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

2004－ＳＬＤＭ－113 （５）
2004／1／22

招待論文 動的再構成可能チップDRPのCコンパイラ

粟島 亨† 戸井 崇雄† 中村 典嗣† 紙 弘和† 加藤 吉之介† 若林 一敏†
宮澤 義幸‡ 李 京‡

†NEC マルチメディア研究所 〒211-8666 川崎市中原区下沼部 1753
‡NEC 情報システムズ 〒213-0012 川崎市高津区坂戸 3-2-1 (かながわサイエンスパーク)
E-mail: †{awash, toi, nnakamur, kami, y-kato, wakaba}@cad.cl.nec.co.jp, ‡{miyazawa, lijing}@ats.nis.nec.co.jp

あらまし 動的再構成可能プロセッサDRP(Dynamically Reconfigurable Processor)のCコンパイル環境について報告する．C言語ベースの動作合成エンジンをフロントエンドとすることでDRPに対するソフトウエアライクな開発環境を実現した．C言語の動作記述から自動スケジューラにより制御回路(FSM)とデータパス回路が合成される．制御回路はDRPの状態遷移コントローラー(STC)にマッピングされ，データパス回路は複数のコンテキストに分割された上でPEアレイにマッピングされる．フロントエンド合成とバックエンド合成は統合開発環境により密に統合され，直観的なGUIが提供される．実チップ上のシンボリックデバッグが可能なオンチップ・デバッガも備えた．
キーワード リコンフィギュラブルプロセッサ，動的再構成，動作合成，コンパイラ

C Compiler for Dynamically Reconfigurable Processor: DRP

Toru AWASHIMA† Takao TOI† Noritsugu NAKAMURA† Hirokazu KAMI†
Yoshinosuke KATO† Kazutoshi WAKABAYASHI† Yoshiyuki MIYAZAWA‡ and Li JING‡

†Multimedia Research Laboratories, NEC Corp.1753, Shimonumabe, Nakahara-ku, Kawasaki, 211-8666, Japan
‡NEC Informatec Systems, Ltd. 2-1, Sakado 3-Chome, Takatsu-ku, Kawasaki, 213-0012, Japan
E-mail: †{awash, toi, nnakamur, kami, y-kato, wakaba}@cad.cl.nec.co.jp, ‡{miyazawa, lijing}@ats.nis.nec.co.jp

Abstract A C Compiler for Dynamically Reconfigurable Processor: DRP is presented. Software like design environment is realized by using our C-based behavior synthesizer as a front-end of the compiler. Finite state machine (FSM) and data paths are generated from C-based behavior description and directly mapped onto State Transition Controller (STC) and multiple context PE arrays of DRP. An integrated development environment (IDE) is provided to support intuitive operations. A symbolic debugger for the actual chip is also provided.
Keyword Reconfigurable Processor, Dynamic Reconfiguration, Behavior Synthesis, Compiler

1. はじめに

NECエレクトロニクスの開発したDRP(Dynamically Reconfigurable Processor)[1]は，複数の回路構成を瞬時に切り替える動的再構成を特徴とするプログラマブルデバイスである．DRPは動的再構成により『並列処理による高スループット化』と，『逐次処理による小面積化』，のトレードオフバランスをとることが可能なアーキテクチャに基づいており，専用ハードウエアと汎用プロセッサのギャップを補完し得るデバイスとして注目に値する．しかしながら，DRPアーキテクチャの可能性を十分に引き出すには，効率の高い開発環境およびコンパイル技術の確立が不可欠となる．

我々は，DRPのアーキテクチャをそれほど意識しなくても，ソフトウエアライクな手法でアプリケーションを開発することのできる効率の高い開発環境の構築に取り組んでいる．代表的なプログラマブルデバイスであるFPGAに対する開発環境はHDLによるRTL記述をエントリとするASICの開発フローに準ずるものであり，ソフトウエア並みの開発効率を確保することは難しい．これに対し，動作合成エンジンをベースとしたフロントエンド合成により，DRPに対するC言語ベースのコンパイル環境を構築することに成功した．フロントエンド，バックエンドの各合成段階は統合開発環境により有機的に統合され，実チップに対するシン

ボリックデバッガを備えるに至った．次節以降で本成果について報告する．

本報告の構成は以下の通りである．まず，動的再構成可能プロセッサ DRP の特徴について整理する．引き続き，DRP コンパイラの基本コンセプト，コンパイルフロー，フロントエンド合成，バックエンド合成，そして統合開発環境・オンチップデバッガについて述べる．まとめとして，予定される機能拡張などについても言及する．

2. 動的再構成可能プロセッサ DRP

DRP(Dynamically Reconfigurable Processor)[1]は NEC エレクトロニクスが開発したプログラマブルデバイスである．粗粒度 PE(Processor Element)アレイアーキテクチャ，マルチコンテキスト型の動的再構成機構，などの特徴をもつ．図 1に DRP の基本構成となる DRP タイルを示す．DRP タイルは 64 個の PE アレイ，中央部の状態遷移コントローラ(STC)，周囲に拡散配置されたデータメモリにより構成される．

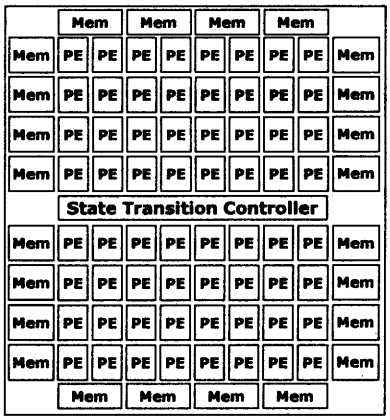


図 1　DRP タイルの構成[1]

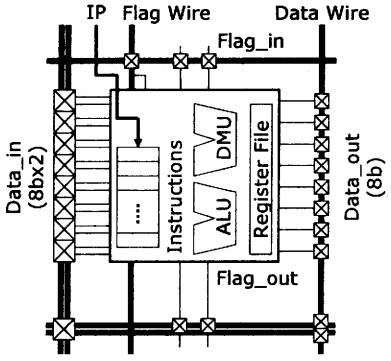


図 2　PE(Processor Element)の構成[1]

DRP タイルを縦横に配置することでスケーラブルな PE アレイが構成できる．プロトタイプチップ DRP-1 は DRP タイルを 2 行 4 列の 8 タイル配列したチップであり 512PE が搭載されている．DRP-1 はチップ内部に最大 16 のコンテキスト情報を保持するマルチコンテキスト機構を持つ．コンテキスト情報は各 PE の命令と PE 間の配線情報（データ転送路設定）により構成される．コンテキストは STC の制御により瞬時に切り替え可能であり時分割多重的な動作が可能となる．マルチコンテキスト機構をいかに活用するかはコンパイラに求められる重要なポイントのひとつであろう．

図 2に演算の基本要素である PE の構成を示す．演算粒度は 8bit を基本とするが，8bit をこえる多 bit の演算も複数の PE の組み合わせで実現できる．また，1 bit 粒度の演算を行う命令やリソースも用意されている．PE は 2 種類の演算ユニット(ALU, DMU)と命令メモリ，バスセレクタ，レジスタファイルなどから構成される．DMU(Data Management Unit)はシフト/マスクなどのビット操作を強化した演算ユニットである．

3. 基本コンセプト

DRP コンパイラのベースとなる技術は一般的なコンパイラ技術というより，むしろハードウエア合成技術に基づくものである．一方，DRP のようなリコンフィギュラブルプロセッサが必要とされる背景として性能とプログラマビリティの両立があり，この視点からソフトウエアライクなコンパイル環境は必須のものといえるだろう．HDL やブロック図入力によるハードウエア的なフロントエンドではソフトウエアプログラマが抵抗なく使えるものとはならない．DRP コンパイラはこのような立場から，C 言語ベースの動作合成エンジンをフロントエンドとし，マッピング，自動配置・配線といった合成エンジンを有機的に統合し，ソフトウエアプログラマに対して違和感のない設計環境を実現することを基本コンセプトとしている．後述するグラフィカルな統合開発環境やオンチップ・デバッガもこのコンセプトに沿ったものである．

4. コンパイルフロー

図 3に DRP のコンパイルフローを示す．DRP コンパイラは大きくフロントエンド合成部とバックエンド合成部にわけられる．フロントエンド合成部は動作合成と呼ばれる合成段階であり，ASIC に対して運用実績のある動作合成エンジン[2]をベースに開発した．動作合成部は C 言語[1]を入力とし，スケジューリング，リソース割り当てを行い，中間コードとして RTL 記述を出

[1]正確には BDL(Behavior Design Language)と呼ばれる動作記述言語を用いている．BDL は入出力の宣言など，C 言語をハードウエア記述用に拡張した言語である．[2]

力する．バックエンド合成部は，マッピング，配置配線，オブジェクトコード生成からなる合成段階である．

動作合成部の出力する RTL は制御回路である FSM(Finite State Machine)とデータパス回路に分かれており，制御部＝状態遷移コントローラ(STC)とデータパス部＝PE アレイが分離された DRP アーキテクチャとよく整合する．フロントエンド合成部の出力する中間コードはバックエンド合成を経て，制御回路は STC に，データパス回路は PE アレイに割り当てられる．

5. フロントエンド合成

フロントエンド合成はＣ言語による動作記述を入力とする動作合成と呼ばれる合成段階である．ここでは DRP 特有の要件，すなわち，状態に基づくコンテキスト分割，DRP 特有の合成制約，リソース共有の考え方，について述べる．

5.1. 動作合成

Ｃ言語による動作記述に対しデータフロー解析，スケジューリング，リソース割り当てを行い，制御回路(FSM)とデータパス回路を出力する．状態分割は，データ依存性，リソース制約，クロック制約を満たし，かつ並列性の最大化，処理サイクル数の最小化を目的に実行される．DRP は STC と PE アレイで構成されており，動作合成の出力形式との親和性は高い．制御回路は STC に，データパス回路は PE アレイにマッピングされる．

DRP コンパイラではデータパス回路をコンテキストに分割する戦略がポイントとなる．動作合成の生成する回路の『状態』と DRP 上の『コンテキスト』の対応を適切に割り当てることで最適なコンテキスト分割を実現する．これにともない，動作合成におけるリソース共有の扱いを拡張する必要が生じる．

なお，以下の説明は DRP コンパイラの自動スケジューリングモードに関するものである．DRP コンパイラはクロック境界を明示して合成を行う手動スケジューリングモードもサポートするが，紙面の都合で本報告の対象とはしない．また，DRP コンパイラの中間コードは Verilog RTL 形式のもので，HDL シミュレータによるシミュレーションが可能であることを付記しておく．DRP コンパイラは後述するオンチップ・デバッガによる実機検証を強く意図するものだが，必要に応じてこの中間コードを利用したサイクル精度のシミュレーションを行うことも可能である．

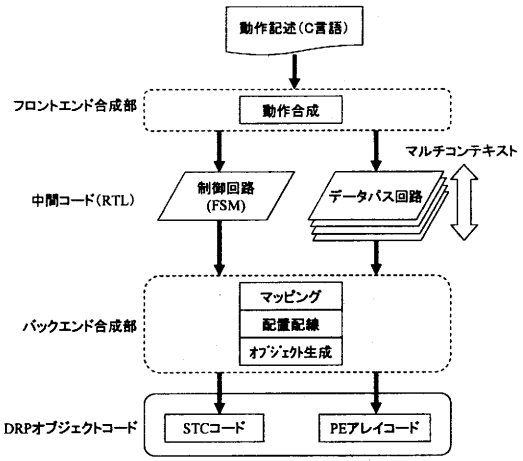
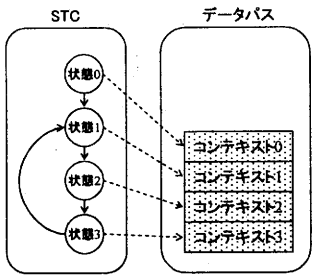


図 3　DRP コンパイルフロー

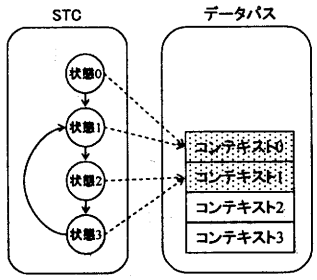
5.2. 状態に基づくコンテキスト分割

動作合成の結果，データパス回路は適当な状態に分割される．ある状態が活性化されたとき，その状態に含まれるデータパス回路も活性となり，それ以外のデータパス回路は非活性となる．マルチコンテキスト機構では，活性化されたコンテキスト上のデータパス回路が有効となり，それ以外のコンテキスト上のデータパス回路は命令メモリに格納されている(隠蔽されている)．そこで，動作合成の結果得られる『状態』の概念と DRP 上の『コンテキスト』という概念の対応付けを行い，マルチコンテキスト分割を実現する．図 4 に状態とコンテキストの対応関係を模式的に示す．状態とコンテキストを 1 対 1 対応の関係で割り当てる，『1 状態→1 コンテキストマッピング』と，多対 1 で割り当てる，『多状態→1 コンテキストマッピング』をサポートする．

『多状態→1 コンテキストマッピング』が必要となる理由は以下の通りである．後述する合成制約のうちコンテキストリソース制約により，状態あたりのリソース数は制限できるが，メモリアクセスなど処理の逐次性により比較的少ないリソース数の状態が生じることがある．一般の動作合成では，回路全体が常に物理的に存在するため，回路全体のサイズが問題になることはあっても，状態毎の回路サイズが問題になることはあまりない．一方 DRP の場合，必要なサイズは，最大のコンテキストサイズにより決定され，少ないリソース数の状態があった場合，その状態に対応するコンテキストのリソース使用率が低くなり全体として効率が落ちてしまう．これをカバーするのが『多状態→1 コンテキストマッピング』モデルであり，これによりコンテキストサイズのバランスが改善され全体的なリソース使用効率が向上する．



(a) 1状態→1コンテキストマッピング



(b) 多状態→1コンテキストマッピング

図 4 状態に基づくコンテキスト分割

5.3. DRP 特有の合成制約

動作合成の基本はデータ依存性の解析に基づき入力記述の並列性を最大限抽出し，処理のサイクル数を最小化することである．これに加え DRP コンパイラでは以下の制約，すなわち，(1)コンテキストリソース制約，(2)クロック制約(最大遅延制約)を考慮する．

5.4. コンテキストリソース制約

1状態1コンテキストマッピングでは，ある状態に割り当て得るリソース量の最大はコンテキストに含まれるリソース量で制約される．例えば DRP タイル1つあたりの PE 数制約は，PE 数≦64，となる．

5.4.1. クロック制約(最大遅延制約)

マルチコンテキストを想定した場合，DRP の動作周波数は全コンテキスト中のクリティカルパス遅延の最大値によって制限される．動作合成では，PE アレイ上での遅延 (PE の演算遅延＋PE 間の配線遅延)を予測し，これが指定したクロック制約を満たさない場合は状態を分割する．

5.5. リソース共有の考え方

状態間のリソース共有として，演算の共有とレジスタの共有について考える．通常の動作合成では，異なる状態に属する演算には排他性(同時に実行されない)があることから，マルチプレクサによる切り替えによって同一の演算器を共有し，演算リソースの有効利用をはかる[2]．一方，レジスタについては，(1)排他性に基づくレジスタ共有と，(2)状態間のデータ授受のためのレジスタ共有，があることに注意を要する．(1)は，生存区間に重なりがない変数同士は同一のレジスタを共有することができる，というものである．(2)は，レジスタを介して互いにデータ授受を行う状態は常にレジスタ共有を行う必要が生じることを示している．DRP の場合はマルチコンテキスト機構により通常の場合と異なるリソース共有の考え方が必要になる．以下に DRP コンパイラにおける，PE 共有，レジスタ共有の考え方についてそれぞれ説明する．

5.5.1. PE 共有の考え方

DRP コンパイラでは，状態がコンテキストに対応するため，排他関係にある演算は異なるコンテキスト上の PE に割り当てられている．したがって，マルチプレクサの挿入による明示的な演算共有は必要なく，演算共有はコンテキストの切り替えにより自然に実現される．後述する自動配置により結果的に同一 PE の異なるコンテキストに割り当てられた命令が当該 PE を共有することになるのである．

5.5.2. レジスタ共有の考え方

(1)排他性に基づくレジスタ共有については，PE 共有の場合と同様，明示的な共有は行わない．DRP-1 の場合 PE 内のレジスタは深さ 16 のレジスタファイルで構成されている．バックエンド合成により同一 PE 内レジスタの異なるアドレスに割り当てられた変数は結果として当該 PE のレジスタを共有することになる．

(2)状態間のデータ授受のためのレジスタ共有，については，共通変数に対しバックエンド合成により同一 PE 内レジスタの同一アドレスが割り当てられ，当該 PE のレジスタを明示的に共有することになる．(2)の場合の共有レジスタを特にコンテキスト間共有レジスタと呼ぶことにする．同様に，コンテキスト間共有メモリ，コンテキスト間共有入出力ポートという概念も必要となるので，ここであげておく．これらを総称してコンテキスト間共有リソースと呼ぶ．

6. バックエンド合成

バックエンド合成は，マッピング，自動配置配線，オブジェクトコード生成からなる合成段階である．

---

[2] 通常，共有する演算器はマルチプレクサよりも回路の大きな演算器に限られる

6.1. マッピング

マッピングは，STC コードの生成，PE の命令マッピング，演算粒度フィッティングおよび論理の最適化などを行う．フロントエンド合成部の出力した中間コードのうち制御回路部分は直接 DRP の STC コードに変換する．データパス回路はコンテキスト毎に DRP リソースへのマッピングを行う．まず，各演算を PE 命令に展開する．次に，階層情報の展開処理を行った後，演算粒度を DRP アーキテクチャに整合させるため，演算分割，メモリ，レジスタ分割を行う．最後にリソース間のデータ転送をネットとして定義し，自動配置・配線が処理可能なネットリストを出力する．マッピングの過程では，冗長ビットの除去，シフト/マスクの最適化，演算段数の最小化，多段ツリーのバランス化など，論理最適化も実行される．

6.2. 自動配置・配線

自動配置は，マッピングの出力したネットリストを入力とし，PE 命令を適切な位置の PE に割り当てる合成段階である．自動配置は基本的に各コンテキスト独立に行われるが，コンテキスト間共有リソース(コンテキスト間のデータ授受に用いられるレジスタ，メモリ，または入出力ポートなど)に対しては特別の考慮が必要となる．これらのリソースは異なるコンテキスト上で同一位置に割り当てられる必要があるからである．したがって，マルチコンテキストの配置問題は，擬似３次元的な配置問題となる（図 5参照）．与えられた PE アレイ領域内で，クロック制約により決まる最大遅延以内で実行可能な命令配置を求めることが自動配置の目的となる．

自動配線は，自動配置により位置決定された PE 間のデータ転送路を決定する合成段階である．DRP のデータ転送路は，配線セグメントとプログラマブルスイッチにより構成される．主要なバスは 8bit であるが 1bit のフラグ線も用意される．自動配置の結果をもとに，与えられた PE アレイ領域内で，クロック制約により決まる最大遅延以内で実行可能なデータ転送路を求めることが自動配線の目的となる．

6.3. オブジェクトコード生成

自動配置・配線の出力を，DRP チップにロード可能なオブジェクトコードに変換する合成段階である．各コンテキストの PE 命令，データ転送路設定に加え，レジスタやメモリの初期値，DRP の各種モード設定(ブートモードの選択，PLL の設定，など)もオブジェクトコードに含めることができる．

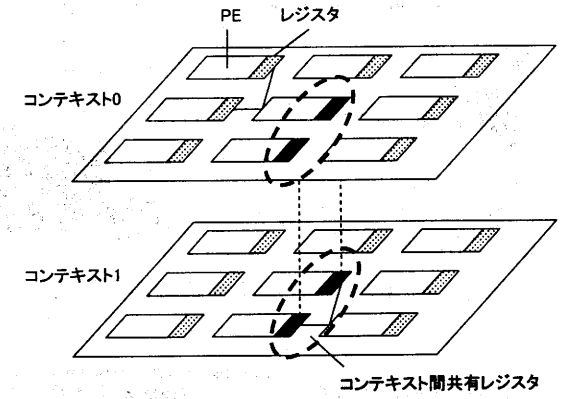


図 5　マルチコンテキスト配置モデル

7. 統合開発環境・オンチップ・デバッガ

よりソフトウエアライクな開発環境の構築を目指してグラフィカルなユーザーインターフェースを備えた統合開発環境の整備を進めている．

7.1. 統合開発環境 Musketeer

統合開発環境 Musketeer は，DRP コンパイラのフロントエンド合成部とバックエンド合成部をシームレスに統合したグラフィカルな開発環境であり，コンパイル操作の簡略化や，PE 使用状況，メモリアクセス状況の視覚化など合成レポートの充実による最適化の支援，などを意図したものである．ソースコード(C 言語)と中間コード，オブジェクトコード間の強力なクロスリファレンス機能も特徴の一つとなっている．ソースコードの変数とレジスタ，メモリの関係だけでなく，ソースコードのどの部分がどの状態にスケジューリングされたかといった情報も視覚化することが可能である．設計者は視覚化されたコンパイラのフィードバック情報に従い，スムースに最適化をすすめることができる．図 6に統合開発環境の画面イメージを示す．

7.2. オンチップ・デバッガ

DRP-1 チップを搭載した評価ボード(PCI ボード)が用意されている．DRP コンパイラはこの評価ボードを用いたオンチップデバッグをサポートしている(図 7参照)．

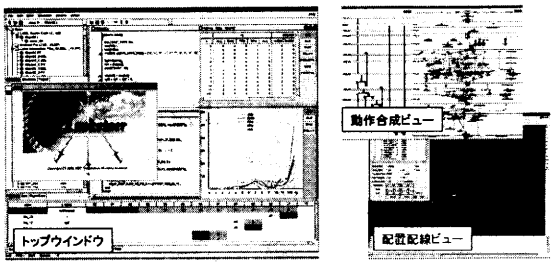


図 6 DRP コンパイラ統合開発環境

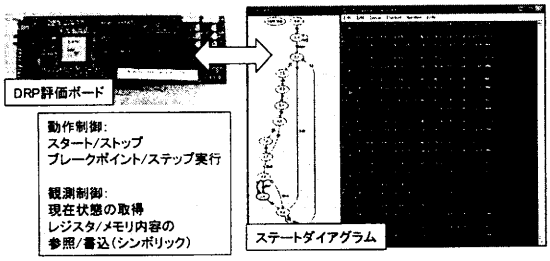


図 7 DRP オンチップ・デバッガ

DRP はデータパスと独立した状態遷移コントローラ(STC)を有すること，全ての命令メモリ，データメモリ，レジスタは統一アドレス空間にマップされること，など，可制御性，可観測性の高いアーキテクチャとなっている．オンチップ・デバッガはこの特性を最大限に活用し，効率的なオンチップデバッグ環境を提供する．動作制御としては，スタート，ストップ，ブレークポイントの設定，ステップ実行などが可能である．DRP の動作状況はステートダイヤグラムによりリアルタイムに視覚化される．そして，最大の特徴はソースコード上の変数名，配列名によるシンボリックデバッグが可能な点である．ソースコード上の変数や配列は，バックエンド合成により DRP 上のレジスタやメモリ粒度(8bit)にフィッティングされる．例えば，32bit 変数 a は，4 つの 8bit レジスタ a0～a3 に分割され，自動配置により任意の PE に割り当てられる．DRP コンパイラはこのような合成の過程を全てトレースしており，シンボル情報としてデバッガに引き渡す．デバッガはこのシンボル情報を用いて，ソース上のシンボル名を使ってリソースにアクセスすることを可能としている．結果として DRP の物理構造は設計者から隠蔽され，ソフトウエアライクなデバッグ作業が可能となる．これは，動作合成からの合成フローを密に統合した DRP コンパイラならではの特徴である．

8. おわりに

動的再構成可能プロセッサ DRP の C コンパイラ，統合開発環境，オンチップ・デバッガについて報告した．我々は，限定ユーザーに対する開発キット(DRP コンパイラ統合開発環境＋評価ボード)の評価リリーズを進めており，評価結果をもとに，統合開発環境のユーザビリティ向上，コンパイラの最適化機能改善に積極的に取り組んでいく．予定している機能拡張の一つとして，デバッガの統合開発環境組み込みによる，ソースレベルデバッグの実現をあげておく．

DRP コンパイラを用いたアプリケーションベンチマークも進展している，我々はすでに，暗号・認証(AES,DES,MD5)，誤り訂正符号(Viterbi,Turbo)，画像処理，ネットワーク処理(パケットフォワーディング，ファイヤーウォール)，などについてベンチマークを行ってきた．これらについては，機会をあらためて報告したい．

謝辞

日頃ご指導いただく，本村真人氏，古田浩一朗氏，藤井太郎氏，安生健一朗氏をはじめとする NEC エレクトロニクスの関係諸氏，ならびに，岡本匠氏をはじめとする NEC マルチメディア研究所の関係諸氏に深く感謝します．また，これまで DRP コンパイラ開発にご協力いただいた，宮田拓雄氏，齋藤雅弘氏，工藤健一氏，冠朋宏氏，池村大輔氏，塩瀬多聞氏，岡本高明氏，横田勲氏，小林敏彦氏，酒井一枝氏，上村哲生氏，ほか関係諸氏に対し心より御礼申し上げます．

文 献

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

[2] 大山，池上，若林，“C 言語ベースのシステムレベル合成・設計手法と開発事例，” 第 13 回軽井沢ワークショップ，2000.