卒業論文

HDL による高速ラベリングの設計

氏　　名：　西垣　亮介
学籍番号：　2210020339-5
指導教員：　山崎　勝弘　教授

提 出 日：　2006 年 2 月 20 日

立命館大学理工学部情報学科

内容梗概

本研究では、株式会社ナノスコープと産学協同プロジェクトの一環として、マハラノビスによる画像判別での前処理であるラベリングを設計した。まず、ソフトウェアによって C 言語でラベリングアルゴリズム実装を行う。次に、ソフトウェアを基にモジュール分割し、ハードウェア記述言語である VHDL によって設計を行った。この検証を行うため、MenterGraphics 社の Modelsim SE によってシミュレーションを行った。

本論文では、まずハードウェア記述言語などのハードウェア設計での現状について述べ、ラベリングアルゴリズムのソフトウェア、ハードウェアの設計、シミュレーションを行う。

図目次

表目次

1. はじめに

近年、フラットディスプレイ(FPD:Flat Panel Display)の需要が高まっており、米DisplaySearch 社によれば、2005 年における FPD の市場規模は対前年比 20%増の 743 億 7000 万米ドルに達する見込みである。2005 年時点で既にディスプレイ市場の 88%を FPD が占めており、伸び率は今後、鈍っていくものの、2009 年まで平均 10%程度の年間成長率で伸長し、2008 年には 1000 億米ドルを突破すると DisplaySearch は予測している。用途別に市場規模を見ると、今後の伸び率が最も高いのは液晶テレビである。2005 年は FPD 市場全体の 15．5%を占めるに留まっているが、2009 年には 28．3%へ上昇すると見込んでいる。PDP(Plasma Display Panel)テレビと合わせると 36%に達すると見込まれている。薄型テレビ需要は台数ベースでも、2005 年の 2710 万台から 2009 年は 9000 万台へ増加する見通しである[3][4] 。

液晶パネルに関しては、シャープや韓国 Samsung Electronics Co.,Ltd.、韓国 LG.Philips LCD Co.,Ltd.、IPS アルファテクノロジなど各社 2006 年度以降、液晶パネルの大増産を計画している[3]。

以上の背景を踏まえて本論文では、株式会社ナノスコープとの産学協同研究において、ハードウェア記述言語 VHDL によってマハラノビス距離を用いた画像判別を行う際の、過程であるラベリングの高速化を行う。これからますます需要の増える FPD などのガラス検査に応用することを目標とする。生産量に対してガラス検査のスピードが遅いと、検査待ちのガラスが増え、出荷までのスピードを遅らせてしまう。本研究では、ガラス検査における前処理であるラベリングを高速化することによって、ガラス検査全体の高速化をすることを目的にしている。まず、ラベリング処理をソフトウェアによって実現する。次に、ハードウェア実装を考慮して処理全体を大まかな機能単位で 4 つに分割し、メインな処理を行う機能をさらに 5 つの機能単位に分割する。そして、各機能ブロックをハードウェアモジュールとして設計する。

なお、今回の設計にあたって使用したハードウェア設計支援ツールは Altera 社の統合開発環境ツール QuartusⅡでバージョンは 4.1 である。シミュレーションツールは Modelsim SE を用いた[7]。

本論文では、本章で研究全体の背景と目的を明らかにし、第 2 章において、システム設計、ハードウェア記述言語、ASIC、及び再構成可能ハードウェアの概念の説明を示す。第 3 章では、ラベリングアルゴリズムのソフトウェアでの実装について述べ、第 4 章では、各モジュールのハードウェア設計について述べる。最後に、第 5 章において、現在までの成果と今後の課題について述べる。

2. ハードウェア記述言語によるシステム設計

2. 1 システム設計

組み込みシステムの設計では、小型化、高性能化、低価格化、さらに低消費電力化が要求されているため、上流工程からソフトウェアとハードウェアの機能を検討しながらの専用 LSI の設計やシステム全体を一つのチップに実装する SoC の設計が必要である。このシステム設計では、システム記述からのソフトウェア／ハードウェアの機能分割や協調検証(コシミュレーション)が重要であり、従来のボトムアップ設計に替わるトップダウン設計が注目されている[2]。

LSI の価格は、設計費と製造費に大きく分けられる。設計費は、LSI にレイアウト(配置配線)するためのマスクを作成する費用で、製造費は、このマスクを利用して LSI を製造する制作費と材料費(シリコン)である。したがって、LSI チップの価格は、高い設計費を多くの製造ロットで割ることによる量産効果で、非常に低価格となる可能性がある。

また、半導体のテクノロジ(微細加工の技術)は、年々進歩しているため、HDL で設計した回路を最新のプロセスで製造することにより、チップ面積が小さくなるために、チップ単価が安くなるとともに、高速化や低消費電力化が可能である。

HDL による設計は、設計効率、ドキュメント性、設計資産の再利用性の向上、及び設計期間・デバッグ期間の短縮が可能である。また、システム設計には、専用に設計する特定用途向け IC の ASIC(Application Specific IC)や特定用途向け標準品の ASSP(Application Specific Standard Product)、さらに命令を拡張した専用プロセッサの ASIP(Application Specific Instruction-set Processor)を利用することが有効である。これらのシステム全体を実装するシステム・オン・チップ(System on a Chip)では、知的財産の IP(Intellectual Property)を効果的に利用し、設計資産の活用や設計期間の短縮を行いながらシステム LSI を設計することが重要である。

2. 2 ハードウェア記述言語

半導体製造技術の進歩により LSI の集積度が、飛躍的に工場しているため、システム LSI の設計には、大規模な回路を短期間で設計し、さらに設計資産の再利用による設計時間の短縮が要求されている。そこで、従来の CAD(Computer Aided Design)によるゲート・レベル設計よりも、上位の言語レベルで記述するハードウェア記述言語(HDL: Hardware Description Language)による設計が必要になってきている[2]。

代表的な HDL には、VHDL と Verilog-HDL がある。

（1）VHDL(VHSIC Hardware Description Language)

米国国防総省の VHSIC(Very High Speed Integrated Circuit)委員会で 1981 年に提案された。当時、国防総省向けの ASIC 開発は長いもので 3 年から 4 年もかかっていたため、開発当初の時点では一番スピードが速い ASIC を使用していても、半導体プロセスの進歩によ

って、開発が完了する時点では時代遅れになってしまうという問題が発生していた。そこで、直接ロジック・ゲートを回路図で入力するのではなく、HDL で記述することによって、開発終了時に一番スピードの速い ASIC を選択して論理合成できるようになりました。現在では、国防総省が調達するすべての ASIC は、電子システムの仕様記述言語として VHDL の記述付きで納入されるように義務付けられている。VHDL は、プログラミング言語 ada を基に開発された言語であり、ディジタル・システムのドキュメント記述から、設計、シミュレーションまでの幅広い範囲を記述することが可能である。VHDL は、その後 IEEE の標準ハードウェア記述言語として採用され、IEEE-1076 仕様が 1987 年に VHDL'87、さらに 1993 年に VHDL'93 として標準化されている。

（2）Verilog-HDL

シミュレーションツール Verilog-XL のためにシミュレーション専用言語として、ゲートウェイ・デザイン・オートメーション社によって開発された。C 言語を基にシミュレーション専用の言語として開発されたため、LSI 設計を重視した言語であり、記述性やシミュレーションの機能が充実している。Verilog-HDL は、IEEE-1364 仕様として標準化されている。

HDL を用いたハイレベル設計では、HDL による機能検証を行った後に、論理合成ツールを用いてゲート回路を自動生成することにより、同じ HDL の記述から設計制約条件によってコスト・パフォーマンスなどに合った LSI の製造が可能となる。

2. 3 ASIC

ASIC(Application Specific IC)は、特定用途向きの集積回路である[2]。ASIC には、特定ユーザ向けの IC の USIC(User Specific IC)と特定用途向け標準費の ASSP(Application Specific Standard Product)の区別がある。USIC は、セミカスタム IC とフルカスタム IC に、さらにセミカスタム IC は、ゲートアレイ、スタンダード・セル・アレイ、エンベディッド・セル・アレイに分類することができる。また、ASSP は、半導体メーカが不特定多数のユーザに販売する LSI で各産業分野で共通に利用可能な標準部品である。

- **ゲートアレイ(GA: Gate Array)**

チップ上に基本セルを縦横のアレイ状に規則正しく配置したマスタウェハを準備しておき、その上にユーザの仕様に合わせて金属の配線のみを行って回路を実現している。したがってゲートアレイは、あらかじめ基本セルの製造工程が終了しているウェハに対して、配線の工程を行うだけで実現するため、開発期間が短く、回路変更が容易で、多品種少量生産に向いている。

- **スタンダード・セル・アレイ(SCA: Standard Cell Array)**

基本セルの配置と配線のすべてをユーザの仕様に合わせて実現するため、チップ面積の縮小や性能の向上など設計の最適化が可能である。ただし、回路が完全に決定してから基本

セルの配置配線を行うため、開発期間が長く、回路の変更も困難である。

- **エンベディッド・セル・アレイ(ECA: Embedded Cell Array)**

ゲートアレイの短い設計期間とスタンダード・セル・アレイの高集積性を合わせて実現している。例えば、回路規模とメガセル(MPU やメモリなど)が決定した時点で、ゲートアレイと同様の基本セルとメガセルの配置配線を行う。その後、ゲートアレイと同様に、ユーザの仕様に合わせて配線のみを行う。

表 1 に GA SCA ECA と FPGA の特徴を示す。

表 1：GA SCA ECA と FPGA の特徴比較

	GA	SCA	ECA	FPGA		
試作・製造期間	○	×	○	◎	◎短い	×長い
設計・開発費用	△	△	△	◎	◎安い	△高い
性能（スピード）	◎	◎	◎	△	◎速い	△遅い
実装できる機能	○	◎	◎	△	◎高い	△低い
量産時のチップ単価	○	◎	○	×	◎安い	×高い

最近の組み込みシステムでは、小型化、高速化、低価格化、及び低消費電力化が要求されているため、ASIC を含む LSI 設計の技術がキーポイントになっている。

2. 4 再構成可能ハードウェア

最近、再構成可能ハードウェアという言葉が使われることがある[2]。これは、従来の回路は、簡単に論理を変更できないハードウェアをかたいハードウェアとすれば、設計データによって柔軟に回路を再構成可能なハードウェアのことを意味している。このように、ユーザが内部論理を変更できる代表的なデバイスとしては、その回路規模や内部構成によって PLD、CPLD、及び FPGA に分類することができる。

- **PLD(Programmable Logic Device)**

入力信号や出力信号を選択できる AND アレイ、OR アレイ、及びフリップフロップと出力を選択できる I/O 機能ブロックから構成されている。回路規模は、標準のゲートやフリップフロップの数個から十数個に相当する機能が実現できる。

- **CPLD(Complex Programmable Logic Device)**

ロジック・エレメントの基本ブロックを複数まとめたロジック・アレイ・ブロックから構成されていて、縦横のインターコネクトの高速ラインで接続されている構造である。CPLD は、PLD を 8 個程度とそれらを接続するプログラム可能な相互配線から構成されている。

- **FPGA(Field Programmable Gate Array)**

コンフィギュアブル・ロジック・ブロック(CLB)の基本ブロックから構成されていて、スイ

ッチ･マトリクス(SM)を経由して縦横のローカル･ラインで配線できる柔軟な構造である。FPGA は、プログラム可能な論理セルの集合体とスイッチ・マトリクスを使って、それらのセルを相互接続できる配線から構成されている。

表 2 に PLD CPLD FPGA の特徴を示す。

表 2：PLD CPLD FPGA の特徴比較

	PLD	CPLD	FPGA
回路規模	小規模	中規模から大規模まで	中規模から大規模まで
内部構造	AND-OR アレイ	PLD とインターコネクト	論理セルとローカル配線
配線時間	高速	規模により中速	配線の自由度が高いため低速
遅延時間	一定	配線によって固定	スイッチの数によって可変
書込方式	PLD プログラマ	ケーブルによる ISP(*)	ケーブルによる ISP

(*)ISP: In-System Programming：ボードに実装したまま内部論理の書き換えが可能

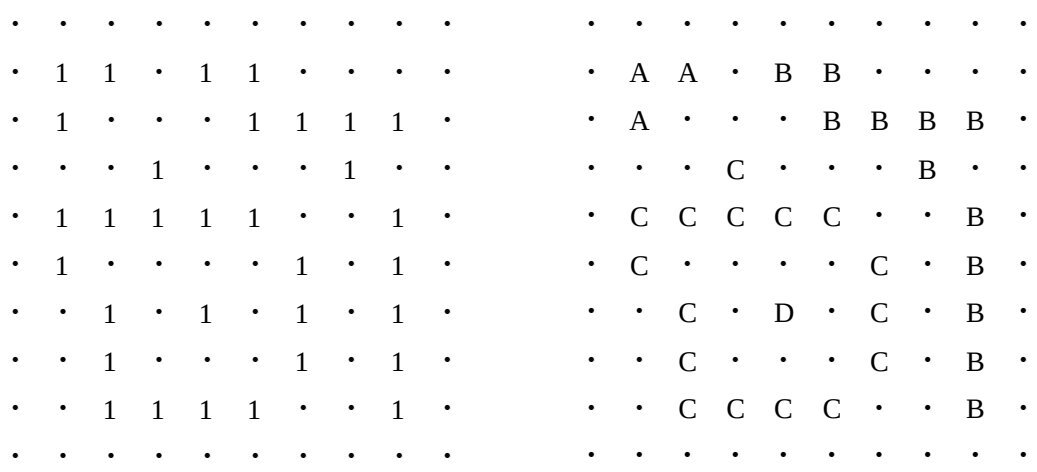
CPLD や FPGA は、パソコンやワークステーションで回路を設計し、ターゲット用に配置配線したデータをケーブルなどで転送してコンフィグレーションすることによって、内部論理を再構成することが可能である。

3. ラベリングアルゴリズムのソフトウェア実装

3.1 ラベリングとは

同じ連結成分に属するすべての画素に同じラベルを割り当て、異なった連結成分には異なったラベルを割り当てる操作を、連結成分のラベリングという。ラベリング操作は、ここの連結成分の属性の解析に先立って各成分を抽出するために、不可欠の操作である[1]。

近傍処理によるラベリングのアルゴリズムを図 1 に示す。



(a) 入力画像　　(b) ラベリング結果

図 1：ラベリング例

3. 2 ラベリングアルゴリズム

ラベリングアルゴリズムのフローチャートを図 2 に示す。

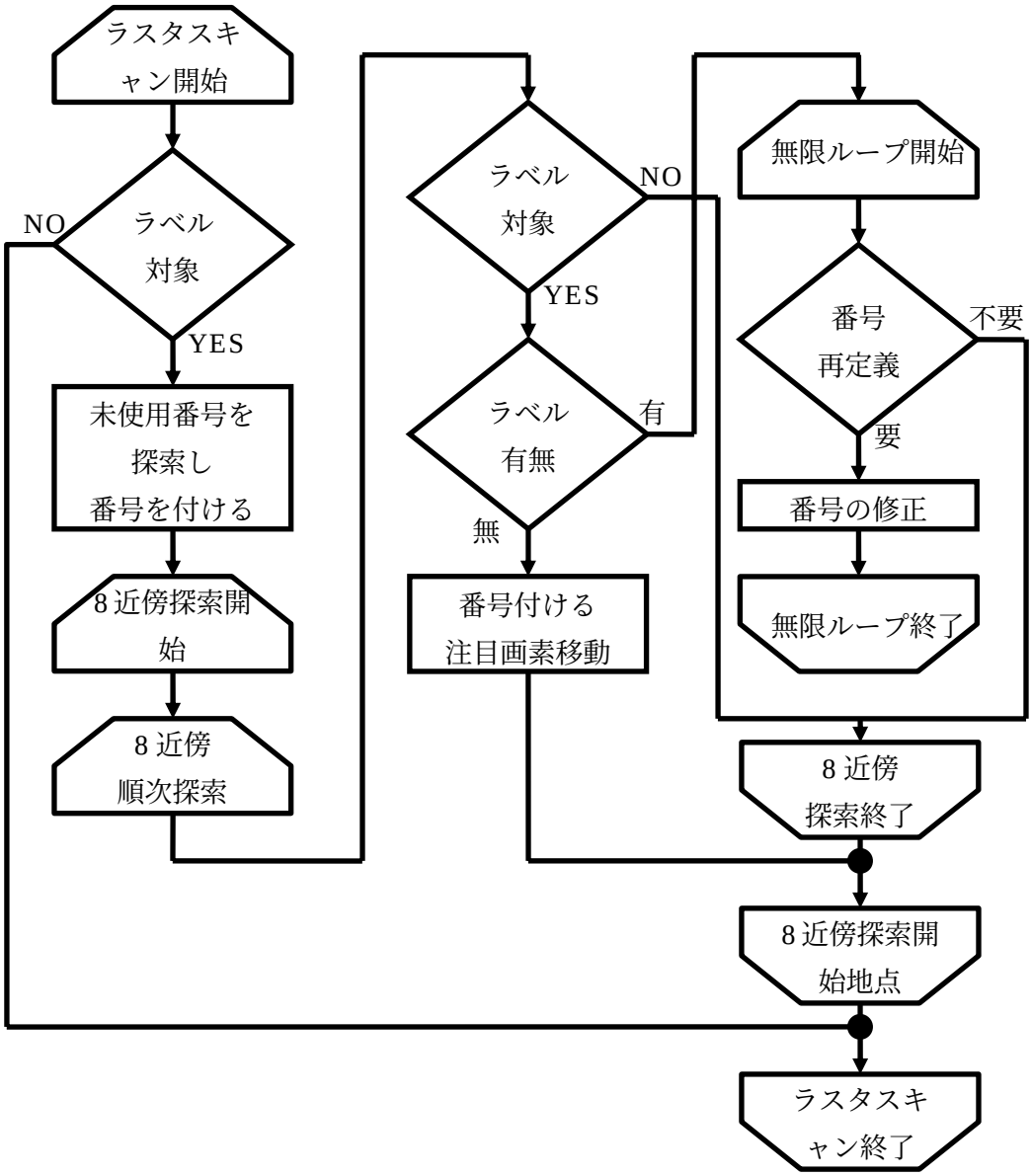


図 2：ラベリングアルゴリズムのフローチャート

最初に対象の画像を加工した画像を保存する画像（ラベルマップ）とラベル番号を管理する表（ラベルテーブル）を用意する。ラベルマップには画素を表すアドレスと輝度値を表す要素があり、すべての要素に 0 を代入し、初期化を行う。ラベルテーブルはラベル番号と参照番号を表す要素があり、すべての要素に 0 を代入し、初期化を行う。

次に、対象画像を左上から右下にラスタスキャンを行う。その際に、それぞれの画素がラベルの対象か否かを判定する。対象の画素を見つけると、その画素から 8 近傍の探索を

行う。注目している画素（注目画素）を判定する前の画素からの方向（進行方向）によって探索を行う順番が異なる。進行方向の左後から左、左前、前、右前、右、右後、後の順に探索を行い、対象の画素を発見すると、その画素に注目を移す。ただし、ラベルの付いていない画素に限る。

ラベルは対象画像の周囲を囲む様に付けられ、8 近傍探索を始めた画素に戻ってくると再びラスタスキャンを行う。ただし、ラベルが付いている画素は無視をする。そして、対象画素を見つけると再び 8 近傍探索を行う。

ラベリング処理を行う過程でラベル番号の違い（衝突）が起こる。その場合、近接するラベル番号の中で参照番号が一番小さい値に参照番号を修正する。

ラスタスキャンが終了すると、ラベルテーブルを修正する処理を行う。今のまま、マップの書き換えを行うと、出力のラベル番号が昇順ではない。したがって、ラベルテーブルの参照番号を昇順になるように修正する処理を行う。

最後に、もう一度ラスタスキャンを行って、ラベルマップのラベル番号と修正されたラベルテーブルの参照番号によってラベルマップの書き換えを行う。ラベリングされた画像と背景を区別させるために、0（ラベリングされていない値）を 255 に変換する処理を並行して行う。これによって、ラベリングされた画像は黒に、背景は白に色分けされた濃淡画像が出力される。

3. 3 具体例による説明

ラスタスキャンは対象画像を左上から右下まで横に探索を行う。ラスタスキャン中にラベル対象の画素を発見し、ラスタスキャンの一時停止状態を図 3 に示す。

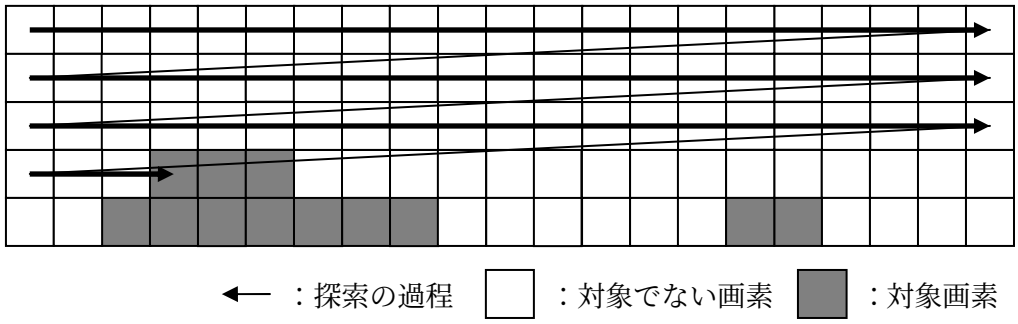


図 3：ラスタスキャン探索

8 近傍の探索の順番は、進行方向によって探索を行う順番が異なる。進行方向の左後から左、左前、前、右前、右、右後、後の順に探索を行う。

進行方向が下と右上のときの探索順番を図 4 に示す。

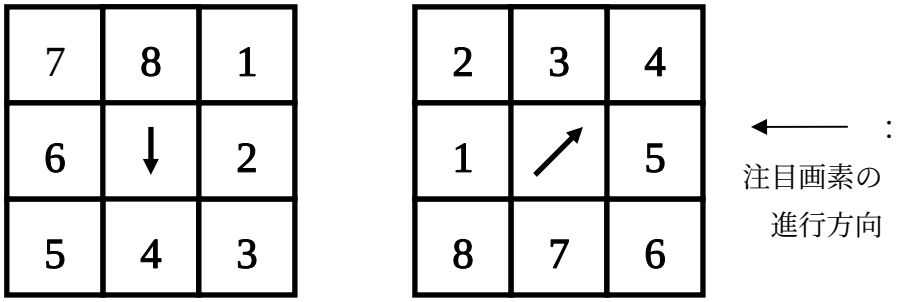


図 4：8 近傍での進行方向に対する探索順番

ラスタスキャンで対象画素を発見し、8 近傍探索を行い、開始画素に戻るまでの様子を図 5 に、ラスタスキャン再開後の探索の様子を図 6 に示す。

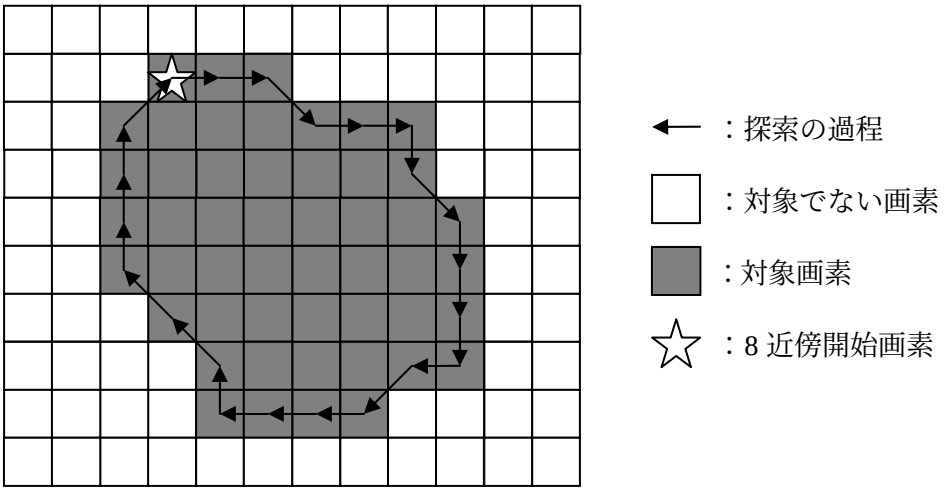


図 5：8 近傍探索の例

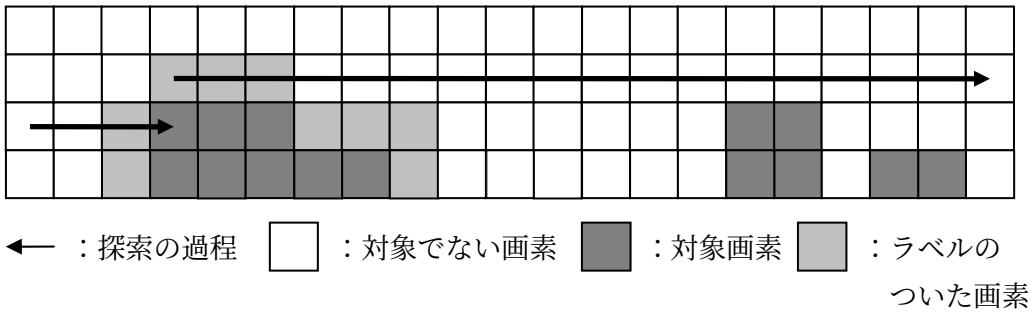


図 6：ラスタスキャン再開後の様子

ラベリング処理を行う過程でラベル番号の衝突時の探索を図 7 に示す。この注目画素は進行方向が右なので、最初に、注目画素の左上を判定するがラベル対象でないので無視す

る。

次に、注目画素の上を判定するが既にラベル番号がついている。この場合、ラベル番号の参照番号同士を比べ、今付けているラベル番号の参照番号より値が小さい場合、その値に参照番号を書き換える処理を行う。そして、右上も同様にラベル番号が付いているが、既に参照番号同士を比べているので、この画素は無視する。続いて、右を判定するとラベル対象であり、ラベル番号がついていないため、注目画素を移し、ラベル番号を付ける。

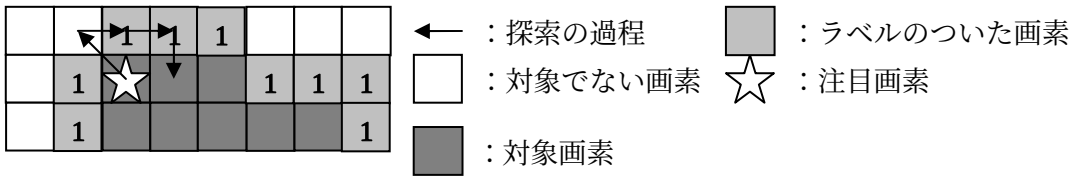


図 7：ラベル衝突時

これまでの処理をラスタスキャン終了まで行ったときの、ラベルマップに付けられたラベル番号の例を図 8 に示す。

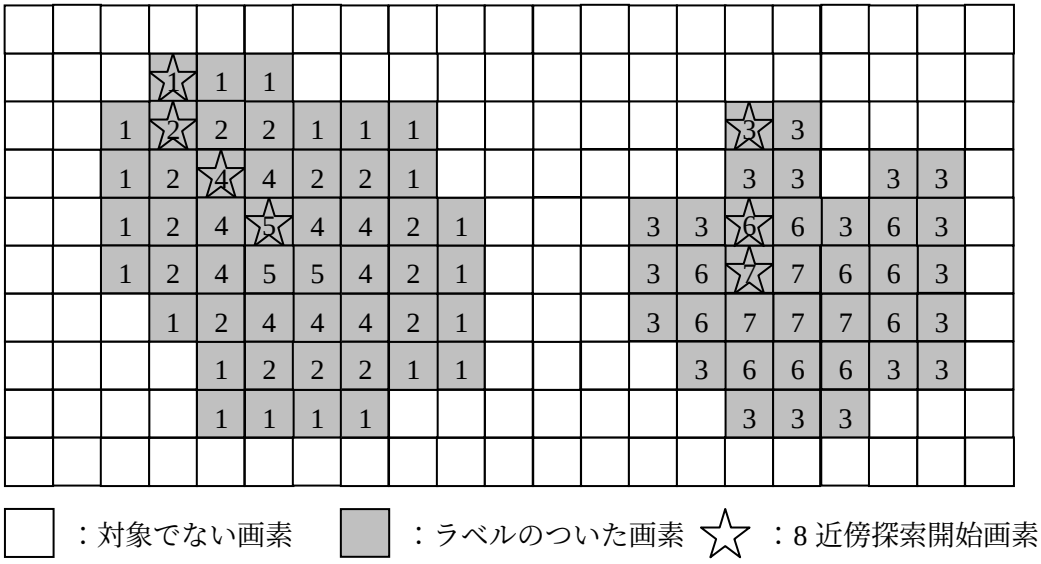


図 8：ラスタスキャン終了時の例

ラスタスキャン終了後、出力する画像のラベル番号を昇順に付けるためラベルテーブルの書き換えを行う。この処理を表 3 に示す。(a)はラベルテーブルを書き換える前であり、ラスタスキャン直後の参照番号となっている。(b)は書き換え後のラベルテーブルであり、この参照番号によってラベルマップを書き換える処理を行う。

表 3：ラスタスキャン後のラベルテーブルの処理前後

(a)処理前

ラベル番号	参照番号
1	1
2	1
3	3
4	1
5	1
6	3
7	3

(b)処理後

ラベル番号	参照番号
1	1
2	1
3	2
4	1
5	1
6	2
7	2

図 8 のラベルマップと表 3 のラベルテーブルを基に、書き換えられたラベルマップを図 9 に示す。

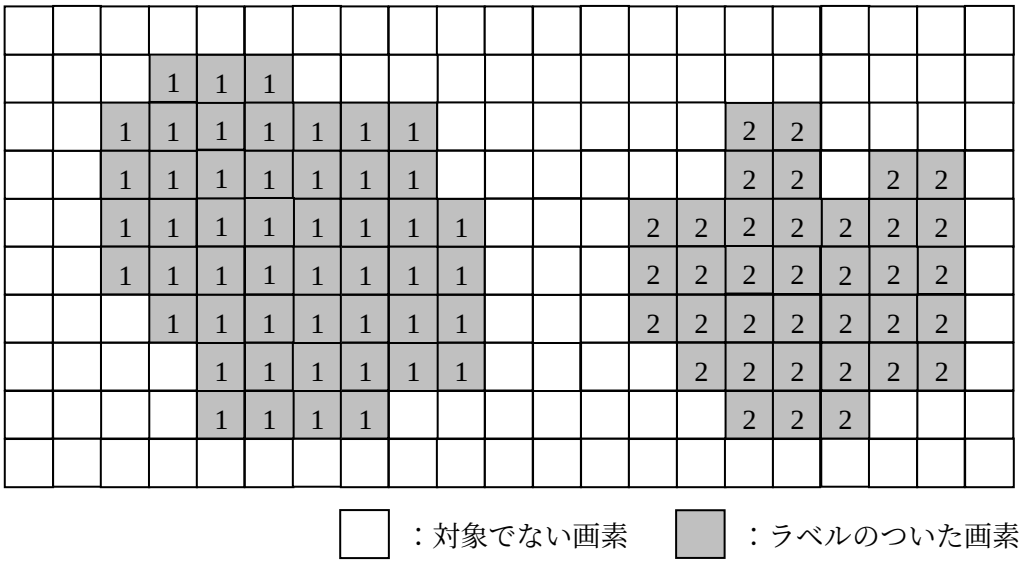


図 9：出力されるラベルマップ

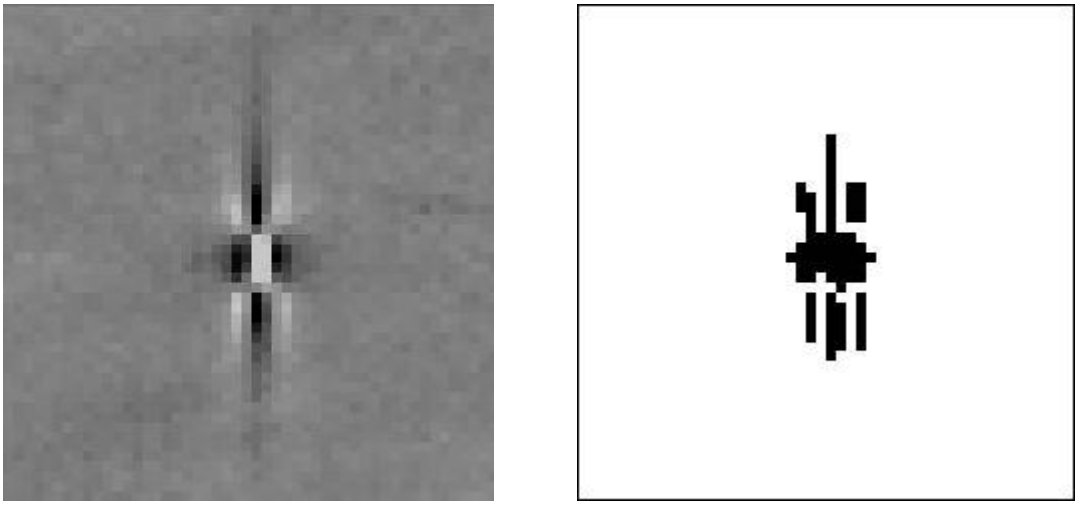
3. 4 C 言語によるソフトウェア実装

3.2、3.3 節で述べたラベリングアルゴリズムを基に、C 言語によってソフトウェアを実現した。開発の流れとしては、Windows 上の Cygwin 環境において開発と検証を行い、コンパイラは gcc-3.3.3 を利用した。ナノスコープに提供していただいた入力画像データは RAW ファイルであり、50bit×50bit の濃淡画像である。Cygwin 環境におけるソフトウェアは、入力画像ファイルをオープンし、各画素の数値をメモリに書き込む。その数値（輝度値）が 150 以上、90 以下の画素をラベリング対象とする[5]。 次に、ラベリングアルゴ

リズムの処理を行い、ラベリングされた画像を BMP ファイルによって出力する。

3. 5 実行結果

Cygwin 環境によって、ソフトウェアの正常な動作を確認した。図 10 に(a)入力画像と(b)出力画像を示す。入力画像はナノスコープに提供していただいた 50×50 画素のビットマップ、RAW ファイルである。出力画像は 50×50 画素のビットマップ、BMP ファイルである。出力画像からはラベリングされているのか判断はできないので、ビットごとに数値化したものを図 11 に示す。



(a)入力画像　　(b)出力画像

図 10：入力画像と出力画像

255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	1	255	255	255	255	255
255	255	255	255	255	1	255	255	255	255	255
255	255	255	255	255	1	255	255	255	255	255
255	255	255	255	255	1	255	255	255	255	255
255	255	255	255	255	1	255	255	255	255	255
255	255	1	255	255	1	255	2	2	255	255
255	255	1	1	255	1	255	2	2	255	255
255	255	1	1	255	1	255	2	2	255	255
255	255	255	1	255	1	255	2	2	255	255
255	255	255	1	255	1	255	255	255	255	255
255	255	255	1	1	1	1	1	255	255	255
255	255	1	1	1	1	1	1	1	255	255
255	1	1	1	1	1	1	1	1	1	255
255	255	1	1	1	1	1	1	1	255	255
255	255	1	1	255	1	1	1	1	255	255
255	255	255	255	255	255	1	255	255	255	255
255	255	255	3	255	1	255	255	4	255	255
255	255	255	3	255	1	1	255	4	255	255
255	255	255	3	255	1	1	255	4	255	255
255	255	255	3	255	1	1	255	4	255	255
255	255	255	3	255	1	1	255	4	255	255
255	255	255	255	255	1	1	255	4	255	255
255	255	255	255	255	1	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255

図 11：出力データ

図 11 は図 10 のラベリング部を注目したデータになっている。ラベル番号はラスタスキャンで発見される順番に昇順になっていることも確認でき、8 近傍においてラベル対象の近接する画素のラベル番号が統一されていることも確認できる。なお、ラベリング対象でない画素の値は 255 になっており、ラベル番号と区別されていることも確認できた。

4.　ハードウェア記述言語によるラベリングモジュールの設計

4. 1 ハードウェア設計の概要

ラベリングをハードウェア記述言語で実装する際のモジュール構成を図 12 に示す[5]。

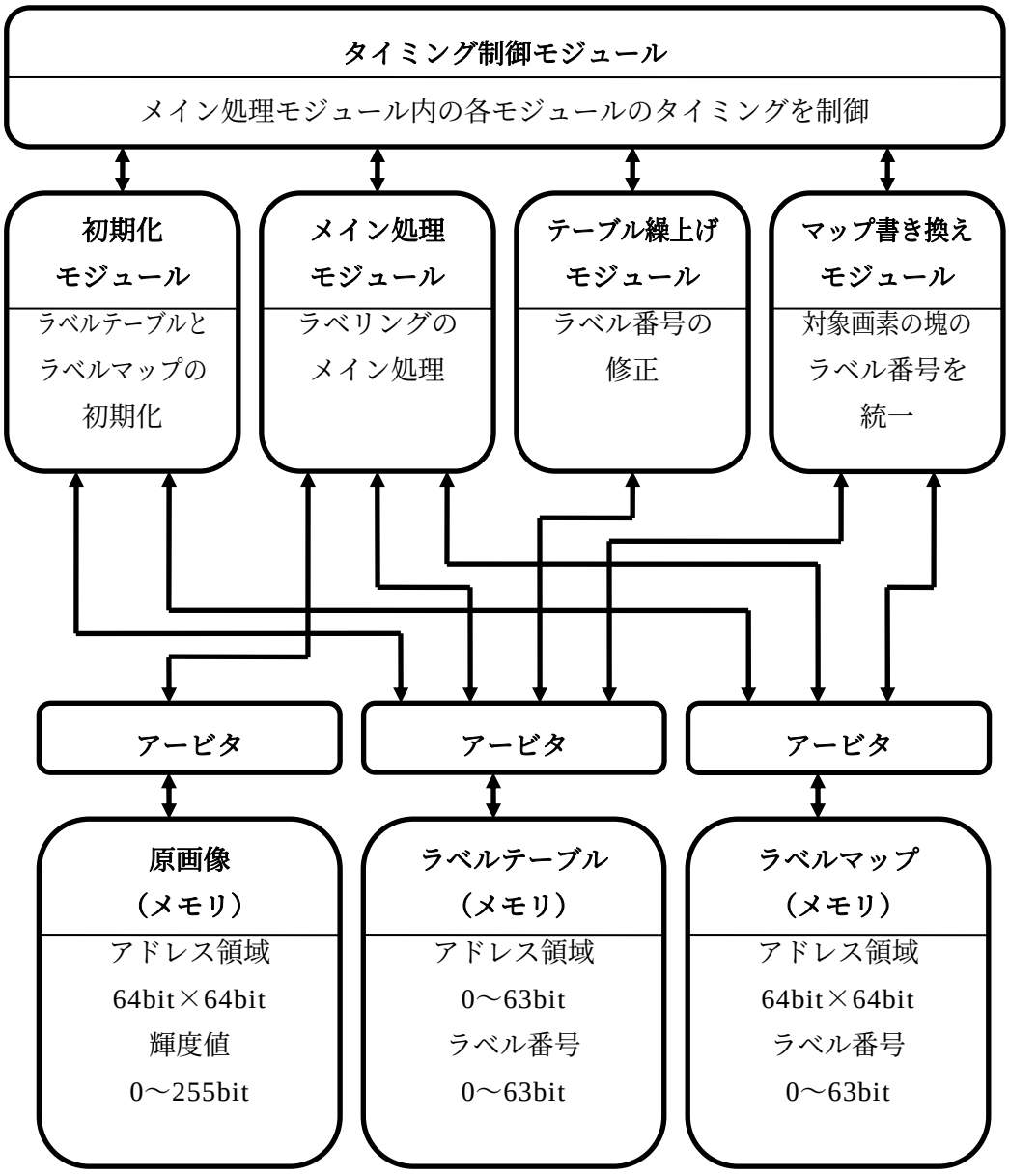


図 12：ラベリングのモジュール構成

タイミング制御モジュールがラベリング処理の主な処理を行う初期化モジュール、メイン処理モジュール、テーブル繰上げモジュール、及びマップ書き換えモジュールを制御している。各モジュールのメモリへのアクセスは各メモリのアービタを通してアクセスできるようになっている。

4. 2 メイン処理モジュール

メイン処理モジュール内のモジュール構成を図 13 に示す[9] [14]。

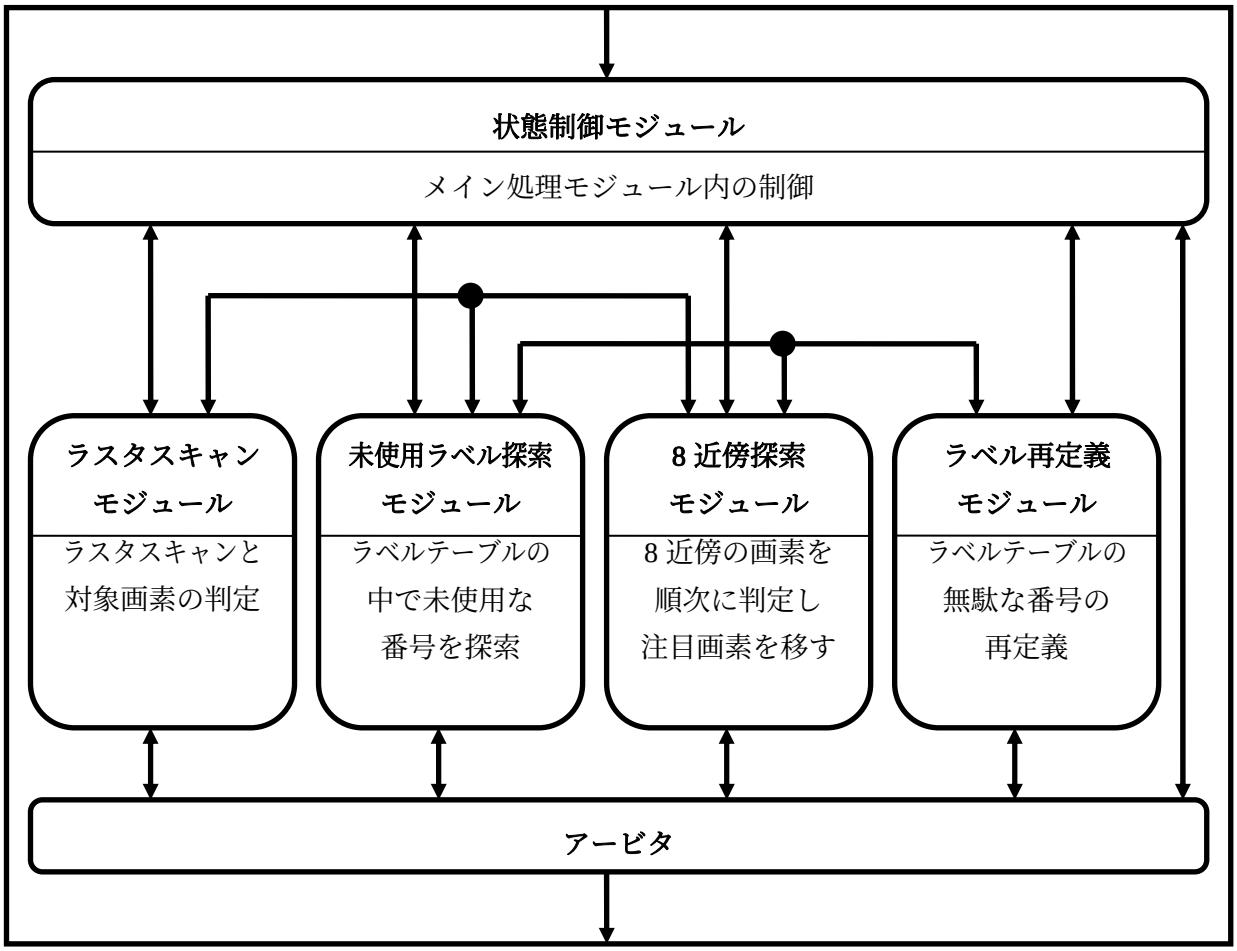


図 13：メイン処理モジュール内のモジュール構成

メイン処理モジュール内は、タイミング制御モジュールから lab_go 信号が状態制御モジュールに送られると開始する。ラスタスキャンモジュール・未使用ラベル探索モジュール・８近傍探索モジュール・ラベル再定義モジュールは状態制御モジュールによって状態遷移される。これらすべてのモジュールは、アービタを通してメイン処理モジュールの出力されている。

（1）状態制御モジュール

状態制御モジュールの入出力インタフェースを図 14 に、信号線の説明を表 4 に示す。

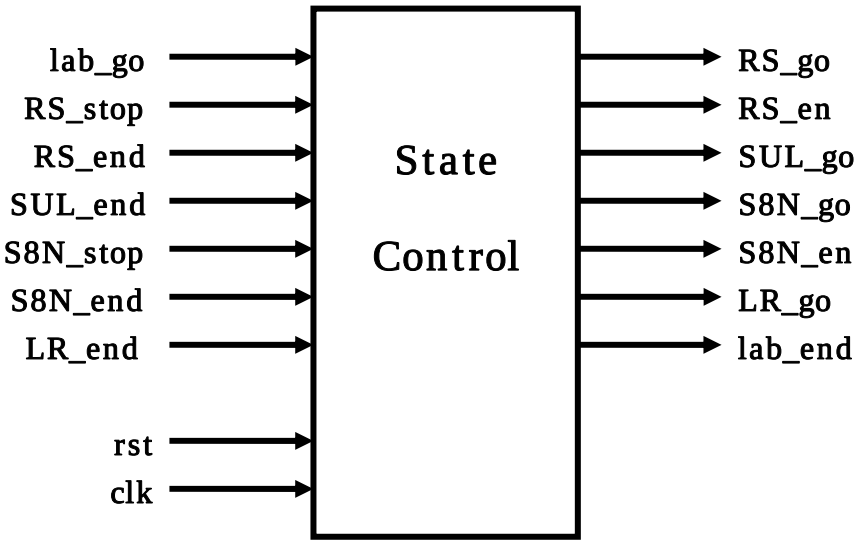


図 14：状態制御モジュールのインタフェース

表 4：状態制御モジュールの入出力信号

信号名	モード	幅	内容
lab_go	input	1bit	メイン処理モジュール開始信号
RS_go	output	1bit	ラスタスキャンモジュール開始信号
RS_stop	input	1bit	ラスタスキャンモジュール停止信号
RS_en	output	1bit	ラスタスキャンモジュール再開信号
RS_end	input	1bit	ラスタスキャンモジュール終了信号
SUL_go	output	1bit	未使用ラベル探索モジュール開始信号
SUL_end	input	1bit	未使用ラベル探索モジュール終了信号
S8N_go	output	1bit	8 近傍探索モジュール開始信号
S8N_stop	input	1bit	8 近傍探索モジュール停止信号
S8N_en	output	1bit	8 近傍探索モジュール再開信号
S8N_end	input	1bit	8 近傍探索モジュール終了信号
LR_go	output	1bit	ラベル再定義モジュール開始信号
LR_end	input	1bit	ラベル再定義モジュール終了信号
lab_end	output	1bit	メイン処理モジュール終了信号
res	input	1bit	リセット信号（Low Active）
clk	input	1bit	クロック信号

状態制御モジュールは、ラスタスキャンモジュール・未使用ラベル探索モジュール・8 近傍探索モジュール・ラベル再定義探索モジュールの 4 つのモジュールの開始を状態遷移によって制御するモジュールとなっている。まず、タイミング制御モジュールから lab_go 信号によって開始し、各モジュールに go 信号を送り、end 信号が返ってくると次のモジュールに go 信号を送る。rw_end 信号が返ってくると、fin 信号によってラベリング処理の終了信号を出力する。

(2) ラスタスキャンモジュール

ラスタスキャンモジュールの入出力インタフェースを図 15 に、信号線の説明を表 5 に示す。

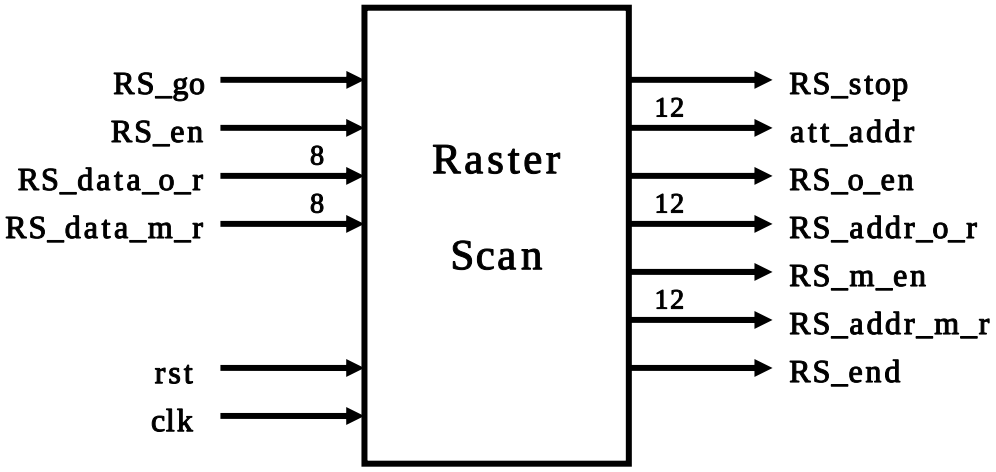


図 15：ラスタスキャンモジュールのインタフェース

表 5：ラスタスキャンモジュールの入出力信号

信号名	モード	幅	内容
RS_go	input	1bit	ラスタスキャンモジュール開始信号
RS_stop	output	1bit	ラスタスキャンモジュール停止信号
RS_en	input	1bit	ラスタスキャンモジュール再開信号
att_addr	output	12bit	注目画素のアドレス値
RS_o_en	output	1bit	ラスタスキャンモジュールでの 原画像のイネーブル信号
RS_addr_o_r	output	12bit	ラスタスキャンモジュールでの 原画像のアドレス読み込み信号
RS_data_o_r	input	8bit	ラスタスキャンモジュールでの 原画像の輝度値読み込み信号
RS_m_en	output	1bit	ラスタスキャンモジュールでの ラベルマップのイネーブル信号
RS_addr_m_r	output	12bit	ラスタスキャンモジュールでの ラベルマップのアドレス読み込み信号
RS_data_m_r	input	6bit	ラスタスキャンモジュールでの ラベルマップのラベル番号読み込み信号
RS_end	output	1bit	ラスタスキャンモジュールの終了信号
res	input	1bit	リセット信号（Low Active）
clk	input	1bit	クロック信号

ラスタスキャンモジュールは、状態制御モジュールから RS_go 信号によって開始し、ラスタスキャンによって各画素がラベル対象か判定し、対象画素を発見するまで行う。対象画素を発見すると一時停止し、RS_stop 信号送り、状態制御モジュールによって未使用ラベル探索モジュールに状態を遷移する。8 近傍探索モジュールによって 8 近傍探索開始画素に戻ってくると、RS_en 信号によって再びラスタスキャンモジュールに状態が遷移され、ラスタスキャンを行う。最後の画素までラスタスキャンすると RS_end 信号によって状態制御モジュールに終了の信号を送る。

（3）未使用ラベル探索モジュール

未使用ラベル探索モジュールの入出力インタフェースを図 16 に、信号線の説明を表 6 に示す。

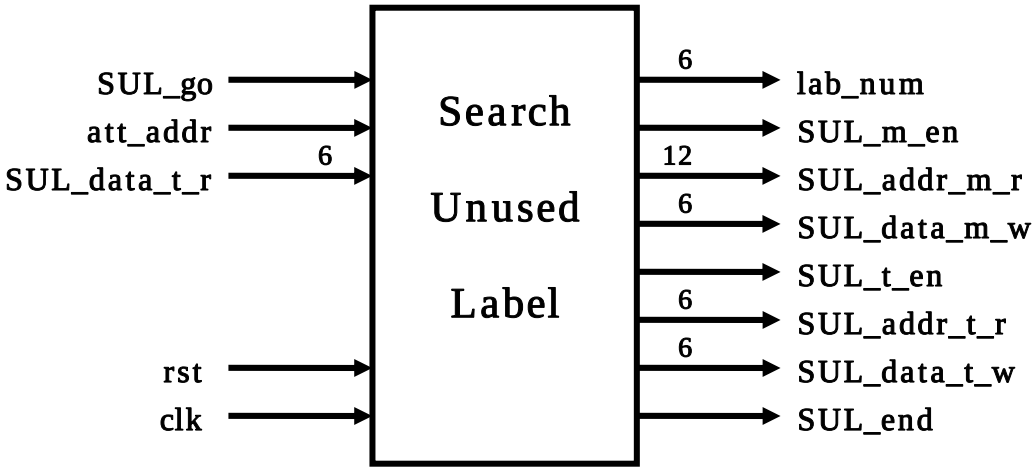


図 16：未使用ラベル探索モジュールのインタフェース

表 6：未使用ラベル探索モジュールの入出力信号

信号名	モード	幅	内容
SUL_go	input	1bit	未使用ラベル探索モジュール開始信号
att_addr	input	12bit	注目画素のアドレス値
lab_num	output	6bit	ラベル番号
SUL_m_en	output	1bit	未使用ラベル探索モジュールでの ラベルマップのイネーブル信号
SUL_addr_m_r	output	12bit	未使用ラベル探索モジュールでの ラベルマップのアドレス読み込み信号
SUL_data_m_w	output	6bit	未使用ラベル探索モジュールでの ラベルマップのラベル番号書き込み信号
SUL_t_en	output	1bit	未使用ラベル探索モジュールでの ラベルテーブルのイネーブル信号
SUL_addr_t_r	output	6bit	未使用ラベル探索モジュールでの ラベルテーブルのアドレス読み込み信号
SUL_data_t_r	input	6bit	未使用ラベル探索モジュールでの ラベルテーブルのラベル番号読み込み信号
SUL_data_t_w	output	6bit	未使用ラベル探索モジュールでの ラベルテーブルのラベル番号書き込み信号
SUL_end	output	1bit	未使用ラベル探索モジュール終了信号
res	input	1bit	リセット信号（Low Active）
clk	input	1bit	クロック信号

未使用ラベル探索は、ラスタスキャンモジュールで対象画素を発見すると状態制御モジ

ュールによって SUL_go 信号が送られる。ここで、ラベルテーブルを参照し、未使用なラベル番号を探索して、そのラベル番号をラベルマップに付ける。次に、SUL_end 信号によって状態制御モジュールに終了の信号を送る。

（4）8 近傍探索モジュール

8 近傍探索モジュールの入出力インタフェースを図 17 に、信号線の説明を表 7 に示す。

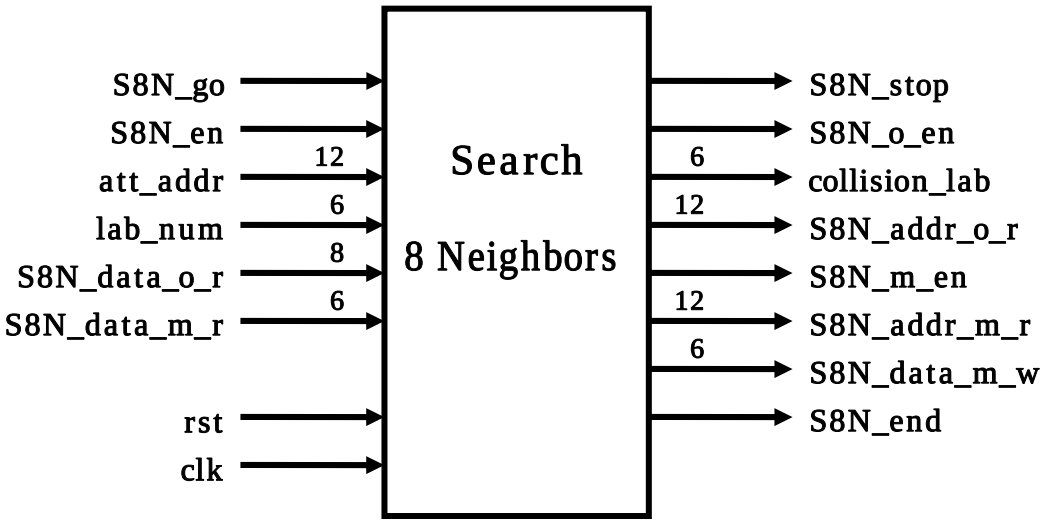


図 17：8 近傍探索モジュールのインタフェース

表 7：8 近傍探索モジュールの入出力信号

信号名	モード	幅	内容
S8N_go	input	1bit	8 近傍探索モジュール開始信号
S8N_stop	output	1bit	8 近傍探索モジュール停止信号
S8N_en	input	1bit	8 近傍探索モジュール再開信号
att_addr	input	12bit	注目画素のアドレス値
lab_num	input	6bit	ラベル番号
collision_lab	output	6bit	衝突時のラベル番号
S8N_o_en	output	1bit	8 近傍探索モジュールでの 原画像のイネーブル信号
S8N_addr_o_r	output	12bit	8 近傍探索モジュールでの 原画像のアドレス書き込み信号
S8N_data_o_r	input	8bit	8 近傍探索モジュールでの 原画像の輝度値読み込み信号
S8N_m_en	output	1bit	8 近傍探索モジュールでの ラベルマップのイネーブル信号
S8N_addr_m_r	output	12bit	8 近傍探索モジュールでの ラベルマップのアドレス読み込み信号
S8N_data_m_r	input	6bit	8 近傍探索モジュールでの ラベルマップのラベル番号読み込み信号
S8N_data_m_w	output	6bit	8 近傍探索モジュールでの ラベルマップのラベル番号書き込み信号
S8N_end	output	1bit	8 近傍探索モジュール終了信号
res	input	1bit	リセット信号（Low Active）
clk	input	1bit	クロック信号

8 近傍探索モジュールは、未使用ラベルモジュールが終了すると状態制御モジュールから S8N_go 信号が送られて開始する。ここで、進行方向を基に 8 近傍の探索を行う。ラベルの衝突でラベル再定義モジュールを参照しなければならない場合、状態制御モジュールに S8N_stop 信号を送り、ラベル再定義モジュールの処理が終了したら状態制御モジュールから S8N_en 信号が送られ、再開する。8 近傍探索を開始した画素に戻ってくると S8N_end 信号を状態制御モジュールに終了の信号を送る。

（5）ラベル再定義モジュール

ラベル再定義モジュールの入出力インタフェースを図 18 に、信号線の説明を表 8 に示す。

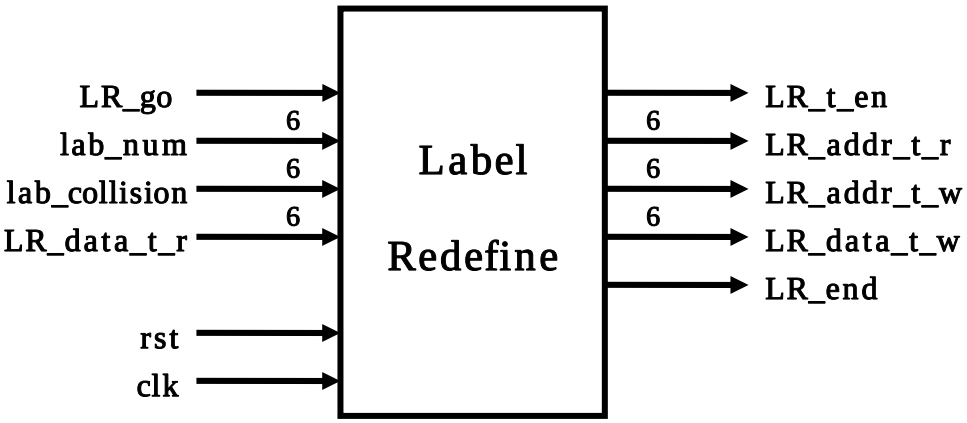


図 18：ラベル再定義モジュールのインタフェース

表 8：ラベル再定義モジュールの入出力信号

信号名	モード	幅	内容
LR_go	input	1bit	ラベル再定義モジュール開始信号
lab_num	input	6bit	ラベル番号
collision_lab	input	6bit	衝突時のラベル番号
LR_t_en	output	1bit	ラベル再定義モジュールでの ラベルテーブルのイネーブル信号
LR_addr_t_r	output	6bit	ラベル再定義モジュールでの ラベルテーブルのアドレス読み込み信号
LR_addr_t_w	output	6bit	ラベル再定義モジュールでの ラベルテーブルへのアドレス書き込み信号
LR_data_t_r	input	6bit	ラベル再定義モジュールでの ラベルテーブルのラベル番号読み込み信号
LR_data_t_w	output	6bit	ラベル再定義モジュールでの ラベルテーブルへのラベル番号書き込み信号
LR_end	output	1bit	ラベル再定義モジュール終了信号
res	input	1bit	リセット信号（Low Active）
clk	input	1bit	クロック信号

ラベル再定義モジュールは、8 近傍探索モジュールでラベルの衝突が起こった場合に、状態制御モジュールから LR_go 信号が送られて開始する。ここで、ラベルマップで衝突したラベル番号を基に、ラベルテーブルの再定義を行う。再定義を終えると LR_end 信号を状態制御モジュールに終了の信号を送る。

4. 3 その他のモジュール

（1）タイミング制御モジュール

タイミング制御モジュールの入出力インタフェースを図 19 に、信号線の説明を表 9 に示す[5][9]。

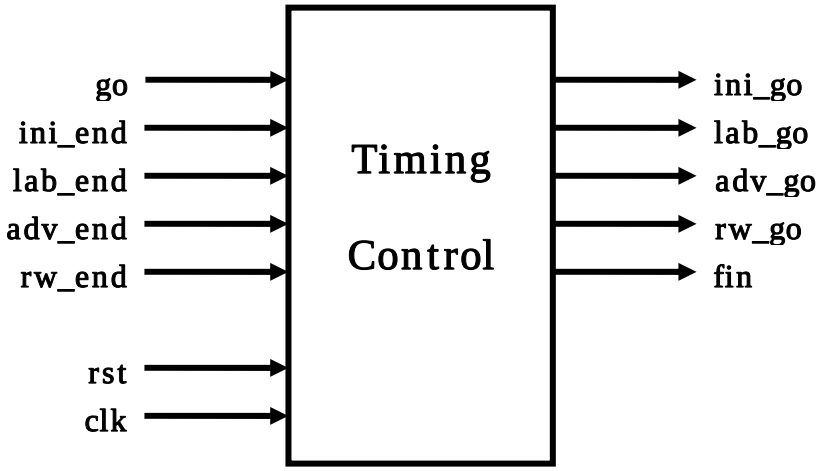


図 19：タイミング制御モジュールのインタフェース

表 9：タイミング制御モジュールの入出力信号

信号名	モード	幅	内容
go	input	1bit	タイミング制御モジュール開始信号
ini_go	output	1bit	初期化モジュール開始信号
ini_end	input	1bit	初期化モジュール終了信号
lab_go	output	1bit	メイン処理モジュール開始信号
lab_end	input	1bit	メイン処理モジュール終了信号
adv_go	output	1bit	テーブル繰上げモジュール開始信号
adv_end	input	1bit	テーブル繰上げモジュール終了信号
rw_go	output	1bit	マップ書き換えモジュール開始信号
rw_end	input	1bit	マップ書き換えモジュール終了信号
fin	output	1bit	タイミング制御モジュール終了信号
res	input	1bit	リセット信号（Low Active）
clk	input	1bit	クロック信号

タイミング制御モジュールは、初期化モジュール、メイン処理モジュール、テーブル繰上げモジュール、マップ書き換えモジュールの 4 つのモジュールの開始を制御するモジュールとなっている。まず、go 信号によってタイミング制御モジュールが開始し、各モジュールに逐次的に go 信号を送り、end 信号が返ってくると次のモジュールに go 信号を送る。

rw_end 信号が返ってくると、fin 信号によってラベリング処理の終了信号を出力する。

(2) 初期化モジュール

初期化モジュールの入出力インタフェースを図 20 に、信号線の説明を表 10 に示す。

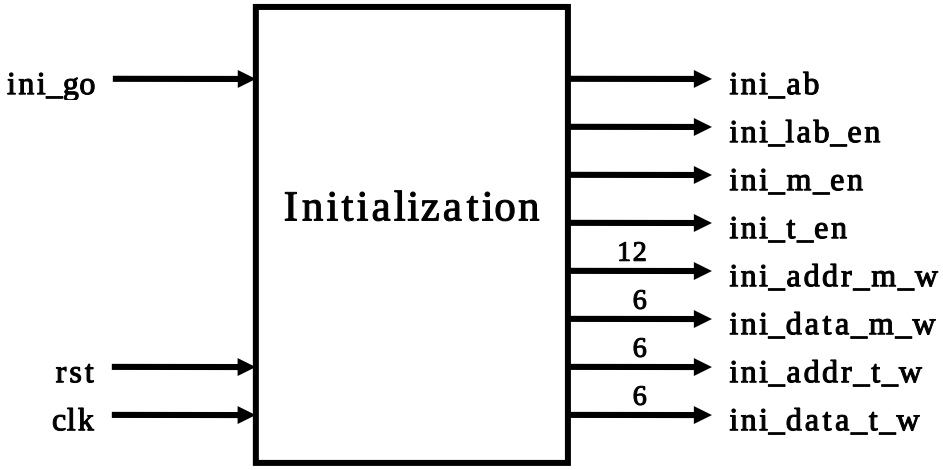


図 20：初期化モジュールのインタフェース

表 10：初期化モジュールの入出力信号

信号	モード	幅	内容
ini_go	input	1bit	初期化モジュール開始信号
ini_ab	output	1bit	初期化モジュールアービタ信号
ini_m_en	output	1bit	初期化モジュールでの ラベルマップのイネーブル信号
ini_addr_m_w	output	12bit	初期化モジュールでの ラベルマップへのアドレス書き込み信号
ini_data_m_w	output	6bit	初期化モジュールでの ラベルマップへのラベル番号書き込み信号
ini_t_en	output	1bit	初期化モジュールでの ラベルテーブルのイネーブル信号
ini_addr_t_w	output	6bit	初期化モジュールでの ラベルテーブルへのアドレス書き込み信号
ini_data_t_w	output	6bit	初期化モジュールでの ラベルテーブルへのラベル番号書き込み信号
ini_end	output	1bit	初期化モジュール終了信号
res	input	1bit	リセット信号（Low Active）
clk	input	1bit	クロック信号

初期化モジュールは、タイミング制御モジュールからの ini_go 信号によって開始し、ラベルマップメモリとラベルテーブルメモリの初期化を行う。初期化が終了すると ini_end 信号によってタイミング制御モジュールに終了の信号を送る。

(3) テーブル繰上げモジュール

テーブル繰上げモジュールの入出力インタフェースを図 21 に、信号線の説明を表 11 に示す。

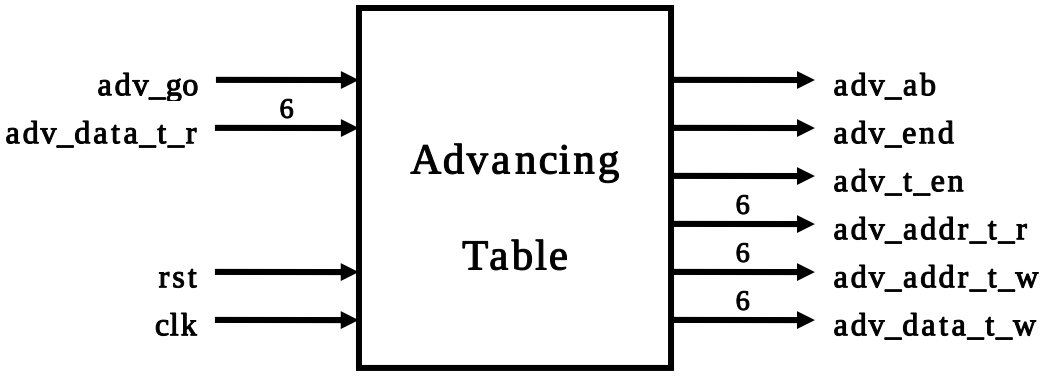


図 21：テーブル繰上げモジュールのインタフェース

表 11：テーブル繰上げモジュールの入出力信号

信号	モード	幅	内容
adv_go	input	1bit	テーブル繰上げモジュール開始信号
adv_ab	output	1bit	テーブル繰上げモジュールアービタ信号
adv_t_en	output	1bit	テーブル繰上げモジュールでの ラベルテーブルのイネーブル信号
adv_addr_t_r	output	12bit	テーブル繰上げモジュールでの ラベルテーブルのアドレス読み込み信号
adv_addr_t_w	output	6bit	テーブル繰上げモジュールでの ラベルテーブルへのアドレス書き込み信号
adv_data_t_r	input	1bit	テーブル繰上げモジュールでの ラベルテーブルのラベル番号読み込み信号
adv_data_t_w	output	6bit	テーブル繰上げモジュールでの ラベルテーブルへのラベル番号書き込み信号
adv_end	output	1bit	テーブル繰上げモジュール終了信号
res	input	1bit	リセット信号（Low Active）
clk	input	1bit	クロック信号

テーブル繰上げモジュールは、メイン処理モジュールでラスタスキャンを行った後のラベルテーブルの参照番号の昇順への修正を行うモジュールとなっている。これも同様に、タイミング制御モジュールからの adv_go 信号によって開始、終了すると adv_end によって終了の信号を送る。

（4）マップ書き換えモジュール

マップ書き換えモジュールの入出力インタフェースを図 22 に、信号線の説明を表 12 に示す。

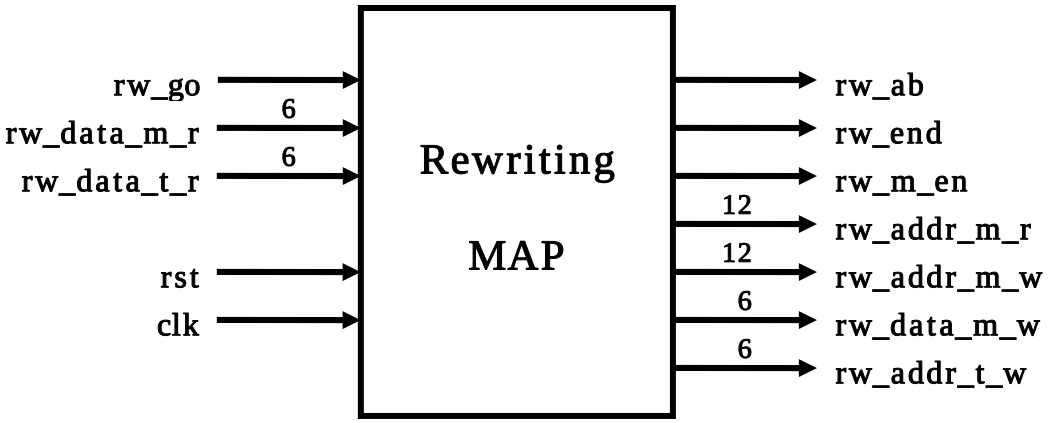


図 22：マップ書き換えモジュールのインタフェース

表 12：マップ書き換えモジュールの入出力信号

信号	モード	幅	内容
rw_go	input	1bit	マップ書き換えモジュール開始信号
rw_ab	output	1bit	マップ書き換えモジュールアービタ信号
rw_m_en	output	1bit	マップ書き換えモジュールでの ラベルマップのイネーブル信号
rw_addr_m_r	output	12bit	マップ書き換えモジュールでの ラベルマップのアドレス読み込み信号
rw_addr_m_w	output	12bit	マップ書き換えモジュールでの ラベルマップへのアドレス書き込み信号
rw_data_m_r	input	6bit	マップ書き換えモジュールでの ラベルマップのラベル番号読み込み信号
rw_data_m_w	output	6bit	マップ書き換えモジュールでの ラベルマップへのアドレス書き込み信号
rw_addr_t_w	output	6bit	マップ書き換えモジュールでの ラベルテーブルへのアドレス書き込み信号
rw_data_t_r	input	6bit	マップ書き換えモジュールでの ラベルテーブルのラベル番号読み込み信号
rw_end	output	1bit	マップ書き換えモジュール終了信号
res	input	1bit	リセット信号（Low Active）
clk	input	1bit	クロック信号

マップ書き換えモジュールは、テーブル繰上げモジュールによって修正されたラベルテーブルとメイン処理モジュールでラベルマップに書き込まれたラベル番号によって、ラベルマップをラスタスキャンし、ラベル番号の書き換えを行う。これも同様に、タイミング制御モジュールからの rw_go 信号によって開始し、rw_end 信号によって終了信号を送る。

4. 4 シミュレーションによる検証

（1）シミュレーション結果とソフトウェアの比較

4.2、4.3 節で紹介したモジュールを VHDL によって設計し、MenterGraphics 社の Modelsim SE によってシミュレーションを行った。その結果を、ソフトウェアと比較した。ランダムに選出された気泡と気流の画像を 10 枚用意し、ラベリング処理を行った。ただし、ランダムに選出した画像は松崎氏と同じ画像を使っている。ソフトウェアの実行環境は、スペックが Intel Pentium 4 CPU 2.53GHz、RAM 1.00GB で、OS は WindowsXP、Cygwin 上で C 言語によって実行し、動作時間を計った。

ソフトウェアの実行時間は、1000 回連続で実行した平均の値を出している[5]。ハードウ

ェアの実行速度は、Modelsim SE によってシミュレーションを行った際の総クロック数から、動作クロック 66MHz を割ることで実行時間を求めている。性能向上比についてはハードウェア実行時間に対するソフトウェア実行時間の比である。

表 13 に気泡、表 14 に気流のソフトウェアとハードウェアの実行時間の結果と、それに対する性能向上比、ラベル総数、及び対象画素数を示す。なお、性能向上比・ラベル総数・対象画素数については最高値から 2 位までと最低値から 2 位までを太字で示している。

表 13：気泡の実験結果

画像種別	ソフトウェア 実行時間(msec)	ハードウェア 実行時間(msec)	性能 向上比	ラベル 総数	対象 画素数
気泡 1	1.53	0.23	6.65	9	172
気泡 2	1.51	0.23	6.57	8	159
気泡 3	1.51	0.17	**8.82**	**1**	**16**
気泡 4	1.50	0.25	6.00	4	116
気泡 5	1.51	0.20	7.55	10	54
気泡 6	1.51	0.22	6.86	3	49
気泡 7	1.53	0.18	**8.50**	**2**	**5**
気泡 8	1.53	0.23	6.65	5	105
気泡 9	1.53	0.20	7.65	4	22
気泡 10	1.56	0.22	7.09	5	52

表 14：気流の実験結果

画像種別	ソフトウェア 実行時間(msec)	ハードウェア 実行時間(msec)	性能 向上比	ラベル 総数	対象 画素数
気流 1	1.54	0.23	**5.13**	**45**	293
気流 2	1.55	0.28	**5.54**	9	383
気流 3	1.56	0.27	5.77	22	**562**
気流 4	1.53	0.26	5.88	15	**488**
気流 5	1.53	0.26	5.88	15	470
気流 6	1.53	0.23	6.65	**27**	260
気流 7	1.53	0.23	6.65	**27**	245
気流 8	1.51	0.20	7.55	13	52
気流 9	1.50	0.20	7.50	**2**	20
気流 10	1.48	0.21	7.05	5	30

（2）考察

シミュレーションによって、性能向上比の最大が 8.82 倍、最低は 5.13 倍、平均で 6.80 倍であることがわかった。松崎氏の性能向上比と比べると、ハードウェアで設計したにも関わらず性能が伸びなかった[5]。ちなみに、松崎氏の性能向上比は最大 233 倍・最低 100 倍・平均 139 倍の性能を得ている。

性能向上比の伸び率が低い原因としては、もともとのソフトウェアでの実行時間が速かった点が挙げられる。次に、メイン処理モジュールの高速化に勤めたため、初期化モジュールでの高速化を実現できていない。従って、初期化モジュールでのタイムロスが挙げられる。この問題については、パイプラインを設けることによって解消されるであろう。

次に、性能向上比の特徴を見ると、ラベル総数・対象画素数が共に小さいと高くなる傾向がある。低くなる傾向に関しては、ラベル総数が極端に高いこと、対象画素数が極端に高いこと、共に高いことが挙げられる。原因としては、ラベル総数が増えることによってラベルテーブルへの参照回数が増えることと、対象画素数が増えることによって対象判定の回数が増えることが考えられる。

5. おわりに

本論文では、ラベリングアルゴリズムの設計とシミュレーションによる検証を行った。アルゴリズムを主に 9 個のモジュールに分割し、ハードウェア記述言語 VHDL によって設計し、MenterGraphics 社の Modelsim SE によってシミュレーションを行った。

今後の課題としては、今回の分割したモジュールは逐次的な処理なので、パイプライン化処理することによって性能をあげることが挙げられる。他にも、ラスタスキャンでラベル対象の画素を発見すると、その対象の画像へのラベル付けを一度に実現することができれば、ラスタスキャンを 2 度せずに済み、ラベルテーブルへの参照も少なくなることから、性能の向上は期待できることなどが挙げられる。

謝辞

本研究の機会を与えてくださり、貴重な助言、ご指導を頂きました山崎勝弘教授、小柳滋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、中谷嵩之氏、松崎裕樹氏、及び高性能計算研究室の皆様に深く感謝したします。

同じく、産学共同研究において、貴重な助言、ご指導を頂きました株式会社ナノスコープの三宅淳司氏、平岡邦廣氏、西田洋隆氏に深く感謝いたします。

参考文献

[1] 田村秀行：コンピュータ画像処理，オーム社，2004.

[2] 仲野巧：VHDL によるマイクロプロセッサ設計入門，CQ 出版社，2002.

[3] Tech-On!：http://techon.nikkeibp.co.jp/.

[4] DisplaySearch 社：http://www.displaysearch-japan.com/.

[5] 松崎裕樹：マハラノビス距離を用いた画像判別とラベリングの高速化の実現，立命館大学理工学部情報学科卒業論文，2006.

[6] 千村亮介：ラベリングとマハラノビス距離による画像判別の実現，立命館大学理工学部情報学科卒業論文，2006.

[7] 川本隆志：画像処理ボード上での高速テンプレートマッチングの実装と検証，立命館大学理工学部情報学科卒業論文，2005.

[8] 的場督永：ハード/ソフト最適分割を考慮した JPEG エンコーダの協調設計，立命館大学理工学部情報学科卒業論文，2005

[9] 古川達久：マルチサイクル・パイプライン方式による教育用マイクロプロセッサの設計と検証，立命館大学理工学部情報学科卒業論文，2004.