オーディオビジュアル複合情報処理　35－11
(2001. 12. 14)

アフィン動きパラメータ推定のための専用ハードウェアに関する考察

仲川　和志†　白石　真一†　長谷山美紀†　北島　秀夫†

† 北海道大学 大学院 工学研究科
〒060-8628 札幌市北区北 13 条西 8 丁目
TEL (011)706-7162

E-mail: †{kazu,shinichi,mikich,kitajima}@media.eng.hokudai.ac.jp

あらまし　動画像からの動き推定法として，アフィン動きモデルを用いた手法が提案されている．この手法を用いることにより，動画像内の回転や伸縮などの複雑な動きを推定可能となるが，反面，多量の計算が必要になるという問題がある．そこで，本文では，専用ハードウェアによりこの計算量の問題を解決することを目的として，アフィン動きパラメータ推定のためのVLSIアーキテクチャを導出する．さらに，本文では，導出されたアーキテクチャを単純化し，かつ高スループット化する手法を提案する．また，本文ではハードウェア記述言語の一つであるVHDLを用いてこのアーキテクチャを設計することにより，FPGA上での実現について検討する．

キーワード　動き推定, アフィン動きパラメータ, VHDL, FPGA, VLSI

A Hardware Architecture for an Affine-Based Motion Estimation

Kazushi NAKAGAWA†, Shin'ichi SHIRAISHI†, Miki HASEYAMA†, and Hideo KITAJIMA†

† Graduate School of Engineering, Hokkaido University
Kita-ku Kita-13 Nishi-8, Sapporo, 060-8628, Japan
TEL +81-11-706-7162

E-mail: †{kazu,shinichi,mikich,kitajima}@media.eng.hokudai.ac.jp

Abstract　This paper proposes an architecture for an affine-based motion estimation algorithm. Although an affine-based motion estimation algorithm can estimate accurate motion from image sequences, it requires high computational power. In order to avoid such computational complexity, we propose a hardware architecture for the motion estimator. Furthermore, we propose a method to achieve high throughput and low hardware complexity in the motion estimator. In this paper,we describe the motion estimator in VHDL and then show the result of design using an FPGA.

Key words　Motion estimation, Affine motion parameters, VHDL, FPGA, VLSI

1. まえがき

動画像からの動き推定は，動画像符号化やコンピュータビジョンの分野において，非常に重要な技術である．この動き推定のための手法には，フレームを方形の領域（ブロック）に分割し，そのブロックごとに動きを推定するブロックマッチング法や，領域の形状を方形に限定せず，領域分割により得られた任意形状の領域の動きを推定する手法 [1] 等がある．これらの手法は，領域の動きモデルとして，単純な平行移動モデルを採用しているが，一方で，より精密な動き推定を行うために，平行移動・回転・伸縮から構成されるアフィン動きモデルを導入した手法も報告されている [2], [3]．このアフィン動きモデルを導入することにより，回転や伸縮などの複雑な動きの推定が可能となるが，反面，これらの動きを表現するアフィン動きパラメータの推定に多量の計算が必要になるという問題が発生する．なぜなら，従来の平行移動モデルを採用した場合では，平行移動を表現する 2 次元のパラメータ空間の探索で動き推定が完了するのに対し，アフィン動きモデルを導入すると，平行移動（2 種類），回転（1 種類），伸縮（2 種類）の計 5 種類，即ち，5 次元のパラメータ空間を探索しなければならなくなるためである．さらに，平行移動パラメータの探索が，加算を主な演算として実行できるのに対し，回転・伸縮パラメータの探索には，計算コストの高い三角関数演算や乗算が必要であるため，動き推定に必要となる計算量は膨大なものとなる．これまで，この計算量の問題を解決するために，アルゴリズムの簡略化や信頼度関数の導入により計算量を削減する試みが，多数なされてきた [2], [4]．しかし，一方で，専用ハードウェアによりこの計算量の問題を解決しようとする手法は，未だ報告されていない．そこで，本文では，アフィン動きパラメータ推定を高速に実行可能で，かつ実現コストが低い VLSI アーキテクチャを提案する．提案するアーキテクチャを利用することにより，FPGA 等の回路規模の小さなデバイスを用いた場合においても，高精度な動き推定のための回路を実現することができる．本文では，まず，回転・伸縮パラメータの探索の中で必要となる回転・伸縮変換に注目し，これを CORDIC(COordinate Rotation DIgital Computer) プロセッサ [5], [6] を用いて実行するアーキテクチャを導出する．CORDIC プロセッサは，回転変換と伸縮変換を共通のアーキテクチャで実行でき，また，乗算器を必要としない単純な構造を持つため，少ないハードウェアで実現が可能である．さらに，本文では，回転・伸縮パラメータ候補の構成方法を工夫することにより，導出されたアーキテクチャが，単純化と高スループット化を実現可能なことを示す．提案する回転・伸縮パラメータの構成手法により，このパラメータを用いた変換を実行する CORDIC プロセッサの反復回数が削減されるため，同プロセッサのスループットが向上する．また，同時に CORDIC プロセッサにおいて角度計算が必要なくなるため，さらに単純なアーキテクチャで実現できる．また，本文では，パソコンなどを用いて簡単に回路の試作が可能な FPGA（Field Programmable Gate Array）デバイスをターゲットとして，導出したアーキテクチャの実現の検討を行う．FPGA による実現のためには，まず，ハードウェア記述言語の一つである VHDL によりアーキテクチャの設計を行う．次に，この VHDL 設計されたアーキテクチャを論理合成し，特定の FPGA デバイスへのマッピングを行う．このマッピング結果から，FPGA に実装した際の回路規模や動作周波数について見積もりを行う．

2. 従来法の紹介

本節では，提案手法の導出の準備として，従来法であるアフィン動きモデルを用いた動き推定と CORDIC アルゴリズムについて説明する．

2.1 アフィン動きモデルを用いた動き推定

動画像内の動きには，従来のブロックマッチング法が扱う平行移動だけではなく，回転・伸縮なども存在する．これらの複雑な動きに対応するために，アフィン動きモデルを用いた動き推定法 [2], [3] が提案されている．以降の説明では，この手法を連続する 2 フレーム（前フレームと現フレーム）からの動き推定に用いるものとして，その概要を示す．

アフィン動きモデルを用いた動き推定法は，アフィン動きパラメータを $\boldsymbol{p}$ とし，評価関数を SAD（Sum of Absolute Difference, 差分絶対値和）とした場合，画像内のある領域 R の動きを，下式で決定されるアフィン動きパラメータ $\boldsymbol{p}_0$ で表現する手法である．

$$\boldsymbol{p}_0 = \arg\min_{\boldsymbol{p}} \sum_{(x,y)\in R} |I_t(x,y) - I_{t-1}(x',y')| \quad (1)$$

ただし，式中の $I_t(x,y)$，$I_{t-1}(x',y')$ は，それぞれ現フレームと前フレームにおける画素値であり，また，前フレームの画素 (x',y') は，現フレームの画素 (x,y) に対して，パラメータを $\boldsymbol{p}$ とするアフィン変換を行うことにより得られる（図 1 参照）．ここで，アフィン動きモデルが，平行移動・回転・伸縮から構成されることを考えると，この (x,y) から (x',y') への座標変換は，式 (2) のように表すことができる．

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad (2)$$

上式において，$(t_x,t_y), \theta, (s_x,s_y)$ は，それぞれ平行

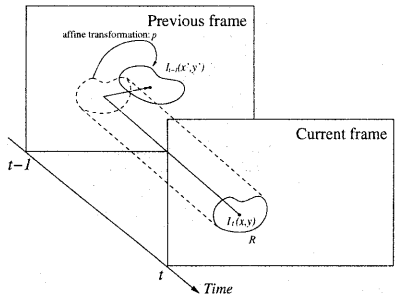

図 1　アフィン動きモデルを用いた動き推定

移動，回転，伸縮パラメータを表し，よって，アフィン動きパラメータ $\boldsymbol{p}$ は，$\boldsymbol{p}=(t_x,t_y,\theta,s_x,s_y)$ となる．以上で述べた手法を用いることにより，動画像内の回転や伸縮などの複雑な動きの推定が可能となる．しかし，一方で，アフィン動きパラメータ，即ち式 (1) を満足する $\boldsymbol{p}_0$ を算出するためには，$\boldsymbol{p}$ の取り得るすべての値で，式 (2) の座標変換を行わなければならず，多量の計算が必要となる（例えば，探索候補の平行移動パラメータ (t_x,t_y) が 16^2 通り，回転パラメータ θ が 8 通り，伸縮パラメータ (s_x,s_y) が 8^2 通りとした場合，$16^2\cdot 8\cdot 8^3=131,072$ 回の座標変換が必要である）．さらに，式 (2) より明らかなように，この変換には計算コストの高い三角関数演算や乗算が必要とされるため，計算量は膨大なものとなる．そこで，本文では，式 (2) に含まれる回転変換や伸縮変換を，専用ハードウェアにより高速に実行するために，次節で説明する CORDIC アルゴリズムを利用する．

2.2 CORDIC アルゴリズム

CORDIC アルゴリズム [5], [6] は，シフトと加減算および定数読み出しという単純な操作の反復を行うことにより，3 つの座標系（線形回転，円形回転，双曲線回転）のいずれかにおいて，ベクトルの回転操作を実行する．座標系をパラメータ m（線形回転 $m=0$，円形回転 $m=1$，双曲線回転 $m=-1$）により設定するものとすると，CORDIC における i 番目の回転操作は，下式のように表される．

$$\begin{cases} x_{i+1}=x_i-m\delta_i 2^{-S_{m,i}}y_i \\ y_{i+1}=y_i+\delta_i 2^{-S_{m,i}}x_i \\ z_{i+1}=z_i-\delta_i\alpha_{m,i} \end{cases} \tag{3}$$

$$\alpha_{m,i}=m^{-\frac{1}{2}}\tan^{-1}(m^{\frac{1}{2}}2^{-S_{m,i}})$$
$$i=0,1,\cdots,N-1$$

ここで，N は反復回数とする．N 回の反復により，およそ $N-1$bit の精度が得られるため，通常 N は，x_i,y_i,z_i の値を保持するレジスタの語長と同程度の大きさに設定する [6]．また，$S_{m,i}$ は，シフト数列である．$\delta_i\in\{-1,1\}$ は，ベクトルの回転方向を表すパラメータであり，残り角 z_{i+1} を 0 に近づけるように選択される．$\alpha_{m,i}$ は，基本回転角と呼ばれる定数であり，通常 ROM に格納され各反復毎に読み出される．また，式 (3) の第一式と第二式は，下式のように書き換えることができる．

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix}=k_{m,i}\begin{bmatrix} \cos(\delta_i\alpha_{m,i}) & -\sqrt{m}\sin(\delta_i\alpha_{m,i}) \\ \frac{1}{\sqrt{m}}\sin(\delta_i\alpha_{m,i}) & \cos(\delta_i\alpha_{m,i}) \end{bmatrix}\begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

$$k_{m,i}=\sqrt{1+m\delta_i^2 2^{-2S_{m,i}}}$$

上式に示されているように，CORDIC アルゴリズムにおける各反復では，ベクトル $[x_i,y_i]^T$ に対する角度 $\delta_i\alpha_{m,i}$ の回転と，$k_{m,i}$ 倍の拡張が行われている．よって，CORDIC アルゴリズムでは，N 回の反復計算を行うことで，目的とする回転角 z_0 の回転操作を，下式に示すように，N 個の基本回転角の回転操作の組み合わせとして実現していることがわかる．

$$z_0=\sum_{i=0}^{N-1}\delta_i\alpha_{m,i} \tag{4}$$

このような反復公式を用いた回転操作により，乗除算，三角関数，双曲線関数などの様々な初等関数演算を実行することが可能となる．ここで一例として，表 1 に，後に本文で用いる直線回転（$m=0$）と円形回転（$m=1$）において計算可能な関数を示す．表 1 に示されるように，円形回転（$m=1$）の場合，N 回の反復の後に得られるベクトル $[x_N,y_N]^T$ は，K 倍に拡張されているため，$1/K$ に縮小する操作（スケーリング操作）が必要になる．このスケーリング操作は，式 (3) に類似した以下の反復公式を用いて行われる [7]．

$$\begin{cases} x_{i+1}=(1-2^{-S_{1,i}})x_i \\ y_{i+1}=(1-2^{-S_{1,i}})y_i \end{cases} \tag{5}$$
$$i=N,N+1,\cdots,M+N-1$$

ただし，M は反復回数であり，$M\simeq N/2$ である [8]．

図 2 に，式 (3),(5) を実行する CORDIC プロセッサのアーキテクチャを示す．図からもわかるように，CORDIC プロセッサは，乗算器を必要としない単

表 1　CORDIC アルゴリズムにより計算可能な関数の例

linear ($m=0$)	$x_N=x_0$ $y_N=x_0z_0+y_0$
circular ($m=1$)	$x_N=K\{x_0\cos z_0-y_0\sin z_0\}$ $y_N=K\{x_0\sin z_0+y_0\cos z_0\}$

$K\equiv\prod_{i=0}^{N-1}k_{1,i}$

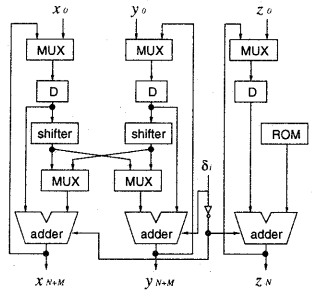

図 2　CORDIC プロセッサのアーキテクチャ

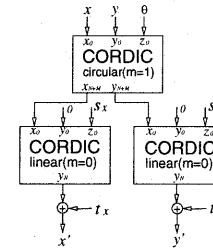

図 3　CORDIC プロセッサを用いたアフィン変換のシグナルフロー図

純な構造でベクトルの回転演算を実行することが可能である．次の節では，このような単純な構造の CORDIC プロセッサを利用して，回転・伸縮パラメータ探索のためのアーキテクチャを導出する．

3. CORDIC アルゴリズムに基づくアーキテクチャの提案

2.1 において述べたように，アフィン動きモデルを用いた動き推定を実行するためには，パラメータ $\boldsymbol{p}$ の取り得るすべての値で，式 (2) の変換を実行しなければならない．さらに，この変換には，計算コストの高い三角関数演算や乗算が含まれているため，パラメータ探索の際の計算量は膨大なものとなる．したがって，高いスループットを要求されるような場合，例えば符号化にリアルタイム性が要求される通信や放送などの場合において，この手法を用いることは困難となる．そこで，この計算量の問題を解決するために，式 (2) 中で多くの計算量を占める回転変換や伸縮変換に着目し，これを**2.2** において説明した CORDIC プロセッサを利用してハードウェア化する．

3.1 CORDIC プロセッサを用いたハードウェア化

CORDIC プロセッサは，下式に示すように，円形回転（$m=1$）においてベクトル $[x_0, y_0]^T$ に対する回転角 z_0 の回転変換を実行可能であり，また，直線回転（$m=0$）において x_0 に対する z_0 の伸縮変換（乗算）を実行可能である．

[回転変換]

$$\begin{bmatrix} x_{N+M} \\ y_{N+M} \end{bmatrix} = \begin{bmatrix} \cos z_0 & -\sin z_0 \\ \sin z_0 & \cos z_0 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{6}$$

[伸縮変換]

$$\begin{bmatrix} x_N \\ y_N \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ z_0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} \tag{7}$$

式 (2) と式 (6),(7) の比較より明らかなように，この CORDIC プロセッサによる回転・伸縮変換を，アフィン動きモデルを用いた動き推定に必要とされる回転・伸縮変換に利用することが可能である．つまり，**2.1** で述べた現フレームの画素の座標値 (x, y) と回転・伸縮パラメータ $\theta, (s_x, s_y)$ を CORDIC プロセッサへ入力することにより，式 (2) 中の回転・伸縮変換を実行できる（図 3 参照）．このように，式 (2) 中の計算コストの高い回転・伸縮変換を実行するために，CORDIC プロセッサを専用ハードウェアとして用いることができる．この CORDIC プロセッサは，乗算器を必要としない単純な構造を持つため，小規模な専用ハードウェアとして実現することが可能である．しかしここで，アフィン変換において用いられる回転 θ・伸縮 (s_x, s_y) パラメータ候補の構成方法を工夫することにより，用いられる CORDIC プロセッサの単純化と高スループット化を実現できる．次節では，回転・伸縮パラメータの構成手法を提案し，そのパラメータを用いた回転・伸縮変換を実行する CORDIC プロセッサのアーキテクチャを示す．

3.2 CORDIC プロセッサの単純化と高スループット化を実現する回転・伸縮パラメータの構成手法

一般的に，アフィン動きパラメータの探索では，あらかじめ，パラメータの探索範囲の決定とパラメータの離散化を行い，その離散化された動きパラメータ候補の中から探索が行われる．そこで，本文では，回転 θ・伸縮 (s_x, s_y) の動きパラメータの探索範囲の決定と離散化を，下式のように CORDIC アルゴリズムの基本回転角を用いて行う．

$$\theta = \sum_{i=g_1}^{h_1} \delta_i \alpha_{1,i} \qquad 0 \leqq g_1 < h_1 \leqq N-1 \tag{8}$$

$$s = \sum_{i=g_0}^{h_0} \delta_i \alpha_{0,i} + 1 \qquad 0 \leqq g_0 < h_0 \leqq N-1 \tag{9}$$

ただし，上式では，2 種の伸縮パラメータ (s_x, s_y) を代表させて，s として記述してある．また，式 (9) には，基本回転角に 1 が加えられているが，これは伸

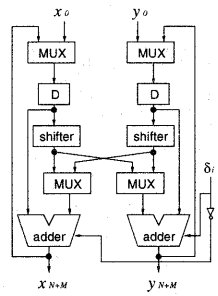

図 4　角度計算のデータパスや ROM を削減した CORDIC プロセッサのアーキテクチャ

縮パラメータ s が常に正の値を取るようにするためである．ここで，探索範囲は，式 (8),(9) から下式のように記述される．

$$|\theta| \leqq \sum_{i=g_1}^{h_1} \alpha_{1,i}$$

$$|s-1| \leqq \sum_{i=g_0}^{h_0} \alpha_{0,i}$$

また，$\delta_i \in \{-1, 1\}$ が2値をとるため，$2^{h_m - g_m + 1} (m = 1, 0)$ 通りの値に離散化される．例えば，$g_1 = g_0 = 3, h_1 = h_0 = 5$ とした場合，それぞれのパラメータの取り得る値は，$\theta \in \{\pm 12.5°, \pm 8.9°, \pm 5.3°, \pm 1.8°\}$, $s-1 \in \{\pm 0.22, \pm 0.16, \pm 0.09, \pm 0.03\}$ の 8 通りとなる．回転・伸縮パラメータを，式 (8),(9) で示すように基本回転角の線形和で構成することによって，CORDIC プロセッサにおける反復回数を削減することができる．これは式 (4) より明らかなように，

$$i = 0, 1, \cdots, g_m - 1, h_m + 1, \cdots, N - 1$$

なる i に関する反復計算（式 (3)）を実行する必要がないためである．よって，$N - (h_m - g_m + 1)$ 回の反復計算を削減することが可能となり，CORDIC プロセッサの高スループット化が実現される．さらに，$2^{h_m - g_m + 1}$ 通りの回転・伸縮パラメータを表現する $(\delta_{g_m}, \delta_{g_m+1}, \cdots, \delta_{h_m})$ は，あらかじめ各回転・伸縮パラメータ毎に算出しておくことが可能である．よって，δ_i を反復毎に算出する必要がないため，ここで用いられる CORDIC プロセッサは，δ_i を算出するための角度計算のデータパスや基本回転角を格納する ROM を設ける必要がなく，汎用的な CORDIC プロセッサ（図 2）と比較して，より単純な回路構造（図 4）で実現することができる．

4. 計算コストの比較

前節では，アフィン動きパラメータ推定に必要とされる回転・伸縮変換が，CORDIC プロセッサを用いてハードウェア化できることを示した．また，回転・伸縮パラメータ候補の構成方法を工夫することにより，用いられる CORDIC プロセッサの単純化と高スループット化が達成されることを示した．本節では，提案手法の有効性を確認するため，**3.2** で提案した構成手法によるパラメータを用いた回転・伸縮変換が，CORDIC プロセッサにより実行される際に必要となる計算コストを算出する．また，比較のため，一般的な手法として，乗除算を用いて回転・伸縮変換を実行する場合について，その計算コストを算出し，提案手法と比較する．ただし，両手法における回転・伸縮パラメータは，式 (8), (9) に示した構成をとるものとし，$g_1 = g_0 = 3, h_1 = h_0 = 5$ とする．また，乗除算を用いる場合では，回転変換に必要とされる三角関数を，下式に示す展開公式を用いて計算するものとする．

$$\sin\theta = \frac{\theta^1}{1!} - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \cdots (-1)^n \frac{\theta^{(2n+1)}}{(2n+1)!}$$
$$= \theta(1 - \frac{\theta^2}{2 \cdot 3}(1 - \cdots (1 - \frac{\theta^2}{2n \cdot (2n+1)}) \cdots))$$

$$\cos\theta = \frac{\theta^0}{0!} - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \cdots (-1)^{n-1} \frac{\theta^{2n}}{(2n)!}$$
$$= 1 - \frac{\theta^2}{1 \cdot 2}(1 - \cdots (1 - \frac{\theta^2}{(2n-1) \cdot 2n}) \cdots)$$

展開公式における項数 n は，小数点以下何 bit までの精度を求めるかによって異なり，また，回転パラメータ θ の大きさによっても異なる．そこで，ここでは，小数点以下 8bit までの精度を求めるものとし，また，各回転パラメータ θ 毎に項数 n を決定した．一方，提案手法における回転変換の場合の反復回数は，回転操作に $h_1 - g_1 + 1 = 5 - 3 + 1 = 3$ 回であり，スケーリング操作に $n/2$ 回（ただし，レジスタの語長を n とする）である．同様に，提案手法における伸縮変換の場合の反復回数は，回転操作に $h_0 - g_0 + 1 = 5 - 3 + 1 = 3$ 回である．上記の条件のもとに，すべての回転・伸縮パラメータの変換を実行した際に必要とされる演算回数を算出した．その結果を表 2 に示す．表より明らかなように，提案手法で用いられる演算の種類は加算とシフトであり，そのままでは乗除算を用いる場合と比較することができない．そこで，本文では，(1) 乗算を n 倍の加算と同一の計算コストとする (2) 除算を乗算と同一の計算コストとする (3) ビットシフトを加算と同一の計算コストとする，という仮定のもとで全ての演算を加算に換算して比較を行う．その結果，両手法の演算は，表中の括弧内に示す加算回数に換算され，加算に換算時の合計が乗除算を用いる場合で

表 2　乗除算を用いる場合と提案手法における演算回数（括弧内は加算に換算した場合の演算回数）

	乗算	加算	除算	ビットシフト	加算に換算時の演算回数
乗除算を用いる場合	60(60n)	10	10(10n)	0	**70n+10**
提案手法	0	96+8n	0	96+8n(96+8n)	**16n+192**

は，$70n+10$ 回であるのに対して，提案手法では，$16n+192$ 回となる．よって，レジスタの語長 n が3以下のような特殊な場合を除いて，提案手法の方がより計算コストが低いことが確認された．

5. FPGA 実現の検討

本節では，**3.** において導出したアフィン動きパラメータ探索のためのアーキテクチャを FPGA 実現する際のコストや処理速度について検討を行う．ただし，ここでは，提案したアーキテクチャにおいて中心的なモジュールとして用いられる CORDIC プロセッサ (図4) に焦点を絞り，その回路規模や動作周波数の見積もりを行う．FPGA による実現には，次の手順が必要である．まず，目的の回路の仕様設計を行い，回路の入出力インターフェースやアーキテクチャなどを決定する．次に，VHDL 等のハードウェア記述言語により RTL(Register Transfer Level) 設計を行う．RTL 設計では，仕様設計をもとに，回路をレジスタと，レジスタ間の組み合わせ回路として設計し，回路の機能を決定する．この際，シミュレータを用いて，回路の機能上の問題を発見し，修正を行う．続いて，RTL 設計された回路を，論理合成ツールを用いて論理合成し，ネットリストを得る．最後にネットリストをマッピングツールへ入力し，特定のデバイスへのマッピングを行う．このマッピング結果から，実際に FPGA へ実装した際の動作周波数などの見積もりを行うことができる．

本節で行う CORDIC プロセッサの設計では，図4に示されている回路に対し，回転操作やスケーリング操作の制御に必要となる周辺回路を加えたものを対象とした．この CORDIC プロセッサを VHDL により RTL 設計し，Accolade 社の論理合成ツールである PeakFPGA を用いて論理合成を行った．次に，Altera 社の MAX+PLUS II を用いてデバイスへのマッピングを行った．設計結果を表3に示す．この表3の結果と **4.** で示した提案手法の反復回数をもとに，設計を行ったアーキテクチャの処理速度を見積もると，1秒あたり約 9.2×10^5 回の回転・伸縮変換を実行することができる．処理速度を上げるためには，CORDIC プロセッサのパイプライン化や回転変換と伸縮変換の並列処理などが有効であると考えられる．

表 3　FPGA デバイスを用いた設計結果

Device	FLEX10K30(30k gates)
Word Length	16bit(n=16)
Logic Cell Usage	398(23%)
Clock Frequency	15.6MHz

6. むすび

本文では，アフィン動きモデルを用いた動き推定法の専用ハードウェア化について考察を行った．アフィン動きモデルを用いた動き推定法は，従来の平行移動モデルだけでは捉えきれなかった回転や伸縮の動きを推定することが可能である反面，多量の計算が必要であるという問題点があった．そこで本文ではハードウェア化によりこの問題点を解決するため，CORDIC アルゴリズムに基づくアーキテクチャを導出した．さらに，動きパラメータ候補の構成方法を工夫することにより，用いられる CORDIC プロセッサの単純化と高スループット化を実現できることを示した．また，本文では，導出したアーキテクチャの FPGA による実現を検討し，その設計結果を示した．今後の課題としては，これまでの設計結果を用いて，FPGA を用いた動き推定の実時間処理の可能性を検討することである．

文　献

[1] 川田 亮一, 小池 淳, 松本 修一, “可変形状の領域分割動き推定方式,” 信学論 (D-II), vol. J82-D-II, no. 11, pp. 1972-1981, Nov. 1999.

[2] 如澤 裕尚, “アフィン変換を用いた動き補償予測に関する検討,” 信学技報, IE94-36, July 1994.

[3] C. S. Fuh, P. Maragos, “Motion displacement estimation using an affine model for image matching,” Optical engineering, vol. 30, no. 7, pp. 881-887, July 1991.

[4] 吉田 俊之, 市川 明男, 酒井 善則, “マルチパラメータ動き推定のための信頼関数とその応用,” 信学論 (D-II), vol. J81-D-II, no. 7, pp. 1536-1544, July 1998.

[5] J. E. Volder, “The CORDIC trigonometric computing technique,” IRE Trans. Electric Computers, vol. EC-8, pp. 330-334, Sep. 1959.

[6] J. S. Walther, “A unified algorithm for elementary functions,” AFIPS Spring Joint Computer Conf, pp. 379-385, May 1971.

[7] Gene L. Haviland, Al A. Tuszynski, “A CORDIC arithmetic processor chip,” IEEE Trans. on Computers, vol. C-29, no. 2, pp. 68-79,Feb. 1980.

[8] Long-Wen Chang, Shen-Wen Lee, “Systolic arrays for the discrete hartley transform,” IEEE Trans. on Signal Processing, vol. 39, no. 11, pp. 2411-2418,Nov. 1991.