卒業論文

画像処理プログラムの HDL 記述と
ハードウェア動作合成システムの検証(Ⅰ)

氏　　名：安倍　厚志
学籍番号：2260040002-2
指導教員：山崎 勝弘 教授
提 出 日：2008 年 2 月 20 日

立命館大学 理工学部 電子情報デザイン学科

内容梗概

本研究ではハードウェア動作合成システムにおいて OpenMP で書かれた画像処理プログラムにより生成された中間コードから verilog-HDL を記述してそのプログラムを手書きによるコード生成とコードジェネレータによる生成を比較して検証を行った。

本論文ではあらかじめ決められたコード生成ルールに従って画像処理プログラムを Verilog-HDL 記述し、画像を出力させる。sobel フィルタによるエッジ検出プログラムで出力された画像における動作の確認とその結果におけるコードジェネレータで生成されたものとの比較検証を行い、コードジェネレータの最適化を検討する。また、hough 変換による直線や円の検出についてもこれを検討する。

目次

１．はじめに 1
２．ハードウェア動作合成システム 3
2.1 ハードウェア動作合成システムの概要 3
2.2 中間表現 4
2.3 コードジェネレータ 5
３．Sobel フィルタ 6
3.1 Sobel フィルタとは 6
3.2 Sobel フィルタのアルゴリズム 6
3.3 HDL 記述と実行結果 7
3.3.1 HDL 記述 7
3.3.2 実行結果 10
４．Hough 変換 11
4.1 Hough 変換とは 11
4.2 Hough 変換のアルゴリズム 13
4.2.1 直線の検出 13
4.2.2 円の検出 14
5. ハードウェア動作合成システムの検証 15
5.1 実験 15
5.2 比較による検証 16
6. おわりに 17
謝辞 18
参考文献 19

図目次

図 1　OpenMp によるハードウェア動作合成システムの構成 3
図 2：中間表現のサンプル 4
図 3　コードジェネレータの構成 5
図 4　Sobel フィルタの係数 6
図 5　Sobel フィルタの出力結果 10
図 6　Hough 変換の考え方 12
図 7　直線を検出するためのハフ変換式の原理 13
図 8　Hough 変換(円の検出)の方法 14

表目次

表 1：実験環境 15
表 2：Sobel フィルタのコードジェネレータでの実行速度 15
表 3：Sobel フィルタの手書きによる記述の実行速度 15
表 4：Sobel フィルタのコードジェネレータでの回路規模 16
表 5：Sobel フィルタの手書きによる記述の回路規模 16

１．はじめに

近年、携帯電話や車など実生活において欠かすことのできないありとあらゆる電子機器に LSI が搭載されるようになり、LSI は電子機器において新しい機能やサービスを実現する最も重要な要素となっている。日々高まるユーザの要求を実現するため、電子機器に搭載される LSI には、さらに高い処理性能、多彩な機能、高い信頼性、低い消費電力などが要求されており、LSI の回路規模や複雑さは著しく増加している。しかし、多様な製品に LSI を供給する多品種少量生産の現代においては、製品の開発サイクルが短縮され、短期間で高性能な LSI を設計する必要があり、設計規模の増大に設計能力が追いつかないという状況が生じ、設計生産性の危機が問題となっている。実際に、現実的な問題として携帯電話の熱暴走などが起こっており、十分設計期間の確保ができない状態でのテスト検証不足による問題の起因は軽視できない。

設計生産性の向上が可能な技術として、LSI の回路の動作を C 言語などのプログラミング言語を用いてより抽象的に記述し、LSI の回路構造を動作の記述から自動合成する動作(高位)合成技術が多数提案されている。動作合成では回路記述を抽象化することで回路設計が容易に行えることに加え、最適化により抽象的な記述から ASIC や FPGA など実装環境に適した回路を生成することで性能のさらなる向上が見込める。

本研究では OpenMP によるハードウェア動作合成システム[1][2][5]において OpenMP から verilog 記述をするのにコードジェネレータを用いる。そのコードジェネレータの生成ルールに基づき、画像処理プログラムとして知られている Sobel フィルタと hough 変換の設計を行う。手書きで設計したものとコードジェネレータで設計されたものを比較してハードウェア動作合成システムの性能を評価する。

sobel フィルタは画像処理の中でも基本的な操作の 1 つで、エッジ(輪郭)を検出するものである。1 次差分型のエッジ特徴抽出オペレータで全方向、水平、垂直の３種類の方向性が指定でき、指定方向のエッジを検出する。この方法は雑音の強調を押さえるため、重み係数を利用している。主に、複雑な画像認識、画像理解のための手がかりとして用いられている。

hough 変換は画像から直線や円を検出する技法として知られている。通常の直交座標上の画像を、極座標の二次元空間（直線検出の場合）に変換したり、三次元の空間（円検出の場合）に変換したりして、そこで最も頻度の高い位置を求め、それを逆変換して、直線や円を検出する。近年、ヒューマンインタフェースや防犯システム、福祉用途を目的とする人物の位置情報検出システムが注目されている。写像デバイスや演算プロセッサの急速な進歩に支えられて、装置の低価格化、高精度化、高信頼化が進んだ結果、画像認識技術が私たちの生活の中で数多く実用化されている。hough 変換はそうした技術における、膨大なノイズに対処できる技術であると言われている。

本論文では、第 2 章でハードウェア動作合成システムについて詳しく説明する。第 3 章で sobel フィルタ、第 4 章で hough 変換、第 5 章でコードジェネレータでの生成と手書きによる生成の比較、検証について述べる。

２．ハードウェア動作合成システム

2.1 ハードウェア動作合成システムの概要

ハードウェア動作合成システム全体の構成を図 1 に示す。このシステムは検証・評価系と実際にハードウェアを出力するハードウェア合成系で構成される。アルゴリズム評価系では、動作記述として書かれた OpenMP プログラムを、OpenMP コンパイラによってマルチスレッドプログラムに変換し、PC クラスタやマルチコア PC などの SMP 環境によってアルゴリズムの検証と並列化の評価を行う。すなわち、プロセッサ数を変化させて実行時間を計測し、速度向上を算出して並列化の効果を明らかにする。並列化アルゴリズムの評価・検証を行い、分析結果を用いて OpenMP プログラムを改善する。SMP 環境により、高速なソフトウェアシミュレーションを行うことが出来るため、検証時間の短縮と並列化アルゴリズムの評価を設計の早期に行うことが可能である。ハードウェア動作合成系では、アルゴリズム評価系の検証後、得られた OpenMP のソースコードの動作合成を行う。トランスレータを通して中間表現に変換した後、コードジェネレータで並列動作ハードウェアを生成する。トランスレータで出力される中間コードには、OpenMP で指定された並列化情報が含まれており、コードジェネレータではそれらを用いて最適化を行い、並列動作ハードウェアを生成する。

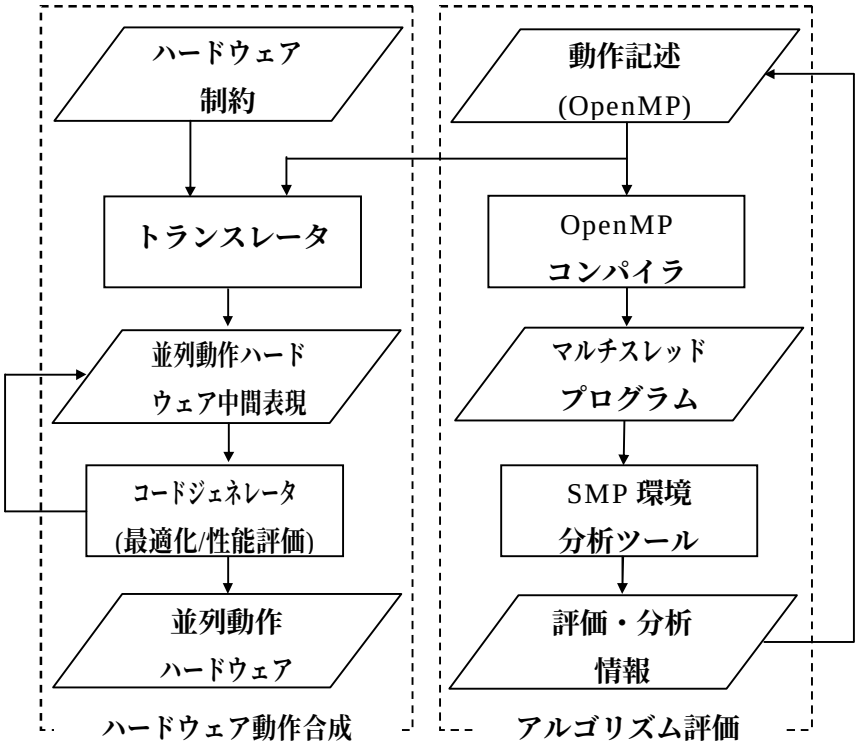

図 1　OpenMp によるハードウェア動作合成システムの構成

2.2 中間表現

HDL 記述に必要な中間表現について説明する。ハードウェア動作合成系におけるトランスレータは、動作記述である OpenMP プログラムを中間表現へと変換する。
トランスレータが生成するレジスタ転送方式である RTL 中間表現を用いてハードウェアの生成を行う。RTL の中間表現では処理を表すシンボルテーブルと制御情報を表す状態遷移表が出力され、両方を合わせてコントロールフローグラフ(CFG)を表す。シンボルテーブルは演算される変数や処理、代入先を示しており、状態遷移表によって次に遷移する状態が示される。

C 言語コードを中間コードのシンボルテーブルと状態遷移表に変換した例を図 2 に示す。サンプルのC言語コードは単純な while の無限ループとループ内で加算と変数への代入を行っている。状態遷移表の#0 で示される状態から、最初に CFG の 5 で示される定数の代入を表す”： =(2 4)”の処理が行われる。”： =(2 4)”ではシンボル 2 で示される変数 i に対し、シンボル 4 で示される定数 0 の代入を示している。次に while ループの条件式である #1 へ遷移し、条件式の判定を行い分岐する。ここでは無限ループの条件であるため、定数であるシンボルテーブルの 6 を参照し、真であることから#3 の状態へ遷移する。#3 では CFG の 8,9 に該当する加算と代入の演算を行った後、状態をループの先頭に当たる#1 へ遷移する。

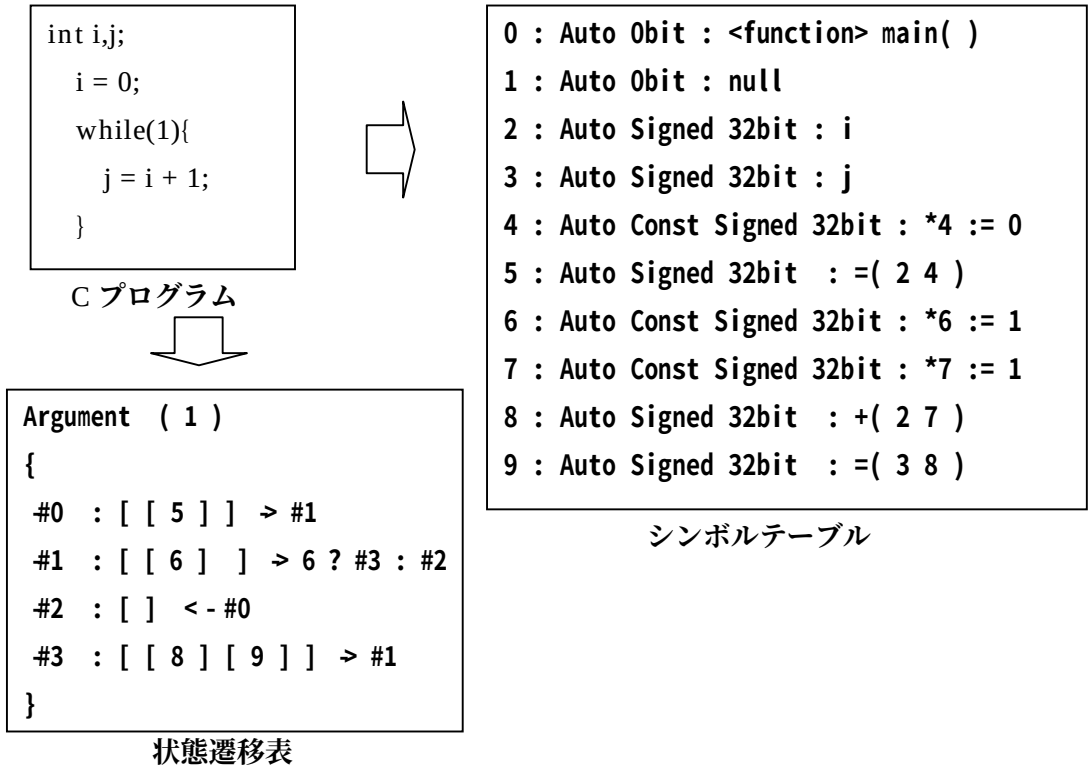

図 2：中間表現のサンプル

2.3 コードジェネレータ

ハードウェア動作合成系におけるコードジェネレータは、トランスレータによって生成された中間表現を用いてハードウェアの生成を行う。コードジェネレータの構成を図 3 に示す。コードジェネレータは以下の５つのモジュールで構成されている。

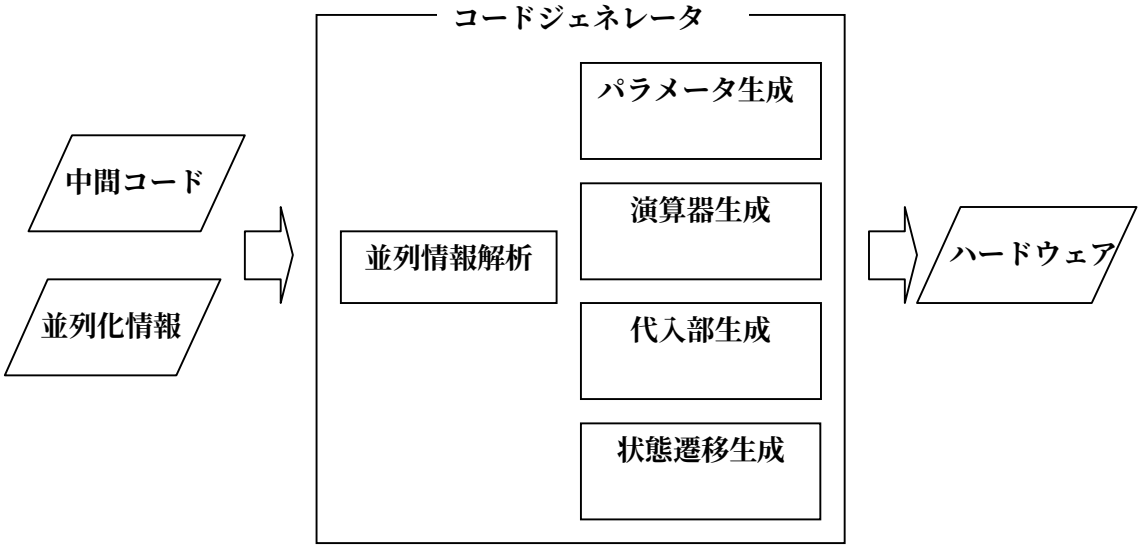

図 3　コードジェネレータの構成

・並列情報解析

前処理として、トランスレータによって生成された中間コードと OpenMP によって指示された並列化情報に用いて、並列処理部の中間コードの複製や内部定数の変換を行う。

・パラメータ生成

内部レジスタ、定数のパラメータ、状態遷移用の定数を生成する。

・演算器生成

用いられる各種演算器の生成とレジスタとの結線を行うコードを生成する。

・代入部生成

演算された結果や、メモリなどの外部からの入出力をレジスタに代入するコードを生成する。

・状態遷移生成

ハードウェアの FSM に当たる内部処理のコントロールを行うコードを出力する。

３．Sobel フィルタ

3.1　Sobel フィルタとは

1次微分フィルタと呼ばれ、画像を対象物体と背景に分割するために、対象物体の輪郭（エッジ）だけを抽出し、認識する手法である。

画像の明るさの変化によりオブジェクトの輪郭を算出することができる。明るさの変化値は、微分演算を利用することで算出できる。しかし、デジタル画像は連続ではないため、厳密には微分演算を行うことができない。そこで、隣接する画素の微分値の近似値を差分により算出する必要がある。隣接する画素の差分は、微分パラメータを用いて画素に重み付けをすることにより算出できる。この差分値が隣接する画素の微分値の近似値となる。

3.2　Sobel フィルタのアルゴリズム

Sobel フィルタは、ある注目画素を中心とした上下左右の 9 つの画素値に対して、図 4 に示すような係数をそれぞれ乗算し、結果を合計する。垂直方向、水平方向の二つの係数行列を用いてこの処理を行う。

-1　0　1 -2　0　2 -1　0　1	-1　-2　-1 0　0　0 1　2　1
水平方向 g_{HS}	垂直方向 g_{VS}

図 4　Sobel フィルタの係数

中央の係数が注目画素の部分になる。

水平方向の合計値を g_{HS}、垂直方向の合計値を g_{VS} としたとき、注目画素の画素値は以下の式で求めることができる。

$$g = (g_{HS}^2 + g_{VS}^2)^{1/2}$$

この注目画素をずらしながらこの計算を行うことによりその画像のエッジを求めることができる。

計算例

10	10	10
0	0	0
0	0	0

$$g_{HS}^2=(-10+0+0+10+0+0)^2$$
$$=0$$
$$g_{VS}^2=(-10-20-10+0+0+0)^2$$
$$=1600$$
$$g = (0+1600)^{1/2}$$
$$= 40$$

例えば、このような画素値があったとすると図 4 の係数を用いて水平方向、垂直方向に対して計算を行うとこのようになる。そこで、g にあらかじめしきい値を設定しておきそのしきい値よりも大きくなればエッジとして検出される。

3.3 HDL 記述と実行結果

3.3.1 HDL 記述

(1)**モジュールのインターフェース**

```
module to_edge( iSTART, oEND, oADDR, oDATA, iDATA, oRD, oWR, iEN, iRUN,
                iSTALL, CLK, XRST);
```

ポート宣言：関数の引数、module **のスタートストップ、アービタへのポート、クロック、リセット(負論理)**

(2)**パラメータ記述**

・定数のパラメータ

中間表現の「Auto Const Signed Xbit : ***数字 := 定数**」**の解釈**

「ConstNum **状態番号」で宣言**

```
parameter   [31:0]ConstNumOne = 32'd1;
parameter   [31:0]ConstNum11   = 32'd0;
parameter   [31:0]ConstNum13   = 32'd256;
```

・状態遷移番号

P_INIT **が初期状態,**P_END **が終了状態, 「**P_STATE **数字(状態番号)」で状態遷移番号を記述していく**

```
parameter   P_INIT       = 8'd0;
parameter   P_END        = 8'd1;
parameter   P_STATE12    = 8'd2;
parameter   P_STATE14    = 8'd3;
```

・配列アクセスの状態遷移

配列アクセスの状態遷移の追加

「遷移元の状態遷移番号_ARRAY_**読み込みなら** R **書き込みなら** W **数字」で宣言**

STALL **は何もしない状態のこと**

```
parameter   P_STATE44_ARRAY_R0        = 8'd171;
parameter   P_STATE44_ARRAY_R1        = 8'd172;
parameter   P_STATE44_ARRAY_R2        = 8'd173;
parameter   P_STATE44_ARRAY_STALL     = 8'd174;
```

・レジスタの宣言

内部変数(レジスタの宣言), 中間コードの「Auto Signed Xbit : **アルファベット」の解釈**

```
reg   [31:0]i;
reg   [31:0]j;
reg   [31:0]l;
reg   [31:0]m;
```

(3)状態遷移部

```
always @ (posedge CLK or negedge XRST) begin
    if(!XRST) begin
        CurrentState <= P_INIT;
    end else begin
        case(CurrentState)
        P_INIT      : begin
                        if(iSTART == 1'b1)
                            CurrentState <= P_STATE12;
                        else
                            CurrentState <= P_INIT;
                    end
         P_STATE12: CurrentState <= P_STATE14;
```

以下同様。基本的に分岐の状態遷移は前の演算結果を参照する。また、配列アクセスの場合は最後に STALL の状態に入り、復帰はアービタからのストールの返値を参照する。

(4)代入部

各状態に対し、一つのレジスタへ演算結果が代入される。左辺には、レジスタ、メモリのいずれかが来る。右辺には、パラメータ、レジスタ、add_wire 等の演算器の接続、汎用レジスタ、メモリのいずれかが来る。

```
always @ (posedge CLK or negedge XRST) begin
    case(CurrentState)
P_STATE12   : i <= ConstNum11;
P_STATE15   : i <= ADD1_RESULT;
P_STATE17 : j <= ConstNum16;
P_STATE20 : j <= ADD1_RESULT;
```

１状態につき一つのレジスタに値が格納される。配列アクセスの際の port への出力は状態遷移記述の部分で出力信号と結線することを考えている。

(5)演算器

各演算器(+,-,*,/,&,etc)に対し、assign 文を二つ宣言し、二項演算の各項に割り当てる。状態遷移を case 文の参照とし wire で結線する。

+の演算器の例：

```
wire [31:0] ADD1_RESULT; //レジスタへの代入で用いる
wire [31:0] ADD1_A, ADD1_B;//加算のオペランド

assign ADD1_RESULT = ADD1_A + ADD1_B;
assign ADD1_A = (CurrentState==P_STATE15) ? i:
                (CurrentState==P_STATE20) ? j:

assign ADD1_B = (CurrentState==P_STATE15) ? 32'd1:
                (CurrentState==P_STATE20) ? 32'd1:
```

各状態に対して一つの演算を行い、その結果を ADD1_RESULT で wire 接続する。これにより、ハードウェアとして演算器が一つ、ポートにセレクタがついた回路が合成される。

3.3.2 実行結果

図 5(a) 原画像

図 5(b) Sobel フィルタ画像

本研究ではよく画像処理に用いられるレナ画像で検証を行った。
図 5 (a)が原画像で図 5(b)が Sobel フィルタを通したときの画像である。

４．Hough 変換

4.1　Hough 変換とは

Hough 変換とは、直線、円、楕円といったパラメトリックな図形を 2 値化された画像から取り出す特殊抽出法である。利点として雑音を含む画像からも特徴を抽出できるという点で優れている。元の画像をパラメータ空間に写像し、パラメータ空間内でピークを検出してそのピークの表す図形が計算結果として検出される。ただし、直線や円は必ずしも連続である必要はない。

Hough 変換による直線検出は、

1. 画像中の雑音に影響されることなく安定的に直線を検出できる。
2. 隠ぺいや画像処理の不完全性によって直線が不完全になっていても検出できる。
3. 複数の直線を同時に検出できる。

などの有益な特徴があるが、以下に示すような問題点も数多く存在する。

1. パラメータ空間を表すための大きなメモリが必要となる。
2. パラメータ空間をデジタル化する際のサンプリング間隔（解像度）を決める基準が明確でない。
3. 画像中の各エッジ点に一本の Hough 曲線を描くため計算量が多くなる。
4. Hough 曲線の集積点（ピーク）を抽出する方法が明確でない。
5. 直線検出後、その直線の中から必要な線分（セグメント）を抽出する方法が明確でない。

・ パラメータ空間の解像度
　この解像度が細かすぎると、一つあたりの要素に投票される回数が減少するのでピーク検出が困難になる。逆に解像度が粗すぎると、検出される図形の精度が下がる。

・ 計算コスト
　パラメータの数の次元の配列が必要なので、必要になるメモリの量と、検出にかかる時間はコンピュータが進歩した現在でもばかにできない。

Hough 変換の基礎概念（直線検出）

x-y 座標のある点(x_0, y_0)を通過する直線群は、傾き *a*、切片 *b* をパラメータとして

$y_0 = a \cdot x_0 + b$　(1)

で表せられる。これは、*a-b* パラメータ空間では、

$b = -a \cdot x_0 + y_0$　(2)

例えば、図 6 のように点　(x,y) = A(1,1),B(2,2),C(4,4) を置き、3 点を通る直線を求めるために、別のパラメータ空間である a-b パラメータ空間に置き換えてみると、3 直線に対する交点が求まる。この点 A(1,0)　a = 1,b = 0 つまり y = x が、x-y パラメータ空間での点 A,B,C における直線となる。

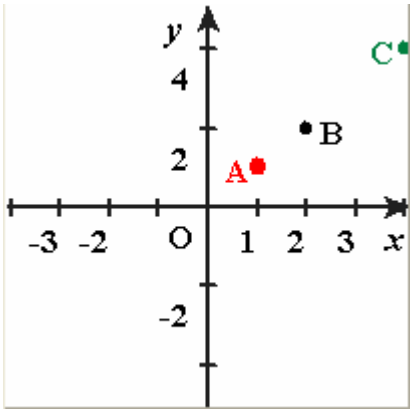

図 6 (a)　x-y パラメータでの要素

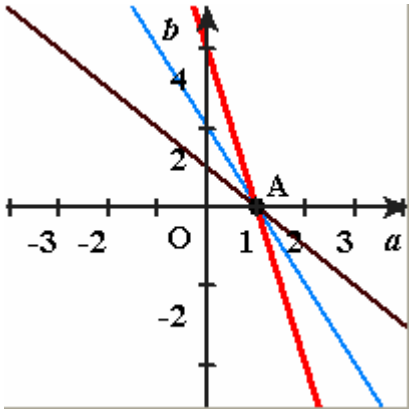

図 6 (b)　a-b パラメータでの直線の交点

このようにして、あるパラメータ空間を別のパラメータ空間に置き換えることで直線を検出できるのが Hough 変換の考え方である。

4.2　Hough 変換のアルゴリズム

4.2.1　直線の検出

直線の場合、パラメータは画像の中心からの距離と傾き 2 つであるから、2 次元の配列を用意する必要がある。この配列は 2 つのパラメータの値を表しており、各要素が 1 本の直線を表している。

画像処理における Hough 変換では図 7 に表すように点(x,y)を通る直線は、その直線に交わる原点からの垂線の長さρと、その垂線でのできる角度θで表すことができる。点（x,y）を通る直線は多数あり、点（x,y）が決まると、θとρの多数の組み合わせが得られる。例えば、別の点が図 7、点(x,y)以外にあり同じ直線上にあるとすれば、その 2 つの点はθとρの同じ組み合わせを持っているということである。θとρの同じ組み合わせが多数あることがわかれば、もとの点は同じ直線上にあるといえる。式(1)は

$$y = -\frac{x}{\tan\theta} + \frac{\rho}{\sin\theta} \qquad (3)$$

に容易に変形でき、これは、傾き-1/tanθ、切片 ρ/sinθ となり、数学的に式(1)と等価である。ところで、に示すように、式(1)を変換式とした場合、垂角 θ→0 になると、切片 b→±∞ になり、また、垂角 θ→π/2 となると、傾き a→±∞となる。従って、パラメータ空間を表現するために確保するな二次元配列は無限大のサイズが必要となる。この意味から、このような a-b パラメータ表現法は禁止的である。

図 7 の右図ではθからρを求める式の原理を表している。

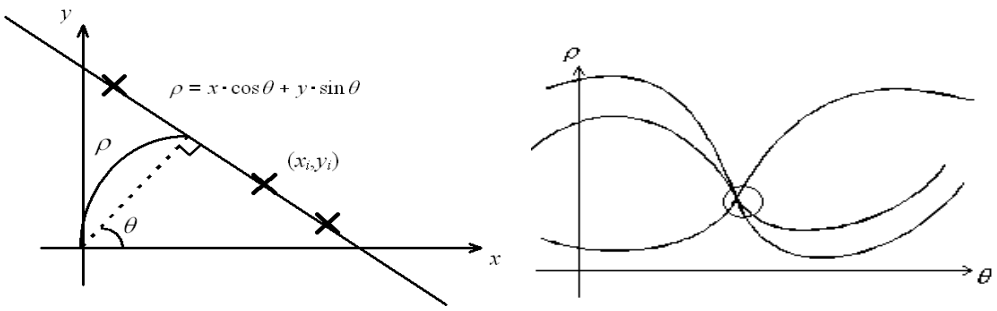

図 7　直線を検出するためのハフ変換式の原理

4.2.2　円の検出

図 8 に示すように直交座標上の点(X,Y)を通るすべての円は、円の中心点(centerX,centerY)と半径 r で表すことができる。

円の検出は、centerX,centerY を変化させながら、

$$r^2=(X-centerX)^2+(Y-centerY)^2$$

の関係により r を求める。

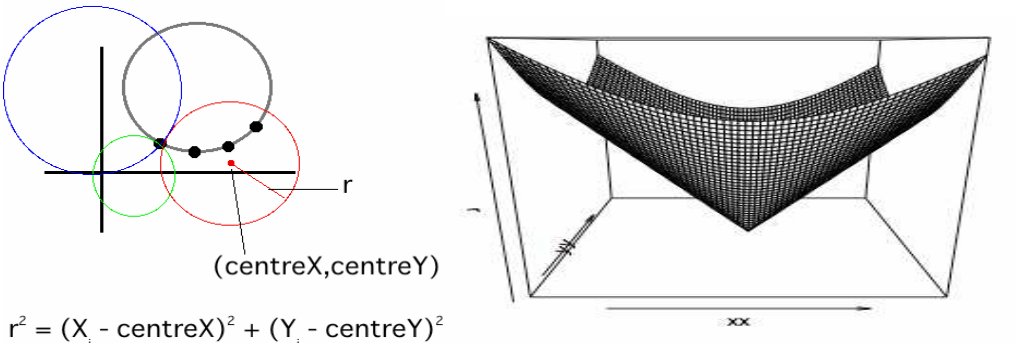

図 8　Hough 変換(円の検出)の方法

上の式を用いて、直角座標上の点（x, y）を新しい三次元空間（centerX, centerY, r）上に変換すると、直角座標上の一点は三次元空間上の 1 枚の面に対応している。直角座標上の点が多数あると、空間上に多数の曲面が得られる。それらの曲面が共有する点があれば、それは元の x-y 直角座標上では一個の円上に並ぶことになる。

5．ハードウェア動作合成システムの検証

5.1 実験

(1)実験環境

動作合成システムの実験環境として、用いた環境を表 1 に示す。

表 1：実験環境

ハードウェア動作合成	論理合成ツール	Xilinx ISE 9.1i
回路シミュレーション	PC 環境	Intel Core2 duo 2.4Ghz,Memory 2GB
	シミュレーションツール	ModelSim XE III 6.1e

(2)実験結果

Sobel フィルタで用いた画像は、解像度 256×256、輝度 0～255 の pgm 画像である。回路シミュレーションを用いて測定した必要な動作クロック数と論理合成によって得られた動作周波数を表 2、表 3 に示す。表 2 はコードジェネレータにより生成されたものである。表 3 は手書きで HDL 記述を行い、生成したものである。クロック減少比は，スレッドが一つの逐次処理の場合に対する動作クロックの比率を示している。

表 2：Sobel フィルタのコードジェネレータでの実行速度

	スレッド数	1	2
回路シミュレータ	動作クロック数(kcycle)	9816	4910
	クロック減少比	1.00	2.00
並列ハードウェア	動作周波数(MHz)	91.66	91.66
	速度向上比	1.00	2.00

表 3：Sobel フィルタの手書きによる記述の実行速度

	スレッド数	1	2
回路シミュレータ	動作クロック数(kcycle)	8203	4105
	クロック減少比	1.00	2.00
並列ハードウェア	動作周波数(MHz)	91.66	91.66
	速度向上比	1.00	2.00

どちらにおいても、動作クロック数については，スレッド数の増加に対して理想的にクロック数が減少した。動作周波数については，想定するハードウェアのデータパスがほぼ同じであるため，同一であった。結果として，スレッド数に対しほぼ理想的な速度向上を得ることができた。

論理合成による回路面積を表 4、表 5 に示す。回路面積比がスレッド数の比率以上に増加するのは，並列化のオーバーヘッドにあたる arbiter やレジスタ，配線などの回路の増加によるものと思われる。

表 4：Sobel フィルタのコードジェネレータでの回路規模

スレッド数	1	2
回路規模(Slices)	2251	6502
回路面積比	1.00	2.89

表 5：Sobel フィルタの手書きによる記述の回路規模

スレッド数	1	2
回路規模(Slices)	2234	6471
回路面積比	1.00	2.90

5.2 比較による検証

表 2、表 3 を比較すると、手書きで記述したものが、コードジェネレータで生成したものより、動作クロック数が 0.84 倍と、減少していることがわかる。これは、コードジェネレータは自動でコード生成するので必要のない記述があるためである。プログラムを見る限りではまだまだ最適化できそうである。

表 4、表 5 を比較すると、回路規模に関してはほとんど変化が見られなかった。それは、コードジェネレータと手書きを比べて違いのあるところは状態遷移生成と代入部生成の部分なので実際の処理内容がほとんど変わってないためだと考えられる。

6．おわりに

本研究ではハードウェア動作合成システムにおいて OpenMP で書かれた画像処理プログラムより生成された中間コードから verilog-HDL を記述してそのプログラムを手書きによるコード生成とコードジェネレータによる生成を比較して検証を行った。

手書きで記述をすることにより、コードジェネレータよりも約 1.2 倍の時間短縮ができた。それは、コードジェネレータは自動でコード生成するので必要のない記述までしてしまうためである。

今後の課題としては、スレッド数を増やしてもっと詳しく検証を行うこと。また、Hough 変換についても実験を行うこと。それにより、まだまだコードジェネレータの最適化ができるものと考えられる。

謝辞

本研究の機会を与えてくださり，貴重な助言，ご指導をいただきました山崎勝弘教授に深く感謝いたします。また本研究に関して貴重なご意見を頂きました、松崎氏、志水氏、そして様々な面で貴重な助言を下さった高性能研究室の皆様に心より感謝いたします。

参考文献

[1] 松崎裕樹：”OpenMP を用いたハードウェア動作合成：画像処理に対する検討とパーサー部の実装” 立命館大学大学院 理工学研究科,修士論文,2008.

[2] 中谷嵩之：”OpenMP によるハードウェア動作合成システムの設計と検証” 立命館大学大学院 理工学研究科,修士論文,2007.

[3] 小林優：”改訂 入門 Verilog-HDL 記述” ,CQ 出版社, 2004.

[4] 酒井幸市：”デジタル画像処理入門” ,CQ 出版社, 2002.

[5] 中谷嵩之、松崎裕樹、山崎勝弘：”OpenMP によるハードウェア動作合成システムの設計と検証” FIT2007, 2007.