卒業論文

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

氏名：　川本隆志
学籍番号：　2210010062-6
指導教員：　山崎　勝弘　教授
提出日：　2005 年 2 月 21 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の産学協同プロジェクトの一環として、LSI 設計の主流となっているハードウェア記述言語 VHDL を用いて、FPGA 上にテンプレートマッチングを実装した。

今回設計したテンプレートマッチングは 2 種類あり、汎用性があるが低速なロジックを用いたテンプレートマッチング（本研究では「簡易テンプレートマッチング」と呼ぶ）と、今回マッチングの対象となるガラスに現れる決まったパターン（テンプレート）に特化した、高速なロジックを用いたテンプレートマッチング（本研究では「高速テンプレートマッチング」と呼ぶ）である。

FPGA 上に実装したのは高速テンプレートマッチングの方であり、簡易テンプレートマッチングについては FPGA 上には実装せず、ビヘイビアレベルでのシミュレーションによる検証に留めた。

一方、高速テンプレートマッチングについては、今回使用するターゲットボードである SBS Technology 社の画像処理ボード TSUNAMI に搭載されている FPGA, Stratix EP1S25 上に実装し、正常な動作を確認した。使用する画像はテスト用の 2 値化済み画像で、あらかじめ TSUNAMI ボード上の SRAM に画像データを格納し、テンプレートマッチングの実行時に SRAM から読み出してくる形式を取る。また、簡易版 C プログラム、高速版 C プログラム、ROM 形式のメモリを回路に含めた VHDL シミュレーション、および実際に FPGA 上で実行した結果の実行クロック数の比較により、性能評価を行った。

なお、今回テンプレートマッチングを設計する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 である。また、生成したコンフィギュレーションデータ（回路情報）の FPGA 上へのダウンロード、及びテンプレートマッチングの開始と結果の確認は TSUNAMI 専用の API を用いて行った。

目次

図目次

表目次

1. はじめに

ハードウェア設計の歴史において、初めて集積回路が一般の人々に普及したのは 1970 年代であるが、この頃、集積回路を用いた製品は電卓、時計やゲーム機といった個人用の電子機器であった。1980 年代に入ると DRAM が普及し、これ以降の半導体の設計規模が線形に増大している。1995 年では設計規模が 10 万ゲートから 50 万ゲートの規模だったが、現在の設計規模はその 10 倍である 100 万ゲートから 500 万ゲートを超える規模にまで達している。また、これまでは単体の機能だけを考慮して設計していたものが、現在ではシステム自身を 1Chip 化するといった設計（SoC:System on Chip）に変わってきているため、設計規模が非常に大きくなっている[1]。

このような半導体の設計規模の拡大に伴って、LSI 設計におけるシミュレーション技術も発展してきた。1970 年代の、LSI 製造のためのレイアウト・マスク設計に関する自動化に始まり、1980 年代には論理ゲートに基づいた回路図入力と論理ゲートレベルのシミュレーションが自動化され、コンピュータ上でのシミュレーションが確立された。しかし、数万ゲート以上にも及ぶ回路設計規模の拡大に伴い、シミュレーションの実行に長時間かかるため、これまでの LSI 設計手法である、仕様に基づいて論理ゲートや記憶素子を接続していき、設計ミスの修正、再確認を繰り返す設計（ボトムアップ設計）手法では、短期間で仕様通りに正確に設計することができなくなり、設計環境や設計フローを改善する必要があった[4]。

1990 年代になると、論理合成ツール及び EDA ツールが開発され実用化されるに至って、上述のような論理ゲートレベルで回路を設計するボトムアップ設計から、HDL（HardWare Description Language）による、機能レベルで回路を記述する上位レベルの設計（トップダウン設計）が可能になった。これにより、機能に関するバグを設計の早期の段階で発見することが可能になり、開発コストと開発期間の大幅な短縮が達成された。また、FPGA（Field Programable Gate Array）を用いることにより、設計した回路を何度でも書きかえることが可能となり、開発段階におけるエミュレーションや、プロトタイプの作成に利用され、一層の開発コストと開発期間の短縮につながった[5]。

ハードウェア記述言語（HDL:Hardware Description Language）は回路設計を一般的なプログラミングの形式で記述するアプローチのための言語で、現在のハードウェア設計において主流となっている。HDL の設計記述レベルには、動作レベル記述、レジスタ遷移レベル（RTL）記述、ゲートレベル記述の 3 つがある[7]。

HDL には主に VHDL、VerilogHDL、SFL（Structure Functional Language）などがあるが、本研究では VHDL を用いて設計した。VHDL の特徴を以下に示す[2][3]。

・VHDL

―VHSIC(Very High Speed Integrated Circuit)ハードウェア記述言語として米国国防総省により開発された

―システムからゲート、スイッチレベルまでハードウェアを階層記述可能

―設計実体はエンティティ宣言とそれに関連したアーキテクチャ本体で定義される
―設計実体を一般的に記述したものをカスタマイズするために、排列指定（Configuration）が用いられる

FPGA（Field Programmable Gate Array）はプログラム書き換え可能な LSI である。論理ゲートをアレイ状に敷き詰めた集積回路で、回路データをダウンロードすることで回路構成を変更することが可能である。FPGA 上に構成された回路は電源を断つか、新たに回路構成データをダウンロードすることで、何度も書き換えることが可能である。従来、設計した回路は実際に LSI に焼き付けるか、シミュレーションによって動作検証を行っていた。しかし、LSI に焼き付けるには時間とコストがかかり、大規模な回路のシミュレーションには膨大な時間が必要であった。FPGA を用いることで設計した LSI をその場で実現し、実機での動作検証を行うことが出来、時間とコストの削減が可能となる[8]。

なお、今回使用する型番 TS-PCI-2501 の TSUNAMI ボード上に搭載されている FPGA は Altera 社の Stratix EP1S25 である。

以上のような背景を踏まえ、本研究では、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 によるテンプレートマッチングの設計を行う。

この共同プロジェクトでは、近年ますます需要が高まっている携帯電話や PC に用いられるフラットパネル・ディスプレイのためのガラス外観検査装置を開発することを目標としている。これまでラインセンサーカメラから取りこんだ画像に対して、それぞれのカメラに 1 台の PC を対応させ、ソフトウェアによって様々な画像処理をさせていた。しかし、複雑な処理をさせるには能力に限界があり、PC の数が多いためシステムの信頼性に問題が出る。そこで、1 台の PC に複数の画像処理ボードを接続し、それぞれの画像処理ボード上の FPGA で画像処理の一部を行うことで、開発コスト及び処理性能を改善する。

プロジェクトの研究テーマとしては、検査装置の完成を目指して、まず様々な画像処理の項目について、画像処理ボード TSUNAMI を用いて上位アプリケーションから呼び出し可能なライブラリー形式の FPGA 画像処理システムを作成することとなり、私はテンプレートマッチングを FPGA 上に実装することにした。

最終的には、ガラスに現れる決まったパターンのテンプレートに特化した高速なアルゴリズムのテンプレートマッチングを実装することが目標である。しかし、設計するにあたっては、まず一般に定義されている、単純なアルゴリズムを用いたテンプレートマッチング（本研究では簡易テンプレートマッチングと呼ぶ）を先に設計し、それに部分的に改良を加えながら高速なテンプレートマッチングを設計するというアプローチを取った。ただ、前者は愚直な処理手順のためハードウェア化するメリットは無いので、TSUNAMI 上の FPGA には実装せず、シミュレーションによる検証に留めた。なお、本研究では、カメラから取りこんだ画像ではなく、あらかじめ用意したテスト用の画像を用いる。用いる画像は、サイズを縮小するために 2 値化を施し、1 画素に 1 ビットを対応させた。テンプレートマッチングを設計するにあたって、Altera 社の統合開発環境ツール QuartusII を使用した。バージョンは 4.1 である。シミュレーションツールとしては ModelSim を用いた。コンフィギュレーションデ

ータ（回路構成情報）の、TSUNAMI 上の FPGA へのダウンロードは、TSUNAMI 専用の API を用いて行った。また、本研究は、ハードウェア設計の基礎の習得、統合開発環境ツールの使用法の習得、高速画像処理の手法の学習、FPGA ボード上への実装の経験を目的とする。

第 2 章で設計した 2 種類のテンプレートマッチングのアルゴリズムを述べる。第 3 章で簡易テンプレートマッチングの設計とシミュレーションによる検証、第 4 章で高速テンプレートマッチングの設計、第 5 章で高速テンプレートマッチングの FPGA 上での検証と性能評価について述べる。

2. ガラス外観検査装置の開発のためのテンプレートマッチング

2.1. ガラス外観検査装置開発プロジェクト

本研究は、株式会社ナノスコープとの産学協同プロジェクト「ガラス外観検査装置の開発」の研究テーマの一環として、テンプレートマッチングを FPGA 上に実装する。このプロジェクトは、ラインセンサーカメラから取りこんだガラスの画像に対して、従来ソフトウェアで画像処理を行っていたものを、ハードウェア（FPGA）で処理させることにより、処理速度とコストの両方を高めることを目的としている。そこで、研究テーマとしてエッジ検出、ラベリング処理などの各種画像処理を FPGA 化し、上位アプリケーションから呼び出し可能なライブラリー形式の画像処理システムを今回用いる画像処理ボード TSUNAMI で作成することになり、私はテンプレートマッチングの FPGA 上への実装を担当した。しかし、本研究に関しては、カメラからの画像を取り込むのではなく、事前に用意した画像を入力とする。図 1 にプロジェクトにおける本研究の位置付けを示す。

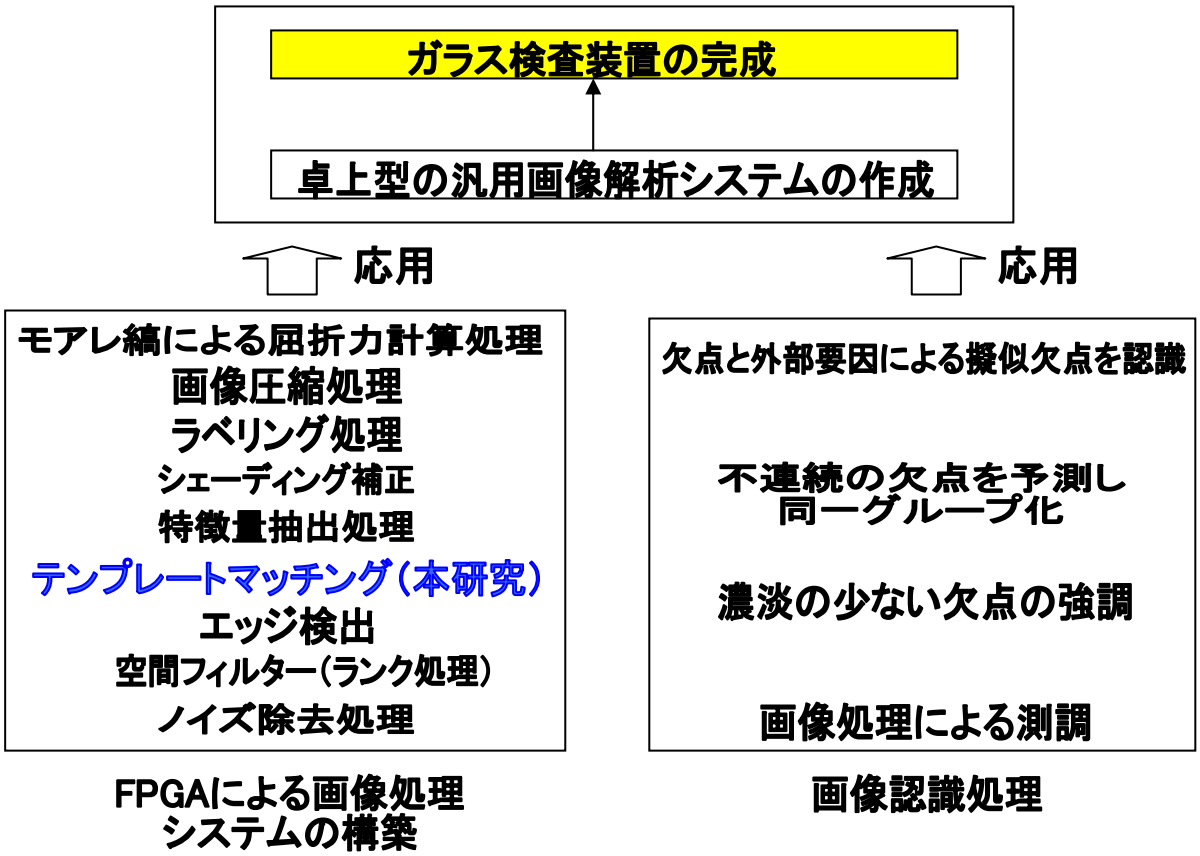

図 1：ガラス外観検査装置開発プロジェクトにおける本研究の位置付け

2.2. テンプレートマッチング

テンプレートマッチングとは、画像認識の手法の一つで、入力画像（対象画像）とテンプレート画像とを重ね合わせることにより比較照合し、両者が一致しているかどうかを判定する処理のことである[9]。

今回用いるテンプレートは、検査対象となるガラス上に現れる、図 2 のような十字型のパターンである。なお、今回使用する画像はテンプレート画像・対象画像ともに 2 値化されたものを扱う。図 2 の右のように、2 値化されたテンプレート画像は 1 画素に 1 ビットが対応しており、十字の部分の画素値が 1 で、それ以外が 0 となるようにした。

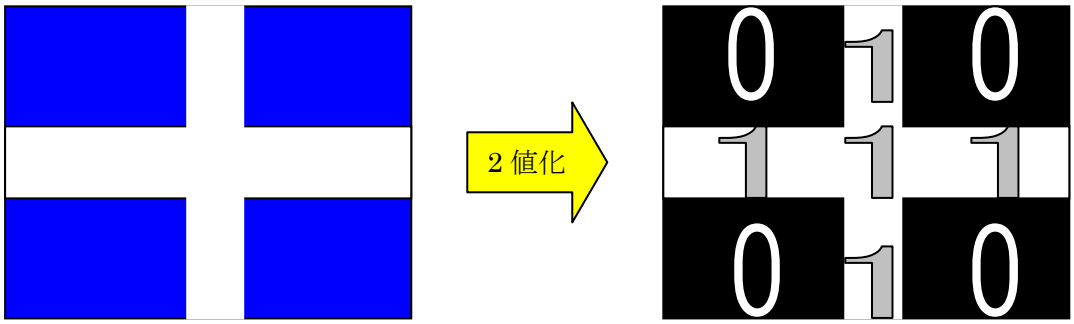

図 2：テンプレート画像（左：濃淡画像、右：2 値化画像）

2.3. 簡易テンプレートマッチング

上で述べたテンプレートマッチングの定義に従って、本研究ではまず始めに、処理速度を考えず、なおかつ、特定のパターンに依存せずどのようなパターンに対してもマッチング位置の検出が可能な簡易な手法として、対象画像とテンプレート画像の全ての画素について差を求めて相違度を計算する形式のテンプレートマッチングを設計した。本研究ではこれを簡易テンプレートマッチングと呼ぶ。

図 3 に、簡易テンプレートマッチングの全体像を示す。図 3 中の「切り出し部分」はテンプレートと同じサイズである。なお、簡易テンプレートマッチングの設計にあたって、対象画像のサイズを 1024x1024 画素、テンプレートのサイズを 8x8 画素とした。それぞれの画像を格納しているメモリを 8 ビット幅とし、1 回のメモリアクセスで、切り出し部分（以下、ブロックと呼ぶ）およびテンプレートの 1 つのラインを読み出せる形式を取っている。

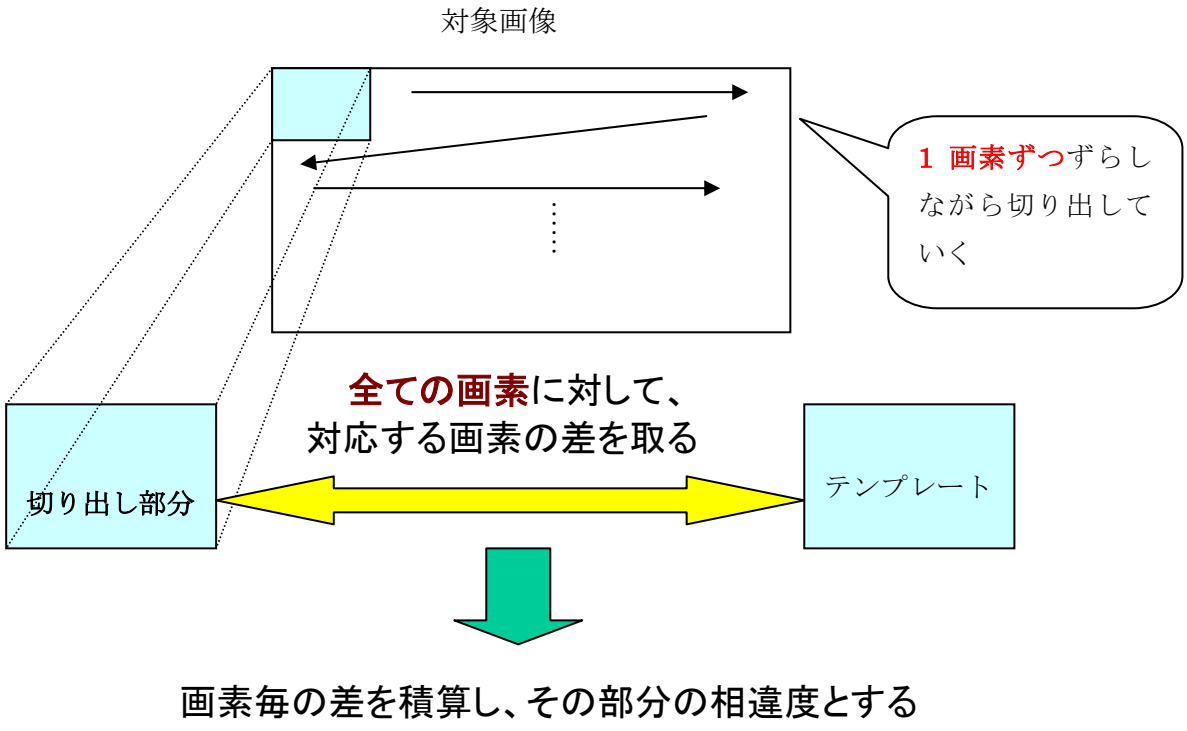

図 3：簡易テンプレートマッチングの全体像

以下、図 4 に示すように、最も左上の座標が（X, Y）のブロックを「ブロック（X, Y)」と、また、ブロックおよびテンプレートの上から I+1 番目のラインを「ライン I」と呼ぶことにする。

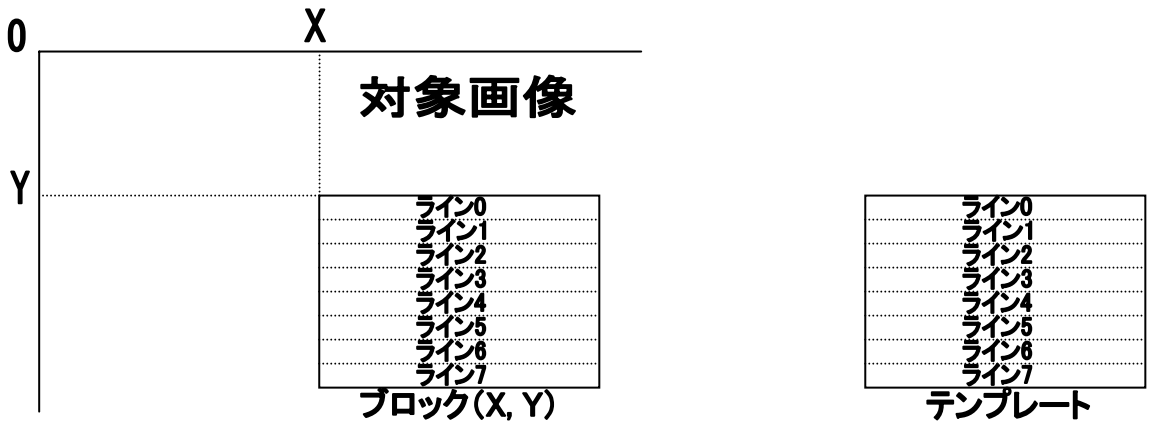

図 4：ブロック（X, Y）とライン I の定義

図 5 に簡易テンプレートマッチングのアルゴリズムを示す。

開始点は（0, 0）で、最小相違度の初期値として取りうる最大の値 8x8=64 を与える。これより小さい相違度を見つけるたびにそれを置き換えていき、テンプレートマッチングの終了

時には最も小さい相違度が保持される。

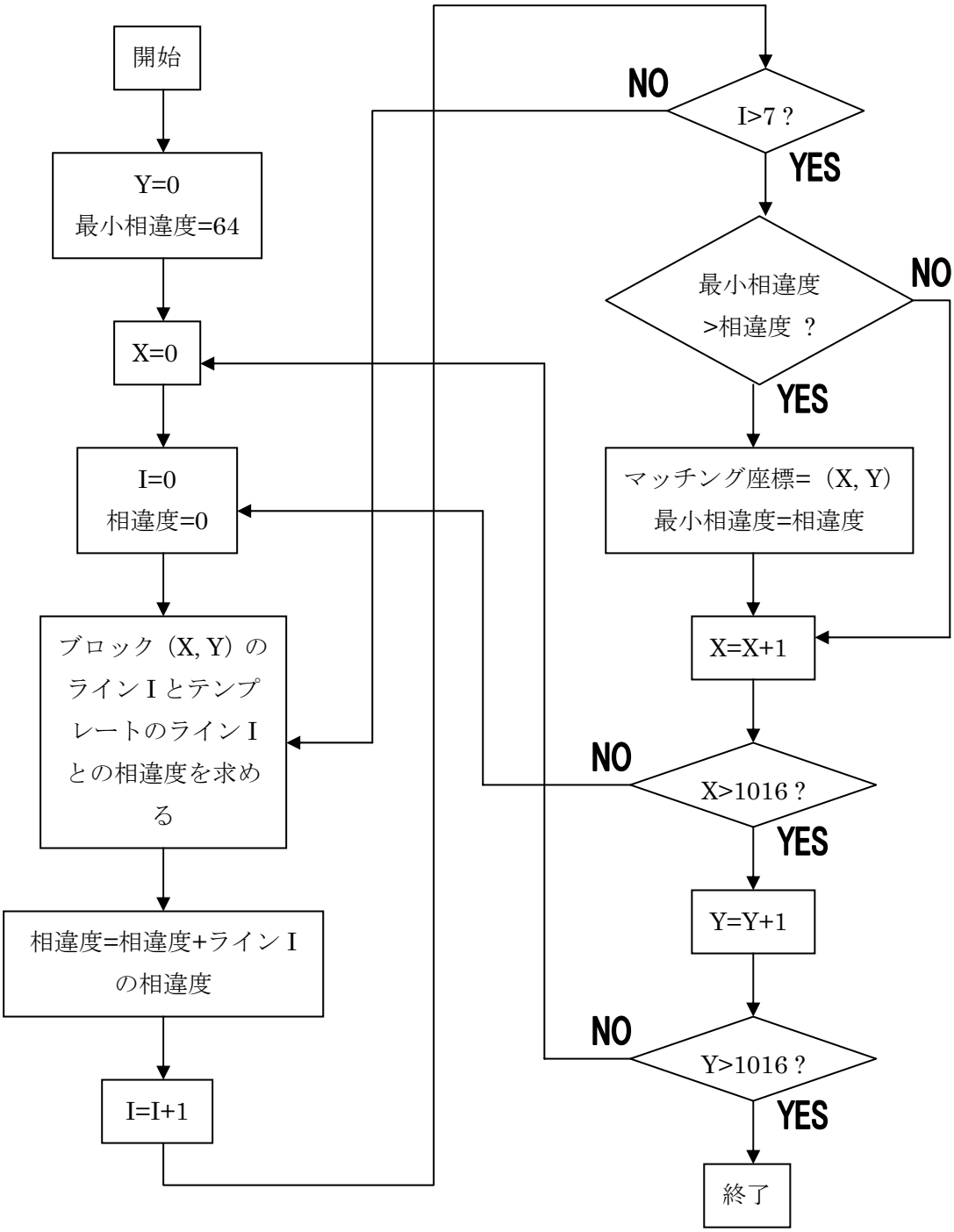

図５：簡易テンプレートマッチングのアルゴリズム

新しいブロックを切り出すたびに相違度を 0 にクリアする。各ライン毎に相違度を積算していき、ブロック全体の相違度が求め終わると、そのブロックの相違度と、現在保持している最小相違度とを比較する。該当ブロックの相違度が最小相違度より小さければ、最小相違度をその相違度で置き換え、マッチング座標を更新する。

マッチング処理を終えるとXの値をインクリメントし､次のブロックを1画素右隣とする。Xが1016のとき、そのブロックは対象画像の右端に位置するので、1016を超えると次のブロックを1画素下の左端とするためにYをインクリメントし、Xを0とする。ここで、同様にYが1016を超えると対象画像の右下まで走査し終わったので、テンプレートマッチングを終了する。

2.4. 高速テンプレートマッチング

（1）簡易版の問題点

簡易テンプレートマッチングはアルゴリズムが単純だが、性能面において次の2つの問題がある。前述のように、簡易テンプレートマッチングは、一般的な定義に忠実な手法であるため、すなわち、

①全ての画素について相違度を求めるため、どのようなテンプレートのパターンに対してもマッチング位置が検出可能な反面、処理速度が低い。

また、

②ブロックを1画素ずつずらして切り出すため、ブロックのスキャン速度が低い。

（2）高速アルゴリズムによる改良

しかし、2．2節で述べたように、今回用いるテンプレートは図2の十字型であり、他のパターンについては考慮しなくてよい。よって､次のようにすることで高速化が可能になる。

①テンプレートのパターンに特化した、ブロック中の特定の位置だけに着目することで、マッチングしているかどうかを判断し、マッチングしないと判断すればそのブロックの相違度の計算を行わない。

また、ほとんどの場合、テンプレートより対象画像の方がはるかにサイズが大きく、対象画像中にマッチング部分は数個しか存在しないため、対象画像から切り出したブロックのほとんどがマッチングしない。そこで、次のようにすることで高速化が可能になる。

②切り出したブロックの画素値のパターンに応じて、次のブロックを複数画素ずらして切り出す。

アルゴリズムの詳細について、図6にフローチャートを示す。図6中の太枠で囲んだ処理の部分はさらに複数の処理から構成されることを示している。なお、高速テンプレートマッチングにおいてはテンプレートのサイズを32x32画素としている。そのため、X増分値・Y増分値の取りうる値も1から32までとなっている。

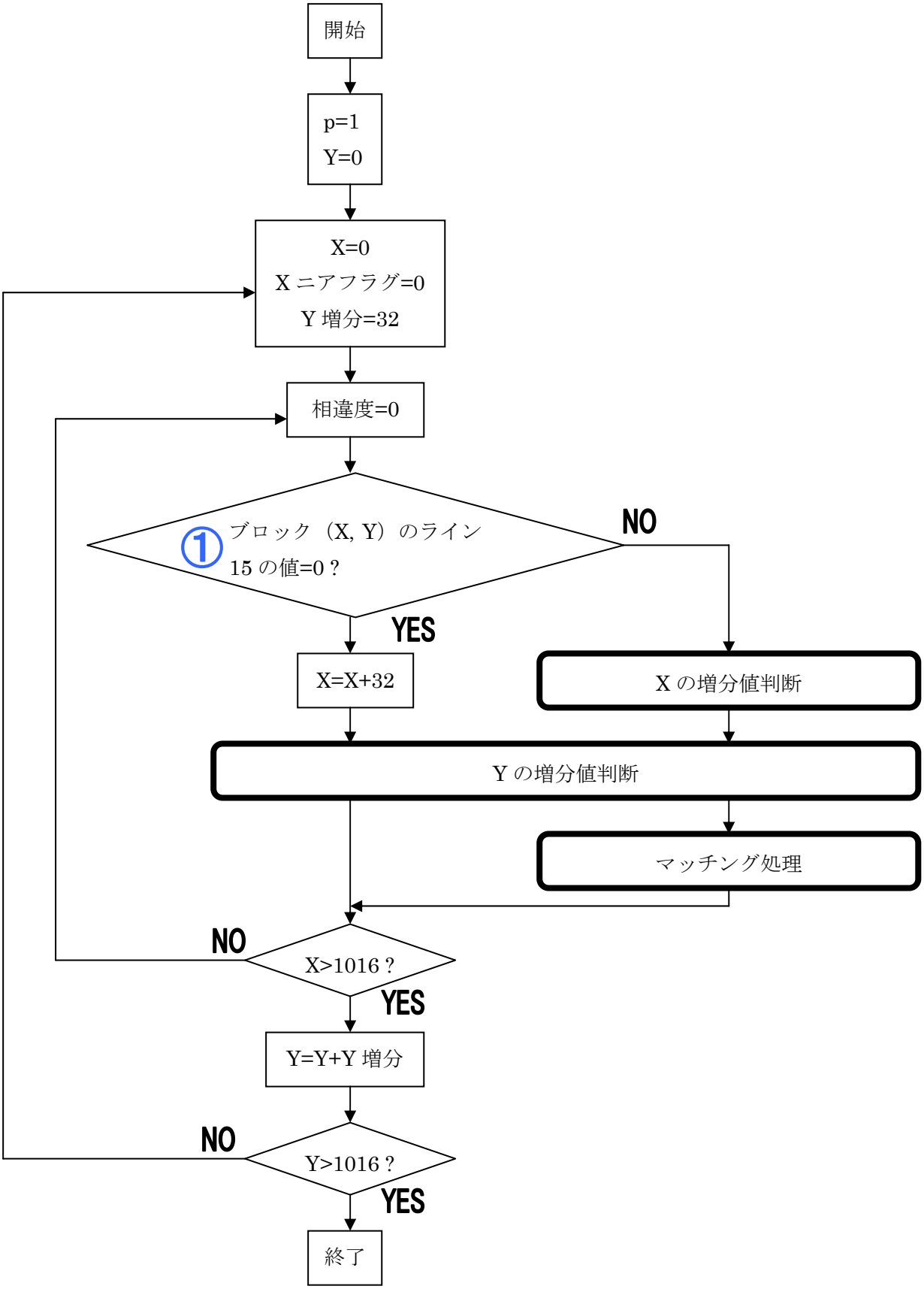

図 6：高速テンプレートマッチングのアルゴリズム

図 6 中の①において、該当ブロックのマッチング処理を行うかどうかを判断している。図 7 に示すように、ライン 15 はブロックの中央にあたり、このラインの値が 0 でない場合、即ち、いずれかのビットが 1 ならば、ライン 15 は十字型の横棒の部分である可能性があるとしてマッチング処理を行う。ライン 15 の値が 0 ならば、ミスマッチしていると判断してそのブロックを捨て、次のブロックを切り出す。

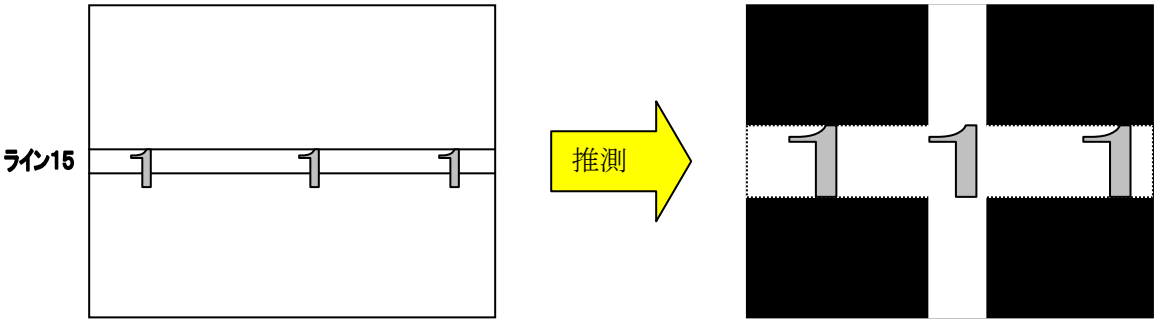

図 7：ブロックのライン 15 の値によるマッチング処理の判断

続いて、マッチング処理に移らないならば X の増分値を 32 とし、マッチング処理に移る場合は以下のように X の増分値の判断を行う。図 8 に X 増分値判断の処理の詳細を示す。なお、ここでは、切り出したブロックの右側にマッチング部分が近づいていることを示すフラグとして、X ニアフラグというものを定義する。まず図 8 中の②のようにライン 15 の最下位ビットが 1 ならば、図 9 のようにマッチング部分が現在切り出しているブロックのすぐ右方にあるかもしれないと推測して、X ニアフラグを 1 にする。X の増分値については更に条件を追加して場合分けする。一方、ライン 15 の最下位ビットが 0 ならば、X ニアフラグを 0 とし、X 増分値を 32 とする。

さらに、ライン 15 の最下位ビットが 1 の場合は、X の増分値を決めるために図 8 中の③の処理を行う。X ニアフラグが 1 の場合は、少なくとも該当ブロックの直前に切り出したブロックにおいて、マッチング部分が近いということを示しているので、X 増分値を 1 とする。また、ライン 15 の 15 ビット目が 1 である場合も、図 10 のように、マッチング部分がかなり近づいていると推測して同様に X 増分値を 1 とする。X ニアフラグが 1 であり、かつライン 15 の 15 ビット目が 0 ならば、図 10 ほど近づいてはいないとして、X 増分値を 12 とする。

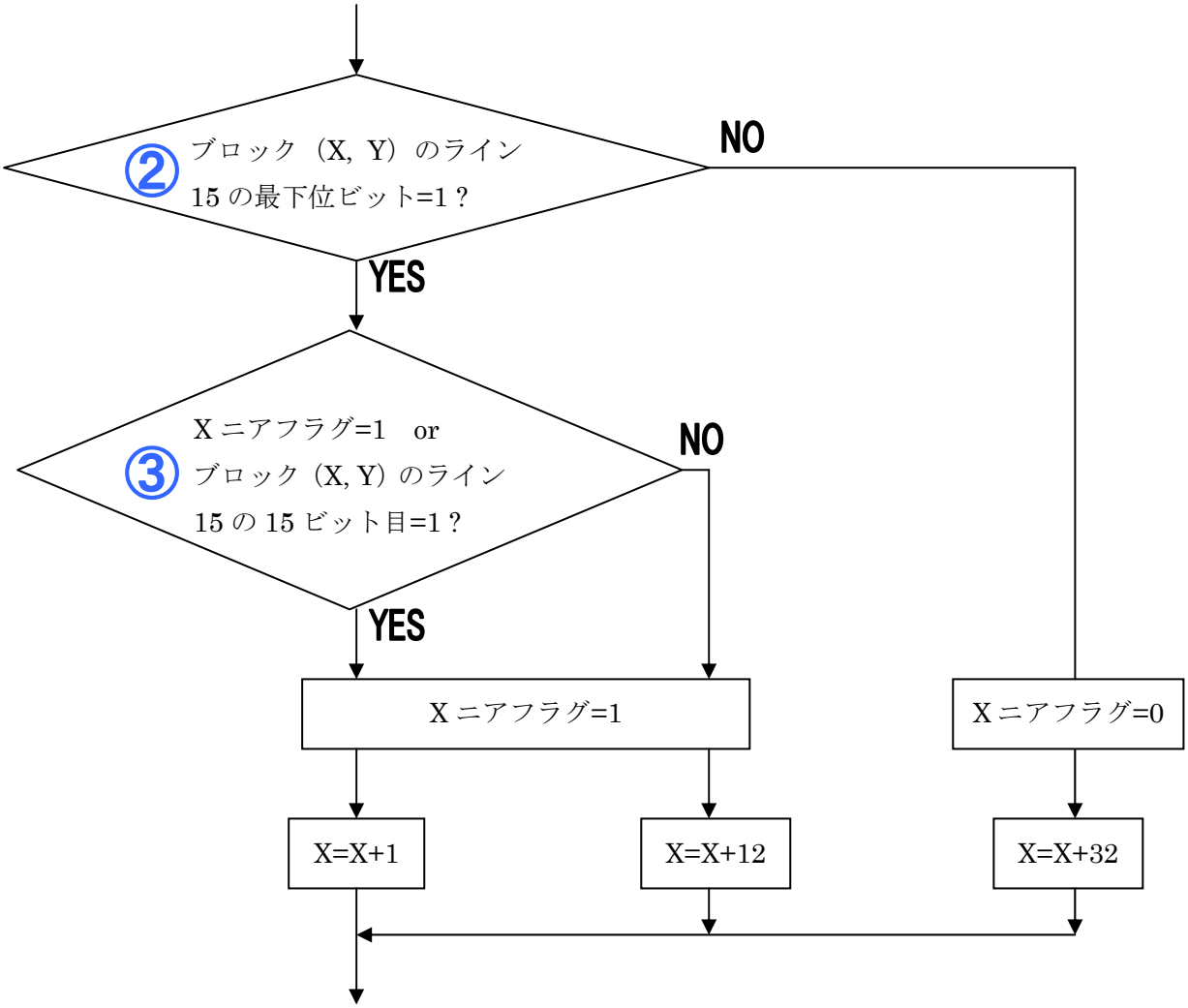

図 8：X 増分値の判断

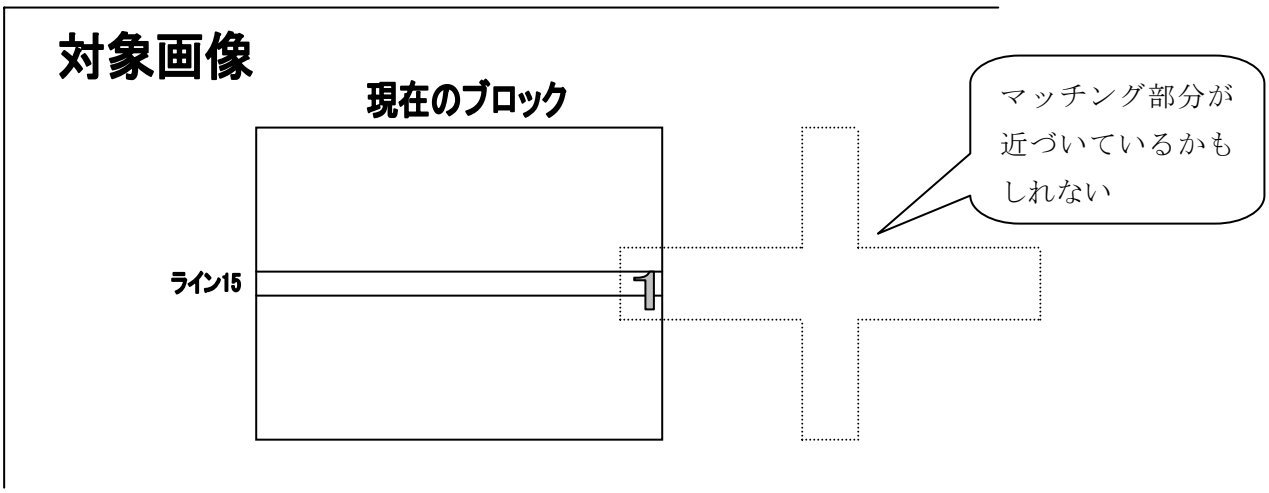

図 9：ライン 15 の最下位ビットが 1 の場合

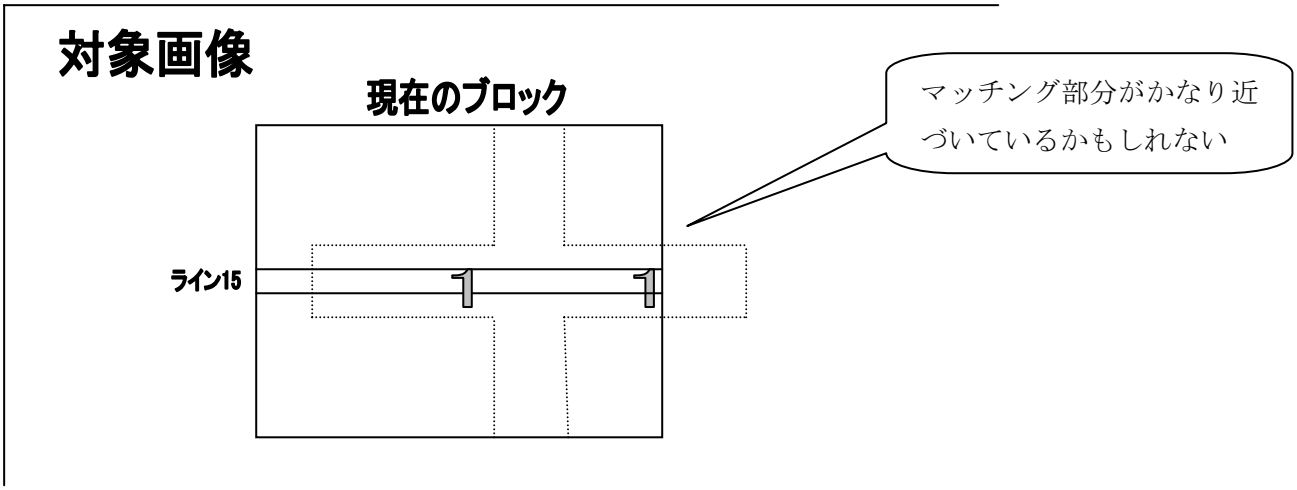

図 10：ライン 15 の最下位ビットと 15 ビット目が共に 1 の場合

該当ブロックのマッチング処理を行う行わないに関わらず、続けてライン 31 を読み出し、Y の増分値の判断を行う。この処理の詳細を図 11 に示す。

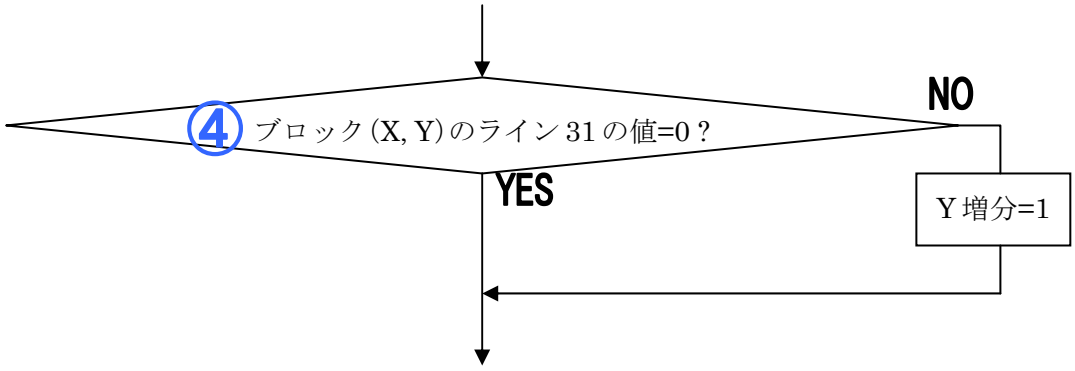

図 11：Y 増分値の判断処理

図 11 中の④において、Y の増分値を判断している。図 12 に示すように、ライン 31 はブロックの最下位ラインにあたり、このラインの値が 0 でない場合、このブロックのすぐ下方にマッチング部分があるかもしれないという予測を立て、デフォルトで 32 であった Y の増分値を 1 とし、Y 方向のずらし方を少しずつにする。一方、ライン 31 の値が 0 ならば、Y の増分値は変更しない。即ち、ブロック（i, Y）(i=0, … , 1016)のすべてのライン 31 が 0 ならば、次の Y の値は現在の Y に 32 を加えたものとなる。

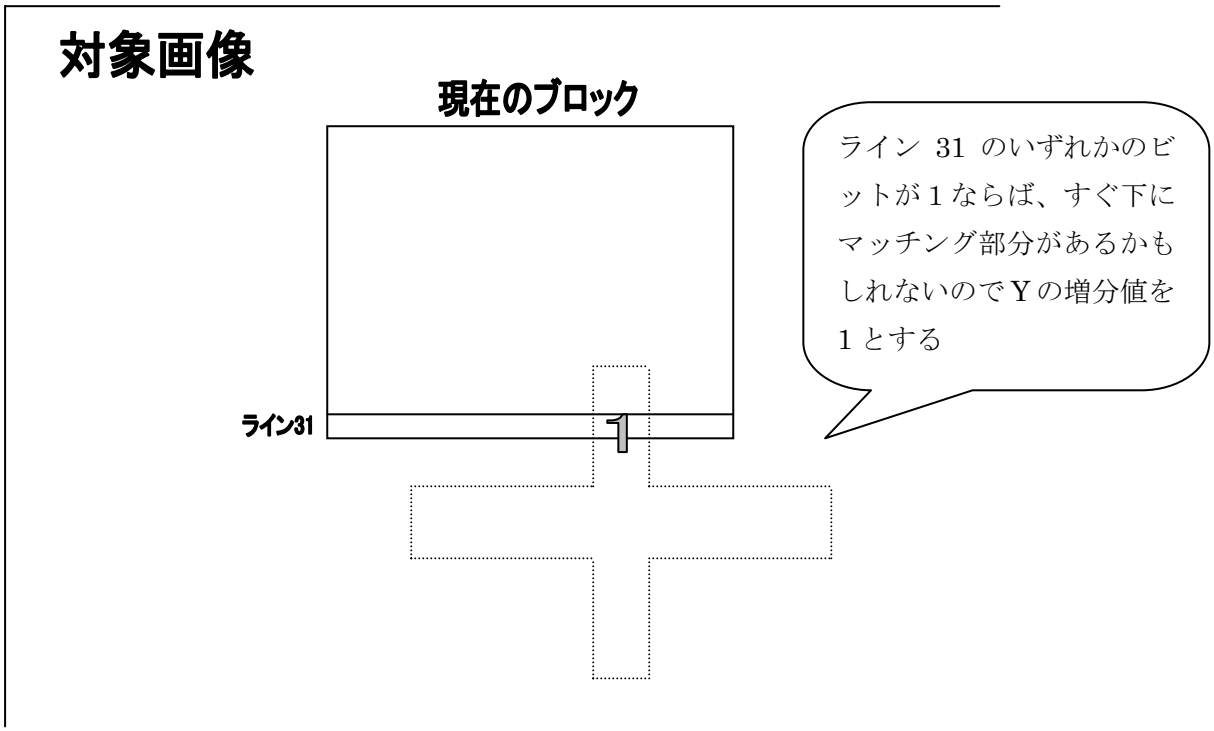

図 12：Y の増分値の判断方法

マッチング処理を行う場合は、Y の増分値の判断が終わるとこの処理に移る。マッチング処理の詳細を図 13 に示す。相違度が最も小さいブロックのみを求める簡易版のマッチング処理とは異なり、高速版においては、図 13 中の⑤のように最大許容相違度を設定して、この値より小さい相違度のブロックの座標を全て結果として保持する。

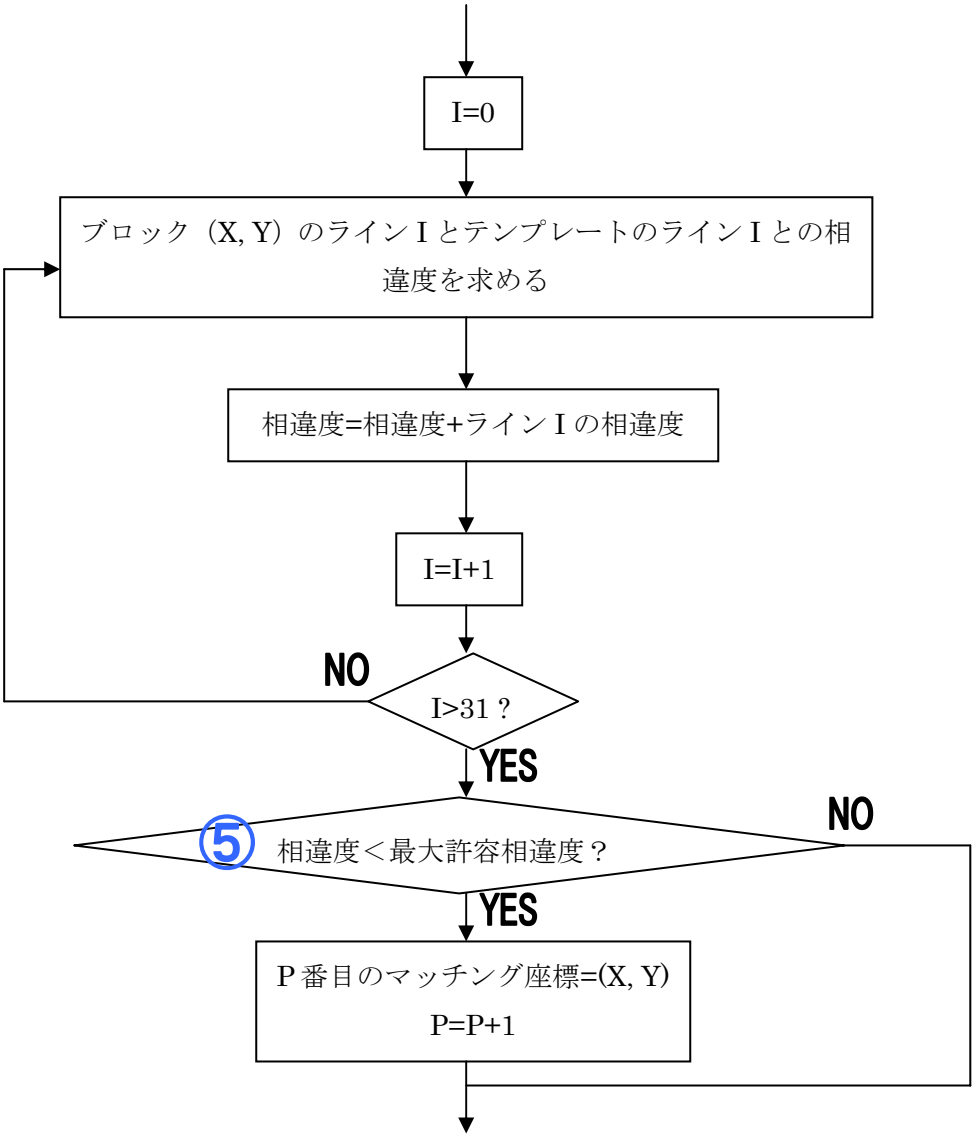

図 13：高速版マッチング処理の詳細

3. 簡易テンプレートマッチングの設計と検証

3.1. 簡易テンプレートマッチングの構成

図 14 に簡易テンプレートマッチングの構成を示す。

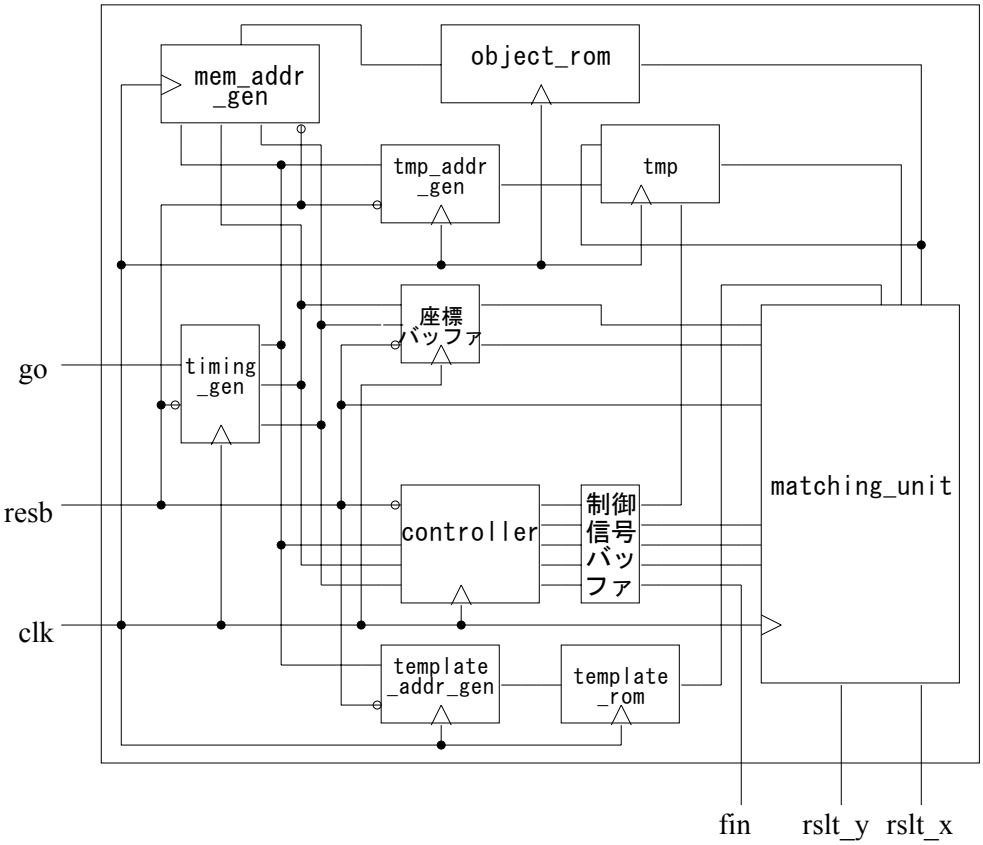

図 14：簡易テンプレートマッチングのブロック図

簡易版で用いる画像はテンプレートが 8x8 ピクセル、対象画像が 1024x1024 ピクセルであり、それぞれ既に 2 値化されているものとする。従って、画像データを一度に 8 ビットずつアクセスするとして、テンプレートのメモリ template_rom（図 14 下部）をユーザロジック内部の 8 つの 8 ビットレジスタ、対象画像のメモリ object_rom（図 14 上部）を 128K エントリの 8 ビットメモリとした。ただ、この簡易版に関しては余りに愚直な処理手順を踏んでいるため、ハードウェア化するメリットはそもそも無いので、「C プログラムとの単純な比較のため」と、「4 章で設計する高速テンプレートマッチングの基本的な骨組みを先に作成するため」の設計として、FPGA 上へは実装しないつもりである。よって、今回は、この対象画像のメモリ及びテンプレートメモリを、ユーザロジック内部の ROM としてモジュール化し、入力アドレスに対して画像データが正しく出力されるように記述した。これらのメモリは簡

易テンプレートマッチングモジュール plain_tm のコンポーネントとして設計した。

簡易テンプレートマッチング plain_tm は、以下の 8 つのコンポーネントを含んでいる。

・タイミングジェネレータ timing_gen
・対象画像メモリアドレスジェネレータ mem_addr_gen
・一時メモリアドレスジェネレータ tmp_addr_gen
・テンプレートメモリアドレスジェネレータ template_addr_gen
・制御ユニット controller
・マッチング処理ユニット matching_unit
・対象画像メモリ object_rom
・テンプレートメモリ template_rom

3.2. 入出力信号の役割とコンポーネント以外の構成要素

（1）入出力信号の役割

表 1 に簡易テンプレートマッチング plain_tm の入出力ポートの説明を示す。以下、信号名に #（シャープ）の付いたものはその信号がアクティブ・ローであることを示す。

表 1：簡易テンプレートマッチング plain_tm の入出力ポート

種別	ポート名	幅	内容
入力	go	1	テンプレートマッチング開始信号
入力	#resb	1	リセット信号
入力	clk	1	クロック信号
出力	rslt_x	10	マッチング部分のX座標
出力	rslt_y	10	マッチング部分のY座標
出力	fin	1	テンプレートマッチング完了信号

クロック信号 clk はそれぞれのコンポーネント及び各メモリ、座標バッファに供給する。リセット信号 resb もそれぞれのコンポーネント、座標バッファに接続される。この信号はアクティブ・ローであり、回路電源投入時の誤動作を防ぐ。

（2）コンポーネント以外の構成要素

簡易テンプレートマッチング plain_tm の各コンポーネントの説明に入る前に、それら以外の構成要素である座標バッファ、制御信号バッファ、一時メモリ tmp について説明する。

（3）座標バッファと制御信号バッファ

座標バッファと制御信号バッファは共にパイプラインレジスタの役割を果たす。それぞれタイミングジェネレータとマッチング処理ユニット、制御ユニットとマッチング処理ユニットの間に置くことで、メモリから読み出した画像データと、その座標位置や制御信号との整合性を保証する。

（4）一時メモリ tmp

一時メモリ tmp はメモリアクセスを常に 1 クロックで完了させるために必要なもので、8x8 ビットのメモリを想定した。

（5）一時メモリを備えていない場合

図 15 に、この一時メモリが無い場合における、最初の切り出し部分（以下、切り出し部分をブロックと呼ぶ）であるブロック 0 とその次のブロック 1 のデータをアクセスする様子を示す。この場合は、対象画像のデータを常に対象画像メモリから読み出さなければならない。

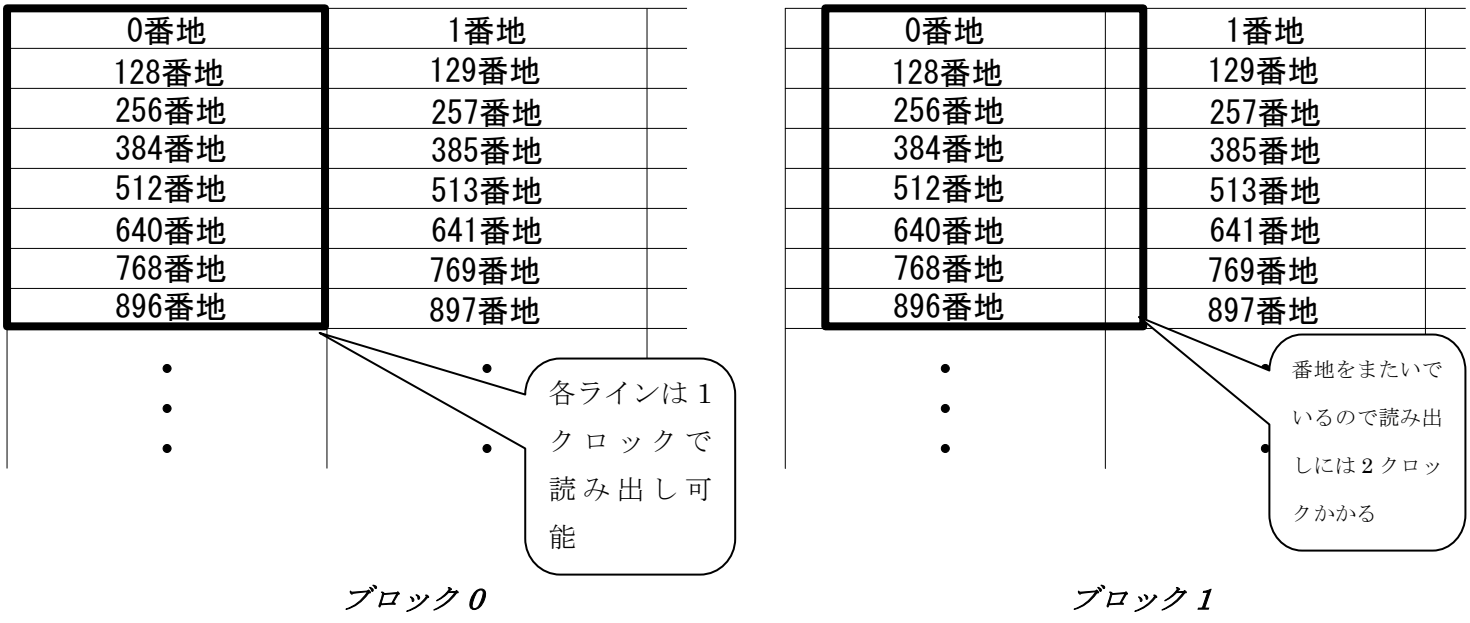

図 15：一時メモリが無い場合のメモリアクセス

ブロック 0 のデータを読み出す場合は、アドレスを順番に 0 番地、128 番地、256 番地、…とすれば、1 クロックで 8 ビットずつアクセス出来るので、この場合は良い。

しかし、次にブロック 1 のデータを読み出す場合、1 画素ずらして読み出すため、8 ビットのデータを読み出すために番地をまたがなければならない。すなわち、1 行目のデータは 0 番地の下位 7 ビットと 1 番地の最上位ビットを連結したものであるため、アクセスに 2 クロ

ックかかる。こうして見ていくと、1 クロックでアクセス可能なのは 8 ブロックにつき 1 ブロックだけなのでメモリアクセスの速度としてはかなり低くなる。

（6）一時メモリを備えている場合

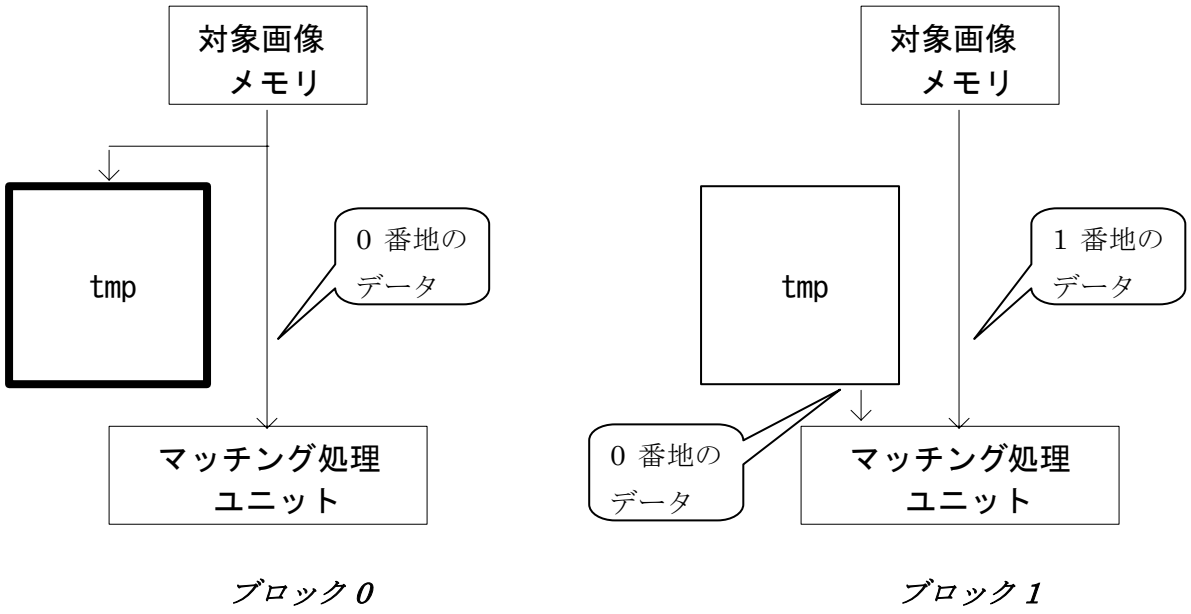

図 16：一時メモリを用いた場合のブロック 0 とブロック 1 のデータの読み出し

そこで、一時メモリを用いるが、図 16 のように、まずブロック 0 においては、対象画像メモリの 0 番地から読み出したデータをマッチング処理ユニットに送ると同時に tmp の 0 番地にも送る。これを 8 サイクル繰り返すとブロック 0 のデータはすべて読み出され、tmp はそのデータで満たされる。次に、ブロック 1 を読み出すが、このブロックの各行の上位 7 ビットは既に tmp に格納されている。よって、tmp と対象画像メモリの両方からデータを取って来れば 1 クロックで所望のデータが得られる。以下、ブロック 2 からブロック 7 まではブロック 1 と同様の方法で読み出す。

次に、ブロック 8 のデータを読み出す場合は、もはや一時メモリだけでは欲しいデータが賄えないので、また対象画像メモリにアクセスし、tmp に格納される。このように、1 クロックでメモリアクセスが行えるようにしている。

3.3. 簡易テンプレートマッチングのコンポーネント

以下、簡易テンプレートマッチング plain_tm の 6 つのコンポーネントについて説明する。

（1）タイミングジェネレータ timing_gen

表 2 にタイミングジェネレータ timing_gen の入出力ポートの説明を示す。このモジュー

ルは、go がアサートされると、idx のカウントを始める。出力信号は後続の 3 つのアドレスジェネレータ mem_addr_gen, tmp_addr_gen, template_addr_gen 及び制御ユニット controller に送られる。VHDL 記述量は 60 行程度であった。

表 2：タイミングジェネレータ timing_gen の入出力ポート

種別	ポート名	幅	内容
入力	go	1	テンプレートマッチング開始信号
入力	#resb	1	リセット信号
入力	clk	1	クロック信号
出力	idx	3	ブロック内のラインインデックス
出力	xpos	10	当該ブロックの左上のX座標（idxがオーバーフローするたびにインクリメントされる）
出力	ypos	10	当該ブロックの左上のY座標（xposが1016になるたびにインクリメントされる）

（2）対象画像メモリアドレスジェネレータ mem_addr_gen

表 3 に対象画像メモリアドレスジェネレータ mem_addr_gen の入出力ポートの説明を示す。このモジュールは、idx, xpos, ypos の値を元に、対象画像メモリへ適切なアドレスを出力する。VHDL 記述量は 40 行程度であった。

表 3：対象画像メモリアドレスジェネレータ mem_addr_gen の入出力ポート

種別	ポート名	幅	内容
入力	#resb	1	リセット信号
入力	clk	1	クロック信号
入力	idx	3	ブロック内のラインインデックス
入力	xpos	10	当該ブロックの左上のX座標
入力	ypos	10	当該ブロックの左上のY座標
出力	addr	17	対象画像メモリのアドレス

（3）一時メモリアドレスジェネレータ tmp_addr_gen

表 4 に一時メモリアドレスジェネレータ tmp_addr_gen の入出力ポートの説明を示す。このモジュールは、一時メモリへ適切なアドレスを出力する。clk の立ち上がり毎に、idx を元に、一時メモリに与えるアドレス addr を生成するが、これはただ単に、addr に idx を代入すれば済む。というのは、一時メモリの各番地はブロック内の各インデックスに 1 対 1 に対応するからである。VHDL 記述量は 30 行程度であった。

表 4：一時メモリアドレスジェネレータ tmp_addr_gen の入出力ポート

種別	ポート名	幅	内容
入力	#resb	1	リセット信号
入力	clk	1	クロック信号
入力	idx	3	ブロック内のラインインデックス
出力	addr	3	一時メモリのアドレス

（4）テンプレートメモリアドレスジェネレータ template_addr_gen

表 5 にテンプレートメモリアドレスジェネレータ template_addr_gen の入出力ポートの説明を示す。

表 5：テンプレートメモリアドレスジェネレータ template_addr_gen の入出力ポート

種別	ポート名	幅	内容
入力	#resb	1	リセット信号
入力	clk	1	クロック信号
入力	idx	3	ブロック内のラインインデックス
出力	addr	3	テンプレートメモリのアドレス

このモジュールは、idx の値を元に、テンプレートメモリへ適切なアドレスを出力する。clk の立ち上がり毎に、idx を元にテンプレートメモリに与えるアドレス addr を生成するが、これもただ単に addr に idx を代入すれば済む。ブロック内のインデックスに対応したテンプレートのラインを出力する必要があるからである。VHDL 記述量は 30 行程度であった。

（5）制御ユニット controller

表 6 に制御ユニット controller の入出力ポートの説明を示す。このモジュールは、idx, xpos, ypos の値を元に、一時メモリ及びマッチング処理ユニットへの制御信号を出力する。clr 信号は、後述するマッチング処理ユニット内部のモジュールである結果アキュムレータに接続される。この信号がアサートされると積算レジスタがクリアされる（0 にクリアされるのではなくて、新しいブロックの 1 ライン目の相違度を書き込む）。これは、ブロック内の相違度を積算した後に、新しいブロックの、相違度の計算のために必要である。

また、cmp 信号は、後述するマッチング処理ユニット内部の最小相違度保持ユニットに接続される。この信号がアサートされると、現在保持している最小相違度と、処理の終わったブロックの相違度を比較し、より小さい方を保持する。それと同時に、テンプレートマッチングが出力するための座標も更新する。

表 6：制御ユニット controller の入出力ポート

種別	ポート名	幅	内容
入力	#resb	1	リセット信号
入力	clk	1	クロック信号
入力	idx	3	ブロック内のラインインデックス
入力	xpos	10	当該ブロックの左上のX座標
入力	ypos	10	当該ブロックの左上のY座標
制御出力	tmp_we	1	一時メモリのライトイネーブル信号（idxが"000"の時1、それ以外の時0）
制御出力	tail	3	一時メモリのデータと対象画像メモリのデータを連結させた16ビットから切り出す8ビットの最下位ビット
制御出力	clr	1	積算レジスタのクリア（idxが"000"の時1、それ以外の時0）
制御出力	cmp	1	相違度の比較（idxが"111"の時1、それ以外の時0）
制御出力	fin	1	テンプレートマッチングの終了信号（idxが"111"かつxposが1016かつyposが1016の時1）

tail 信号はマッチング処理ユニット内部の 8 ビットセレクタに接続され、図 17 のように、一時メモリから読み出した値と対象画像メモリから読み出した値を連結させた、内部の 16 ビットレジスタ tmp_mem から所望の 8 ビットを選ぶために、その 8 ビットの最下位ビットを指定するものである。

xpos の下位 3 ビットが”000”でない時、tail はその値に基づいて適切な値が代入される。また、この場合は、tmp から読み出す値も使うため、tmp_we はネゲートしておく。VHDL 記述量は 60 行程度であった。

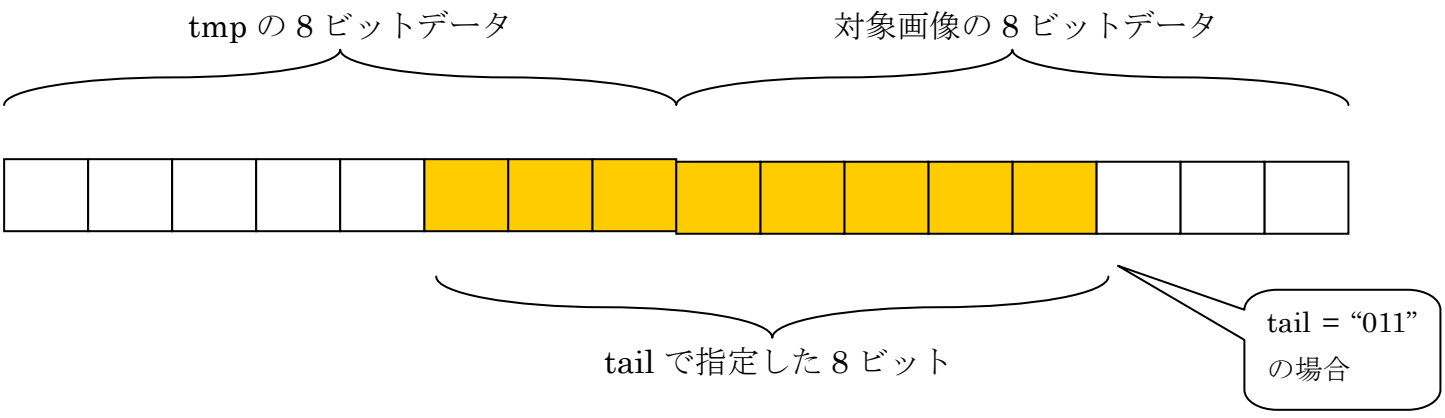

図 17：8 ビットセレクタ内部の 16 ビットレジスタ tmp_mem と tail 信号による 8 ビットの指定の例

（6）マッチング処理ユニット matching_unit

図 18 にマッチング処理ユニット matching_unit のインターフェースを示す。

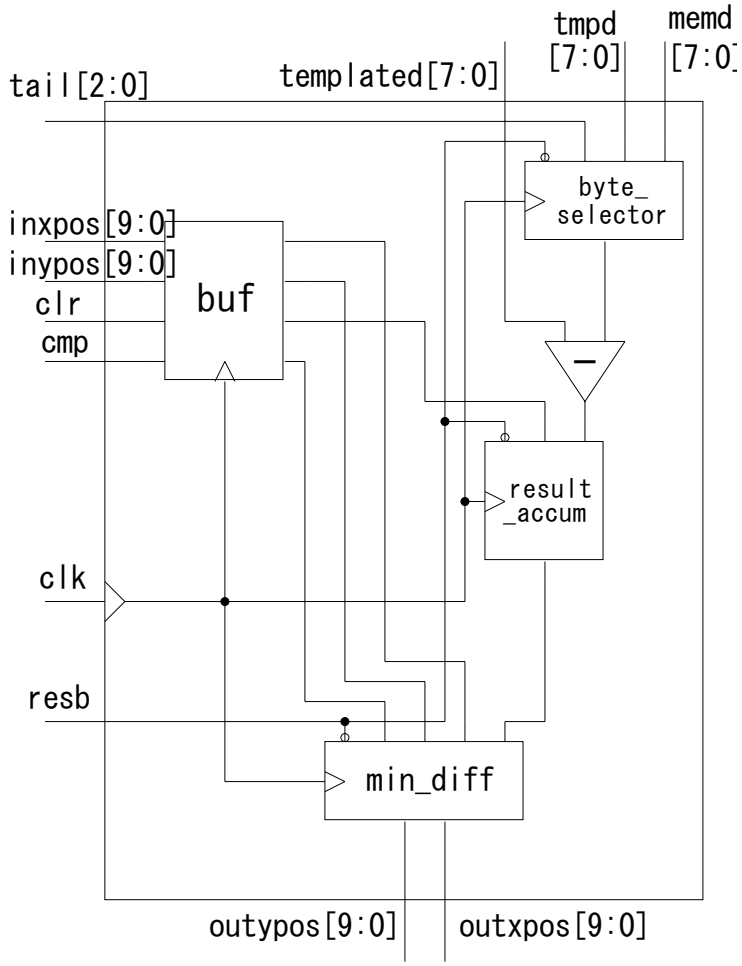

図 18 : マッチング処理ユニット matching_unit のインターフェース

このモジュールは、以下の 3 つのコンポーネントを含んでいる。

・8 ビットセレクタ byte_selector
・結果アキュムレータ result_accum
・最小相違度保持ユニット min_diff

表 7 にマッチング処理ユニット matching_unit の入出力ポートの説明を示す。clk が立ち上がる毎に、対象画像メモリからのデータ memd と一時メモリからのデータ tmpd から 8 ビットセレクタ byte_selector によって 8 ビットデータを選択し、テンプレートメモリからのデータ templated との差を取り、それを後続の結果アキュムレータ result_accum へ送る。同じタイミングにおいて、結果アキュムレータから出力される積算値を、最小相違度保持ユニット min_diff が現在保持している最小相違度と比較し判定するとともに、必要ならばマッチング座標の更新を行う。

表 7：マッチング処理ユニット matching_unit の入出力ポート

種別	ポート名	幅	内容
入力	#resb	1	リセット信号
入力	clk	1	クロック信号
入力	inxpos	10	当該ブロックの左上のX座標
入力	inypos	10	当該ブロックの左上のY座標
入力	templated	8	テンプレートからのデータ
入力	tmpd	8	一時メモリからのデータ
入力	memd	8	対象画像メモリからのデータ
制御入力	tail	3	一時メモリのデータと対象画像メモリのデータを連結させた16ビットから切り出す8ビットの最下位ビット
制御入力	clr	1	積算レジスタのクリア
制御入力	cmp	1	相違度の比較
出力	outxpos	10	マッチング部分のX座標
出力	outypos	10	マッチング部分のY座標

なお、図 18 中にある buf は、座標値や各制御信号のための、パイプラインレジスタの役割を担うものであり、画像データとの同期を取る。VHDL 記述量は 140 行程度であった。

（7）対象画像メモリ object_rom

表 8 に対象画像メモリ object_rom の入出力ポートの説明を示す。このモジュールは、対象画像メモリアドレスジェネレータから指定されたアドレスのデータを出力する。ROM を想定したので、入力データポートは無い。VHDL 記述量は 50 行程度であった。

表 8：対象画像メモリ object_rom の入出力ポート

種別	ポート名	幅	内容
入力	clk	1	クロック信号
入力	address	17	アドレス
出力	q	8	出力データ

（8）テンプレートメモリ template_rom

表 9 にテンプレートメモリ template_rom の入出力ポートの説明を示す。このモジュールは、テンプレートメモリアドレスジェネレータから指定されたアドレスのデータを出力する。ROM を想定したので、入力データポートは無い。VHDL 記述量は 40 行程度であった。

表 9：テンプレートメモリ template_rom の入出力ポート

種別	ポート名	幅	内容
入力	clk	1	クロック信号
入力	address	3	アドレス
出力	q	8	出力データ

3.4. マッチング処理ユニットのコンポーネント

以下、マッチング処理ユニット matching_unit が含む 3 つのコンポーネントを説明する。

（1）8 ビットセレクタ byte_selector

表 10 に 8 ビットセレクタ byte_selector の入出力ポートの説明を示す。

表 10：8 ビットセレクタ byte_selector の入出力ポート

種別	ポート名	幅	内容
入力	#resb	1	リセット信号
入力	clk	1	クロック信号
制御入力	tail	3	一時メモリのデータと対象画像メモリのデータを連結させた16ビットから切り出す8ビットの最下位ビット
入力	tmpd	8	一時メモリからのデータ
入力	memd	8	対象画像メモリからのデータ
出力	data	8	切り出した8ビットデータ

このモジュールは、一時メモリから読み出した 8 ビットのデータと、対象画像から読み出した 8 ビットのデータを連結させた、16 ビットの中から適切な 8 ビットのデータを選び出力する。

clk が立ち上がる毎に、図 17 のように、16 ビットから tail によって選び出す 8 ビットの最下位ビットを指定して、適切な 8 ビットを出力する。VHDL 記述量は 40 行程度であった。

（2）結果アキュムレータ result_accum

表 11 に結果アキュムレータ result_accum の入出力ポートの説明を示す。このモジュールは、処理対象となるブロック内のライン毎の相違度を取りこみ、ブロック毎に積算して出力する。VHDL 記述量は 40 行程度であった。

表 11：結果アキュムレータ result_accum の入出力ポート

種別	ポート名	幅	内容
入力	#resb	1	リセット信号
入力	clk	1	クロック信号
制御入力	clr	1	積算レジスタのクリア（1の時、diffの値を内部のアキュムレータに代入、0の時、内部アキュムレータにdiffの値を積算）
入力	diff	7	ライン毎の相違度
出力	diff_ac	7	積算された相違度

（3）最小相違度保持ユニット min_diff

表 12 に最小相違度保持ユニット min_diff の入出力ポートの説明を示す。

表 12：最小相違度保持ユニット min_diff の入出力ポート

種別	ポート名	幅	内容
入力	#resb	1	リセット信号
入力	clk	1	クロック信号
制御入力	cmp	1	相違度の比較（1の時、diff_acと内部の最小相違度とを比較する、0の時、比較はしない）
入力	inxpos	10	該当ブロックのX座標
入力	inypos	10	該当ブロックのY座標
入力	diff_ac	7	積算された相違度
出力	outxpos	10	最小相違度にとなったブロックのX座標
出力	outypos	10	最小相違度にとなったブロックのY座標

このモジュールは、現在保持している最小の相違度と新たに積算された相違度とを比較して、常に最小の相違度とそのブロックの座標位置を内部に保持し、座標位置だけを出力する。非同期リセット信号 resb がアサートされると、最小相違度レジスタだけは初期値として 127 が代入され、それ以外の内部信号は 0 にクリアされる。VHDL 記述量は 50 行程度であった。

3.5. シミュレーションによる検証と性能評価

（1）ModelSim によるビヘイビアシミュレーション

この簡易テンプレートマッチングに関しては上述のように、FPGA 上への実装は行わず、シミュレーションツール ModelSim による動作レベルの検証に留めた。検証の様子を図 19 に示す。この検証においては、対象画像中のマッチング座標を（557, 186）とした。図 19 中のウインドウの左半分に設計した回路に含まれる信号が羅列されており、ウインドウの右半分にそれぞれの信号に対応する波形が表示される。この図では、黄色の枠で囲んだ部分を見

て結果を確認する。

図では、テンプレートマッチング終了信号 fin が途中で 1 になっており、テンプレートマッチングがここで終了していることを確認できる。このとき、マッチング座標の X 座標、Y 座標を示す rslt_x,rslt_y がそれぞれ 557, 186 となっており、テンプレートマッチングが正常に実行できていることが確認できる。

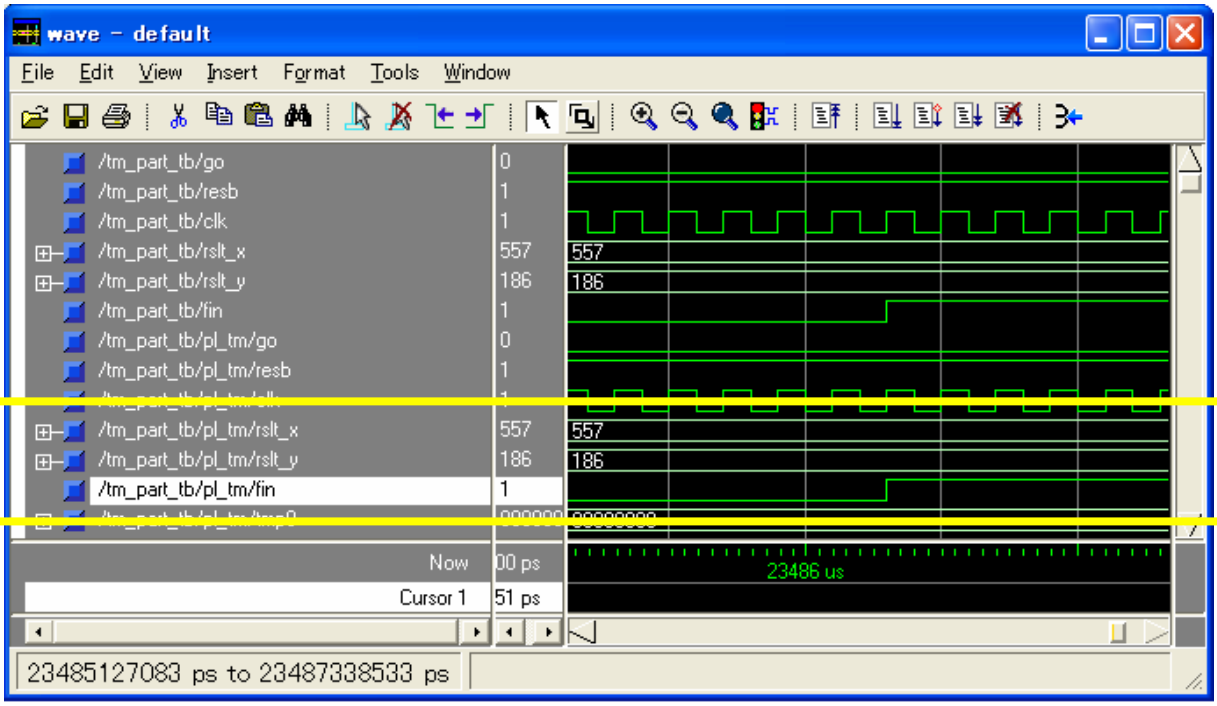

図 19：ModelSim を用いた簡易テンプレートマッチングの動作レベル検証の様子

（2）C プログラムとの性能比較・評価

動作レベルのシミュレーションではあるが、今回記述した VHDL 版の簡易テンプレートマッチングが C 言語で記述したプログラムとどのくらいの性能の差があるかを確かめるために、それぞれの実行時間を測定した。表 13 に、簡易テンプレートマッチングの、C プログラムと VHDL ビヘイビアレベルの実行時間を示す。

表 13：C プログラムと VHDL ビヘイビアレベルの実行時間

	実行時間（ms）
C プログラム	2969
VHDL ビヘイビアレベル	1655

C プログラムのコンパイル条件は Borland C++ Compiler 5.5 で速度最適化オプション-O2 とした。実行環境は本研究室 PC の Bronto（Pentium4 2.8GHz, 2GB RAM, 240GB HDD）

とした。一方、VHDL ビヘイビアレベルのほうは、クロック周波数を 5MHz（100ns ごとに反転）とした。

実行クロック数で見ると、VHDL ビヘイビアレベルが 5.0(MHz)×1.655(s)≒8(百万クロック)、C プログラムが 2800(MHz)×2.969(s)≒8313(百万クロック)であり、VHDL の方が C プログラムに比べて約 1000 分の 1 以下となり、かなりの性能向上が得られていると考えられる。

4. 高速テンプレートマッチングの設計

4.1. 高速テンプレートマッチングの構成

（1）高速テンプレートマッチングの全体的な構成

図 20 に高速テンプレートマッチングの構成を示す。

簡易版と異なり、対象画像・テンプレート画像を共に FPGA 外部の SRAM（1MB）から読んでくる形式を取る。ただ、対象画像のサイズは 1024x1024 画素のままとしたが、テンプレート画像のサイズは 32x32 画素とした。

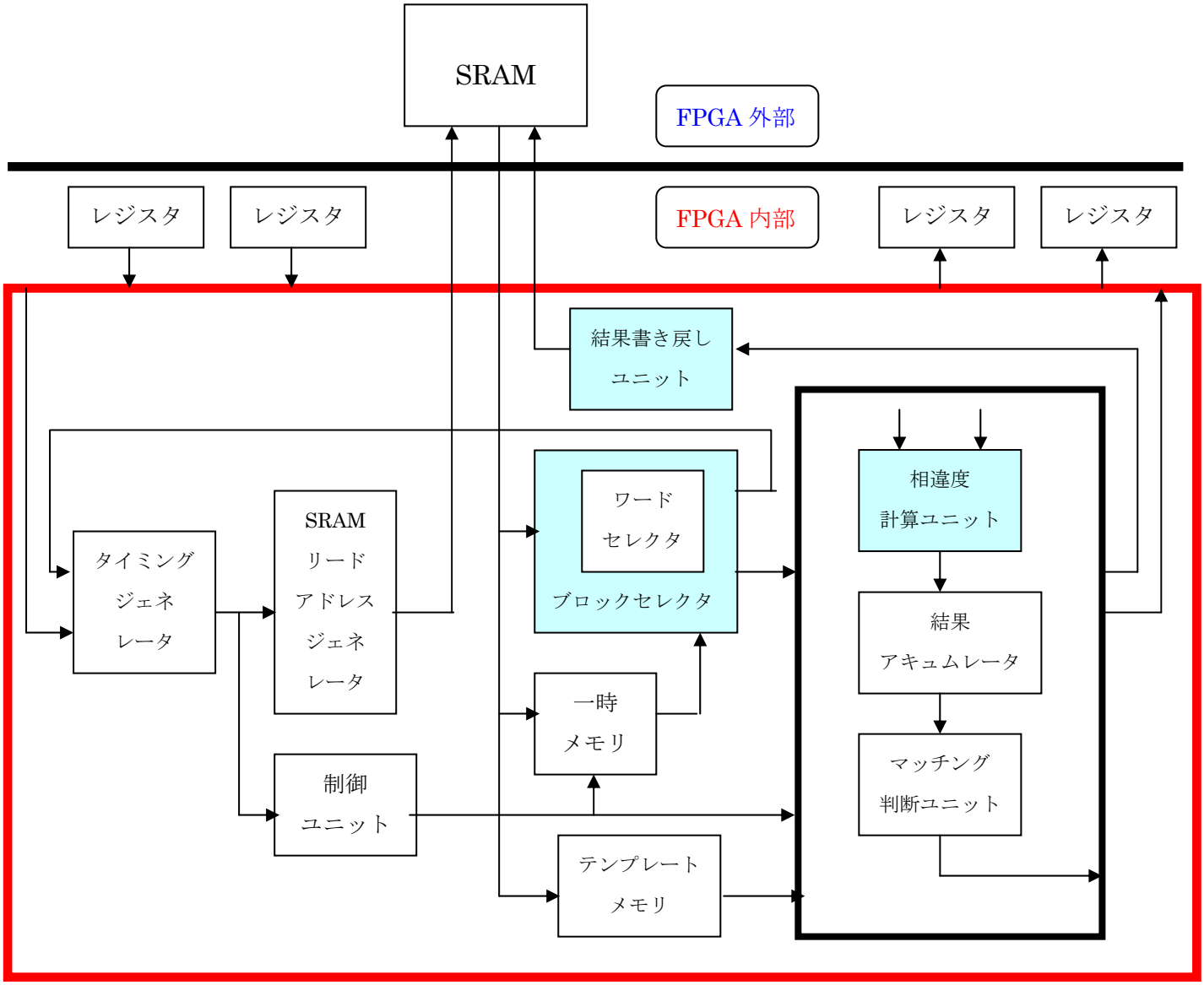

図 20：高速テンプレートマッチングの構成

図 20 の赤枠部分が、設計した高速テンプレートマッチングのモジュールである。高速テンプレートマッチングのモジュールの外側にあるレジスタは、次章で述べる、テンプレートマッチングの実行を開始させたり、終了を確認するために設けたもので、ユーザインターフェースの役割を果たす。

図 20 中の水色で示した部分は、簡易版に対して新しく作成したモジュールである。特にブロックセレクタは２．４節で述べた高速テンプレートマッチングのアルゴリズムを実現しており、高速化の鍵を握っている部分である。

他に、SRAM リードアドレスジェネレータは簡易版における対象画像メモリアドレスジェネレータと同様の機能を持つ。ワードセレクタは簡易版における 8 ビットセレクタに相当するもので、テンプレート画像のサイズ変更に伴い、簡易版に対して変更を加えた。高速版では、このモジュールをマッチング処理ユニットのコンポーネントではなく、ブロックセレクタのコンポーネントとし、マッチング処理ユニットの構成をシンプルにした。一時メモリアドレスジェネレータとテンプレートメモリアドレスジェネレータは、簡易版に対する回路規模の最適化の結果、冗長となったため削除した。

（２）SRAM のアクセス方法

上述の通り、高速版においては画像データを実際にボード上の SRAM から読み出して実行する。そのため、簡易版とは異なり、正確なデータを読んでくるために SRAM に対して正確なリードアクセス手順を踏む必要がある。図 21 に SRAM リードアクセスシーケンスを、表 14 に各信号の説明を示す。

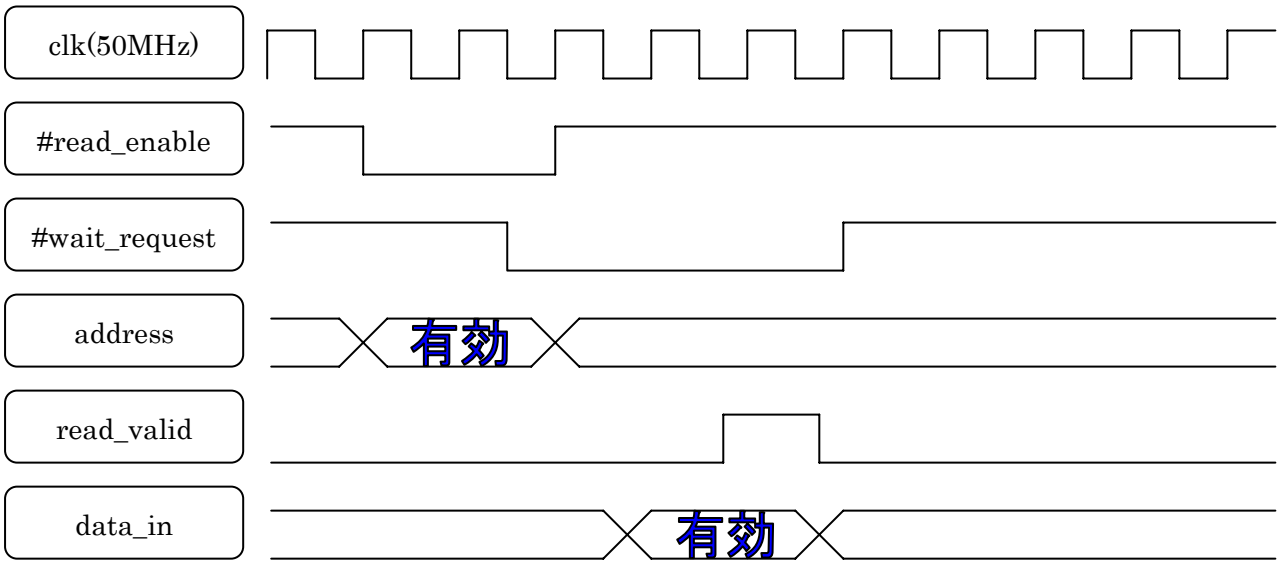

図 21：SRAM リードアクセスシーケンス

表 14：SRAM リードアクセスに用いる信号

信号種別	信号名	幅	内容
出力	#read_enable	1	SRAMに対するリードイネーブル
入力	#wait_request	1	この信号がアクティブになるまでread_enableをアクティブにしておかなければならない
出力	address	20	アドレス
入力	read_valid	1	この信号がアクティブの時data_inは有効
入力	data_in	32	読み出しデータ

また、SRAM ライトアクセスにおいては、マッチング候補の座標を SRAM に書き戻す。図 22 に SRAM ライトアクセスシーケンスを、表 15 に各信号の説明を示す。

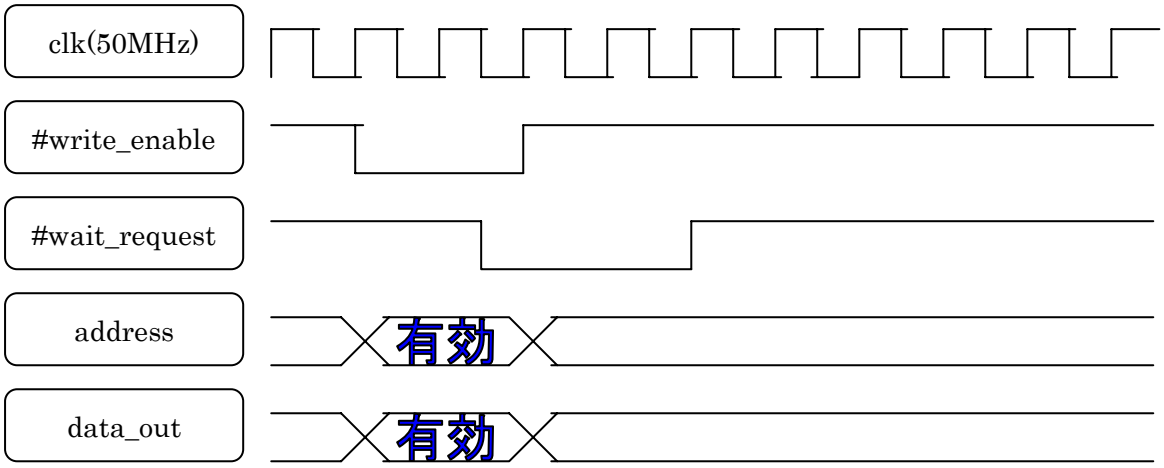

図 22：SRAM ライトアクセスシーケンス

表 15：SRAM ライトアクセスに用いる信号

信号種別	信号名	幅	内容
出力	#write_enable	1	SRAMに対するライトイネーブル
入力	#wait_request	1	この信号がアクティブになるまでwrite_enableをアクティブにしておかなければならない
出力	address	20	アドレス
出力	data_out	32	書きこみデータ

4.2. 変更したモジュール

（1）高速テンプレートマッチング quick_tm

表 16 に高速テンプレートマッチング quick_tm の入出力ポートの説明を示す。青色で示した部分は簡易版に対して新しく追加したものである。

表 16：高速テンプレートマッチング quick_tm の入出力ポート

種別	ポート名	幅	内容
入力	clk	1	クロック信号
入力	#resb	1	リセット信号
入力	st	1	テンプレートマッチング開始信号
出力	rslt	1	1つでもマッチング候補が存在した場合に1を出力
出力	run	1	テンプレートマッチング実行中に1を出力
出力	fin_int	1	テンプレートマッチング終了割り込み信号
入力	org_img_adr	32	SRAM上の対象画像の先頭アドレス
入力	org_img_w	11	対象画像の横幅（画素数）
入力	org_img_h	11	対象画像の縦幅（画素数）
入力	cmp_img_adr	32	SRAM上のテンプレート画像の先頭アドレス
入力	cmp_img_w	10	テンプレート画像の横幅
入力	cmp_img_h	10	テンプレート画像の縦幅
入力	border	10	最大許容相違度
出力	address	20	SRAMのアドレス
入力	data_in	32	SRAMから読み出すデータ
出力	#read_enable	1	SRAMリードイネーブル
入力	read_valid	1	アクティブ時、SRAMから読み出すデータが有効
出力	data_out	32	SRAMに書きこむデータ
出力	#write_enable	1	SRAMライトイネーブル
入力	#wait_request	1	この信号がアクティブの時は、読み出し時は#read_enableを、書きこみ時は#write_enableをアクティブにしておかなければいけない

機能については、外部レジスタ（５．２節（２）で後述）とのインターフェースと SRAM インターフェースを追加した。

外部レジスタとのインターフェースについては、入力部分では、レジスタから対象画像・テンプレート画像のサイズや SRAM 上の先頭アドレスなどを読み出し、各コンポーネントにそれらを渡す。出力部分では、実行状態を示す run やマッチング部分が存在したかどうかを示す rslt をレジスタに書き込む。

SRAM インターフェースについては、画像データを読み出したり、マッチング部分の座標を書き戻すために設けた。VHDL 記述量は 900 行程度となった。

（２）タイミングジェネレータ timing_gen

表 17 にタイミングジェネレータ timing_gen の入出力ポートの説明を示す。青色で示した部分は簡易版に対して新しく追加したものである。

表 17：高速版タイミングジェネレータ timing_gen の入出力ポート

種別	ポート名	幅	内容
入力	clk	1	クロック信号
入力	#resb	1	リセット信号
入力	clk	1	テンプレートマッチング開始信号
入力	nxt	1	0 の 時 、idx=idx+1 と し 、1 の 時 、idx=0 か つ xpos=xpos+xincとする
入力	xinc	6	X座標の増分値
入力	yinc	6	Y座標の増分値
入力	ob_xmax	11	対象画像の横幅
入力	ob_ymax	11	対象画像の縦幅
入力	tm_xmax	10	テンプレート画像の横幅
入力	tm_ymax	10	テンプレート画像の縦幅
入力	active	1	1の時、このモジュールを動作させる
出力	idx	5	ブロック内のラインインデックス
出力	xpos	10	当該ブロックの左上のX座標（idxがオーバーフローするたびにインクリメントされる）
出力	ypos	10	当該ブロックの左上のY座標（xposが1016を超えるたびにインクリメントされる）
出力	TMorOB	1	0の時、idx,xpos,yposはテンプレート画像を読み出すためのもので、1の時、idx,xpos,yposは対象画像を読み出すためのものである

新しく追加した入力信号 nxt,xinc,yinc は後述するブロックセレクタから受け取る信号で、nxt は現在のブロックを捨て、次のブロックへ移るための信号である。xinc,yinc はそれぞれ次のブロックを切り出す際の X 座標・Y 座標の増分値である。また、active は 1 回の SRAM のリードアクセスが終わるまで、このモジュールの動作を止めるための信号である。今回使用する TSUNAMI ボード上の SRAM リードアクセスには 1 回で数クロックかかるため、該当データを読み終えるまで、次に読み出すデータに対応するカウンタ値 idx,xpos,ypos の出力を待たせなければいけない。出力信号 TMorOB は、現在出力しているカウンタ値がテンプレート画像と対象画像のどちらに対応しているのかを示す信号である。VHDL 記述量は 120 行程度となった。

（3）SRAM リードアドレスジェネレータ mem_raddr_gen

表 18 に SRAM リードアドレスジェネレータ mem_raddr_gen の入出力ポートの説明を示す。青色で示した部分は簡易版に対して新しく追加したものである。

表 18：SRAM リードアドレスジェネレータ mem_raddr_gen の入出力ポート

種別	ポート名	幅	内容
入力	clk	1	クロック信号
入力	#resb	1	リセット信号
入力	idx	5	ブロック内のラインインデックス
入力	xpos	10	当該ブロックの左上のX座標
入力	ypos	10	当該ブロックの左上のY座標
入力	TMorOB	1	0の時、アドレスはテンプレート画像に対応しており、1の時、アドレスは対象画像に対応している
入力	ob_badr	32	SRAM上の対象画像の先頭アドレス
入力	tm_badr	32	SRAM上のテンプレート画像の先頭アドレス
入力	org_img_w	11	対象画像の横幅
入力	go	1	テンプレートマッチング開始信号
出力	addr	19	SRAMアドレス
出力	#r_en	1	SRAMリードイネーブル信号

新しく追加した入力信号 TMorOB により、テンプレート画像のアドレス、あるいは対象画像のアドレスの何れかを生成する。VHDL 記述量は 90 行程度となった。

（4）制御ユニット controller

表 19 に制御ユニット controller の入出力ポートの説明を示す。青色で示した部分は簡易版に対して新しく追加したものである。

表 19：高速版制御ユニット controller の入出力ポート

種別	ポート名	幅	内容
入力	clk	1	クロック信号
入力	#resb	1	リセット信号
入力	idx	5	ブロック内のラインインデックス
入力	xpos	10	当該ブロックの左上のX座標
入力	ypos	10	当該ブロックの左上のY座標
入力	ob_xmax	11	対象画像の横幅
入力	ob_ymax	11	対象画像の縦幅
入力	tm_xmax	10	テンプレート画像の横幅
入力	tm_ymax	10	テンプレート画像の縦幅
入力	st	1	テンプレートマッチング開始信号
出力	tmp_we	1	一時メモリのライトイネーブル信号（idxが0の時1、それ以外の時0）
出力	clr	1	積算レジスタのクリア（idxが0の時1、それ以外の時0）
出力	cmp	1	相違度の比較（idxが31の時1、それ以外の時0）
出力	run	1	テンプレートマッチング実行中に1を出力

制御ユニットは簡易版に対してほとんど変更を加えなかった。ただ、簡易版に含まれていた出力信号 tail は、これを利用するモジュールにおいて、他の信号で置き換えることが出来たため、削除した。VHDL 記述量は 80 行程度となった。

（5）ワードセレクタ word_selector

表 20 にワードセレクタ word_selector の入出力ポートの説明を示す。青色で示した部分は簡易版に対して新しく追加したものである。

表 20：ワードセレクタ word_selector の入出力ポート

種別	ポート名	幅	内容
入力	clk	1	クロック信号
入力	#resb	1	リセット信号
入力	tmpd	32	一時メモリからのデータ
入力	memd	32	SRAMからのデータ
入力	xpos	10	当該ブロックの左上のX座標
入力	st	1	テンプレートマッチング開始信号
出力	data	32	切り出した32ビットデータ

このモジュールは、簡易版の 8 ビットセレクタを拡張したもので、一時メモリから読み出した 32 ビットデータと SRAM から読み出した 32 ビットデータを連結させた 64 ビットから所望の 32 ビットを選択する。簡易版での tail の役割は xpos の下位 5 ビットで実現する。

（6）マッチング処理ユニット matching_unit

表 21 にマッチング処理ユニット matching_unit の入出力ポートの説明を示す。青色で示した部分は簡易版に対して新しく追加したものである。

簡易版では対象画像メモリから読み出したデータ memd と一時メモリから読み出したデータ tmpd を取り込み、内部で 8 ビットセレクタにより所望の 8 ビットデータを選択していたが、高速版ではワードセレクタとして後述のブロックセレクタ内部に組み込んだため、入力となる対象画像のデータは ob_data ただ一つである。ob_data が有効となるのは dvalid が 1 の時のみで、それ以外の時はマッチング処理を止めている。また、TMorOB が 0 の時は、テンプレート画像の読み出し段階のため、この時もマッチング処理を行わない。

新たな出力信号として dscrd と match を追加したが、これは該当ブロックの相違度の計算を終えてマッチしたかどうかを示す信号である。上位モジュール quick_tm は、dscrd がアクティブになる（マッチしなかった場合）か、match がアクティブになり（マッチした場合）、後述する結果書き戻しユニット rtn_mrslt がマッチング座標を SRAM に書き戻し終えたことを示す信号がアクティブになるまで、タイミングジェネレータ timing_gen の入力信号 active

を非アクティブのままにする。これは、マッチした場合の SRAM へのマッチング座標の書き込みと、次のブロックの読み出しが競合するのを避けるためである。

また、高速版では、相違度計算のために新しいモジュールとして相違度計算ユニット calc_diff を作成した。VHDL 記述量は 200 行程度となった。

表 21：高速版マッチング処理ユニット matching_unit の入出力ポート

種別	ポート名	幅	内容
入力	clk	1	クロック信号
入力	#resb	1	リセット信号
入力	inxpos	10	当該ブロックの左上のX座標
入力	inypos	10	当該ブロックの左上のY座標
入力	ob_data	32	対象画像のデータ
入力	tm_data	32	テンプレートのデータ
入力	clr	1	積算レジスタのクリア（idxが0の時1、それ以外の時0）
入力	cmp	1	相違度の比較（idxが31の時1、それ以外の時0）
入力	dvalid	1	1の時、ob_dataは有効値
入力	TMorOB	1	1の時のみマッチング処理を行う
入力	border	1	最大許容相違度
入力	st	1	テンプレートマッチング開始信号
出力	outxpos	10	マッチング部分のX座標
出力	outypos	10	マッチング部分のY座標
出力	dscrd	1	該当ブロックの相違度計算が終了し、マッチしなかった時、1を出力
出力	match	1	該当ブロックの相違度計算が終了し、マッチした時、1を出力

（7）結果アキュムレータ result_accum

表 22：高速版結果アキュムレータ result_accum の入出力ポート

種別	ポート名	幅	内容
入力	clk	1	クロック信号
入力	#resb	1	リセット信号
入力	clr	1	積算レジスタのクリア（1の時diffの値を内部のアキュムレータに代入、0の時内部アキュムレータにdiffの値を積算）
入力	diff	6	ライン毎の相違度
入力	valid_i	1	1の時だけ、積算を行う
入力	st	1	テンプレートマッチング開始信号
出力	diff_ac	11	積算された相違度
出力	valid_o	1	後続のマッチング判定ユニットへ送るvalid信号（valid_iを1クロック遅らせて出力する）

表 22 に結果アキュムレータ result_accum の入出力ポートの説明を示す。青色で示した部分は簡易版に対して新しく追加したものである。簡易版に対して加えた変更は、valid_i が 1 の時だけ積算を行うようにしたのみである。VHDL 記述量は 40 行程度であった。

（8）マッチング判定ユニット judge_matching

表 23 にマッチング判定ユニット judge_matching の入出力ポートの説明を示す。青色で示した部分は簡易版に対して新しく追加したものである。

表 23：マッチング判定ユニット judge_matching の入出力ポート

種別	ポート名	幅	内容
入力	clk	1	クロック信号
入力	#resb	1	リセット信号
入力	cmp	1	相違度の比較（1の時、diff_acと最大許容相違度とを比較する、0の時、比較はしない）
入力	clr	1	新しいブロックの相違度判定に入っていることを示す信号
入力	inxpos	10	該当ブロックのX座標
入力	inypos	10	該当ブロックのY座標
入力	diff_ac	11	積算された相違度
入力	TMorOB	1	1の時だけマッチング判定を行う
入力	border	10	最大許容相違度
入力	st	1	テンプレートマッチング開始信号
入力	dvalid	1	1の時だけマッチング判定を行う
出力	outxpos	10	マッチしたブロックのX座標
出力	outypos	10	マッチしたブロックのY座標
出力	dscrd	1	該当ブロックの相違度計算が終了し、マッチしなかった時、1を出力
出力	match	1	該当ブロックの相違度計算が終了し、マッチした時、1を出力

このモジュールは、簡易版の最小相違度保持ユニットに相当するものである。高速版においては、相違度が最小となるマッチング部分を求めるのではなく、最大許容相違度より小さい相違度のブロックをすべてマッチング候補として求める方式に変更したため、このようにモジュール名を変えた。VHDL 記述量は 200 行程度となった。

4.3. 追加したモジュール

（1）ブロックセレクタ block_selector

表 24 にブロックセレクタ block_selector の入出力ポートの説明を示す。

表 24：ブロックセレクタ block_selector の入出力ポート

種別	ポート名	幅	内容
入力	clk	1	クロック信号
入力	#resb	1	リセット信号
入力	idx	5	ブロック内のラインインデックス
入力	xpos_in	10	当該ブロックの左上のX座標（入力）
入力	ypos_in	10	当該ブロックの左上のY座標（入力）
入力	memd	32	SRAMから読み出したデータ
入力	tmpd	32	一時メモリから読み出したデータ
入力	valid_in	1	1の時、memdは有効値
入力	TMorOB_in	1	1の時だけこのモジュールを動作させる
入力	clr_in	1	clr信号の入力
入力	cmp_in	1	cmp信号の入力
入力	st	1	テンプレートマッチング開始信号
出力	nxt	1	1の時、現在のブロックを捨て、次のブロックに移る
出力	xinc	6	X座標の増分値
出力	yinc	6	Y座標の増分値
出力	data	32	対象画像のデータ
出力	valid_out	1	後続のマッチング処理ユニットへ送るvalid信号（valid_inを2クロック遅らせて出力させる）
出力	TMorOB_out	1	後続のマッチング処理ユニットへ送るTMorOB信号（TMorOB_inを2クロック遅らせて出力させる）
出力	clr_out	1	後続のマッチング処理ユニットへ送るclr信号（clr_inを2クロック遅らせて出力させる）
出力	cmp_out	1	後続のマッチング処理ユニットへ送るvalid信号（cmp_inを2クロック遅らせて出力させる）
出力	xpos_out	10	当該ブロックの左上のX座標（出力）
出力	ypos_out	10	当該ブロックの左上のY座標（出力）

このモジュールは第 2 章で述べた高速版のアルゴリズムを実現しており、ワードセレクタをコンポーネントに持つ。VHDL 記述量は 180 行程度であった。

（2）相違度計算ユニット calc_diff

表 25 に相違度計算ユニット calc_diff の入出力ポートの説明を示す。このモジュールはライン毎の相違度を計算し、後続の結果アキュムレータへ渡す。簡易版では 1 クロックで 1 ラインの相違度の計算を行っていたが、複雑であったので処理を 2 クロックに分割し、モジュール化した。VHDL 記述量は 200 行程度であった。

表 25：相違度計算ユニット calc_diff の入出力ポート

種別	ポート名	幅	内容
入力	clk	1	クロック信号
入力	#resb	1	リセット信号
入力	xor_in	32	対象画像データとテンプレートデータの排他的論理和
入力	valid_i	1	1の時、xor_inが有効
入力	st	1	テンプレートマッチング開始信号
出力	diff	6	ライン毎の相違度
出力	valid_o	1	後続の結果アキュムレータへ送るvalid信号（valid_iを2クロック遅らせて出力する）

（3）結果書き戻しユニット rtn_mrslt

表 26 に結果書き戻しユニット rtn_mrslt の入出力ポートの説明を示す。

表 26：結果書き戻しユニット rtn_mrslt の入出力ポート

種別	ポート名	幅	内容
入力	clk	1	クロック信号
入力	#resb	1	リセット信号
入力	match_x	10	マッチングX座標
入力	match_y	10	マッチングY座標
入力	wvalid	1	書き込み有効信号
入力	fin_int	1	テンプレートマッチング終了割り込み信号
入力	#w_req	1	この信号がアクティブになるまで#w_enをアクティブにしておかなければならない
出力	data	32	書き込みデータ
出力	addr	20	書き込みアドレス
出力	#w_en	1	SRAMライトイネーブル信号
出力	w_fin	1	SRAM書き込み終了信号

このモジュールは、該当ブロックの相違度の計算が終了した後、マッチしているならばそのブロックの座標を SRAM に書き戻す。wvalid が 1 の時だけ動作し、match_x,match_y の順番に書き込む。fin_int が 1 の時は、テンプレートマッチングが終了したので、最後に対象画像中に幾つマッチング部分があったかを書き込む。それぞれの書き込みが完了すると w_fin を 1 にし、次のブロックの読み出しを開始させることによって、SRAM に対する読み出しと書き込みの競合を回避する。

5. FPGA ボード上での検証と性能評価

5.1. 画像処理ボード TSUNAMI

今回、テンプレートマッチングを実装したターゲットボードは、SBS Technology 社の画像処理ボード TSUNAMI である。このボードの特長は以下の通りである[10]。

・Altera Stratix を最大 5 個まで搭載可能
・ホスト CPU に負荷をかけることなく、テラフロップスレベルの処理能力を所有
・外部入出力：3G バイト／秒
・FPGA 間の内部バス：2G バイト／秒
・ボード間通信用 I/O：1.2G バイト／秒
・Wave FPGA ソフトウェア開発ツール

また、使用した TSUNAMI ボードの型番は TS-PCI-2501 で、Stratix EP1S25 FPGA, 1MB SRAM, 256M SDRAM を搭載しており、今回は、この SRAM に画像データを用意する。図 23 に TSUNAMI ボードの構成を示す。

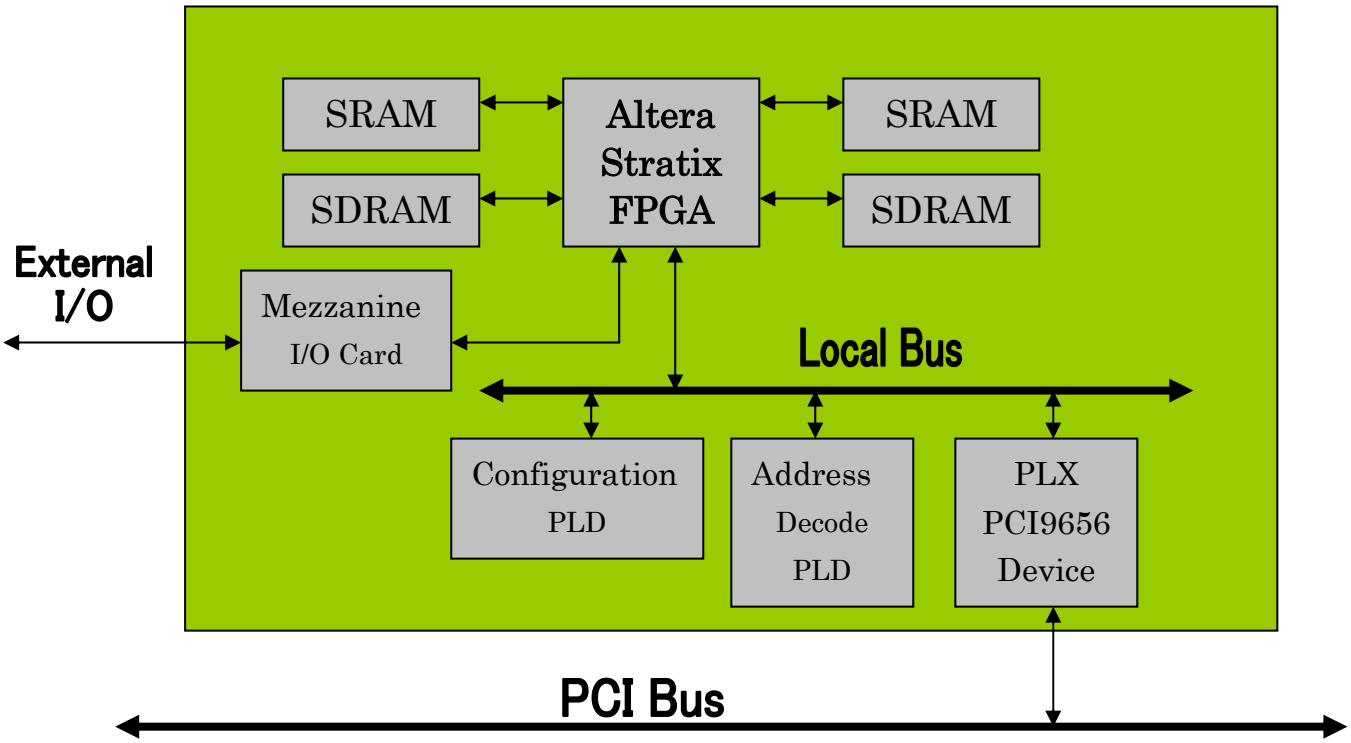

図 23：TSUNAMI ボードの構成

5.2. 検証方法

（1）回路構成情報の生成

作成した高速テンプレートマッチングの論理合成、及びフィッティング（FPGA 上の資源の割り当て）は Altera 社の統合開発環境ツール QuartusII を用いて行った。フィッティングが完了すると回路構成情報を含む RBF ファイルを出力する。図 24 に QuartusII でのコンパイル（論理合成からフィッティングまでの、一連の過程）の様子を示す。

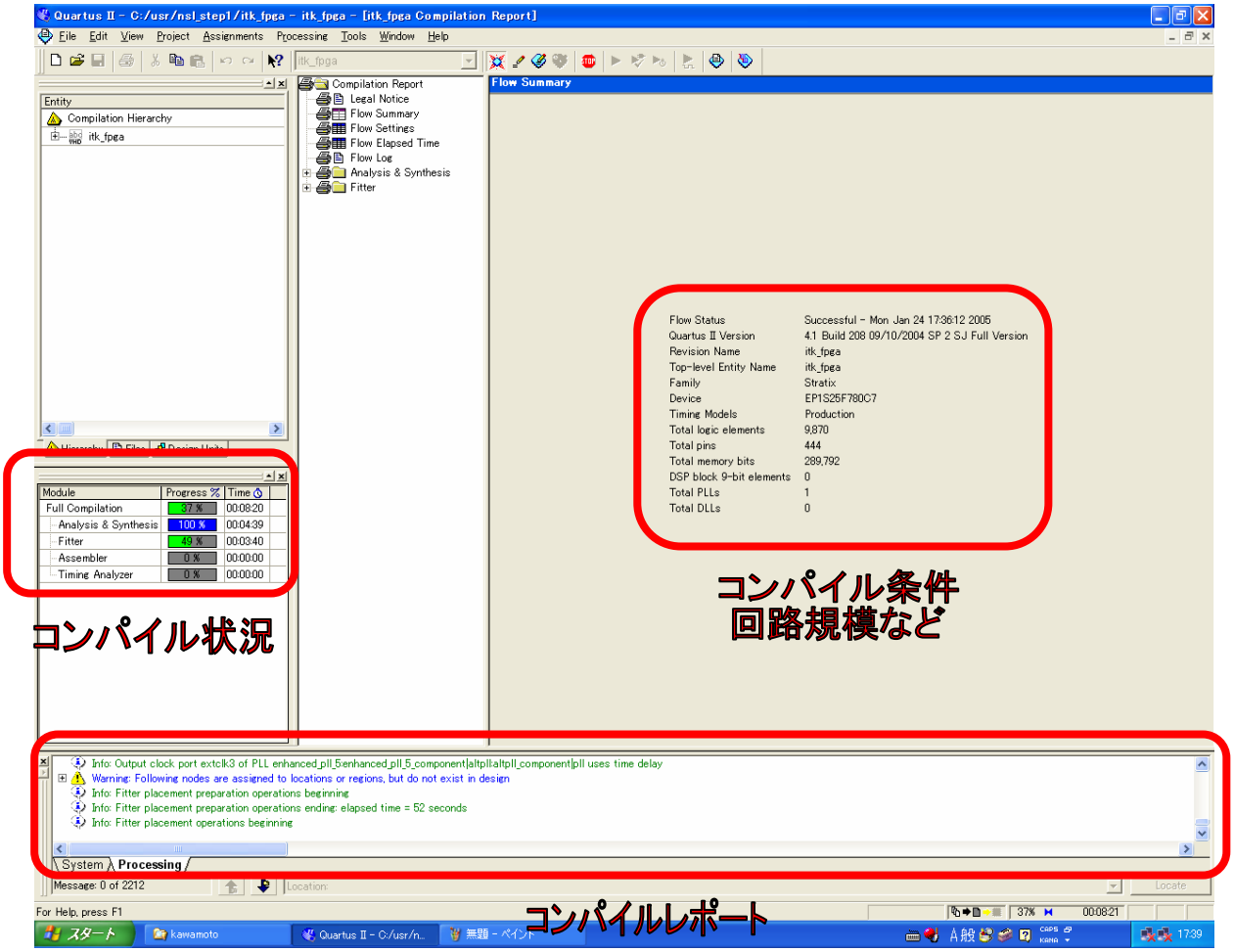

図 24：QuartusII を用いたコンパイルの様子

（2）FPGA へのダウンロードとテンプレートマッチングの開始

QuatusII により回路構成情報の RBF ファイルを出力すると、WaveShellAPI という TSUNAMI 専用の API を用いてこの回路構成情報ファイルを FPGA 上へダウンロードする。

続けて、FPGA 上に設けた専用のレジスタに、対象画像・テンプレート画像のサイズや SRAM 上での先頭アドレス、最大許容相違度などの設定情報を書きこむ。図 25 に設定情報を書きこむレジスタのシステム空間における配置を、表 27 にレジスタの内容を示す。

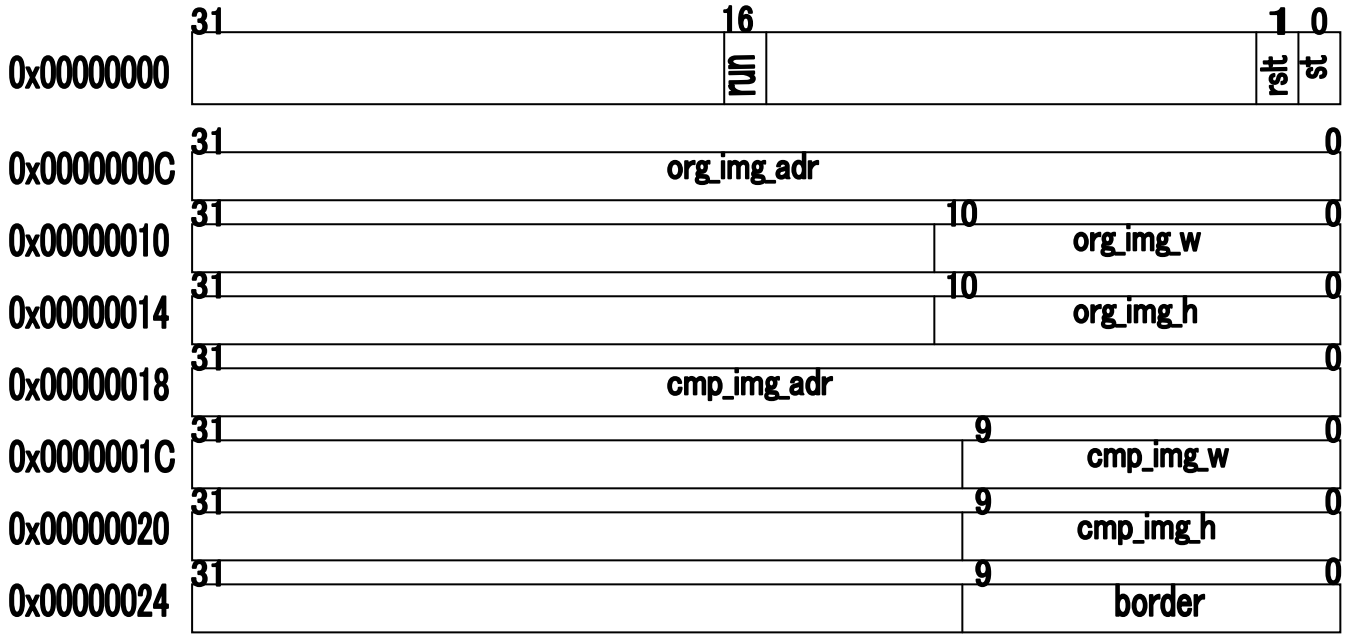

図 25：レジスタの配置

表 27：レジスタの内容

種別	設定情報名	幅	内容
設定情報	st	1	テンプレートマッチング開始信号
結果	rslt	1	1つでもマッチング候補が存在した場合に1を出力
結果	run	1	テンプレートマッチング実行中に1を出力
設定情報	org_img_adr	32	SRAM上の対象画像の先頭アドレス
設定情報	org_img_w	11	対象画像の横幅（画素数）
設定情報	org_img_h	11	対象画像の縦幅（画素数）
設定情報	cmp_img_adr	32	SRAM上のテンプレート画像の先頭アドレス
設定情報	cmp_img_w	10	テンプレート画像の横幅
設定情報	cmp_img_h	10	テンプレート画像の縦幅
設定情報	border	10	最大許容相違度

設定情報の書き込みに続いて DMA 転送により対象画像・テンプレート画像の SRAM への書き込みを行う。これでテンプレートマッチング開始に必要な準備が整ったので、図 25 のレジスタの st に 1 を書き込み、テンプレートマッチングを開始させる。

（3）結果の確認

テンプレートマッチングが終了すると API 側は終了割り込み信号を受け、終了を確認する。続いて、TSUNAMI 上の SRAM からマッチング座標を DMA 転送により読み出し、図 26 に示すように、コマンドプロンプト上に、実行クロック数、マッチング部分が存在したかどうかの情報、マッチング部分が存在した場合のマッチング X 座標・Y 座標を出力する。ここで

は、対象画像中のマッチング座標が(128, 256)である画像を用いて検証しており、正しく実行できていることが確認できる。

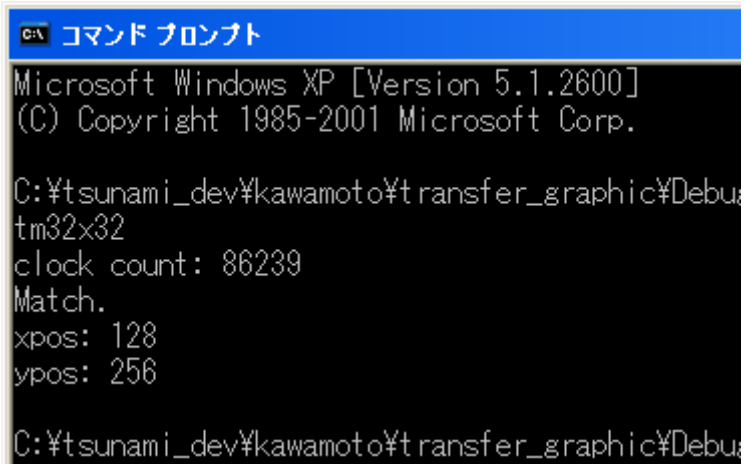

図 26：高速テンプレートマッチングの実行結果の確認

5.3. 検証結果と性能評価

第 4 章で設計した、マッチング候補を複数検出する高速テンプレートマッチングを TSUNAMI TS-PCI-2501 ボードの FPGA・Stratix EP1S25 に実装したところ、正常な動作を確認した。表 28 に、4 つの実行条件においてマッチング候補の数を以下のように変化させて実行した時の実行クロック数を示す。

・マッチング候補数 1：(111, 111)
・マッチング候補数 2：(111, 111), (222, 222)
・マッチング候補数 3：(111, 111), (222, 222), (333, 333)
・マッチング候補数 4：(111, 111), (222, 222), (333, 333), (444, 444)
・マッチング候補数 5：(111, 111), (222, 222), (333, 333), (444, 444), (555, 555)
・マッチング候補数 6：(111, 111), (222, 222), (333, 333), (444, 444), (555, 555), (666, 666)

表 28：テンプレートマッチングの実行クロック数

実行条件 / マッチング候補の数	簡易版 Cプログラム	高速版 Cプログラ	高速版VHDL (シミュレーショ	高速版 (TSUNAMI)
1	39480000	167132	34898	270246
2	39480000	277810	44042	464103
3	39480000	384440	68354	685894
4	39480000	489132	77498	868199
5	39480000	576184	89898	1078680
6	39480000	676816	99042	1249433

簡易版においては、常に 1 画素ずつずらしながらブロックを切り出すため、マッチング候補数に関わらず実行クロック数は一定である。アルゴリズムの比較では、マッチング候補数が 1 つの場合は､簡易版 C プログラムに対して高速版 C プログラムが約 236 分の 1 の実行クロック数となっている。

また、高速版 VHDL の実行にあたっては、C プログラム実行環境（本研究室 PC）のメモリアクセスレイテンシの条件となるべく揃えるために、TSUNAMI 上だけではなく、簡易版で用いたような、1 クロックでメモリアクセスが可能な ROM 形式のモジュールを用いたシミュレーションによる検証も行った。そのシミュレーションによる高速版 VHDL と高速版 C プログラムを比較すると、マッチング候補数が 6 つの場合に、VHDL が C プログラムに比べて約 6.8 分の 1 の実行クロック数となった。

しかし、高速版 VHDL を実際に TSUNAMI 上で実行させたところ、シミュレーションと比較すると、ほとんどの場合において実行クロック数が 10 倍以上となり、高速版 C プログラムと比較しても 2 倍近く多く、期待した性能は得られなかった。これは、TSUNAMI 上の SRAM に対する 1 回のリードアクセスに 10 クロック近く掛かってしまうことが原因であると考えられる。

このように、メモリアクセスがボトルネックとなっているため、実行クロック数を抑えるためにはメモリアクセス回数をなるべく少なくすることが重要である。ただ、該当ブロックのマッチング処理を行うかどうかの判断に要するメモリアクセスを 2 回に抑えるなど、無駄なメモリアクセスは極力行わないようにしているため、今後大幅な実行クロック数の縮小はかなり難しいと考えられる。

6. おわりに

本論文では、株式会社ナノスコープとの産学協同研究の一環として、ハードウェア記述言語による、ガラス外観検査装置開発のための、テンプレートマッチングの設計について述べた。一般的な定義に忠実な手法である簡易テンプレートマッチングと、十字型のテンプレートに特化した手法の高速テンプレートマッチングの 2 種類を設計し、高速テンプレートマッチングのみを画像処理ボード TSUNAMI 上の FPGA に実装し、正常な動作を確認した。性能評価にあたっては、FPGA 上のシステムと C プログラムを処理する汎用 CPU の動作周波数にはかなりの差があるため、設計したテンプレートマッチングがいかに実行クロック数を少なく抑えられているかに着目した。高速版 C プログラムに対する高速版 VHDL の、実行クロック数の比較では、TSUNAMI 上の SRAM を用いた場合、メモリアクセスがボトルネックとなり、高速版 C プログラムの 2 倍の実行クロック数が掛かってしまい、全体としては期待した性能向上は得られなかった。しかし、1 クロックでアクセス可能なメモリを用いたシミュレーションでは、高速版に対して約 10 分の 1 の実行クロック数に抑えることが出来、性能向上が得られた。今後は画像処理ボードの性能も上がり、メモリアクセスレイテンシの問題も解消されると考えられる。

今後の課題は、テンプレートマッチングに関しては、様々な画像サイズに対応できるように拡張すること、傾いた対象画像に対しても正確にマッチング部分を検出できるように、アフィン変換の機能を追加することである。また、産学協同プロジェクト全体としては、第 2 章の冒頭で述べたようなテンプレートマッチング以外の画像処理についても FPGA 上に実装することである。

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、古川達久氏、藤原淳平氏、同じハードウェアグループの梅原直人氏、中谷嵩之氏、難波翔一朗氏、船附誠弘氏、的場督永氏、及び様々な面で貴重な助言や励ましを下さった研究室の皆様に深く感謝いたします。

同じく、共同研究において、研究全体にわたって貴重な助言、ご指導を頂きました株式会社ナノスコープの平岡邦廣氏、西田洋隆氏、岡元英樹氏、また、VHDL によるハードウェア設計と FPGA への実装に関して多くのアドバイスを頂いた株式会社ジーニックの久野啓祐氏、笠井久史氏、及び研究を行う上でサポートして下さったプロジェクトのメンバーの皆様に深く感謝いたします。

参考文献

[1]立命館大学　VLSI センター：社会人向け VLSI 設計セミナー、2001
[2]Z・ナバビ著、佐藤一幸訳：VHDL の基礎：国際標準ハードウェア記述言語によるディジタル・システム設計、日経 BP 出版センター、1994
[3]長谷川裕恭：VHDL によるハードウェア設計入門：言語入力によるロジック回路設計手法を身につけよう、CQ 出版、1995
[4]池田修久:ハードウェア記述言語による単一サイクル／パイプラインマイクロプロセッサの設計、立命館大学理工学部卒業論文、2002
[5]大八木睦:ハードウェア記述言語によるマルチサイクル／パイプラインマイクロプロセッサの設計、立命館大学理工学部卒業論文、2002
[6]中村央志：ハードウェア記述言語による教育用マイクロプロセッサの設計（I)、立命館大学理工学部卒業論文、2003
[7]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卒業論文、2003
[8]藤原淳平：ハードウェア記述言語による教育用マイクロプロセッサの設計（II)、立命館大学理工学部卒業論文、2003
[9]田村秀行：コンピュータ画像処理、オーム社，2003
[10]株式会社ソリトンシステムズ：TSUNAMI 簡易仕様書
[11]天野英晴:マルチコンテキストデバイスを用いた動的適応型ハードウェアの提案、情報処理学会研究報告、2002-ARC-150, Vol.2002, No.112, pp.59-64, 2002
[12]北岡稔朗、天野英晴、安生健一朗：DRP 上での仮想ハードウェア実現のための構成情報の圧縮､第１回リコンフィギャラブルシステム研究会論文集, pp.213-218, 2003 年 3 月
[13]児玉祐悦、片下敏宏、佐谷野健二：リコンフィギュアラブルシステム REX への並列計算機 EM-X の実装、情報処理学会論文誌、Vol.2003 No.027, pp.109-114, 2003 年 3 月
[14]出口勝昭、山田裕、天野英晴：DRP でのウェーブレットフィルタの実装、電子情報通信学会技術研究報告、VLD2002-142, Vol.142, No.609, pp.73-78, 2003 年 1 月
[15]谷川一哉、吉田哲生、児島彰、弘中哲夫、吉田典可：PARS アーキテクチャの詳細設計に関する一考察、情報処理学会研究報告、Vol.2001-ARC-144, pp.31-36, 2001 年 1 月
[16]天野英晴、犬尾武、紙弘和：DRP 上での仮想ハードウェア機構の検討、電子情報通信学会技術研究報告 VLD2003-122, Vol.103, No.578, pp.47-52, 2004 年 1 月