2009－MPS－73 (39)
2009／3／6

FPGAを用いた複数定数乗算回路の最適設計の高速化

中山 雅雄† 佐々木孝雄† 豊嶋 久道†

† 神奈川大学工学部

あらまし 複数定数乗算 (MCM) 回路の設計において, 回路規模や遅延時間・消費電力等のコストを最小とする問題はNP完全問題として知られている. そのため, 遺伝的アルゴリズム (GA) などの最適化アルゴリズムによる設計手法が提案されている. しかしながら, この設計アルゴリズムをソフトウェア実装した場合, 定数やそのビット数が増加すると, 最適化に非常に時間がかかる. そこで本研究では, MCM回路の最適設計を高速化するために, ソフトウェア実装ではボトルネックとなる繰り返し処理に着目し, FPGAの構造に適した並列化方法や, 並列化・パイプライン化の粒度を持つアルゴリズムを提案する. さらにFPGAに実装し, 提案法の有効性について検証する.

Shortening of processing time of optimal design of multiple constant multiplication using FPGAs

Masao NAKAYAMA†, Takao SASAKI†, and Hisamichi TOYOSHIMA†

† Faculty of Engineering, Kanagawa University

Abstract Problem of designing multiple constant multiplication (MCM) circuits with minimum cost is known to be an NP-complete problem. Several techniques using combinatorial optimization algorithms such as genetic algorithms (GAs), etc., have been proposed. However, if implemented as software, as the circuit scale increases, a great deal of time is needed for optimization. The purpose of this study is to shorten the time spent on optimizing MCM circuit design. We propose a hardware-oriented algorithm suitable for FPGAs for both circuit synthesis and optimization.

1. はじめに

複数定数乗算 (Multiple Constant Multiplication: MCM) 回路は, 1入力変数に対し複数定数の乗算を行う回路であり, 音声・画像など多くの信号処理で使用されている. しかし, 乗算回路は多くのハードウェアコストを要するため, 回路規模や遅延時間・消費電力の制限がある場合は, 乗算を加算器とシフトで構成し冗長部分を共有化し, コストを削減する方法 [1], [2] を用いることが多い. この共有化の組み合わせは, 定数の個数やビット数が多くなると膨大になる. そのため, MCM回路の加算器数や加算段数が最小になる組み合わせの探索問題は, 組み合わせ最適化問題として知られており, 最適化アルゴリズムを用いた設計手法が提案されている [3].

最適化アルゴリズムは解生成と評価を大量に繰り返すため, 実用的な時間で大規模な回路設計を行う場合は, 高速化を図る必要がある. 一般的に最適化アルゴリズムは, ソフトウェア実装されるため, コンパイラオプションや, 計算処理の改善などの方法によって高速化を行う. しかし, いずれの方法も数% ～数倍程度の高速化しか見込めないことが多いのが現状である. また, 複数のプロセッサ, コンピュータの並列処理により高速化を図る手法もあるが, 消費電力や設備投資, 通信などの問題がある.

これに対し, 近年の再構築可能なLSI(Field Programmable Gate Array : FPGA) の普及により, アルゴリズムをハードウェア上に実装し, 高速化を図ることが容易に行えるようになった. しかし, 現在のFPGAの動作クロックはCPUに比べ圧倒的に遅いため, 単純にアルゴリズムをその動作どおりにハードウェア実装する手法 (Software-Based Haedware : SBH) [4] では高速化は困難であり, ソフトウェア実装より低速になることもある.

そこで本研究では, 数十MHzで動作するようなFPGAにおいて, MCM回路の最適設計を高速化するために, ソフトウェアではボトルネックとなる繰り返し処理に着目

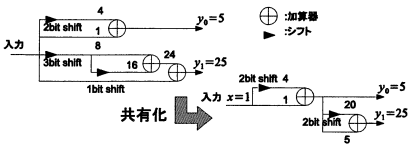

図 1　MCM 回路における演算の共有

し, FPGA の構造に適した並列化方法や, 並列化粒度を持つアルゴリズムを提案する. さらに FPGA に実装し, 提案法の有効性について検証する.

2. 複数定数乗算 (MCM) 回路の最適設計

MCM 回路とは, x を入力, a_i $(i = 0, 1, 2, \cdots N-1)$ を N 個の定数とすると

$$y_i = a_i x \quad (i = 0, 1, 2, \cdots N-1) \tag{1}$$

と表現される. この回路の合成には, 定数を個々に作成し入力変数と乗算させる単純な方法が考えられるが, 生成される回路は冗長部分が多く, 必要となる加算器数が増えてしまう. そのため, 回路規模や遅延時間・消費電力の制限がある場合は, 乗算を加算器とシフトで構成し冗長部分を共有化する (図 1). 共有化は定数の個数やビット数が大きくなれば組み合わせは膨大になる. そのため, 共有化は様々な方法が提案されており, 代表的なものとして定数を 2 進数で表し, 共通部分を共有化する方法 (CSE) [1] や, 加減算器とシフトからなる非巡回有向グラフを用いて共通部分を共有化する方法 (GD) [2] がある.

これらの回路合成アルゴリズムは, 決定的手法であり一意の解を求めることができるが, 必ずしもコストを最小にするものではない. 加算器数や加算段数などの設計条件に厳しい制約がある場合, 決定的手法では不十分で, コストを最小にする設計方法が必要になる. そのため, 遺伝的アルゴリズム (GA) 等の最適化アルゴリズムを取り入れる手法が提案されている [3]. 最適化アルゴリズムを用いると, コスト的にさらに優れた MCM 回路を設計することが可能となるが, 回路規模の増加に伴い, 設計に要する時間が非常に長くなるという問題がある. そのため, 設計アルゴリズムの高速化が必要となるが, ソフトウェア, SBH 実装では, 数桁倍以上の高速化は困難である.

そこで本研究では, 最適設計アルゴリズムをハードウェア実装する際に, ソフトウェア実装ではボトルネックとなる繰り返し処理に着目し, FPGA の構造に適した並列化方法や, 並列化・パイプライン化の粒度を持つアルゴリズムによって MCM 回路の最適設計の高速化を提案する.

3. ハードウェア実装のための MCM 回路最適設計アルゴリズム

本研究では, ハードウェア実装を前提として, 以下のアルゴリズムを用いる. 最適化アルゴリズムとしては GA を用いる. また, MCM 回路を生成し, 加算器数を適応度とした評価部分に用いる回路合成アルゴリズムには, 文献 [1](IMA) を基にしたものを使用する.

GA を用いる背景には, 単点探索よりも多点探索のほうが並列実装できるため高速化が期待でき, さらに他の多点探索のアルゴリズムに比べて構造が簡単であるという理由がある. また, IMA は共有部分の探索に定数を 2 進数で表現し AND 演算を行い, 非零ビットを数える方法を用いる. これは非常にハードウェアとの親和性が高いという理由から基にするアルゴリズムとして選択した.

ここでは GA の構成として, 評価に用いる回路合成アルゴリズム, そして遺伝子表現と適応度計算について記す.

3.1 MCM 回路合成アルゴリズム

IMA をハードウェアで実装する場合, 定数をシリアル転送することができる. これを積極的に利用すると, 定数のビットシフトが可能になり, 配線数も減少するので回路規模削減や, 高速動作が期待できる. このアルゴリズムの動作は以下のとおりである.

対象の定数列を $A = \{a_i | i = 0, 1, \cdots, N-1\}$ とし, 作業用定数列 $B = \{b_i | i = 0, 1, \cdots, N_b - 1\}$ に, 初期値 $N_b = N$, $b_i = a_i (i = 0, 1, \cdots, N_b - 1)$ を代入する.

STEP 1　定数列 B 中の異なる 2 つの定数 b_i, b_j に対して,b_i を s ビット左シフトさせた値と,b_j との論理積を $c = (b_i \ll s)$ & b_j として求める. c の非零ビット数を n_c とし, n_c が最も多くなるような $\{i, j, s\}$ の組み合わせを探索する. $n_c < 3$ の場合は終了する.

STEP 2　STEP 1 で探索した $\{i, j, s\}$ を基に, $b_i = b_i - (c \gg s)$, $b_j = b_j - c$, $b_{N_b} = c \gg s$, $N_b = N_b + 1$ のように定数列 B を更新し, STEP 1 へ戻る.

3.2 遺伝子表現

IMA は定数の共有化の繰り返しにおいて, 前半の組み合わせのほうが合成される回路への影響力が大きい. そこで GA で用いる遺伝子を, 前半 G 回の共有化の組み合わせとした (図 2). この $\{i, j, s\}$ を G 個まとめたものを 1 つの遺伝子とする. つまり先述のアルゴリズムにおいて前半 G 回は, 遺伝子が持つ $\{i, j, s\}$ を使用し, STEP 2 のみを行うことになる. そして G 回より後半は回路合成アルゴリズムによって共有化が行われる.

3.3 適応度計算

回路合成アルゴリズムによって作成された回路の加算器数を $adder$ とすると, 適応度 $Fitness$ は $Fitness = \frac{1}{adder}$ と定義する. この値が大きいほど適応度が高い.

4. 最適設計アルゴリズムのハードウェア実装

GA を用いた最適設計アルゴリズムを実現するための

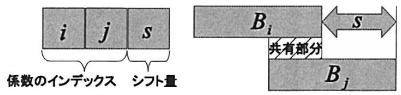

図 2　GA の遺伝子表現

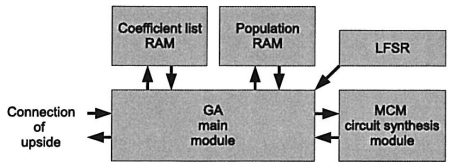

図 3　最適設計アルゴリズムのハードウェア構成

ハードウェア構成を図 3 に示す. GA main module は世代カウンタ, 選択回路, 交叉回路, 突然変異回路などから構成され, 評価には MCM 回路合成モジュールを使用している. その他に定数を保存する Coefficient list RAM, 個体群を保存する Population RAM, そして乱数を生成する LFSR モジュールからなる. 以下に詳細を述べる.

4.1　回路合成モジュール

回路合成アルゴリズムのハードウェア構成を図 4 に示す. shared module は, 合成対象の定数の入力, 作業用定数の更新, 合成結果の出力など行う. 合成アルゴリズム中で最も時間のかかる部分が STEP 1 であり, これを実行するモジュールは search module として独立している.

search module は並列化された複数の systolic array module から成り立っている. 図 4 の "1" と記された systolic array module には $B_0 \sim B_{N-1}$ の定数が接続され, 順次 $B_1 \sim B_{N-1}$ と続き最後には $B_{N-2} \sim B_{N-1}$ と接続されるようになっている. それぞれの systolic array module のハードウェア構成を図 5 に示す. shared module から転送された定数は, selector を通り shift register に保存され systolic array に転送される.

systolic array は各 Processing Element(PE) が単純な構造で, PE 同士の通信は隣接間のみ, データの入出力は端にある PE に行えばよいので, 集積回路として実現しやすい特徴を備えており, とりわけアイランドタイル FPGA と親和性が高い. 本研究の systolic array は, 図 5 の一番上のシフトレジスタに入る定数と, 他の定数との共通ビット数を求める演算を行う. 実際には, シリアル転送された定数を PE によって AND 演算後, 非零ビット数を数える. PE からのもう 1 つの出力である 1 クロック遅延させた定数は, selector に戻り, 再び shift register に入力される. これにより定数のビットシフトと同等の機能を実現することができる. 共通ビット数をカウントした後, comparison module で共通ビット数が最も多い組み合わせ $\{i, j, s, n_c\}$ を探索し, shared module に返す.

shared module では search module によって探索された組み合わせ $\{i, j, s, n_c\}$ を受信し, 共通ビット数 $n_c \geqq 3$

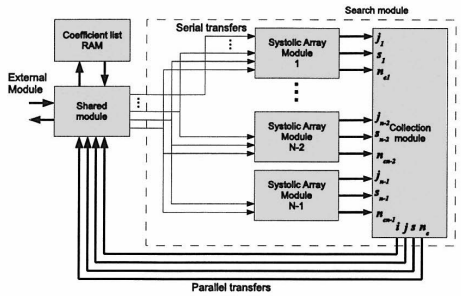

図 4　回路合成モジュールの構成

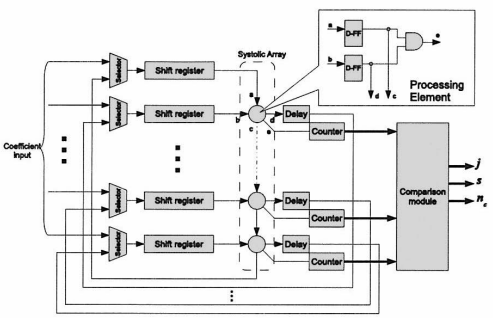

図 5　systolic array module の構成

であれば, $\{i, j, s\}$ によって定数列 B の更新と追加を行う. $n_c < 3$ の場合は終了する.

4.2　LFSR

LFSR(Linear Feedback Shift Register) は, 数個のシフトレジスタと 1 つの XOR で構成される疑似乱数生成器である. 構造が単純なためハードウェア実装に適している. 1 クロックで乱数が発生できるので, 効率が良い.

4.3　選択回路

乱数を個体選択用レジスタ $SEREG$ に保存し, Population RAM のポインタとすることで実現する.

4.4　交叉回路

親遺伝子 G_a, G_b, 子孫遺伝子 G_x, G_y とすると $G_x = G_a \cdot COREG + G_b \cdot \overline{COREG}, G_y = G_a \cdot \overline{COREG} + G_b \cdot COREG$ と定義する. $COREG$ は, 遺伝子長分のシフトレジスタで構成され, 乱数を保存する. 実装にはセレクタを用いた. 1 回の動作で一様交叉が実現する.

4.5　突然変異回路

突然変異率 MR と乱数 R を整数値化し, $MR > R$ のときに突然変異を行う. 突然変異率は $0 < MR < 1$ であるので定数 R を乗じて整数値化する. 突然変異は 遺伝子 G_a と乱数との XOR によって実現させる. 交叉回路と同様に, 1 回の動作で突然変異が実現できる.

4.6　世代交代モデル

世代交代モデルには Elite Recombination (ER) [5] を

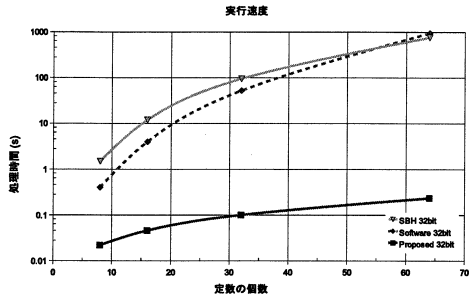

図 6　32bit の定数における動作速度

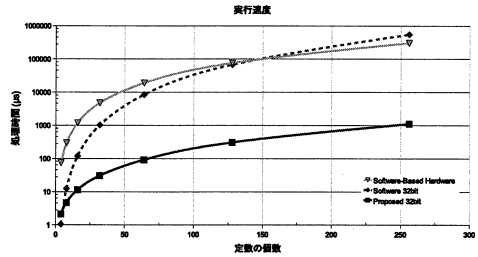

図 7　回路合成モジュール単体の動作速度

使用した. 通常の世代交代モデルのように適応度順序を求め, 保持する必要が無いためレジスタの量を削減できる.

5. 結果と考察

提案法の有効性を評価するために, FPGA に実装し動作速度の比較を行った. コンピュータ上にソフトウェア実装したもの (以下 Software) とソフトウェアをベースにハードウェア実装したもの (SBH) を比較対象とした.

提案法と SBH には以下のハードウェア環境を使用した.

- 高速演算 FPGA ボード TD-BD-Bioler3 :
  FPGA Xilinx VirtexII Pro 33088Slices 66MHz
  (Xilinx ISE Foundation 8.2i)

また, Software には以下の環境を使用した.

- PC WindowsXP Xeon 3.4GHz Memory:1GB
  (Microsoft Visual C++ 2005)

定数のビット数を 32bit とし, 定数の数を 8,16,32,64 個の場合の評価を行った. その時の GA パラメータは 200 世代, 個体数はそれぞれ, 5, 10, 20, 40 個, 交叉率 0.8, 突然変異率 0.08, 遺伝子長 3 とした.

それぞれの手法による動作速度の結果を図 6 に示す. 提案法は定数の個数が増えるに従い, Software に対して高速化の割合が増加しており, 定数が 64 個の場合にはおよそ 4000 倍の高速化が実現できている. 一方で, 同じハードウェア実装である SBH は高速化できていない. 定数が 64 個の場合で, Software に対して 1.2 倍程度高速になるものの, それ以下では Software よりも遅いという結果になっている. このようにハードウェア実装の方法によって動作速度は大きく左右されることがわかる. この結果は, 以下のように考察される.

まず, 最適設計アルゴリズムの実行時間における評価部分が占める時間と割合を求めた. 定数が 32bit, 64 個の場合の Software では, 評価処理が全体の 98.3% を占める結果となった. このことから評価処理の大部分を占める回路合成モジュールの高速化が, 全体の最適設計の高速化に大きく寄与していることがわかる.

回路合成モジュールの実装方法による消費クロックの違いは次のように説明できる. SBH の消費クロック数 CLK_S は, 定数の個数 N, 定数のビット数 L とすると, $CLK_S = \frac{1}{2}(N^2 - N) \cdot L$ と近似できる. これに対して, 提案法の消費クロック数 CLK_P は $CLK_P = L^2 + L + 3N$ と近似できる. ここで, L は 16bit や 32bit 程度と固定されることが多いので, 実際上 N が大きくなるに従い提案法の高速化の割合が大きくなることがわかる.

実際に回路合成モジュール単体での動作速度を比較すると図 7 のようになる. 図 6 の結果と同様に, 提案法は定数の個数が増えるに従い, Software, SBH に対しての高速化の割合が増加している.

6. むすび

本研究では MCM 回路の最適設計の高速化のため, FPGA の構造に適した並列化方法や, 並列・パイプラインの粒度を持つアルゴリズムを提案し FPGA 上に実装した. その際に GA の評価部分の回路合成アルゴリズムが, ソフトウェア実装ではボトルネックとなる繰り返し処理であることに着目し, それらを systolic array を用いた並列構造によって FPGA に適用させ, 高速化を実現した.

文　献

[1] M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, "Multiple constant multiplication: Efficient and versatile framework and algorithms for exploring common subexpression elimination", IEEE Trans. Integrated Circuits & Syst., vol.15, no.2, pp.151-165, Feb. 1996.

[2] D. Bull, D. Horrocks, "Primitive operator digital filters", IEEE Proceedings-G Vol.138, pp.401?411, Jun. 1991.

[3] T.Sasaki and H.Toyoshima, "An Optimal Synthesis Method for Multiple Constant Multiplication Circuits With Consideration of the Order of Coefficients to be Synthesized", Proc. 10th Int. Symp. Integrated Circuits, Devices & Systems, F1025, pp.1-4, Sep. 2004.

[4] S.V. Wunnava and V.L. Patil, "VHDL: Software Based Hardware Designs", Proceedings IEEE Southeastcon '98, pp.392–396, Apr. 1998.

[5] Thierens, D. and Goldberg, D.E., "Elitist Recombination: an integrated selection recombination GA", Proceedings of the First IEEE Conference on Evolutionary Computation, pp.508-512, Jun. 1994.