Vol.2010-SLDM-146 No.6
2010/10/5

動的パッチ読み出し機構を備えた製造後機能修正可能アクセラレータ

吉田浩章[†1,†2] 藤田昌宏[†1,†2]

SoC の開発コスト増大と開発期間短縮に伴い，仕様変更や設計誤りによる製造後修正を可能とする技術の重要性が増している．最近になり，制御部分をプログラマブルにすることにより製造後修正を可能とするプログラマブルアクセラレータが注目されている．このプログラマブルアクセラレータは制御回路をメモリで実現しているため，結線論理によるアクセラレータに比べて電力効率が非常に低い．そこで，我々は小規模の機能修正を対象に，制御回路の大部分を結線論理で実現した上で部分的にパッチを当てることで制御を修正するプログラマブルアクセラレータを提案している[1]．しかしながら，この手法は機能修正が小規模の場合にのみ有効であり，大規模な機能修正の場合には効率が悪化してしまう，もしくは機能修正が不可能となってしまう．本稿では，動的パッチ読み出し機構を導入することでこの問題を解決する方式を提案する．

A Highly Energy-Efficient Accelerator Enabling Post-Silicon Engineering Changes With Dynamic Patch Loading Mechanism

Hiroaki Yoshida [†1,†2] and Masahiro Fujita[†1,†2]

With the rising cost of SoC development and the shorter time-to-market, the demand for post-silicon programmability has been increasing. Recently, programmable accelerators have attracted more attention as an enabling solution for post-silicon engineering change. However, programmable accelerators suffer from low energy efficiency mainly due to their extensive use of memories. More recently, we have proposed a highly-efficient programmable accelerator which enables post-silicon engineering change. The proposed accelerator acheives high efficiency by implementing the controllers mostly by hard-wired logic and providing a control patching mechanism. However, it is effective only when the size of engineering change is comparatively small. In this paper, we overcome this drawback by introducing a dynamic patch loading mechanism.

1. はじめに

近年の SoC 開発コストの増大と開発期間の短縮に伴い，高位合成を利用した設計手法の導入が進んでいる．高位合成によって生成された固定機能アクセラレータは高性能と高効率の両立が求められる様々な分野において利用されている．一方で仕様変更や設計誤りによる製造後修正がますます重要となっており，製造後修正を可能とするプログラマブルアクセラレータが注目されている[2),3)]．

ASIC 開発においては，開発コストや開発期間の短縮を目的として設計後の修正を行う engineering change (EC) 手法が用いられてきた．EC は全体回路に対して十分小さいため，この目的には FPGA 等の高い柔軟性を提供するプログラマブル素子は面積オーバーヘッドも大きく向いていない．現時点では EC は配線の入れ替えやスペアセルを用いた回路レベルの変更で実現されているが，高位合成手法の普及が進むにつれて高位における EC が重要になってくると思われる．このような背景の下で，プログラマブルアクセラレータの高位設計手法が提案されている[2)–7)]．No-Instruction-Set Computer (NISC)[2)] はマイクロコード方式のプログラマブル制御回路とカスタムデータパスからなるプログラマブルハードウェアである．単一アプリケーションに対して最適なカスタムデータパスを生成する手法も提案されている[6)]．この手法では，まず初期スケジューリング結果に基づいて十分な数の FU を割り当てる．次に，FU の使用頻度に応じて FU を段階的に削減することでカスタムデータパスを生成する．プログラマブルループアクセラレータ (PLA)[3),7)] は，MOV 命令やグローバルバス，ポート交換などアーキテクチャ的にプログラマビリティを向上させる工夫をしている．

上記のプログラマブルアクセラレータは全て制御回路をメモリを用いた水平型マイクロコード方式で実現している．一般的に水平型マイクロコード方式は非常に大きなメモリを必要とするため，結線論理で実現した制御回路に比べて面積や電力効率が悪い．このため，アクセラレータの優位点である高性能と高電力効率を両立することが難しくなってしまう．我々は可能な機能修正の範囲を小規模に限定することで，高性能と高電力効率を両立することが可能なプログラマブルアクセラレータ方式を提案した[1)]．提案方式では制御回路の大部分を結線論理で実現し，部分的に制御信号にパッチを当てることで製造後の機能修正を可能としている．しかしながら，この手法は機能修正が小規模の場合のみ有効であり，機能修正

†1 東京大学 大規模集積システム設計教育研究センター (VDEC)
†2 科学技術振興機構 戦略的創造研究推進事業 CREST

Vol.2010-SLDM-146 No.6
2010/10/5

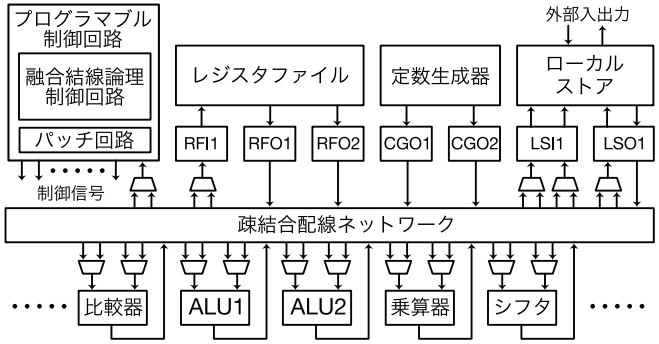


図 1 プログラマブルアクセラレータの基本構成

の規模が大きくなると効率が悪化してしまう，もしくは機能修正が不可能となってしまう．本稿では，動的パッチ読み出し機構を導入することでこの問題を解決する方式を提案する．次の第 2 節で従来方式[1] である製造後機能修正可能なアクセラレータとパッチコンパイル手法を説明する．その後，第 3 節で従来方式の問題点について述べた後，提案方式である動的パッチ読み出し機構の回路構成と動作原理について説明する．

2. 製造後機能修正可能な高電力効率アクセラレータ

2.1 全体の構成

本稿で対象とするプログラマブルアクセラレータ (以下，アクセラレータ) の基本構成を図 1 に示す．データパス部分は潜在的な機能修正を考慮した合成手法[8] によって生成することができる．アクセラレータは機能ユニット (FU)，FU 間を接続する配線部分，およびプログラマブル制御回路からなる．各 FU はあらかじめ決められた 1 つ以上の種類の演算を行うことが可能であり，制御信号によって演算の種類を決定する．典型的な FU としては，ALU や乗算器，比較器，バレルシフタなどがある．レジスタファイルやローカルストア，定数生成器などのメモリ素子では，メモリ素子の書き込みポートや読み込みポートをそれぞれ FU とみなす．このようにすることで，合成およびコンパイルの際にメモリ素子へのアクセスを算術演算と同様に扱うことが可能である．各 FU の入力は他の FU の出力もしくはマルチプレクサの出力と接続される．同様に各 FU の出力も他の FU の入力やマルチプレクサの入力に接続される．マルチプレクサの入力は FU の出力と接続されている．マルチプレクサの制御信号によって各 FU は入力信号を選択する．レジスタファイル (RF) はレジスタの集合であり，複数の書き込みポート (RFI) および読み込みポート (RFO) を持つ．レジスタは局所変数値の格納に使用される．各 RF ポートの制御信号によって，どのレジスタにアクセスするかを決定する．定数生成器はあらかじめ決められた定数を出力することが可能であり，レジスタファイル同様，読み込みポート (CGO) への制御信号に基づいて定数を生成する．ローカルストアは主に配列やグローバル変数値を格納する RAM であり，また外部とのデータのやり取りにも使用される．ローカルストアも書き込みポート (LSI) と読み込みポート (LSO) を持つが，他のメモリ素子とは異なり，ポートはアドレスとデータの 2 つの信号線を持つ．ローカルストアの書き込みポートは書き込みイネーブルの制御入力を持つ．

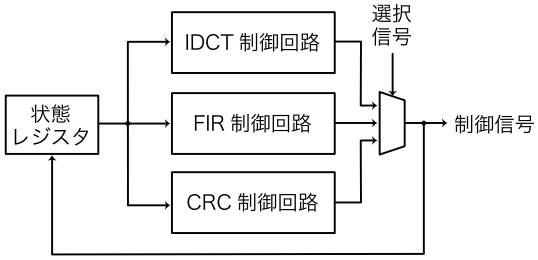


図 2 融合結線論理制御回路の基本構成

プログラマブル制御回路 (以下，制御回路) は融合結線論理制御回路とパッチ回路からなり，現在の状態に基づいて FU やマルチプレクサへの制御信号を生成する．制御回路はデータパスが生成した 1 ビットの制御信号に基づいて次状態を決定する．融合結線論理制御回路とパッチ回路については次節以降で詳細に説明する．

2.2 融合結線論理制御回路

融合結線論理制御回路は，初期設計記述に対応する制御回路を結線論理で実現した回路である．図 2 にその構成例を示す．初期設計記述として複数の設計記述が与えられた場合には，まず各設計記述に対応する結線論理制御回路を生成し，これらの出力をマルチプレクサで結合したものが最終的な制御回路となる．マルチプレクサの選択信号を変更することで，機能を変更することが可能となっている．実装時には，全体の論理回路に対して最適化を適用することで異なる機能間での共有が行われるため，各制御回路を実装した後に単純に結合した場合に比べて，面積効率の良い回路を実装することができる．融合結線論理制御回路は固定機能を実現するものであり，次に説明するパッチ回路を付加することにより機能修正が可能になる．

Vol.2010-SLDM-146 No.6
2010/10/5

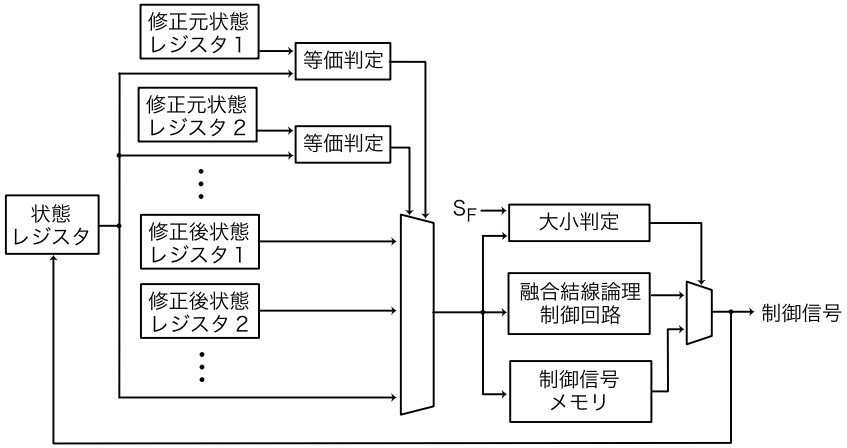
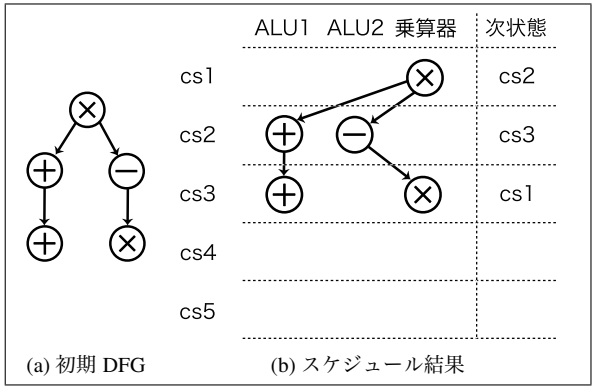


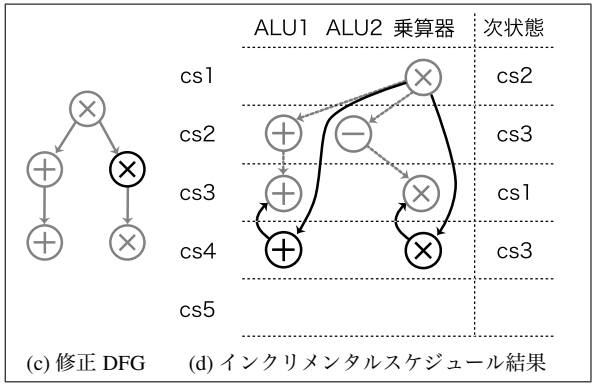
図 3 パッチ回路の基本構成

2.3 パッチ回路

パッチ回路は，融合結線論理制御回路のいくつかの状態に対して生成する制御信号を修正することができる．修正されない状態に対しては，融合結線論理制御回路が生成する制御信号がそのまま出力される．図 3 に示すようにパッチ回路は状態パッチ部分と制御信号パッチ部分からなる．ここで，融合結線論理制御回路に実装されている状態を $s_1, ...s_n$，パッチ回路に実装されている状態を $s_{n+1}, ..., s_{n+m}$ とする．状態パッチ部分では，融合結線論理制御回路のいくつかの状態をパッチ回路の状態に変換する．また，制御信号パッチ部分はパッチ回路内の各状態 $s_{n+1}, ..., s_{n+m}$ の制御信号を生成する．

パッチ回路の動作原理を図 6 を用いて説明する．ここでデータパスは 2 つの ALU と 1 つの乗算器を備えている．図 6(a) に示す初期設計のデータフローグラフ (DFG) のスケジュール結果が図 6(b) であるとする．このスケジュールでは 3 つの状態 cs1, cs2, cs3 があり，状態遷移は cs1→cs2→cs3→cs1→ ··· の順で繰り返される．これらは融合結線論理制御回路で実装されている．次に機能修正後のデータフローグラフを図 6(c) に示す．この機能修正では減算が乗算に修正されている．修正された演算に対応する状態は cs2 であるため，cs2 に含まれている加算とともに再スケジュールを行う必要があり，新しい状態 cs4 を導入した再スケジュール結果は図 6(d) になる．この cs4 のスケジュール情報はパッチ回路内の制御信号メモリに格納され，また図 6(e) に示すように修正元状態レジスタ 1 を cs2 に，修正後状態レジスタ 1 を cs4 にする．修正元状態レジスタ 2 は使用しないため，到達しない状態を設定しておくことで，状態変換を無効化する．こうすることで，状態遷移は cs1→cs4→cs3→cs1→ ··· となり，機能修正が実現できる．





レジスタ番号	変換元状態	変換後状態
1	cs2	cs4
2	—	—

(e) 状態変換表

図 4 パッチ回路の動作例

2.4 パッチコンパイル手法

修正前後の設計記述は，修正前後のデータフローグラフを融合したグラフ $G=(V,E)$ で表現される．ここで，演算ノードの集合 V は，修正されない演算ノードの集合 V_F，除去された演算ノードの集合 V_R，追加された演算ノードの集合 V_N の和集合 $V=V_F\cup V_N\cup V_R$ として表される．つまり，修正前の演算ノード集合は $V_F\cup V_R$，修正後の演算ノード集合は $V_F\cup V_N$ である．各データ依存辺 $e\in E$ は各演算ノード間のデータ依存関係を示す．データパスは FU の集合 $F=\{f_1,f_2,...\}$ とレジスタファイルポートの集合 $P=\{p_1,p_2,...\}$ からなる．制御ステップ $S=\{s_1,s_2,...\}$ は制御回路の各状態に対応しており，修正前の各演算 $v\in V_F\cup V_R$ の実行される制御ステップを $S_o(v)$，使用する FU を $F_o(v)$ とする．また，修正可能な制御ステップの総数の最大値を M_{max} とする．これは制御信号メモリのワード数に対応している．パッチコンパイル問題は，各追加演算ノード $v\in V_N$ の制御ステップ $S(v)$ と使用する FU $F(v)$ を求める問題である．

修正前の演算はすでに制御ステップにスケジュールされているとする．次に各制御ステップ間に空の制御ステップを挿入する．この空ステップにスケジュールされた演算はパッチ回路に実装されることになる．各制御ステップ間に挿入される空の制御ステップの数は制御信号メモリのワード数もしくは最も悲観的にスケジュールした場合に必要な制御ステップ数の小さい方となる．

次に，制約式で用いる変数について説明する．これらの変数はすべて 2 値変数である．$B_{i,j,k}$ は制御ステップ s_k において演算ノード v_i が FU f_j を使用しているときに 1 となる変数であり，$G_{j,k,q,t}$ は制御ステップ s_k において FU f_j の t 番目の入出力信号線がレジスタポート p_q を使用する場合に 1 となる変数である．また，M_k は制御ステップ s_k に修正がある場合に 1 となる変数である．制約式は以下の 7 種類に分類される．

制約 *1* (演算使用制約) データフローグラフにおける各追加演算はある制御ステップに一度のみスケジュールされなくてはならない．これを制約式にすると以下のようになる．

$$\sum_{j,k} B_{i,j,k}=1 \quad \forall i \tag{1}$$

制約 *2* (資源制約) 各制御ステップにおいて，各 *FU* は高々一度のみ使用可能である．これを制約式にすると以下のようになる．

$$\sum_{i} B_{i,j,k}\le 1 \quad \forall j,k \tag{2}$$

制約 *3* (データ依存制約) データフローグラフの各データ依存辺について，始点の演算は終点の演算よりも先にスケジュールされなければならない．これを制約式にすると以下のようになる．

$$\sum_{k} k(\sum_{j} B_{l,j,k})+1\le \sum_{k} k(\sum_{j} B_{m,j,k}) \quad \forall (o_l,o_m)\in E \tag{3}$$

ここで，左辺の最初の項は始点の演算の制御ステップ，右辺は終点の演算の制御ステップに対応する．

制約 *4* (修正制御ステップ制約) 変数 M_k は制御ステップ s_k が修正された際に *1* となる．これを制約式にすると以下のようになる．

$$B_{i,j,k}\le M_k\le 1 \quad \forall i,j,k \tag{4}$$

ここで，制御ステップ k が修正されない場合に必ずしも M_k が *1* とならないことに注意されたい．

制約 *5* (除去演算制約) 除去する演算 $o_r\in O_r$ がスケジュールされている制御ステップは無条件に修正制御ステップとなる．これを制約式にすると以下のようになる．

$$M_{step(o_r)}=1 \quad \forall o_r\in O_r \tag{5}$$

制約 *6* (最大修正制御ステップ数制約) 修正可能な制御ステップ数の上限はパッチ回路内の制御信号メモリのワード数で決定される．これを制約式にすると以下のようになる．

$$\sum_{i} M_k\le M_{max} \tag{6}$$

制約 *7* (レジスタポート制約) ここでは説明の簡略化のためチェイニングを考慮しない．つまり，各 *FU* はレジスタファイルから値を読み出し，演算結果をレジスタファイルに格納する．よって，各 *FU* の入出力はレジスタファイルポートに接続されている必要がある．これを制約式にすると以下のようになる．

$$\sum_{j,t} G_{j,k,q,t}\le 1 \quad \forall k,q \tag{7}$$

$$\sum_{i} B_{l,j,k}\le \sum_{q} G_{j,k,q,t} \quad \forall j,k,t \tag{8}$$

各変数のレジスタ割り当てについては，整数線形計画法問題を解いた後に従来手法[9] を用いて求める．上記の制約式を整数線形計画法によって解くことによって，各演算の制御ステップと FU が求まる．もし解が求まらない場合には，与えられた制御信号メモリのワード数では機能修正が不可能であることを示している．

3. 動的パッチ読み出し機構

3.1 従来方式の問題点と解決法

第 2 節で説明したアクセラレータ方式では，全体の電力効率はパッチアクセスの一回当たりの電力量と頻度で決まる．結線論理制御回路の 1 サイクル当たりの消費電力量を E_h，アクセス回数を N_h，パッチメモリの 1 アクセス当たりの消費電力量を E_p，アクセス回数を N_p をすると，制御回路の消費電力量 E_{ctrl} は以下の式で表される．

$$E_{flat} = E_h \cdot N_h + E_p \cdot N_p \quad (9)$$

パッチを当てる制御ステップ数が総ステップ数に比べて十分に小さい場合には E_p，N_p ともに小さくなるため，結線論理制御回路方式と同等の電力効率を実現できる．しかし，パッチを当てる制御ステップ数が大きくなると，パッチメモリの規模が水平型マイクロコード方式のメモリと同等になり，電力効率の優位性が失われてしまう．また，提案手法ではパッチメモリ量は設計段階であらかじめ決定しておく必要があり，製造後の機能修正の規模がパッチメモリ量よりも大きい場合には，機能修正は不可能になる．

一般的な処理ではデータアクセスと同様，命令アクセスに関しても空間的および時間的局所性があることが知られている．この性質を利用し，命令メモリを階層化することで消費電力を削減する手法が提案されている[10)–12)]．ここで k 次メモリの 1 アクセス当たりの消費電力量，アクセス回数を E_{p_k}，アクセス回数を N_{p_k} をすると，パッチメモリを 2 階層にした場合の消費電力量 E_{hier} は以下の式で表される．

$$E_{hier} = E_h \cdot N_h + E_{p_1} \cdot N_{p_1} + E_{p_2} \cdot N_{p_2} \quad (10)$$

このとき N_{p_2} が N_{p_1} に比べて十分に小さい場合には，2 次パッチメモリの消費電力量は無視できる．ここで 1 次パッチメモリの大きさを適当に選択することで，パッチ規模が比較的大きい場合においても高い電力効率を実現可能である．

3.2 回路構成と動作原理

図 5 に示すように，提案方式の回路構成は 1 次パッチメモリと 2 次パッチメモリの 2 階層のパッチメモリからなる．2 次パッチメモリはすべてのパッチを含んでおり，必要に応じてその内容の一部を 1 次パッチメモリに転送する．ここで，1 次パッチメモリが図 3 中の制御信号メモリに対応する．2 次パッチメモリから 1 次パッチメモリへの転送は DMA コントローラによって制御される．転送が必要な場合にはパッチ回路がパッチ更新命令を DMA コントローラに送り，これに基づいて 2 次パッチメモリの内容が 1 次パッチメモリへ送られる．第 2.2 節で説明した通り，パッチ回路内の結線論理制御回路は現在の状態に基づいて

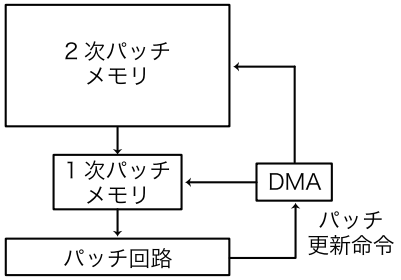


図 5 動的読み出し機構を備えたパッチ回路

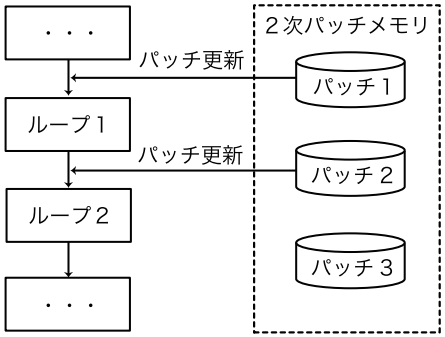


図 6 動的読み出し機構を備えたパッチ回路の動作例

データパスの制御信号を生成する．一方で，制御信号メモリは修正後制御ステップにおけるデータパスの制御信号だけでなく，前述のパッチ更新命令も生成する．パッチ更新命令は 2 次パッチメモリの転送元アドレスと 1 次パッチメモリの転送先アドレス，転送ワード数の 3 つのフィールドからなる．転送ワード数に応じて複数サイクルで転送を行い，また転送ワード数が 0 の場合には何も実行しない．パッチ更新命令の挿入は演算のスケジューリングと同時に行う．

ここでは図 6 の例を用いて動作を説明する．図の例ではループ 1 とループ 2 の 2 つのループを含んでおり，それぞれのループに対して機能修正が行われた結果，パッチ 1 とパッチ 2 がコンパイルされている．ここでパッチ 1 とパッチ 2 は規模の制約により同時に 1 次パッチメモリに格納できないものとする．このとき，あらかじめパッチ 1 とパッチ 2 を 2 次パッ

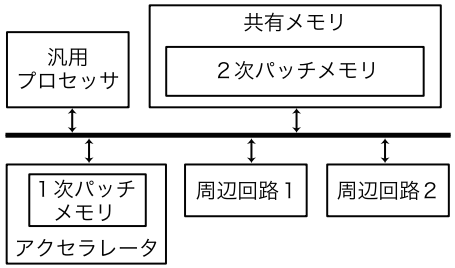


図 7 SoC 内の共有メモリを 2 次パッチメモリとして利用

チメモリに格納しておき，ループ 1 の実行の直前にパッチ更新命令を実行し，パッチ 1 を 1 次パッチメモリに転送する．ループ 1 の実行が終了した後，ループ 2 の実行の直前に再びパッチ更新命令を実行し，パッチ 2 を 1 次パッチメモリに転送する．前節で説明したように，ループ 1 やループ 2 の繰り返し回数が多い場合には，2 次パッチメモリのアクセス回数は 1 次パッチメモリに比べて小さくなるため，電力効率を下げることなく，パッチ規模を大きくすることができる．1 次パッチメモリの大きさは原則として各ループ内で想定される機能修正の規模で決定され，また 2 次パッチメモリの規模は想定される機能修正全体の上限によって決定されるべきである．

この階層的メモリ構造を導入することによって電力量を削減することは可能であるが，一方で面積は増加してしまう．最近の SoC では大規模なメモリを搭載し，機能ブロック間で共有することが一般的となっている．このとき，このメモリの一部を 2 次パッチメモリとして使用することで，独立した 2 次パッチメモリを実装する必要がなくなる．この場合には他の機能ブロックと同様，共有メモリのアクセス制御を行う必要がある．

4. まとめと今後の課題

従来我々が提案した製造後機能修正可能なアクセラレータは高性能と高電力効率を両立可能であるが，実現可能な機能修正は小規模なものに限定されていた．本稿では，動的パッチ読み出し機構を用いることで電力効率を失うことなくパッチ規模を大きくすることが可能な方式を提案した．今後の課題としては，全体の消費電力量を最小化するように大規模パッチを分割し，パッチ更新命令のスケジュールを行う自動コンパイル手法の開発が挙げられる．

参 考 文 献

1) 吉田 浩章, 藤田 昌宏, “製造後機能修正可能な高電力効率アクセラレータの高位設計手法,” 情報処理学会 DA シンポジウム 2010 論文集, pp. 45–50, 2010 年 9 月.
2) M. Reshadi and D. Gajski, “A cycle-accurate compilation algorithm for custom pipelined datapaths,” in *Proc. IEEE/ACM Int. Symp. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Sep. 2005, pp. 21–16.
3) K. Fan, M. Kudlur, G. Dasika, and S. Mahlke, “Bridging the computation gap between programmable processors and hardwired accelerators,” in *Proc. Int. Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2009, pp. 313–322.
4) A. Kifli, G. Goossens, and H. DeMan, “A unified scheduling model for high-level synthesis and code generation,” in *Proc. European Design and Test Conf. (ED&TC)*, Mar. 1995, pp. 234–238.
5) M. Benmohammed and A. Rahmoune, “Automatic generation of reprogrammable microcoded controllers within a high-level synthesis environment,” *IEE Proc. Computers and Digital Techniques*, vol. 145, no.3, pp. 155–160, May 1998.
6) J.Trajkovic and D.Gajski, “Automatic data path generation from C code for custom processors,” in *Proc. IFIP Int. Embedded Systems Symp. (IESS)*, May 2007, pp. 379–384.
7) K.Fan, H.Park, M.Kudlur, and S.Mahlke, “Modulo scheduling for highly customized datapaths to increase hardware reusability,” in *Proc. IEEE/ACM Int. Symp. on Code Generation and Optimization (CGO)*, Apr. 2008, pp. 124–133.
8) 吉田 浩章, 藤田 昌宏, “潜在的多様性を考慮したプログラマブルハードウェアの高位合成手法,” 電子情報通信学会技術研究報告, vol. 109, no. 462, pp. 67–72, 2010 年 3 月.
9) P. Brisk, F. Dabiri, R. Jafari, and M. Sarrafzadeh, “Optimal register sharing for high-level synthesis of SSA form programs,” *IEEE Trans. Computer-Aided Design*, vol.25, no.5, pp. 772–779, May 2006.
10) J.Kin, M.Gupta, and W.H. Mangione-Smith, “The filter cache: An energy efficient memory structure,” in *Proc. Int. Symp. on Microarchitecture (MICRO)*, 1997, pp. 184–193.
11) L.H. Lee, B.Moyer, and J.Arends, “Instruction fetch energy reduction using loop caches for embedded applications with small tight loops,” in *Proc. Int. Symp. on Low Power Electronics and Design (ISLPED)*, 1999, pp. 267–269.
12) J.Park, J.Balfour, and W.J. Dally, “Maximizing the filter rate of l0 compiler-managed instruction stores by pinning,” *Technical Report 126, Concurrent VLSI Architecture Group, Stanford University*, 2009.