II pro (Xilinx社)[10]、ACM (QuickSilver社)[11]、DAP/DNA (IPFlex社)[12]などが発表され、実用的なレベルで普及してきている。

さて、C言語ベースで記述されたアプリケーションの一部をハードウェアへオフロードする手法は、おおむね以下のようなステップで行われる。

A) プロファイラ等を用いて、オフロードする箇所(CPUにとって高負荷な箇所)を特定する。
B) SWとHW間のインタフェースを定める。
C) HWの動作モデルをSWに組み込む。

D) SW の全体検証を行う。
E) HW を詳細設計する。
F) HW 動作モデルを置き換える。
G) SW の全体の動作検証・性能検証を行う。

しかし、このような設計手法では、以下のような問題が発生する。

- HW にオフロードする箇所や、SW と HW 間のインタフェースの種類で SW の仕様が変化する。
- オフロードしようとした箇所が HW に入りきらない。

後者は HW への実装を最適化することで回避できる可能性があるが、オフロードする箇所の選択時(初期評価時)にこの問題が発生すると初期評価のために実装最適化を行わなくてはならず、設計時間を増加させる原因となる。

そこで我々は、DRP とローカルメモリからなる最も単純な HW アーキテクチャを対象とすることで SW 側の仕様を固定し、その HW の動作モデルを DRP のエミュレーションで実現して HW/SW の全体検証を可能にするプロトタイプツール(関数オフローダ)を開発した。

本報告では、DRP の概要、関数オフローダの動作概要、DRP のコンテキストを仮想化したエミュレーション方法、関数オフローダの適用例と拡張例、まとめと今後の課題の順で説明する。

なお本報告では、分割前のソースコードをオリジナルコード、分割後の SW 側のコードを SW コード、分割後の HW 側のコードを HW コード、結合後のコードを統合コードと呼ぶ。

2. 動的再構成プロセッサ DRP

DRP はプログラム可能なプロセッサエレメント(PE)とユーザメモリをアレイ状に並べた IP コアであり、データ処理(データパス)を PE のアレイで実現し、データ制御を状態遷移コントローラ(STC)で実現する。図 1 に DRP コアの基本構成を示す。

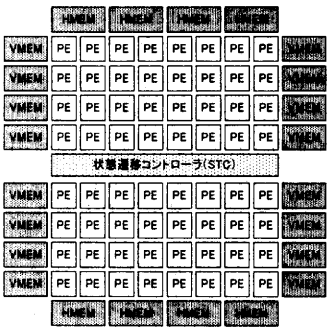

図 1 DRP コアの基本構成

DRP コンパイラは、DRP 単体で動作するアプリケーションを開発するツールである。図 2 に示すように、DRP コンパイラでは、C で記述されたアプリケーションを状態遷移マシン(FSM)とデータパスに分割して合成し、それぞれ STC コードと PE/メモリアレイコードとして出力する。

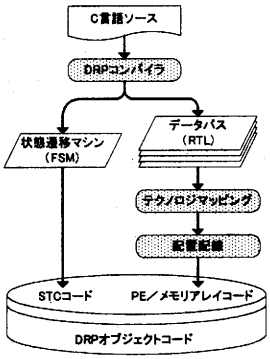

図 2 DRP コンパイラ

これらの DRP オブジェクトコードを DRP に書き込み、動作を開始させると、FSM ごとに分割された複数の回路構成を記憶した PE アレイは、STC から出力された制御信号によって、記憶している処理の中の 1 つを選択してデータ処理を行う。以下、STC によって選択されるデータ処理をコンテキストと呼ぶ。図 3 に DRP の動作を模式的に示す。

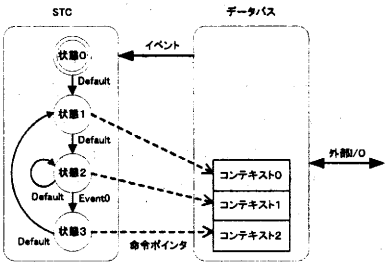

図 3 DRP の動作模式図

この DRP コアを内蔵したプロトタイプチップ DRP-1 は、512 個の PE と 2M ビットのメモリを搭載し、11~133MHz で動作する。各 PE は 16 個のコンテキストを有する。図 4 は DRP-1 を実装した DRP 評価ボードの写真である。

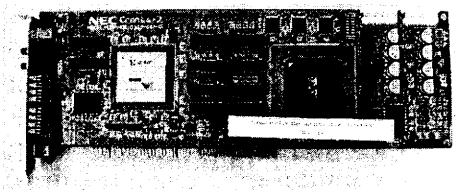

図 4 DRP 評価ボード

3. 関数オフローダ

SW と HW を分割する場合には、分割した HW が分割前の SW に対して、どのくらい性能等の向上があるかを知るために初期評価を行わなければならない。また、分割した HW コードはオリジナルコードと全く同じ動作をすることを確認できなければならない。

そこでオフロードする HW をオリジナルコード内の関数に限定して、図 5 に示すようなフローで HW/SW を協調して開発する手法を検討した。

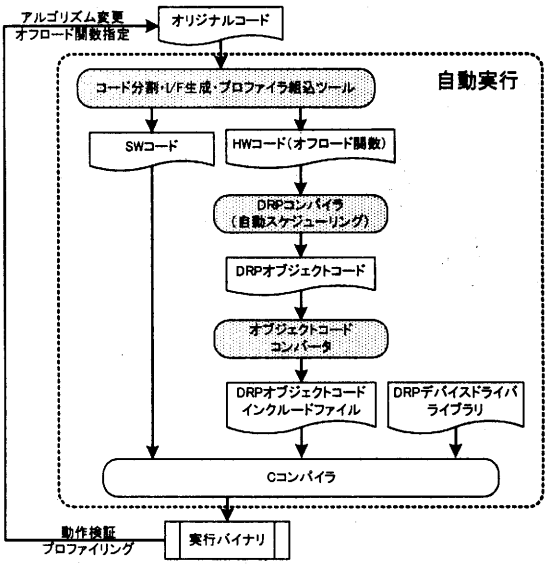

図 5　開発フロー

オリジナルコード中でオフロードする関数を指定するだけで、自動的に統合コードの実行バイナリを生成する。そのバイナリを実行することで分割した HW を含めた HW/SW 全体の動作検証と、分割した HW のプロファイリングを同時に行う。これにより、迅速な初期評価および動作検証が可能になる。

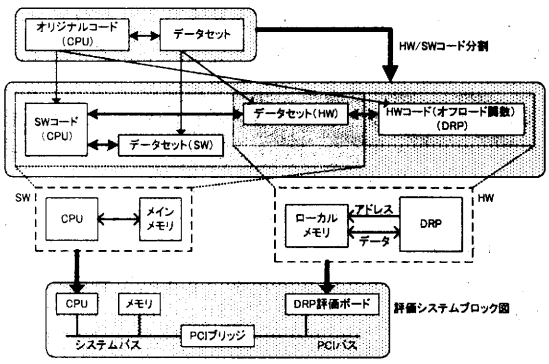

図 6　HW/SW コード分割

3.1. HW/SW コード分割

コード分割と実装イメージを図 6 に示す。関数単位の分割に加えて、オリジナルコードが扱うデータセットのうち、オフロード関数が使用するデータセットも分割する。関数のオフロード先の HW アーキテクチャとして、図 6 に示すような、最も単純なメモリ共有型のアーキテクチャを DRP 評価ボードで実現する。

3.2. データセットと SW-HW 間インタフェース仕様

ある関数とその外部とのデータのやりとりは以下の 3 種類である。

- 引数
- 戻り値
- 大域変数

SW コードで扱っているこれらのデータセットをオフロード先のローカルメモリ上のアドレスにリマップする。このアドレス変換テーブルが SW と HW 間のインタフェース仕様となる。本関数オフローダでは上記データセットを一次元配列化している。

メモリ共有型インタフェースにより、DRP と CPU 間のデータ共有が簡単に実現でき、DRP コンパイラによるメモリアクセスの自動スケジューリングの恩恵を受けることができる。しかし、DRP が CPU から受け取るパラメータが増えると、メモリアクセスをするための DRP の状態数が増える傾向にある。これは、例えば DRP が連続でアクセスできるようにアドレス変換テーブルやアルゴリズム/データ構造を修正することで改善することが可能である。

3.3. HW コード生成

前節のアドレス変換の仕様を反映した HW コードが生成される。関数本体の処理の他にデータセットを授受するためのコードが追加・置換される。

3.4. SW コード生成

SW コードは、オフロードする関数以外はオリジナルコードのままである。SW コードのオフロードする関数内のコードは、データセットを DRP と授受するためのコード(データ授受コード)、及び DRP を制御するためのコード(DRP 制御コード)に置き換えられる。

DRP の動作モードには通常モードとデバッグモードの 2 種類がある。DRP をデバッグモードで動作させた場合には 1 クロックサイクル毎に DRP は停止する。

そこで、SW コードでは以下の 2 種類の方法で DRP を制御する。

- エミュレーションモード
- リアルモード

エミュレーションモード(図 7)では、DRP のデバッグ機能を利用して、DRP をサイクルアキュレートで制御する。DRP の出力ピン(アドレス、read/write 信号、write データ等)を監視し、その値に対する応答(read データ)を DRP の入力ピンに与えることで DRP から見た外部 I/O(ローカルメモリ)をソフトウェアでエミュレートする。

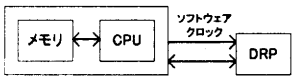

図 7　エミュレーションモード

リアルモード(図 8)では、DRP を通常動作させる。DRP と同期したメモリを DRP の外部に接続し、そのメモリをソフトウェアで管理する。

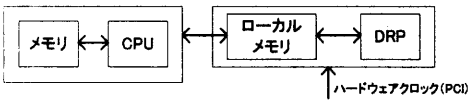

図 8　リアルモード

3.5. 動作概要

3.5.1. エミュレーションモード

エミュレーションモードでは、DRP はステップ実行で制御し、DRP の I/O はソフトウェアでエミュレートするため、SW コードの DRP 制御コード内に直接アドレス変換のためのコードを埋め込む。本関数オフローダでは、コード分割時にこの作業を行う。

図 9 に動作概要を示す。SW コードでオフロード関数が call されると、まず DRP をオープンし、DRP の出力値をサンプリングする。メモリライト要求の場合はデータセットを更新し、動作終了要求の場合は、DRP をクローズして return する。動作終了要求以外の場合は、DRP をステップ実行する。メモリリード要求であった場合には、ステップ実行後に、対応するデータを DRP のリードデータ入力ピンに与える。ステップ実行後は再び DRP の出力値のサンプリングから同じ動作を繰り返す。

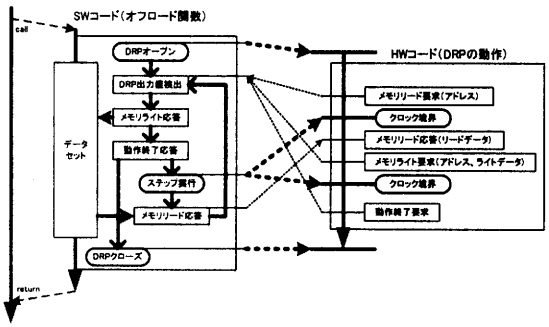

図 9　エミュレーションモードの動作概要

3.5.2. リアルモード

リアルモードでは、DRP は通常動作し、DRP のメモリアクセスは DRP と同期したローカルメモリに対して行うため、SW コードのオフロード関数の前後にローカルメモリへアクセスするコードを追加する。

図 10 に動作概要を示す。SW コードでオフロード関数が call されると、データセットをローカルメモリにダウンロードして、DRP をオープンする。その後 DRP の終了要求をポーリングで待ち、DRP をクローズする。その後、ローカルメモリからデータセットを取り出し、アップロードする。

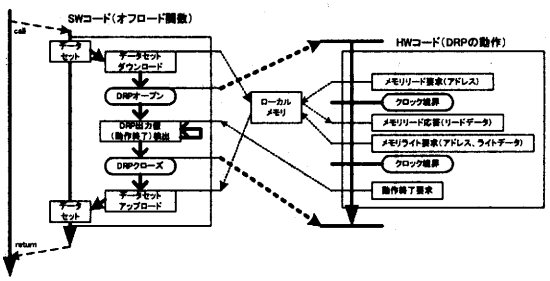

図 10　リアルモードの動作概要

3.6. 実行バイナリ生成

HW コードは、DRP コンパイラでコンパイルされる。従って、オフロードする関数は DRP コンパイラでコンパイルできるコードである必要がある。DRP コンパイラでコンパイルできるコードには、クロック境界を記述したサイクルアキュレートな手動スケジューリング(cyberI)向けのコードとクロック境界を記述しない自動スケジューリング(cyberII)向けのコードの 2 種類がある。通常、オリジナルコードにはクロック境界は記述されていないため、本関数オフローダでは自動スケジューリング向けコードとして DRP コンパイラに渡し、DRP オブジェクトコードを生成する。

なお、DRP の統合開発環境を用いることで HW コードから DRP オブジェクトコードを生成することができる。従って、HW-SW 間インタフェース仕様を保ったまま、HW コードを手動スケジューリングで記述し直すことで更なる最適化が可能である。

DRP オブジェクトコードは、本関数オフローダのオブジェクトコードコンバータによって SW コードへインクルード可能なファイルに変換される。

SW コードを C コンパイラでコンパイルすることで、上記 DRP オブジェクトコードと DRP 評価ボードのデバイスドライブラリがインクルードされて実行バイナリが生成される。

3.7. プロファイリング

エミュレーションモードでは、DRPをサイクルアキュレートでI/Oを含めて制御するため、クロックサイクル毎のDRPの動作情報を記録することができる。本関数オフローダでは、以下の項目がクロックサイクル毎にログとして記録される。

- DRPからローカルメモリへのI/Oアクセス
- 実行した状態番号と発生したイベント
- 各コンテキストの実行ステップ回数　等

なお、リアルモードではこれらのプロファイリングは行えない。DRPのクロックサイクル毎の詳細な動作検証を行うにはエミュレーションモードで動作させる必要がある。

4. DRPのコンテキストの仮想化

関数オフローダを用いることで、容易な初期評価が可能となる。しかしながらDRP評価ボード上で実行するためには、使用するコンテキストを16以下に抑える必要があり、すべてを自動化することができない。そこでDRPのコンテキストを仮想化して、この問題を解決する。エミュレーションモードではDRPをステップ実行で制御するため、ここに仮想化の機能を追加する。

4.1. 制御概要

図11にコンテキストを仮想化するための制御の模式図を示す。以下、DRPコンパイラで生成したオブジェクトコードを論理STC及び論理コンテキストと呼び、DRP内に格納しているSTC及びコンテキストを物理STC及び物理コンテキストと呼ぶ。

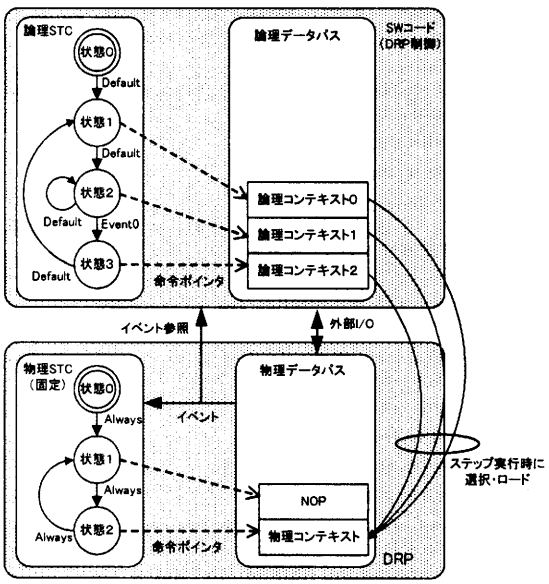

図11　コンテキスト仮想化のための制御

コンテキストを仮想化するために、論理STCの代わりに、図11で示している固定の物理STCをDRPに書き込む。この物理STCはイベントに関係なく遷移し、状態1では何も動作しないコンテキスト(NOP)を指す。そして状態2の命令ポインタが指す物理コンテキストに、論理コンテキストをステップ実行毎に上書きする。上書きする論理コンテキストは現在の状態番号、および物理データパスから出力されるイベント(DRP外部から参照可能)で論理STCを参照して決定する。

4.2. コンテキストキャッシュ型の仮想化制御

前節の仮想化方法では、ステップ実行の度に物理コンテキストを上書きする必要があるため、エミュレーション速度が遅い。

そこで図12に示すような改良を加えて、DRP内にコンテキストのキャッシュ機能を追加した。

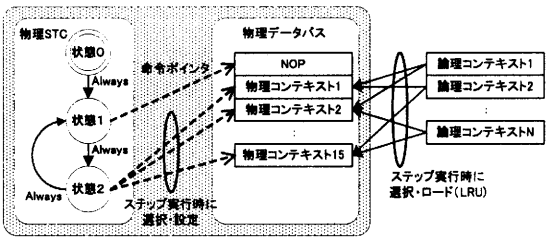

図12　コンテキストキャッシュ型仮想化制御

コンテキストキャッシュ型の仮想化制御では、物理STCの命令ポインタをステップ実行毎に更新する。遷移先の論理コンテキストがすでに物理コンテキストに存在する場合は物理STCの状態2の命令ポインタのみを更新し、含まれない場合は物理コンテキストをLRU(Least Recently Used)アルゴリズムで選択して論理コンテキストで上書きし、同時に命令ポインタも更新する。これによって、使用頻度の高い15コンテキストがDRP内にキャッシュされる。この改良により、物理コンテキストの上書き回数が削減され、エミュレーション速度が向上する。

また、コンテキストを仮想化して制御する場合の実行時プロファイルでは、論理状態番号及び論理コンテキスト番号でログが記録される。加えて、論理コンテキストを書き込んだ物理コンテキストの番号や書き込んだデータ量、物理STCの更新情報等も記録される。

5. 適用例

画像認識処理で用いられる特徴ベクトル計算[13]に関数オフローダを適用した。特徴ベクトル計算は、画像の各画素とその周囲の8画素との間で最大3次の積を計算し総計を求める。ベクトルは35次元の値となる。

Cの関数として記述された32x32画素に対する特徴ベクトル計算を、本関数オフローダを用いてDRPへオフロードした。表1に全く手を加えずにオフロードした結果と、C言語レベルで最適化した結果を示す。いずれも自動スケジューリングでコンパイルした。

表1　関数オフローダ適用結果

種類	コンテキスト総数	状態総数	入出力状態数	最大PE数(*1)	PE総数	メモリブロック数	乗算器数	動作周波数(MHz)	実行ステップ数	処理時間(us)
初期評価	69	70	17	166	5388	12	8	43.6	47000	1078
最適化後	14	15	10	226	1325	8	8	18.5	11000	595

*1：コンテキスト当りのPE数の最大値

表1において、コンテキスト総数から動作周波数まではDRPコンパイラが出力したコンパイル結果であり、実行ステップ数はプロファイラから取得したものである。処理時間は動作周波数と実行ステップ数から換算した。

手を加えなかった場合(初期評価)では、コンテキスト数が69となってしまい、これまでは評価できなかった。しかし本関数オフローダを用いて、コンテキストを仮想化したエミュレーションモードで動作させることで、正常動作の確認が可能となった。また、図5で示した開発フローを繰り返してアプリケーションをある程度最適化した結果、コンテキスト数を14まで抑えることができ、DRP評価ボード上でリアルモードで動作させることができた。

処理に必要なPE数は時間的(各コンテキストで使用するPE数の合計)に見ると5388個から1325個に減っており、空間的(PEを最も多く使用しているコンテキストにおけるPE数)に見ると166個から226個に増えている。これはコンテキスト内の使用率が上がったことを意味しており、そのために動作周波数が下がっている。本報告では、処理時間の加速が主眼ではないため動作周波数に関する最適化は行っていないが、DRPコンパイラの配置配線等を人手で行うことで動作周波数を大きく改善することが可能である。

6. マルチファンクションオフローダ

以上、オリジナルコード内の1つの関数に対するオフロード手法を紹介したが、本関数オフローダは複数の関数のオフロードにも容易に適用可能である。リアルモードで動作させるには、オフロードする関数が個々に使用するコンテキスト数の合計が16以下になるように最適化すればよい。エミュレーションモードに関してはそのまま適用可能である。

7. おわりに

本関数オフローダを用いることで、HW/SW協調動作検証が可能となる。特にエミュレーションモードを用いることで、HWコードのクロックサイクル毎の詳細な動作検証が可能となる。さらに、コンテキストを仮想化する手法を用いることで、最適化を施していないHWコードを、実装するHWのリソースに制約されずに動作検証することができ、実行時のプロファイリングによって迅速な初期評価が可能となる。

今後は、本関数オフローダをCPUアクセラレータとして適用する検討を行う。DRPをCPUアクセラレータとして使用するためには、データセットの転送オーバヘッドを考慮する必要がある。また、DRPはリアルモードで動作させる必要があり、現状ではコンテキスト数は16以内に制限される。

転送オーバヘッドに関しては、データセットのプリロード/ポストストアを関数の境界を超えてスケジューリングする手法を検討している。コンテキスト数制約に関しては、リアルモードで動作するコンテキストの仮想化手法を検討している。これらの課題を解決し、本関数オフローダをより実用的なアプリケーション開発手段として展開する予定である。

謝辞

本研究に当たって、NECエレクトロニクス株式会社の本村真人氏、古田浩一朗氏、安生健一朗氏、藤井太郎氏をはじめ多くの方々に多大なご協力を頂いたことに深く感謝します。

文　献

[1] M.Motomura, "A Dynamically Reconfigurable Processor Architecture," Microprocessor Forum, Oct.2002.

[2] "The National Technology Roadmap for Semiconductors: Technology Needs," 1997 Edition, Semiconductor Industry Association, 1997.

[3] http://www.mepcore.com/

[4] http://www.tensilica.com/

[5] http://www.celoxica.com/

[6] 大山他, "C言語ベースのシステムレベル合成・検証手法と開発事例" 第13回軽井沢ワークショップ, 2000.

[7] John R. Hauser, John Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor," Proc. of FCCM, 1997.

[8] Elliot Waingold, et al., "Baring it all to Software: Raw Machines," IEEE Computer, pp. 86-93, September 1997.

[9] http://www.altera.com/

[10] http://www.xilinx.com/

[11] http://www.quicksilvertech.com/

[12] http://www.ipflex.com/

[13] F.Goudail, et al., "Face recognition system using local autocorrelations and multiscale integration," IEEE TRans. Pattern Analysis and Machine Intelligence, vol.18, no.10, 1996.