社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

Impulse C を用いた Line-Based アーキテクチャ二次元離散ウェーブレット変換の FPGA への実装

宮島　敬明†　　新井　正敏††　　天野　英晴†

† 慶應義塾大学理工学部 〒 223-8522 神奈川県横浜市港北区日吉 3-14-1
†† カルソニックカンセイ (株) 先行開発本部 〒 331-8501 埼玉県さいたま市北区日進町 2-1917
E-mail: †vision@am.ics.keio.ac.jp, ††ARAI-MASHI@ck-mail.com

あらまし　二次元の離散ウェーブレット変換は JPEG 2000 で画像の圧縮に採用されており、従来の JPEG に用いられる離散コサイン変換よりも原理的にノイズの発生が少ない。しかし、離散コサイン変換に比べ圧縮率が低く、その計算過程もメモリアクセスが頻繁で組み込み用途には不向きである。画像処理のようなホストプロセッサとのデータ通信が欠かせないアプリケーションのハードウェア実装では、通信方式やオフロードする処理の選定に一定の実装経験とプロトタイピングが必要となる。その際、コストや開発期間の短縮のため FPGA が用いられるが、複雑なアプリケーションでは、プロトタイピングそのものに時間がかかってしまう。その解決策として近年注目されている高位合成言語は、標準 ANSI-C から RTL を自動生成することでコスト・開発期間の短縮化を図るものである。本研究では、高位合成言語 Impulse C を用いて、Daubechies のウェーブレットを基底とし、実用的な Line-Based アーキテクチャの二次元離散ウェーブレット変換を FPGA 上への実装を試みた。その結果、ソフトウェアでの実行時間と比較して 5.29 倍程度の性能を達成した。

キーワード　FPGA，離散ウェーブレット変換，Impulse C, Daubechies のウェーブレット, 高位合成言語, Line-Based Architecture

An FPGA Implementation of Line-Based Architecture 2-Dimensional Discrete Wavelet Transform Using Impulse C

Takaaki MIYAJIMA†, Masatoshi ARAI††, and Hideharu AMANO†

† Faculty of Science and Technology, Keio University 3-14-1, Hiyoshi, Kohokuku, Yokohama, 223-8522, Japan
†† CalsonicKansei Co. 2-1917, Nisshincho, Kitaku, Saitama, 331-8501, Japan
E-mail: †vision@am.ics.keio.ac.jp, ††ARAI-MASHI@ck-mail.com

Abstract　2 Dimensional Discrete Wavelet Transform(2D-DWT) that is used for Image compression on JPEG2000, makes theoretically less noises than Discrete Cosine Transform(DCT) that is used on traditional JPEG. However, the compression ratio of DWT is lower than DCT and the procces of computation requires high frequently memory accesses, so that DWT is not suitable for embedded system. Hardware implementations of specific applications that have data transfer from host processor require certain implementation experience and prototyping to select a data transfer manner and a offload process. At the case of Prototype, FPGA is used to save cost and developing time. Although developing a prototype of large and complicated system takes long time iteself. High-Level Synthesis Language being gathered attention as a solution for this problem. It automatically generates RTL through Standard ANSI-C so as to shorten developing time. In this paper, we implemented practical Line-Based Architecture 2D-DWT that consists of Daubechies' wavelet using ImpulseC. As a result, we achived approximatly 5.29 times faster result than software execution.

Key words　FPGA, Discrete Wavelet Transform, Impulse C, Daubechies' Wavelet, High-Level Synthesis, Line-Based Architecture

1. はじめに

近年、携帯電話などの組み込み機器に搭載されるプロセッサの演算能力の向上、動画像専用の命令の追加により H.264 などの高負荷な圧縮・伸張アルゴリズムの処理にかかる時間も現実的なものとなってきている。

また、高精細テレビの普及に伴い、より高画質で高精細な動画像のニーズが高まっている。加えて、ストレージの大容量・低価格化、インターネット回線の高速化などにより動画像の容量面での制約はなくなりつつある。

しかし、動画像圧縮方式として最も普及している JPEG や H.264 では、離散コサイン変換を基礎としている。離散コサイン変換は処理が容易ではあるが、圧縮前の画像と圧縮後の画像が同一のものではない不可逆変換である。

また、離散コサイン変換は 8x8 画素のブロックに対して個々に行われるため、近傍画素との相関が強いという画像処理の特徴を効果的に利用していない。そのため、圧縮率を優先した場合、画像全体にブロックノイズと呼ばれる矩形ノイズが発生してしまう。加えて、離散コサイン変換はコサイン関数を元にしているおり、有限時間でインパルス応答を完全に再現できないため、急激な色や明るさの変化に弱く、それが原因となりモスキートノイズと呼ばれるもや状ノイズも発生してしまう。これは、JPEG 形式（JFIF 形式）の規格策定された 1980 年代半ばは、演算性能に大きな制約があったため、動画像の精細さよりも演算量の削減が優先されたからである。

そこで、動画像の標準使用策定団体である Joint Photographic Experts Group は 2000 年により高品質な画像圧縮を目指し、JPEG 2000 [1] [2] と呼ばれる新しい規格を策定した。この規格の中で最も注目すべき点は、可逆変換である離散ウェーブレット変換 (Discrete Wavelet Transform: DWT) を採用したことである。これにより理論的には、圧縮前と圧縮後の画像に違いがなくなった。また、変換の処理を行うブロックのサイズを任意に設定することができるようになったことで、ブロックノイズの低減に大きく貢献している。加えて、DWT はウェーブレット関数と呼ばれるある条件を満たす関数を元に信号を高周波成分と低周波成分に次々に分解していくもので、離散コサイン変換と違い急激な信号の変換にも耐性を持つ。しかし、DWT を組み込みハードウェア実装する場合の問題として、ブロックサイズの拡大などに伴うメモリ使用量の増加が挙げられる。

FPGA(Field Programmable Gate Array) を用いたリコンフィギャラブルシステムは、ハードウェア上で実装手法を素早く試作するものとして広く利用されている。また 2000 年以降、RTL よりも抽象度が高く、C 言語に似た記述を行う高位合成言語が実用化段階に入っている。両者の特性を生かすことで、プロトタイピングの高速化を計ることが可能とされている。Impulse C [3] は FPGA 専用の高位合成言語であり、ANSI-C から RTL を自動的に生成する。モジュールの回路記述を行う従来の HDL と異なり、シーケンシャルに動作を記述することで所要の機能をハードウェア化できる。これにより、アルゴリズムのソフトウェアからハードウェアへの移植が容易に行えるようになっている。

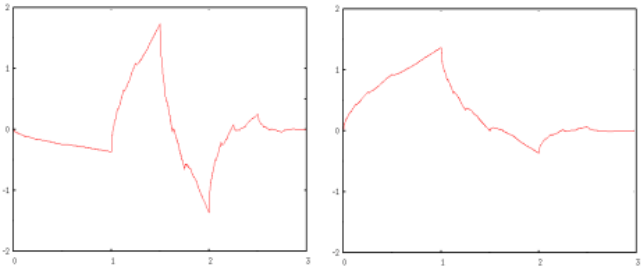

図 1　N=2 の Daubechies のウェーブレット (左) とスケーリング関数 (右)

本研究では、Zervas [4] らがまとめた DWT の基礎的な実装手法の中で、実用的とされる Line-Based アーキテクチャを用い、画像処理に最適な Daubechies' のウェーブレットをマザーウェーウレットとした離散ウェーブレット変換を ImpulseC を用いて FPGA 上に実装し、その評価を行った。

2. 離散ウェーブレット変換

2.1 ウェーブレットとスケーリング関数

DWT は、マザーウェーブレットから導かれるある条件を満たすウェーブレット $\psi_{j,k}(t)$…式 (1) とスケーリング関数 $\varphi_{j,k}(t)$…式 (2) をもとに、入力信号を高周波成分 H と低周波成分 L とに分解し、分解された低周波成分 L を再び高周波成分 LH と低周波成分 LL に連続的に分解していく変換である。

$$\psi_{j,k}(t) = 2^{\frac{-j}{2}}\psi(2^{-j}t - k), \tag{1}$$

$$\varphi_{j,k}(t) = 2^{\frac{-j}{2}}\varphi(2^{-j}t - k) \tag{2}$$

両者を、k シフト（左右移動）、j ダイレーション（高さの変更）するとき、$j = k$ であれば互いにが直行条件を満たし、以下の式 (3) と式 $reftwoscale2$ で表されるツースケール関係を満たす。

$$\psi_{j,k}(t) = \sum_n q_n \psi_{j-1,2k+n} \tag{3}$$

$$\varphi_{j,k}(t) = \sum_n p_n \psi_{j-1,2k+n} \tag{4}$$

p_n はスケーリング関数を表す係数列、q_n はウェーブレットを表す係数列である。

2.2 Daubechies のウェーブレット

Ingrid Daubechies によって発見された [5]Daubechies のウェーブレット、スケーリング関数は、図 1 の様に複雑で既知の関数で表すことができない。ウェーブレットを表す係数列 q_n とスケーリング関数を表す係数列 p_n から再帰的な計算を行い、順次精度を高めていくことで求められる。また、Daubechies のウェーブレットはより単純な Haar のウェーブレットに比べて、急激な変化に対応することができ画像の解析・分解に適している。表 2.2 に N=2 の Daubechies のウェーブレットの元

表 1　N=2 の Daubechies の p_n, q_n と使用した値

p_n	q_n	使用した p_n	使用した q_n
0.482962913145	-0.129409522551	0.482962608	-0.129408836
0.836516303738	-0.224143868042	0.836515427	-0.224143028
0.224143868042	0.836516303738	0.224143028	0.836515427
-0.129409522551	-0.482962913145	-0.129408836	-0.482962608

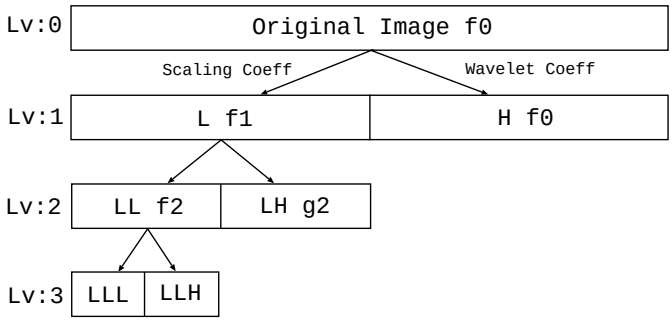

図 2　一次元の多重解像度解析

となる係数列 p_n と q_n を示す。

また、上記のスケーリング係数とウェーブレット展開係数は下式 (5)、(6) のようになる

$$s_k^j = \sum_n \overline{p_{n-2k}} s_n^{j-1} \tag{5}$$

$$w_k^j = \sum_n \overline{q_{n-2k}} w_n^{j-1} \tag{6}$$

2.3　多重解像度解析

多重解像度解析は、DWT を信号処理に用いる際に最も重要な概念である。所与の信号 $f_0(x)$ をスケーリング関数 $\varphi_{j,k}(t)$ との一次結合で近似し、レベルと呼ばれる近似の精度を順に落としていくことで行われる。あるレベル J の近似 $f_{j-1}(t)$ とそれより 1 レベル低い（粗い）レベル $J+1$ の近似 $f_j(t)$ の差分をウェーブレット成分 $g_j(t)$ と呼び、式 (7) で表され、スケーリング関数 s_k^j とウェーブレット展開係数 w_k^j を用いて $g_j(t)$ で表される。

$$f_{j-1}(t) = f_j(t) + g_j(t) \tag{7}$$

$f_{j-1}(t)$ を次々に $f_j(t)$ と $g_j(t)$ に分けていくと式 (8) の様になる。

$$f_0(t) = g_1(t) + \cdots + g_J(t) + f_J(t) = \sum_{j=1}^{J} g_j(t) + f_J(t) \tag{8}$$

これよりり、様々な解像度から成るウェーブレットを用いて元の波形を解析することが出来る。また、$f_j(t)$ をローパスフィルタ、$g_j(t)$ をハイパスフィルタと考えると、低周波成分を次々と高周波成分と低周波成分に分けていく処理と同値である。図 2 に一次元の多重解像度解析の概要を示す。

2.4　二次元 DWT

二次元 DWT は、近傍画素との関連が強いという特性を利用し、図 3 に示すように離散ウェーブレット変換を列方向にかけた後、行方向にも変換を行う処理である。その結果は、以下の

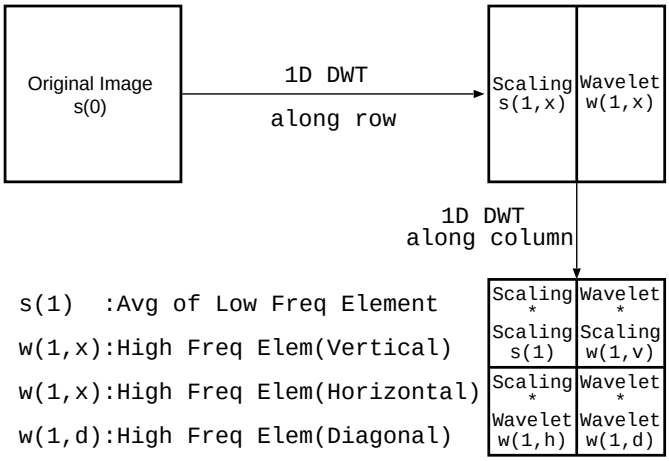

図 3　二次元離散ウェーブレット変換

成分を表す

左上：平均化した低周波成分

$$s_{m,n}^{j+1} = \sum_l \sum_k \overline{p_{k-2m}} \cdot \overline{p_{l-2n}} s_{k,l}^j \tag{9}$$

左下：水平方向の高周波成分

$$w_{m,n}^{j+1,h} = \sum_l \sum_k \overline{p_{k-2m}} \cdot \overline{q_{l-2n}} s_{k,l}^j \tag{10}$$

右上：垂直方向の高周波成分

$$w_{m,n}^{j+1,v} = \sum_l \sum_k \overline{q_{k-2m}} \cdot \overline{p_{l-2n}} s_{k,l}^j \tag{11}$$

右下：対角方向の高周波成分

$$w_{m,n}^{j+1,d} = \sum_l \sum_k \overline{q_{k-2m}} \cdot \overline{q_{l-2n}} s_{k,l}^j \tag{12}$$

また、左上の平均化した低周波成分 $s_{m,n}^{j+1}$ に再び二次元 DWT をかけることで、$j+2$ の二次元の多重解像度解析となる。

3.　Impulse C

3.1　概　　要

Impulse C は、Impulse Accelerated Technologies [3] より提供される FPGA 用の高位合成言語である。これは、HW プロセスの並列化モデル概念、ネットリスト、通信チャネル、各メモリへのアクセス手法を ANSI-C の基準に沿ってライブラリ化したものである。また、その Impulse C から HDL への変換を行い、プロファイルなどを得る開発環境として CoDeveloper が付属する。開発の流れは、C のアルゴリズムのプロファイルから HW で高速化したい部分を見つけ、メモリアクセスやダイナミック・ポインタなどの HW 化する上での制約を考慮に入れて Impulse C でその部分の手直しを行う。その後、CoDeveloper で HDL とプロファイルを出力し、得られたパイプライン段数、レイテンシ、速度、面積のレポートを参考に詳細を詰めていく。

Verilog におけるモジュールに相当するものをプロセスと呼び、プロセス間の通信チャネルには、stream（制御付き FIFO）、signal（ready 制御付きレジスタ）、register（単純なレジスタ）を利用する。また、チャネルや変数・配列は 1〜64bit 幅のものを自由に利用することが出来る。

3.2 SW-HW 分割

Impulse C では、SW プロセスと HW プロセスを容易に分離することが可能である。SW プロセスは完全な C 言語で記述され、MicroBlaze などのソフトプロセッサ上で実行される。処理すべきデータに対する前処理など負荷の小さな処理などが行われる。データは stream 経由で HW プロセスに送られ、ハードウェア処理される。本研究では、SW プロセスの testData Producer でビットマップのヘッダ部分と RGB 値の実データの分割を、testDate Consumer で処理結果をテキストファイルに書き出した。

3.3 Pragma

Impulse C では、pragma を用いて CoDeveloper にパイプライン段数やステートマシンの作成を指示する。以下に、本研究で仕様した Pragma について記述する。

3.3.1 CO PRIMITIVE

HW プロセス内で共通に利用するマクロを定義する。ソフトウェアにおける関数や Verilog における function のに相当するもので、呼び出した回数だけインスタンス化（回路実態化）される。本研究では、一次元離散ウェーブレット変換処理の入力値とスケーリング係数／ウェーブレット展開係数乗じる処理をマクロ（dwt_scaling, dwt_wavelet）として定義する際に利用した。

3.3.2 CO PIPELINE

Impulse C では、for ループのパイプライニングは自動的に行われず、CO PIPELINE pragma が使用された場合のみパイプライニングを行う。本研究では、”Prepare Second 1D-DWT” プロセス内の配列読み書きを es パイプライン化するために利用した。

3.3.3 CO SET StageDelay

CO PIPELINE pragma とセットで使用され、パイプライン内の各ステージディレイの制約を CoDeveloper に指示する。この pragma を利用することで、パイプライン段数の増減・効率化を図ることが可能である。本研究では、上記のパイプラインの最適化を行うために利用した。

3.4 小数点演算

Impulse C では、8, 16, 32bit の四則演算の固定小数点演算をサポートする。小数点以下の bit 桁数を指定し、FXADD/FXSUB/FXMUL/FXDIV の各マクロを用いて演算を行う。

なお、本研究では使用しなかったが Xilinx Core Generation FloatingPoint による浮動小数点演算も可能である。

4. 実　　装

本研究では、Impulse C を用いて Daubechies のウェーブレットをマザーウェーブレットとした離散ウェーブレット変換を Line-Based アーキテクチャで実装した。

4.1 実 装 環 境

FPGA は Xilinx Virtex5(XC5VLX85, 51840slices) を使用し、Xilinx ISE 11.4 を用いて、論理合成・配置配線を行った。Impulse C による RTL の自動生成には、CoDeveloper Ver3.60 を用いて、Generic_VHDL を書き出した。

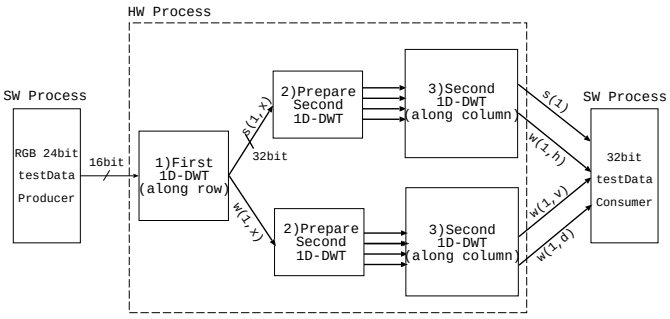

図 4　各要素のブロック図

CoDeveloper の最適化オプションは、以下の三つを適用した。

- “Enable constatnt propagation”：定数伝播を許可する
- “Scalarize array variables”：ローカル配列を値に変換しメモリではなくレジスタに値を置く
- “Relocate loop invariant expressions”：可能ならループ内の代入式を外に出す

また、ソフトウェア実装そしては C 言語で記述された中野ら [6] のものを利用した。実行環境は、Linux Kernel 2.6.22、gcc ver.3.46、使用したオプションは -O -Wall である。

4.2 アルゴリズム

4.2.1 全　体　像

Line-Based 方式では、入力画像を列ごとにスキャンし、列方向の一回目の離散コサイン変換処理が完了した後、直ちに行方向の二回目の離散コサイン変換の処理に進む手法である。これは、1D-RPA アルゴリズムを元にしており、このコンセプトに沿ったアーキテクチャは、[7] [8] に記載されている。各レベルの処理結果を即座に利用することで遅延を最小限に抑える。

本研究では、各 8bit の RGB 実データは SW プロセスでビットマップ画像から分離され、stream 経由で HW プロセスに送られる。HW プロセスは、二次元の離散ウェーブレット変換を処理ごとに 3 つのプロセスとして分離し、RGB を別々に並列処理できるよう全てのプロセスを三つずつ用意した。各処理プロセスは 32bit 固定小数点で行い、結果も 32bit とした。なお、逆変換は行っていない。

固定小数点の形式は符号ビット 1bit、整数部 11bit、仮数部 (FRACBITS)20bit の計 32bit(1s11.20) で行った。仮数部が 20bit であるのは、ウェーブレットを表す係数列 q_n とスケーリング関数を表す係数列 p_n が小数点 10 桁と小さい値であり、入力値に係数を 4 回乗じた後の誤差を小数点 5 桁以下にするためである。また、実際に使用した値を表 2.2 に示す。

図 4 に本アーキテクチャの RGB 一要素分のブロック図を示す。

4.2.2 dwt_scaling, dwt_wavelet マクロ

dwt_scaling は入力値とスケーリング係数、dwt_wavelet は入力値とウェーブレット展開係数とを乗ずるもので、#pragma CO PRIMITIVE を用いて作成されたマクロである。First 1D-DWT, Second 1D-DWT プロセスで呼び出されており、呼び出された回数だけ回路実態を作成する。また、結果は 1clk で出

力可能である。以下に実際の Impulse C のコードを示す。

```
co_int32 dwt_scaling(  co_int32 pix_in0,  co_int32 pix_in1,
                       co_int32 pix_in2,  co_int32 pix_in3)
{
// mean value of (p[0] * pix_in0)
  co_int32 scl_mean_0, scl_mean_1, scl_mean_2, scl_mean_3;
// mean value of (scl_mean_0 * scl_mean_1)
  co_int32 scl_mean_0_1, scl_mean_2_3;
  co_int32 scl_result;

  scl_mean_0 = FXMUL32(pix_in0, FX_p_0, FRACBITS);
  scl_mean_1 = FXMUL32(pix_in1, FX_p_1, FRACBITS);
  scl_mean_0_1 = FXADD32(scl_mean_0, scl_mean_1, FRACBITS);

  scl_mean_2 = FXMUL32(pix_in2, FX_p_2, FRACBITS);
  scl_mean_3 = FXMUL32(pix_in3, FX_p_3, FRACBITS);
  scl_mean_2_3 = FXADD32(scl_mean_2, scl_mean_3, FRACBITS);

  scl_result = FXADD32(scl_mean_0_1, scl_mean_2_3, FRACBITS);

  return(scl_result);
}
```

4.2.3 First 1D-DWT プロセス

First 1D-DWT プロセスでは、列方向に対する一次元 DWT 処理を行う。testData Producer プロセスから処理する要素 (8bit) を 16bit として読み出す。離散ウェーブレット変換は 2pixel 間隔で処理を行うため、新たに 2pixel の読み出しが必要となる。この時、stream から 8bit で二回読み出すのではなく、16bit として一回で読み出した値を HW プロセス内で二つの 8bit に分割した。stream は FIFO(BRAM) で作られており、そのアクセスに Impulse C では 2clk 必要とする。よって、二回アクセスした場合に比べ 1clk 分の高速化が可能である。新規に読み出した 2pixel は 8bit から 32bit に変換され、前の処理から再利用可能な既に 32bit に変換された 2pixel の計 4pixel に対し一回分の処理が行われる。4pixel がそれぞれ dwt_wavelet ／ dwt_scaling マクロでウェーブレット展開係数／スケーリング係数と掛け合わされ、$w(1,x)$ と $s(1,x)$ として列方向の一次元離散ウェーブレット変換が行われる。また、stream からの読み出し、8bit 分割、係数との乗算は#pragma CO PIPELINE によって 2 段のパイプラインになっている。

4.2.4 Prepare Second 1D-DWT プロセス

Prepare Second 1D-DWT プロセスでは、列方向に向いて入力された pixel を行方向に転地して出力する。

前処理として、入力画像の幅と同じ長さのバッファ配列:buff を 8 つ用意し、その 6 つの配列を値で満たす。次に、以下の処理を入力画像の高さまで続ける。

1-1　buff0〜buff3 の index=0(右端)4 値を stream で次のプロセスへ書き出す

1-2　buff0〜buff3 の index=0 に buff2〜buff5 の値を代入

1-3　前のプロセスから stream で 2 つの 32bit 値を読み込み buff6〜buff7 に保持していく

1-4　index++し、index=WIDTH(左端) まで 1-1〜1-3 の処理を続ける

ここまでで、buff0〜buff3 には buff2〜buff5 の値がコピーされ、buff6,buff7 には四つ先の列が保持されている。

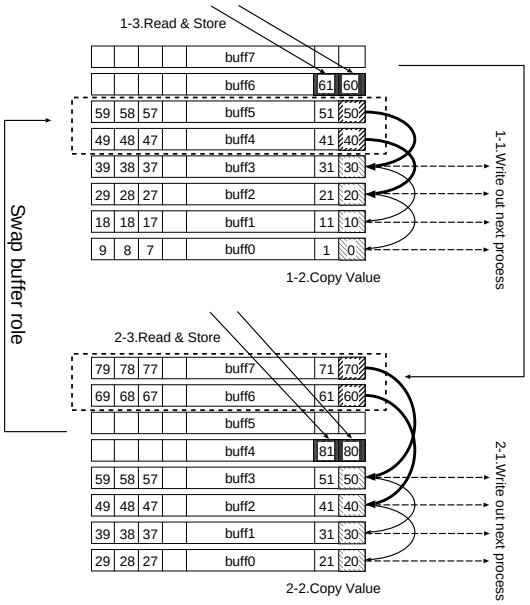

図 5　Prepare Second 1D-DWT プロセス

2-1　buff0〜buff3 の index=0(右端)4 値を stream で次のプロセスへ書き出す

2-2　buff0,buff1 に buff2,buff3 を buff2,buff3 に buff6,buff7 の値を代入

2-3　前のプロセスから stream で 2 つの 32bit 値を読み込み buff4〜buff5 に保持していく

2-4　index++し、index=WIDTH(左端) まで 1-1〜1-3 の処理を続ける

1-1〜1-4 と 2-1〜2-4 で、buff4,buff5 と buff6,buff7 の役割が入れ替わっている。また、この部分は 7 段パイプラインとなっている。図 5 に WIDTH=10 の時の処理手順の概略を示す。

読み出す配列がなくなる最後とその前の配列の処理は、最初と二番目の配列のコピーを保持しておき、それらを利用して 4 値とした。また、このプロセスは 2 つが実体化されている。これは、First 1D-DWT からの値が元画像を左右に分割したもので、それぞれは並列に処理が可能だからである。

4.2.5 Second 2D-DWT プロセス

Second 1D-DWT プロセスでは、行方向に対する一次元 DWT 処理を行う。First 1D-DWT プロセスと異なるのは、一回の処理に必要な 4pixel が Prepare Second 1D-DWT プロセスから 4 本の stream で一度に供給される点である。供給された 4pixel は dwt_scaling ／ dwt_wavelet マクロでそれぞれスケーリング係数／ウェーブレット展開係数が乗じられ、$s(1)$ と $w(1,h)$、もしくは、$w(1,v)$ と $w(1,d)$ してが出力される。32bit の出力結果は、SW プロセスである testData Consumer に stream 経由で送られ、入力画像への 1 レベル分の 2D-DWT 処理は終了となる。また、Prepare Second 1D-DWT と同様の理由で、プロセスは 2 つが実体化されている。

5. 評　　価

5.1 実装した二次元離散ウェーブレット変換ハードウェアの評価

実装した二次元離散ウェーブレット変換全体とプロセス単体の評価結果を表 2,3 に示す。

表 2　実装したハードウェア全体の評価（面積）

Slice Logic Utilization	Used[Utilization]
Occupiles Slices	12,104/12,960[93%]
Slice Register	8652/51,840[16%]
Slice LUTs	44,809/51,840[86%]
Block RAM(18k)	21/96[%]
Slice LUTs:as Dual Port RAM	1,200
Slice LUTs:as Single Port RAM	10,752
動作周波数	61.3[MHz]

表 3　実装したハードウェアのプロセスごとの評価（処理レート）

プロセス	処理レート × プロセス数	全処理レート
First 1D-DWT	0.5[clk/pix] × 1	0.50[clk/pix]
Prep 2nd 1D-DWT	2.0[clk/pix] × 2	1.00[clk/pix]
Second 1D-DWT	0.5[clk/pix] × 2	0.25[clk/pix]
全体	2.0[clk/pix] × 1	1.00[clk/pix]

Prepare Second 1D-DWT と Second 1D-DWT については、プロセスを二つ実体化し処理を担当する領域を二つに分けている。よって、全体としての処理レートは 2 倍となる。

表 3 から、全体のボトルネックは Prepare Second 1D-DWT プロセスであり、1pixel の出力に 1clk 必要であることがわかる。評価に用いた画像 (H:400*W:400=160000pix) に 1clk の時間（動作周波数 61.3[MHz] の逆数) を乗じ、2D-DWT にかかる処理時間の見積りを表 4 に記す。

表 4　二次元離散ウェーブレット変換の処理時間見積り

プロセス	全処理レート	全 pix の処理時間
First 1D-DWT	0.50[clk/pix]	1.35[ms]
Prep 2nd 1D-DWT	1.00[clk/pix]	2.60[ms]
Second 1D-DWT	0.25[clk/pix]	0.68[ms]
全体	1.00[clk/pix]	2.60[ms]

5.2 ソフトウェア実行との比較

ここでは、2D-DWT に関してソフトウェア実行との行う。ソフトウェア側の評価環境は、Intel Core 2 Duo E6600 2.40GHz である。表 5 に比較結果を示す。評価には H:400*W:400=160000pix の画像を用いた。

表 5　ソフトウェア実行との処理時間の比較

処理	ソフトウェア	実装した回路	速度比較
動作周波数	2.4[GHz]	61.3[MHz]	x0.026
一回目の 1D-DWT	3.60[ms]	1.35[ms]	x2.67
配列の転地	1.61[ms]	2.60[ms]	x0.62
二回目の 1D-DWT	3.60[ms]	0.68[ms]	x5.34
全体	8.81[ms]	2.60[ms]	x5.29

回路全体として、5.29 倍の高速化を達成することができた。

1D-DWT 処理については、入力値のスケーリング係数／ウェーブレット展開係数との乗算を並列化していることが高速化の要因と考えられる。

配列の転地 (Prepare Second 1D-DWT プロセス) について、今回実装した回路はソフトウェア実行よりも遅くなっている。これは、一度に保持する配列の数を抑えたこと、パイプライン化した結果、4.2.4 に示した 1-1 と 1-2 の処理が同じ配列にアクセスし、コンフリクトを起こしているからだと考えられる。

6. まとめと今後の課題

本研究では、高位合成言語である Impulse C を用いて、Daubechies のウェーブレットを基底とし、Line-Based アーキテクチャの二次元離散ウェーブレット変換を FPGA 上に実装した。その結果、ソフトウェア実行に対し、5.29 倍程度の高速化を達成した。

今後の課題は、逆変換の実装の他、より効率の良い Lifting Base アーキテクチャや Block Base アーキテクチャの実装とそれを踏まえた新しいアルゴリズムの探求を行っていきたい。また、Impulse C の効果的な記述についても考察を深めていきたい。

謝　　辞

研究の環境だけでなく、有益なアドバイスと議論をいただいたカルソニックカンセイ先行開発部門の皆様と有限会社インターリンクの皆様に感謝する。

文　　献

[1] ISO/IEC IS 15444-1: “Information Technology - JPEG2000 image coding system - Part 1：Core coding sytem”, ISO/IEC JTC1/SC29/WG1 (Dec 2000).

[2] ISO/IEC IS 15444-1: “Information Technology - JPEG2000 image coding system - Part 1 : Core coding sytem”, AMENDMENT1: Codestrean restrictions (Mar 2002).

[3] Impulse Accelerated Technologies. http://www.impulseaccelerated.com.

[4] N.D.Zervas, G.P.Anagnostopoulos, V.Spiliotopoulos, Y. Andreopoulos, C.E.Goutis: “The Development of the UP-ACS CFD EnvironmentEvalutation od design alternatives for the 2-D discrete wavelet transform”, IEEE Trans. on Circuit Systems for Video Technology, vol.11, issue.3, pp. 1246–1262 (Mar.2000).

[5] Ingrid Daubechies: “Orthonormal bases of compactly supported wavelets”, Communications on Pure and Applied Mathematics, pp. 909–996 (1988).

[6] 中野宏毅, 山本鎭男, 吉田靖夫：“ウェーブレットによる信号処理と画像処理”, 共立出版株式会社 (1999).

[7] Line-Based, Reduced Memory, Wavelet Image Compression: “IEEE Trans. Circuits and Syst. Video Tech.”, Vol,9, pp. pp.378–389 (2000).

[8] Wen-Shiaw Peng, Chen-Yi Lee: “An Efficient VLSI Architecture For Separatable 2-D Discrete Wavelet Transform”, Proceedings of the 1999 International Conference on Image Processing(ICIP99), pp. pp.754–758 (1999).