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

吉田 浩章　　藤田 昌宏

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

概要　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．本稿で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する．また，修正前後の設計記述からパッチをコンパイルする手法を提案する．例題を用いた評価結果を通じて提案方式が電力効率の面で従来方式よりも優位であることを示す．

High-Level Synthesis for Highly-Efficient Accelerators Enabling Post-Silicon Engineering Change

Hiroaki Yoshida　　Masahiro Fujita

VLSI Design and Education Center, the University of Tokyo
CREST, Japan Science and Technology Agency

Abstract　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. This paper first proposes highly-efficient programmable accelerators which enable post-silicon engineering change. The proposed accelerators acheive high efficiency by implementing the controllers mostly by hard-wired logic and providing a control patching mechanism. Then, we propose a patch compilation technique from a given set of an original design and a modified design. Experimental results demonstrate that the proposed accelerators can acheive higher energy efficiency than the existing programmable accelerators.

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 [1, 2].

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．現在では 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あるものの，高位合成手法の普及に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 [3, 4, 1, 5, 6, 2]. No-Instruction-Set Computer (NISC) [1] 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 [5]．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) [6, 2] 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．

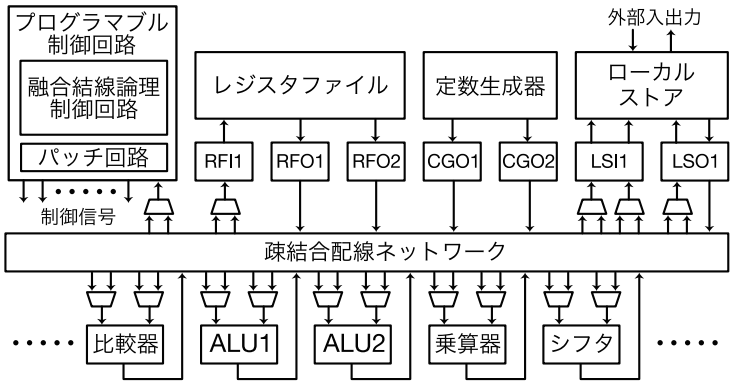


図 1　プログラマブルアクセラレータの基本構成

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．本稿では，可能な機能修正の範囲を限定することで，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する．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．本稿では，初期設計記述と機能修正後の設計記述の差分からパッチをコンパイルする手法を提案する．提案パッチコンパイル手法は整数線形計画法に基づいており，厳密解を求めることが可能となっている．また例題を用いた評価を通じて，提案アクセラレータ方式が従来方式に比べて面積や電力効率の点で優れていることを示す．

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．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

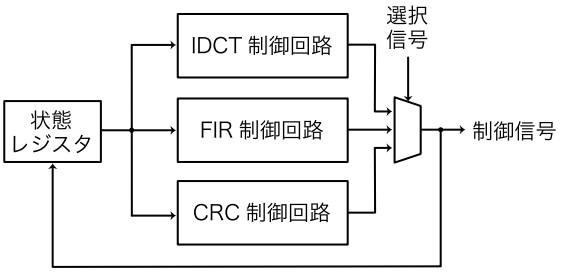


図 2　融合結線論理制御回路の基本構成

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．

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

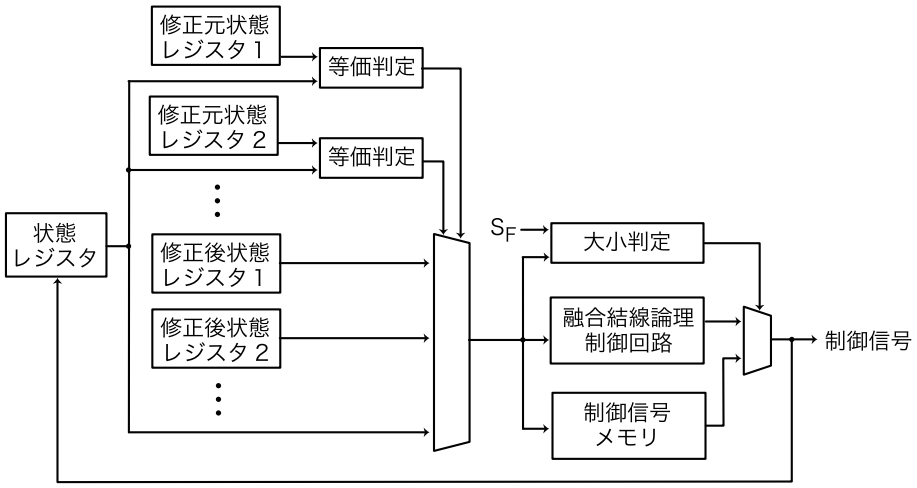


図 3　パッチ回路の基本構成

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，各制御回路を単純に実装した後に結合した場合に比べて，効率的な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って，機能変更が可能になる．

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．

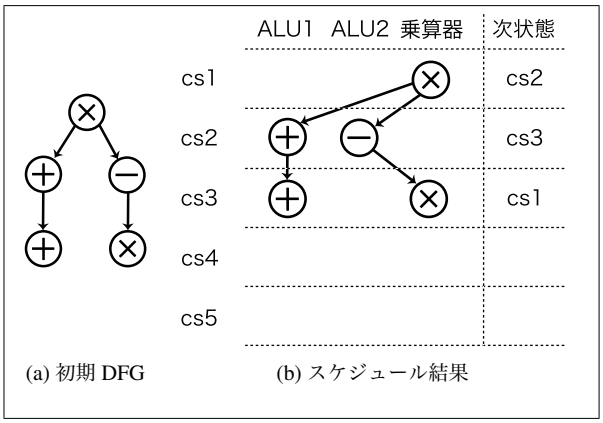
パッチ回路の動作原理を図 4 を用いて説明する．ここでデータパスは 2 つの ALU と 1 つの乗算器を備えている．図 4(a) に示す初期設計のデータフローグラフ (DFG) のスケジュール結果が図 4(b) であるとする．このスケジュールでは 3 つの状態 cs1, cs2, cs3 があり，状態遷移は cs1→cs2→cs3→cs1→ ⋯ の順で繰り返される．これらは融合結線論理制御回路で実装されている．次に機能修正後のデータフローグラフを図 4(c) に示す．この機能修正では減算が乗算に修正されている．修正された演算に対応する状態は cs2 であるため，cs2 に含まれている加算とと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 4(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 4(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．

3　整数線形計画法に基づくパッチコンパイル手法

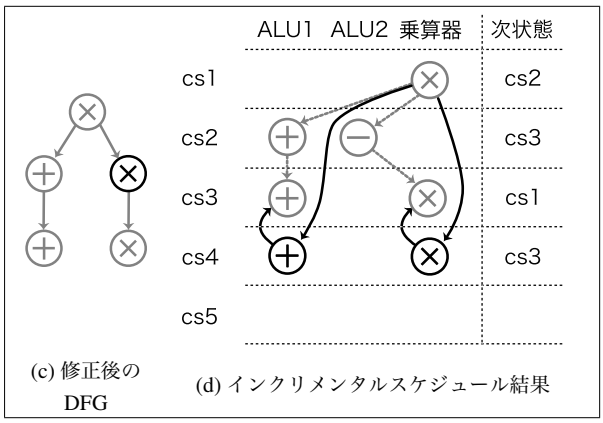
3.1　問題定式化

本節では，機能修正前後の設計記述に基づいてアクセラレータの機能を修正するためのパッチメモリの内容をコンパイルする手法を説明する．提案アクセラレータではパッチ回路内のメモリ量によって可能な機能修正の規模が決まるため，修正規模が大きい場合には機能修正が不可能である場合がある．製造後において，機能修正が可能であるかどうかは極めて重大な問題であり，正確に見極める必要がある．そのため本稿では整数線形計画法を用いて厳密解を求める手法を提案する．一般的に，整数線形計画法を用いた高位合成手法は実用的な規模の問題に適用することが難しいことが知られており，そのためスケジューリング，バインディングなど各段階別に整数線形計画法を適用する場合が多い．本稿で扱う問題はインクリメンタルな合成問題であり，機能修正の規模が比較的小さく，また多くの FU がすでにバインドされているため，実用的な規模の問題でも適用可能であると考えられる．

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



(a) 初期 DFG　(b) スケジュール結果



(c) 修正後の DFG　(d) インクリメンタルスケジュール結果

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

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．

3.2　整数線形計画法に基づく解法

本節では上記のパッチコンパイル問題を整数変数を用いた制約式で表現し，整数線形計画法で解く手法について説明する．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．

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

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．

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

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．

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

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．

$$\sum_i M_k \leq M_{max} \tag{6}$$

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．

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

$$\sum_i B_{l,j,k} \leq \sum_q G_{j,k,q,t} \quad \forall j, k, t \tag{8}$$

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 [7] などの手法を用いて求める．

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．

3.3 修正前後のデータフローグラフの差分の計算方法

提案手法では修正前後の設計記述が与えられた際に，データフローグラフの差分を求める必要がある．一般的に合成ツールでは設計記述をデータフローグラフに変換する際に様々な最適化を適用する．記述修正によって最適化の影響範囲が変わってしまうため，設計記述では同じ記述の部分であっても最適化によって差分が生じる可能性がある．このため，単純なグラフ間の比較では差分が大きくなってしまうことが考えられる．修正可能な制御ステップ数には上限があるため，データフローグラフにおける差分は小さい方が望ましい．これを実現する方法として以下の 3 通りが考えられる．

1. 設計記述の差分のみをデータフローグラフに変換し，修正前のデータフローグラフと結合する．
2. 設計記述の差分に対応する部分にのみ最適化を適用するように最適化手続きを変更する．
3. 機能修正の際に設計者がデータフローグラフの差分を指示する．

設計効率の面では方法 1 または方法 2 が望ましいが，これらの実現にはデータフローグラフへの変換や最適化の手続きを大幅に変更する必要がある．よって，現時点では方法 3 を想定しており，データフローグラフの差分を明示的に与えることによって機能修正を指定する．方法 2 と方法 3 については今後の課題とする．

表 1 設計例題の概要

例題	演算ノード数	説明
bubble_sort	55	バブルソート
adpcm_decoder	178	ADPCM デコーダ
bdist2	182	MPEG-2 bdist2() 関数
idct	286	8x8 逆離散コサイン変換
mpeg_pred	369	MPEG-1 予測関数

4 提案アクセラレータ方式の評価結果

本稿で提案した手法を我々が開発している SORA 合成・最適化フレームワーク上に実装した．入力 C プログラムを解析し，SSA 形式の CDFG を構築する処理には LLVM コンパイラ・インフラストラクチャ [9] を用いている．データパス合成には潜在的多様性を考慮したプログラマブルアクセラレータの高位合成手法 [11] を用いている．この手法では複数機能の実行に最適化されたデータパスを合成することが可能となっている．提案手法で用いる整数線形計画法のソルバとしては Gurobi Optimizer[10] を用いた．

本節では評価を通じて提案アクセラレータ方式が面積と消費電力の面において従来方式よりも優れていることを示す．例題設計として表 1 に示す C 言語で記述された 5 つの設計を用いた．面積・消費電力評価には 45nm プロセスの仮想テクノロジである FreePDK45[8] を用いた．また，スタンダードセルライブラリは Nangate 社提供のものを用いた．論理合成には Synopsys 社 Design Compiler を，静的タイミング解析，面積と消費電力の評価には Synopsys 社 PrimeTime を用いた．

本評価では以下の 8 種類の回路を設計し，面積と消費電力を評価した．公平な比較のため，全ての回路の動作周波数は 200MHz となるように設計してある．最初の 5 回路は各例題を固定機能アクセラレータとして高位合成した回路である．次の 3 回路は 5 例題の機能をすべて実行可能な提案アクセラレータ方式の回路であり，最大修正制御ステップ数 M_{max} がそれぞれ 3，10，50 である．最後の回路は制御回路をメモリで実現する従来方式であり，ここでは完全プログラマブル方式と呼ぶ．評価結果を表 2 に示す．表では各回路の部分毎の面積と消費電力を示している．$M_{max} = 3$ の提案アクセラレータ方式は完全プログラマブル方式に比べて面積で 78%，消費電力で 83% 小さい．また，固定機能アクセラレータの中で最も大きな面積を持つ回路である bdist2 に比べて面積で 18%，消費電力で 13% のオーバーヘッドであった．このように，提案アクセラレータ方式は複数機能の実行時の変更や製造後の機能修正を可能にしつつ，固定機能アクセラレータに匹敵する面積・消費電力を実現可能であることを確認した．

表 2　各方式の面積・消費電力の比較 (FreePDK 45nm テクノロジ)

回路	面積 [μm^2]						消費電力 [mW]					
	制御	MUX	算術	RF	LS	合計	制御	MUX	算術	RF	LS	合計
bubble_sort	208	393	1028	3882	25657	31168	0.03	0.04	0.05	0.58	5.50	6.20
adpcm_decoder	538	1116	2580	8519	35127	47881	0.05	0.04	0.06	1.14	7.47	8.75
bdist2	791	647	6589	33960	32814	74802	0.07	0.05	0.38	2.82	7.02	10.34
idct	1239	1323	11651	24942	16435	55591	0.09	0.07	0.34	1.98	3.41	5.89
mpeg_pred	1619	1813	8689	26128	32799	71048	0.10	0.05	0.17	2.30	7.00	9.63
提案方式 (M_{max} = 3)	6624	1946	11335	33100	35087	88091	1.20	0.07	0.27	2.67	7.44	11.65
提案方式 (M_{max} = 10)	17447	1946	11335	33100	35087	98914	3.23	0.07	0.27	2.67	7.44	13.68
提案方式 (M_{max} = 50)	79292	1946	11335	33100	35087	160759	14.78	0.07	0.27	2.67	7.44	25.23
完全プログラマブル	313868	1990	11415	33083	35116	395471	58.61	0.03	0.12	2.93	7.41	69.11

5　まとめと今後の課題

本稿では高性能と高電力効率を両立可能なプログラマブルアクセラレータ方式を提案した．提案方式では，小規模な機能修正を対象として，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効率的な機能修正を可能としている．また，初期設計記述と機能修正後の設計記述の差分からパッチをコンパイルする問題を整数線形計画法で定式化し，厳密解を得る手法を提案した．例題を用いた評価では，提案アクセラレータ方式が従来方式に比べて電力効率の点で優れていることを示した．

今後の課題としては，3.3 節で説明したデータフローグラフの差分の計算方法について，設計効率を失わずに機能修正を指定する方法について検討していく予定である．提案アクセラレータ方式の実用に向けた課題としては，機能修正規模に制約がある，またあらかじめその修正可能規模を決定しなくてはならない，という点が挙げられる．この課題の解決方法として，外部メモリから動的にパッチを制御信号メモリに転送する手法を検討している．各ループ内の機能修正は制御信号メモリに収まるが，複数ループで機能修正が必要な場合などは，外部メモリからの転送の回数は少なく済むため，この方法が有効であると考えられる．

参考文献

[1] 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.

[2] 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.

[3] A. Kifli, G. Goossens, and H. De Man, “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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] *FreePDK45,* `http://www.eda.ncsu.edu/wiki/FreePDK45:Contents`. North Carolina State University, 2010.

[9] C. Lattner and V. Adve, “LLVM: A compilation framework for lifelong program analysis & transformation,” in *Proc. IEEE/ACM Int. Symp. on Code Generation and Optimization (CGO)*, May 2004, p. 75.

[10] *Gurobi Optimizer Reference Manual, Version 3.0*. Gurobi Optimization, Inc., 2010.

[11] 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.