

JPEG2000 用ウェーブレット変換器の アーキテクチャ設計とその評価

上田恭子[†], 武内良典[†], 北嶋暁^{††}, 今井正治[†]

[†]大阪大学 大学院基礎工学研究科

大阪府豊中市待兼山町 1-3

Tel : 06-6850-6626

^{††}大阪電気通信大学 総合情報学部情報工学科

大阪府寝屋川市初町 18-8

Tel : 072-824-1131

E-mail : k-ueda@ics.es.osaka-u.ac.jp

あらまし 本稿では, JPEG2000 用ウェーブレット変換器の VLSI 向きアーキテクチャを提案する. 提案するウェーブレット変換器はウェーブレット変換の 4 つのステップを並列に行ない, また乗算結果を再利用するアルゴリズムの改良により, 従来型の変換器よりも乗算回数を減らすことができる. 評価の結果, 提案するアーキテクチャは, これまでに提案されていたアーキテクチャよりも実行時間および面積の点で優れていることが知られた.

キーワード JPEG2000, ウェーブレット変換, リフティング法

Architecture design and evaluation of wavelet transform unit for JPEG2000

Kyoko Ueda[†], Yoshinori Takeuchi[†], Akira Kitajima^{††}, Masaharu Imai[†]

[†]Graduate School of Engineering Science,

Osaka University

Machikaneyama 1-3, Toyonaka, Osaka,

560-8531 Japan

Tel : 06-6850-6626

^{††}Faculty of Information Science and Technology,

Osaka Electro-Communication University

18-8, Hatsu-cho, Neyagawa, Osaka,

572-8530, Japan

Tel : 072-824-1131

E-mail : k-ueda@ics.es.osaka-u.ac.jp

Abstract This report proposes a VLSI architecture to implement wavelet transform for JPEG2000 image processing. By processing 4 steps in the wavelet transform in a pipelined manner, the wavelet transform unit can transform an image more efficiently than sequential implementation. In addition, the improved architecture can reduce the number of multiplications by reusing a multiplied result, compared with a conventional wavelet transform architecture. According to experiments, the proposed architecture can achieve shorter processing time with smaller area compared to a conventional wavelet transform unit.

key words JPEG2000, wavelet transform, lifting scheme

1 はじめに

近年、音声、画像、映像などを含めた様々なメディアがデジタル化され、通信応用、蓄積目的においても各種データはメディアの種類に依存しない単一的取り扱いが可能となってきた。しかしながら、そのデータ量は膨大なものとなっており、データ圧縮アルゴリズムが必要不可欠となっている [1]。とくに、音声、画像、映像などデジタル化された信号に対しては、圧縮率の高い非可逆アルゴリズムが提案されている。これら圧縮率の高い非可逆アルゴリズムは、各種メディアの特徴を利用したデータ圧縮アルゴリズムとなっている。

これまで、静止画像データの圧縮アルゴリズムとしては、JPEG が用いられていた [2]。JPEG アルゴリズムは、離散コサイン変換 (DCT) の直交変換を利用したアルゴリズムで、DCT の後、量子化、エントロピー符号化を行っている。DCT は、近年の急速なハードウェア技術により、それまでの Hadamard 変換に代わり、最近の信号処理アルゴリズムでは、頻繁に用いられるようになっていく。しかしながら、JPEG アルゴリズムで用いられている DCT は、演算量低減のため、8x8 ピクセル程度の大きさの画像に対して、処理を行うため、ブロック境界に歪みを発生させてしまうという問題がある。

JPEG2000 では、離散コサイン変換に代わりウェーブレット変換に基づくアルゴリズムが使用されている [3]。JPEG2000 は、新しい画像圧縮アルゴリズムで、JPEG よりも圧縮率が高くかつ品質の良いアルゴリズムとして制定された。しかしながら、JPEG2000 には、JPEG と比較して、演算量、使用するメモリ量が増加するなどの問題点がある。本稿では、これらの問題を解決するための、ウェーブレット変換の VLSI 向きアーキテクチャの提案を行う。

本稿の構成は次のとおりである。まず、第2節で JPEG2000 アルゴリズムで用いられるウェーブレット変換の概要について説明する。次に、第3節では従来及び提案するウェーブレット変換器の VLSI 向きアーキテクチャについて説明する。4節では、第3節で提案する変換器の評価を行い、第5節で本稿をまとめる。

2 JPEG2000 とウェーブレット変換

2.1 JPEG2000

JPEG2000 での画像圧縮手順は次のとおりである [3]。

1. 領域変換
2. ウェーブレット変換
3. 量子化
4. エントロピー符号化

従来の JPEG で用いられている離散コサイン変換では画像を 8 ピクセル四方のブロックに分解して処理が行なわれるため、量子化を粗く行なった場合にブロック歪みが生じることがある [2]。しかし JPEG2000 では、画像をブロックに分割せずに処理を行なうウェーブレット変換を採用しているため、ブロック歪みは発生しない。また、JPEG2000 は従来の JPEG アルゴリズムよりも圧縮率が高いことが知られている。しかし、JPEG2000 アルゴリズムは JPEG アルゴリズムに比べて数倍の計算量を必要とするため、低コスト、低消費電力、高速な処理を実現するためには、専用演算器の導入などの工夫が必要となる [4]。そこで、本稿では、JPEG2000 処理の特徴的な処理であるウェーブレット変換の VLSI 向きアーキテクチャについて検討した。

2.2 ウェーブレット変換

2.2.1 ウェーブレット変換の概要

ウェーブレット変換とは、任意の関数を、マザー・ウェーブレットと呼ばれる基準となる関数で表すことのできる部分とそれ以外の部分に分解することである [5]。ウェーブレット変換は連続ウェーブレット変換、離散ウェーブレット変換と離散時間ウェーブレット変換の 3 つに大別される。画像処理など、デジタル信号の圧縮や解析には離散時間ウェーブレット変換が用いられており、JPEG2000 でも離散時間ウェーブレット変換が用いられる。以降、離散時間ウェーブレット変換を単にウェーブレット変換と表す。

ウェーブレット変換は図 1 のフィルタバンクを用いて実現できる [6]。図 1 の回路の順変換では、入力データをハイパスフィルタ (High Pass Filter, HPF) とローパスフィルタ (Low Pass Filter, LPF) によって変換し、ダウンサンブラによって間引きを行なう。HPF の出力データがマザー・ウェーブレットを用いて表すことのできる部分となる。逆変換は、まず、アップサンブラを用いてデータのアップサンプリングを行ない、その後 HPF, LPF 処理を行なったデータを足し合わせることで実行される。

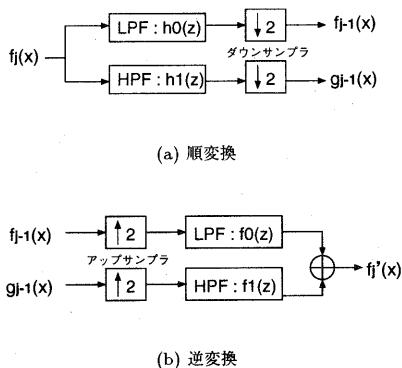


図 1: ウェーブレット変換のフィルタバンク

2.2.2 リフティング法

2.2.1節で述べたフィルタバンクで、フィルタの伝達関数が n 次の式 $a_0 + a_1 z^{-1} + \dots + a_n z^{-n}$ で表される場合には、計算する時刻の n 単位時間前までの入力データが必要である。また、フィルタバンクでは HPF, LPF の出力をダウンスンプラによって半分に間引くため、HPF, LPF のすべての出力が必要とされるわけではない。そこで、フィルタの複雑さを軽減し不要な計算を省くためにリフティング構成が提案されている [7]。

リフティング法は図 2 のブロック図で表現される。ここで、 $s_i(z), t_i(z)$ はフィルタであり、 z は先見を行なう素子、 z^{-1} は遅延器である。この方法では、入力データをダウンスamplingと先見によって偶数列のデータ、奇数列のデータに分けて変換を行なう。リフティング法では偶数列、奇数列間の演算の中間結果を相互に利用して演算を進め、最後に K 倍または $1/K$ 倍によるスケーリングを行なう。データの更新はステップ、更新の回数はステップ数と呼ばれる。リフティング法はフィルタバンクと違い、入力を先に分割する方法を採用しているため不要な計算を行なうことがない。

一般に、リフティング法を用いることで、フィルタの複雑さが軽減され、乗算器や遅延器の削減が可能になる。したがって、リフティング法はハードウェアによる実装に適した計算方法であると言える。また、リフティング法を用いる場合、各ステップは並列に動作させることが可能であり処理の高速化が実現できるため、ウェーブレット変換は専用演算器としての実現が有効であると考えられる。そこで、リフティング法に基づくウェーブレット変換を専用演算器として実現することにした。

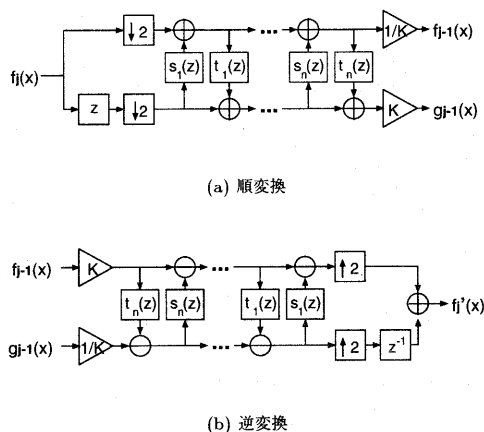


図 2: リフティング法のブロック図

2.2.3 入力データの補完

JPEG2000 におけるウェーブレット変換のデータの更新は、式 (1) に従って実行される。

$$y(2n+1) = Ax(2n) + x(2n+1) + Ax(2n+2) \quad (1)$$

ここで、 $x(n), y(n)$ はそれぞれ、時刻 n における入力データ、出力データである。この時、入力データが $x(1)$ から始まる場合、 $y(1)$ を得るための $x(0)$ が存在しないことになる。そこで入力データは折り返していると仮定し、不足データを式 (2) で補う。

$$x(i) = \begin{cases} x(2m-i), & i < m \\ x(2n-i), & i > n \end{cases} \quad (2)$$

ここで、 m, n はそれぞれ、入力データの開始点、終了点を表す。

3 ウェーブレット変換器のハードウェアアルゴリズム

3.1 2入力加算に分解したアルゴリズム

フィルタのステップは式 (1) で表されるため、図 3(a) に示すとおり 3 個の値を加算することで出力データが得られる。ここで、図 3 の A は辺を通過するデータの A 倍の乗算を表す。しかし入力データが連続して入力される場合、必要な値を求めるには 2 時刻前までの入力データを保持しておく必要がある。しかし、図 3(b) に示すよう

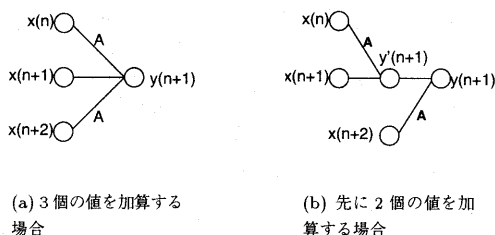


図 3: データの加算方法

に先に到着する 2 個の値を加算した結果を保持しておくことで、保持する必要がある値が 1 個減りハードウェアの面積を小さくすることが可能である。この方法は式 (3) で表わされる。

$$\begin{aligned} y'(n+1) &= Ax(n) + x(n+1) \\ y(n+1) &= y'(n+1) + Ax(n+2) \end{aligned} \quad (3)$$

以後このアルゴリズムに基づいたアーキテクチャを ARCH1 と呼ぶこととする。

3.2 提案するアルゴリズム

入力データの補完が必要である場合、フィルタのステップでは図 4(a) に示すように式 (4) の演算を行なう。

$$\begin{aligned} x(n) &= x(n+2) \\ y(n+1) &= Ax(n) + x(n+1) + Ax(n+2) \end{aligned} \quad (4)$$

$x(n) = x(n+2)$ であるから 2 つの乗算結果は全く同じものである。そこで式 (5) の方法で乗算結果を再利用することが考えられる (図 4(b))。

$$\begin{aligned} y'(n+2) &= Ax(n+2) \\ y(n+1) &= y'(n+2) + x(n+1) + y'(n+2) \end{aligned} \quad (5)$$

乗算結果の再利用により乗算回数が 1 回減少し、より少ない実行サイクル数で処理が可能となる。乗算結果を再利用する方法を用いる場合、ARCH1 では、式 (3) の $y'(n+1)$ の値を遅延させずに用いる必要がある。そこで 3 個の値を加算するという構造を持たせた上で乗算結果を再利用するアルゴリズムを提案する。

以後このアルゴリズムに基づいたアーキテクチャを ARCH2 と呼ぶこととする。

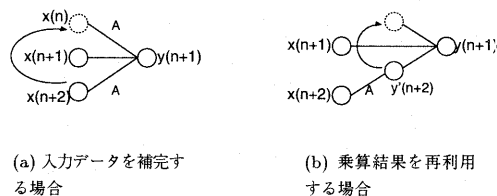


図 4: データの補完

4 ウェーブレット変換器の実装と評価

4.1 演算精度

浮動小数点演算は固定小数点演算に比べて処理量が多く構造も複雑になるため、今回の実装では固定小数点演算を用いることにした。面積や遅延を考えた場合、ビット幅は短いことが望まれるが、画像の質を保つためには長いビット幅が必要である。そこで、両者のトレードオフを考慮し、適切なビット幅を求めるために、式 (6) で与えられる信号対雑音比 (Signal to Noise Ratio, SNR)[8] を用いて画質を評価した。

$$SNR[\text{dB}] = 10 \log \sum_{i=1}^n \frac{x_i^2}{(x_i - x'_i)^2} \quad (6)$$

ここで、 n は画素数、 x_i は原画像の画素値、 x'_i は伸長画像の画素値を表す。今回の実験では変換後の SNR が 30dB 以上を満たすビット幅を採用することとした。小数部分のビット幅を変化させて SNR を求めた結果の一部を表 1 に示す。表 1 より、小数部分に 5 ビットを割り当てることによって画像の質を 30dB 以上に保った処理が可能であると判断した。原画像の 1 画素を -128 から 127 の整数値で表すため、データの整数部分は、2 ビットのガードビットを含めた 10 ビットで表現することにした。以上の結果より、データを、整数部分 10 ビット (符号含む)、小数部分 5 ビットの計 15 ビットで表現することにした。

4.2 ウェーブレット変換器の構造

4.2.1 ウェーブレット変換器全体の構造

ウェーブレット変換器を実装するにあたって、図 5 の構造を持たせることにした。この構造は ARCH1, ARCH2 に共通である。

図 5 の各ブロックは次の処理を行なう。

CORE1 ステップ 1 の処理を行なう。入力データを nxt

表 1: 固定小数点ビット幅と画像品質

画像名 (Grayscale)	小数部分のビット幅 [bits]				
	3	4	5	6	7
baboon	31.45	34.09	38.96	42.88	43.55
barbara	30.19	29.28	34.46	39.99	40.92
lena	31.36	32.36	37.16	41.71	42.30
milkdrop	27.99	27.64	32.93	38.56	39.54
peppers	29.39	27.41	32.53	38.87	39.72

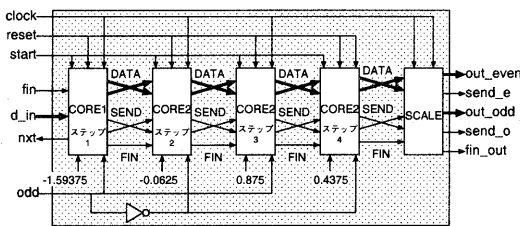


図 5: ウェーブレット変換器の構造

信号によって制御し、出力データをステップ 2 に渡す。

CORE2 ステップ 2, 3, 4 の処理を行なう。入力データは前のステップから順時送られる。

SCALE スケーリングを行なう。

各ブロック間の DATA, SEND, FIN はそれぞれ、次のブロックへの入力データ、入力データの到着時に ON となる信号、最後の入力データである時に ON となる信号である。各ステップの出力は偶数列と奇数列に分割しているため、ステップ 2 以降は偶数列、奇数列の入力データを分割したまま入力する。また、odd 信号は入力データが奇数列からである時に ON となる信号である。入力信号の始まりは前のステップと逆になるため、ステップ 1, 3 では odd 信号をそのまま採用し、ステップ 2, 4 ではその否定を採用している。図 5 の各ステップに入力されている実数は、 9×7 フィルタで用いられている係数 (式 (1) の A) である。

4.2.2 ARCH1 の内部構造

ARCH1 の内部構造を示す。図 5 の CORE1 および CORE2 の構造をそれぞれ図 6 および図 7 に示す。

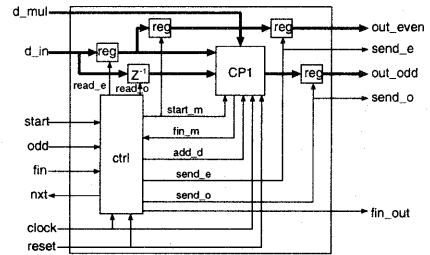


図 6: ARCH1/CORE1 の構造

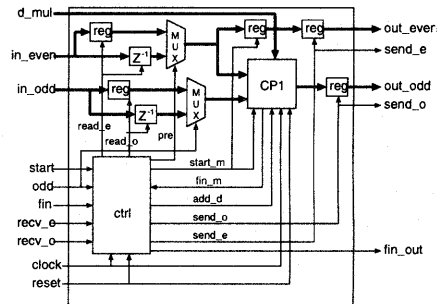


図 7: ARCH1/CORE2 の構造

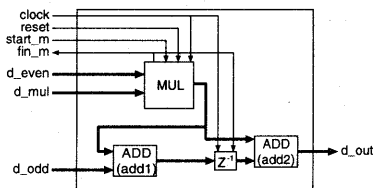


図 8: CP1 の構造

各ブロックは次の処理を行なう。

ctrl CORE の動作を制御する
 CP1 式 (3) の演算を行なう
 reg レジスタ
 z^{-1} 遅延器
 MUX セレクタ

CORE1 では nxt 信号によって入力データを制御することで、CORE2 では入力データと入力データを遅延させたデータのどちらかを選択することで入力データの補完を行なう。

CORE1 および CORE2 内の CP1 の構造を図 8 に示す。CP1 の MUL, ADD はそれぞれ逐次型乗算器と加算器である。偶数列の入力データを乗算した結果と、奇数列の入力データを CP1 の add1 で加算し、その結果を遅延器で遅延させる。遅延させた値と乗算結果を add2 で加算することで式 (3) の $y(n+1)$ が得られる。

4.2.3 ARCH2 の内部構造

ARCH2 の CORE1 および CORE2 の構造を図 9, 図 10 に示す。CORE1, CORE2 の CP2 は式 (5) の演算を行なうブロックである。

CP2 の構造を図 11 に示す。CP1 のように入力データを補完するのではなく、乗算結果を再利用する。時刻 n において補完を行なう必要がある場合は、MUX の出力を時刻 n における乗算器 MUL の出力、すなわち式 (5) の $y'(2n+2)$ とすることで式 (5) の $y(2n+1)$ が得られる。

4.3 ウェーブレット変換器の実行サイクル数

入力データの補完が必要となる場合は、奇数列のデータから入力された場合である。したがってステップ 1 への入力データが奇数列からの場合、ステップ 3 への入力データも奇数列から、ステップ 2 と 4 へは偶数列からとなる。したがって、データの拡張は 2 回必要である。ステップ 1 への入力データが偶数列からの場合も、同様に 2 回の補完が必要となるため、常に補完が 2 回必要となる。したがって、ARCH2 は ARCH1 と比べて乗算回数が 2 回少ないことになる。その結果、乗算器の実行サイクル数を K とすると、ウェーブレット変換の実行サイクル数は少なくとも $2K$ サイクル減らすことが可能となる。ARCH1 ではデータの補完の際に入力データの制御が必要となるため実際には ARCH2 の実行サイクル数は ARCH1 よりも $2K+10$ サイクル少なくなる。表 2 に両

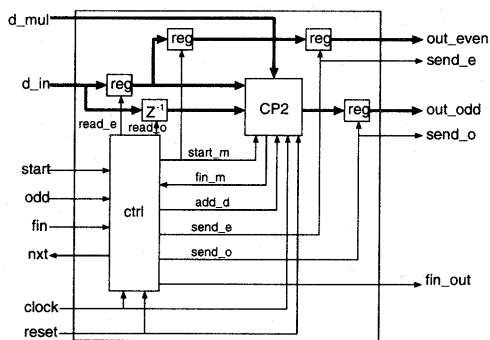


図 9: ARCH2/CORE1 の構造

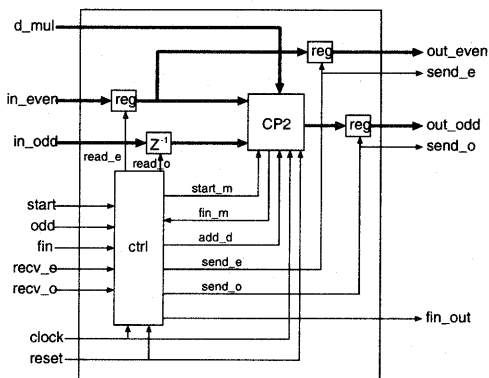


図 10: ARCH2/CORE2 の構造

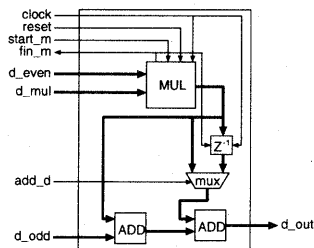


図 11: CP2 の構造

表 2: ウェーブレット変換器の実行サイクル数

	実行サイクル数
ARCH1	$22 + 9 * K + \left\lceil \frac{K+2}{2} \right\rceil * \text{入力データ数}$
ARCH2	$12 + 7 * K + \left\lceil \frac{K+2}{2} \right\rceil * \text{入力データ数}$

表 3: ウェーブレット変換器の面積 (単位:gates)

クロック周期 [ns]	ARCH1	ARCH2
5	15182	13886
10	14346	12533

者の実行サイクル数を示す。

今回の実装では実行サイクル数が10サイクルの逐次乗算器を用いたため、ARCH2の実行サイクル数はARCH1よりも30サイクル少なくなった。

4.4 ウェーブレット変換器の面積および性能評価

3節で述べたアルゴリズムを持つ変換器をVHDLで記述し、次のライブラリとツールを用いて論理合成を行った。

ライブラリ

CMOS 0.18 μ m スタンダードセルライブラリ

論理合成ツール

Synopsys 社 Design Compiler 2000.06

クロック周期を5, 10nsと設定した場合の面積の合成結果を表3に示す。

両者のアーキテクチャを比較すると、ARCH2の方が約10%面積が小さいことがわかる。ARCH1ではデータの補完のための遅延器などが影響して面積が大きくなると考えられる。また実行サイクル数はARCH2の方が少ないため実行時間も短縮される。一般に、面積と処理時間の間にはトレードオフが存在するにもかかわらず、提案するARCH2はARCH1よりも面積と処理時間の両面において有利であると知られた。

4.5 考察

4.5.1 ウェーブレット変換器の導入による効果

ウェーブレット変換器を導入した場合の効果について考察する。1秒間に30枚処理可能な画像のサイズを、汎用プロセッサを用いた場合と今回提案したARCH2のウェーブレット変換器を導入した場合とで比較した。

表 4: 汎用プロセッサの仕様

プロセッサ名	PentiumII
クロック周波数	200MHz
メモリ	128MB
C コンパイラ	gcc version 2.95.2

表 5: ウェーブレット変換の処理時間 (動作周波数:200MHz)

	汎用 CPU	変換器
処理時間 [ms]	910.8	64.28

表 6: 1秒間で30枚処理可能な画素数 (動作周波数:200MHz)

	汎用 CPU	変換器
画素数 [画素]	9968	14645

表4に示す動作環境で、汎用プロセッサを用いて1024x768サイズの画像を圧縮する場合の、ウェーブレット変換の処理時間を表5に示す。"汎用 CPU"が汎用プロセッサで実現した場合、"変換器"がウェーブレット変換器を導入した場合である。

処理時間はほぼ画像サイズに比例すると考えられる。そこで、表5に示した処理時間より、1秒間に30枚処理可能な画像のサイズを推定した結果を表6に示す。1秒間で30枚処理可能な画素数は、汎用プロセッサの場合は約10000画素、ウェーブレット変換器を導入した場合は約15000画素である。よって1秒間で処理可能である画像の画素数が約1.5倍となるという結果が得られた。

5 おわりに

本稿ではJPEG2000の処理を行なうハードウェアの実現について検討した。ハードウェアとしての実現の第一歩として、JPEG2000の大きな特徴であるウェーブレット変換を専用演算器として実装した。新たにアーキテクチャを提案し、従来のアーキテクチャとの比較を行なった。比較の結果、実行サイクル数、面積共に従来のアーキテクチャよりも削減可能であることが確認された。また、JPEG2000の処理全体を汎用プロセッサで実現した

場合と実装したウェーブレット変換器を導入した場合を想定して比較を行なった。動作周波数 200MHz とした場合、1 秒間に 30 枚処理可能な画像のサイズが、ウェーブレット変換器を導入することで、汎用プロセッサで実現した場合の約 1.5 倍となることを確認した。ウェーブレット変換部分のハードウェア化により、汎用プロセッサでウェーブレット変換を行なう場合に比べて高速化が可能であることを示した。

今後の課題は、JPEG2000 における他の処理のハードウェア化について検討し、JPEG2000 全体の処理をリアルタイムで行なうことが可能であるハードウェアを実現することである。

謝辞

本研究を進めるにあたり、貴重なコメントを頂いた大阪大学 VLSI システム設計研究室の諸氏に深謝する。

参考文献

- [1] "JPEG2000 は今後の画像圧縮の主流となりうるか?,"
'<http://www.zdnet.co.jp/magazine/pcmag/9904/t990421a.html>'.
- [2] K.R.Rao, P.Yip, "画像符号化技術 -DCT とその国際標準-",
オーム社, 1992.
- [3] "JPEG2000 Part I Final Committee Draft Version 1.0,"
16 March 2000.
- [4] "より小さくきれいに, JPEG-2000 制定へ,"
'<http://www.zdnet.co.jp/news/0012/04/jpeg2000.html>'.
- [5] 榊原進, "ウェーブレットビギナーズガイド,"
東京電機大学出版局, 1996.
- [6] Hitoshi KIYA, Hiroyuki KOBAYASHI and Osamu WATANABE, "Design of Integer Wavelet Filters for Image Compression,"
IEICE Trans.Fundamentals, vol.E83-A, no.3, pp.487-490, March 2000.
- [7] I.Daubechies and W.Sweldens, "Factoring Wavelet Transforms into Lifting Steps,"
J.Fourier Anal.Appl., vol.4, pp.247-269, 1998.
- [8] Brian A. Basister and Thomas R. Fischer, "Quantization Performance in SPIHT and Related Wavelet Image Compression Algorithms,"
IEEE Signal Processing Letters, vol.6, no.5, May 1999.