修 士 論 文

FPGAを用いた画像検索回路の設計

北陸先端科学技術大学院大学
情報科学研究科情報処理学専攻

高道 悦子

2003年3月

修 士 論 文

FPGAを用いた画像検索回路の設計

指導教官　中野 浩嗣 助教授

審査委員主査　中野 浩嗣 助教授
審査委員　浅野 哲夫 教授
審査委員　金子 峰雄 教授

北陸先端科学技術大学院大学
情報科学研究科情報処理学専攻

110072 高道 悦子

提出年月: 2003 年 2 月

概 要

本論文では, FPGA を用いた高速画像検索システムを提案する. このシステムは, 質問となるテンプレート画像 T が与えられたときに, 大量のデータベース画像 $I_1, I_2, \ldots$ の中から, T に類似する部分画像を含む画像を全て列挙するものである.

システムは, 本研究で開発したハードウェアジェネレータと FPGA ベンダが提供する各種ツールから構成される. まず, ハードウェアジェネレータが, 与えられたテンプレート画像 T に対し, 入力画像 I_i が T に類似する部分画像を持つかどうか判定するための回路記述を Verilog HDL ファイルで出力する. 次に, FPGA ベンダが提供するデザインツールによって, Verilog HDL ファイルから回路を合成し, FPGA にダウンロードする. このようにして FPGA に構築される画像検索回路は, 生成後にテンプレート T を変更することができない, インスタンスに特化したものであるが, インスタンスを固定することで計算量や計算時間を大幅に削減できる.

また我々は PCI 接続の Xilinx 社の FPGA とタイミング解析ツールを用い, 画像検索回路の性能評価を行った. その結果, 本手法は従来のソフトウェアを用いて検索する手法に比べ, 最大で 3000 倍の高速化に達した.

目 次

第1章　はじめに

近年のコンピュータの発達やインターネットの普及によってさまざまなマルチメディア情報を送受信したり, 大量のデータを蓄えることが可能になった. それと同時に, 身の周りの情報だけでなく, 文化財情報や図書館の電子化も着実に進んでいる. これに伴い, 溢れんばかりの情報の中から必要な情報を効率よく検索する技術が重要になってきた.

本論文では, このような背景をふまえ, 大量の画像データベースから質問画像に類似する画像を検索する問題を扱う. この問題は, 物体識別や目標物の数え上げなど, 様々な分野で応用される有用な問題である.

このような大量データの処理を効率よく行うためには, ハードウェアアクセラレータを用いた処理の高速化が非常に有効である. 特に, FPGA(Field Programmable Gate Array) などで知られる回路データが書き換え可能な LSI は, ハードウェア開発時間の短縮や低コスト化に大いに役立っている.

そこで本研究では, 大量の画像データベースから質問画像に類似する画像を効率よく検索するために, FPGA 上に画像検索回路を設計する手法を提案する.

1.1　FPGA とは

FPGA(Filed Programmable Gate Array) は, ユーザが設計した回路を瞬時に構築することができる, 回路データが書き換え可能な VLSI である. FPGA は主な構成要素として, 書き換え可能なロジックセル, メモリブロック, 再編成可能な配線などを持っている. ロジックセルは, 通例 4 つの入力を持つ論理関数と, いくつかの Flip Flop から成っている. メモリブロックは, 通例 dual-port RAM と呼ばれる, 同時に 2 つの異なるアドレスのデータに読み書きすることができる RAM である.

FPGA に回路を設計する流れを図 1.1 に示す. まず最初に, ユーザは回路の仕様 (入出力や動作など) を HDL(Hardware Description Lauguage) と呼ばれる言語によって記述する. HDL には, 主に Verilog HDL や VHDL などがあり, 本論文では Verilog HDL を使用している.

次に, FGPA ベンダが提供するデザインツールによって回路を合成する. 回路の合成には, 大きく分けて二つの段階がある.

一つは, HDL コードから論理回路への変換である. ここでは, ユーザによって記述された HDL コードの文法チェックを行い, AND ゲートや OR ゲートなどの組み合わせ回路や Flip Flop などの順序回路へ変換する.

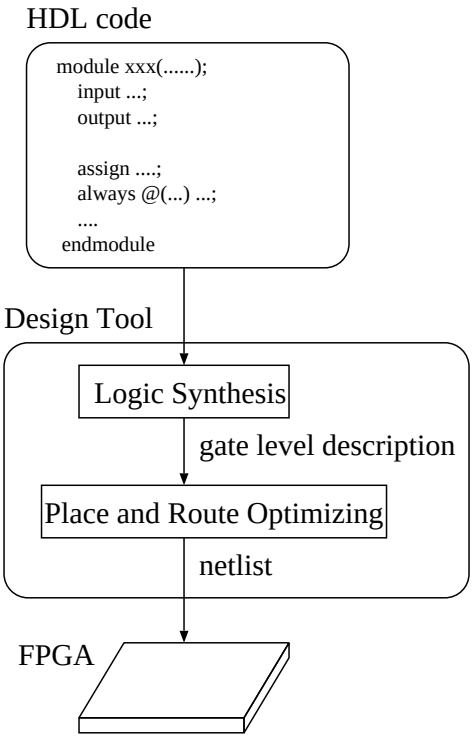


図 1.1: 回路設計の流れ

もう一つは, 論理回路から個々の FPGA に対応するネットリストへの変換である. ここでは, 一つめの段階で得られた論理回路を回路面積や回路遅延について最適化する. そして, 個々の FPGA に合わせてネットリストを出力する.

ユーザは, 各段階で回路動作のシミュレーションや回路面積や回路遅延などが満たされているかをタイミング解析ツールで行い, もし仕様が満たされていなければ前の段階に戻って修正する作業を繰り返す. そして, 全ての回路の仕様が満たされれば, FPGA にネットリストをダウンロードし, 回路を構築する.

本論文では, ユーザが HDL ソースを記述する部分を自動化している. 検索したい画像を入力すると, その画像を検索するための回路の Verilog HDL コードが自動生成される. これによって, ユーザの負担および回路の開発時間を大きく減らすことができる.

1.2 部分計算について

本論文の FPGA を用いた高速計算は, 部分計算 [2, 3] の概念に基づいている. ここで部分計算について次の関数 $f(x, y) = x^3 + x^2y + y$ を例に簡単に説明する. この関数が $x = 2$ について繰り返し何度も評価される場合を想定しよう. このとき, $x = 2$ で固定された次の関数 $f_2(y) = 8 + 5y$ を導入する. そして, $f(2, y)$ が評価されるかわりに $f_2(y)$ を評価す

るのである. 実際, 2 つの整数の乗算について, その一方が固定されている場合は効率よく計算できることが知られている [5]. このように, 関数 $f(x, y)$ をある x で固定して関数 f_x とする最適化を部分計算と呼ぶ.

従来は, この最適化された関数 $f_x(y)$ をソフトウェア上で計算していた [2, 3]. これに対し本論文では, $f_x(y)$ を計算するためのハードウェアを設計し, さらなる高速化を図る. 本研究では, 次のような特性の関数 $f(x, y)$ を含む問題に対し, FPGA を使用するインスタンスに特化した手法を提案する.

- どのように x の値を固定するかは問題のインスタンスに依存する
- 様々な y の値に対し, $f(x, y)$ を繰り返し評価する

本論文で提案する FPGA を使用するインスタンスに特化した手法は, 関数 f_x を計算するためのハードウェアを用いて $f_x(y)$ を評価する. 扱う問題が先ほど述べた性質を満たすとき, 提案手法は効率よく計算することができる.

例として, ある文脈自由文法 G が与えられたとき, $f_G(w)(= f(G, w))$ を, 文法 G が語 w を生成するか決定する関数であるとする. この関数 $f_G(w)$ について, FPGA を使用するインスタンスに特化した方法は, 従来通りにソフトウェアで計算する方法よりもはるかに高速に計算する [1].

本論文で扱う画像検索問題についてこの部分計算を適応すると, このようになる. 大容量の画像ベータベースから, テンプレート画像 T に類似する画像を検索するという問題は, テンプレート画像 T を固定することによって効率よく検索することができる. より具体的には, 第 2 章で導入する T と画像 I との相違度 (距離) を表す関数 $D(T, I)$ の評価のかわりに, T を固定した関数 $D_T(I)$ 導入し, $D_T(I)$ を計算するために設計されたハードウェアを用いることによって高速化を図る.

1.3 類似研究との比較

FPGA を用いた画像検索回路は他にも提案されているが [4], インスタンスに特化したものではなく, 扱えるのは白黒 2 値画像である. 本論文で提案するものは, それらを改良し, インスタンスに特化させており, さらに白黒濃淡画像も扱うことができる.

第2章　相違度関数

ここでは, テンプレート画像 T と画像 I との相違度 (距離) 関数を定義する. テンプレート画像 T は, $m \times m$ 画素の L 階調白黒濃淡画像である. 但し T はドントケアを含むことができ, T の位置 (i, j) の画素 $T_{i,j}$ がドントケアであることを $T_{i,j} = d$ と記す. また, ドントケアでない画素を有効画素と呼び, 有効画素の数を e で表す.

ここで, T と $m \times m$ 画素 L 階調白黒濃淡画像 I との相違度関数 $D(T, I)$ を次式で定義する.

$$D(T, I) = \sum_{T_{i,j} \neq d} |T_{i,j} - I_{i,j}| \tag{2.1}$$

特に $L = 2$ の場合, 即ち T が白黒 2 値画像のときは,

$$D(T, I) = \sum_{T_{i,j} \neq d} T_{i,j} \oplus I_{i,j} \tag{2.2}$$

となる. ここで $\oplus$ は排他的論理和を表す.

画像 I が $n \times n$ $(n > m)$ 画素の場合は, I の全ての $m \times m$ 画素部分画像との相違度を求め, その最小値とする. 即ち,

$$D(T, I) = \min_{1 \leq x, y \leq n-m+1} D(T, I[x, y]). \tag{2.3}$$

ここで, $I[x, y]$ を画像 I の位置 (x, y) で始まる $m \times m$ 画素の部分画像とする.

$$I[x, y] = \{I_{i', j'} | x \leq i' \leq x + m - 1, y \leq j' \leq y + m - 1\} \tag{2.4}$$

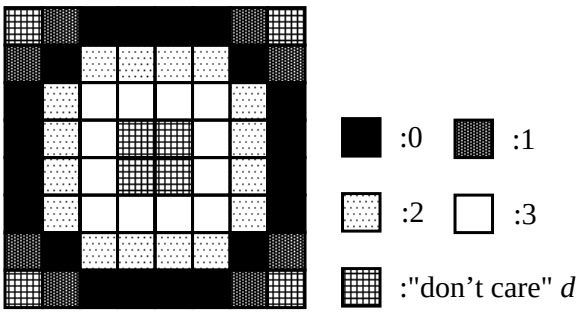


図 2.1: テンプレート画像の例

もし, I が T に類似する部分画像を持つならば, $D(T, I)$ が小さな値をとることは明らかである.

データベースの全画像 I_k $(k = 1, 2, \cdots)$ について相違度 $D(T, I_k)$ を求め, 設定された閾値以下であるものを列挙することによって, T に類似する画像が得られる.

第3章　ソフトウェアによる相違度の計算

ここでは, 従来通りのソフトウェアによる方法でテンプレート画像 T と画像 I との相違度 $D(T, I)$ を計算する方法を述べる.

3.1　単純な方法

L 階調 $m \times m$ 画素のテンプレート画像 T と $n \times n$ $(n > m)$ 画素の画像 I との相違度 $D(T, I)$ を計算するプログラムは次のようになる.

```
d=-1; /*ドントケア*/

/*テンプレート T と画像 I との相違度 D_TI の計算*/
D_TI=INT_MAX; px=1; py=1;
for(x=1;x<=n+m-1;x++)
    for(y=1;y<=n+m-1;y++){
        d_ti=0;
        for(i=1;i<=m;i++)
            for(j=1;j<=m;j++)
                if(T[i][j]!=d) /* T_i,j がドントケアでない*/
                    d_ti+=(T[i][j]>I[x+i-1][y+j-1])?
                          (T[i][j]-I[x+i-1][y+j-1]):
                          (I[x+i-1][y+j-1]-T[i][j]);
        if(d_ti<D_ti){
            D_TI =d_ti; px=x; py=y;
        }
    }
```

プログラムでは, 画像 I にサイズ $m \times m$ の窓をかけて部分画像 $I[x, y]$ $(1 \leq x, y \leq n-m+1)$ を取り出し, T との相違度を計算している. 部分画像と T との相違度は, T の各画素 T_{ij} $(1 \leq i, j \leq m)$ について, $T_{i,j} \neq d$ ならば, 対応する I の画素 $I_{x+i-1,y+j-1}$ との差の絶対値を求め, それらを合計する. そして, 現在見ている部分画像と T との相違度が, 今までのものより小さい場合, T と I の相違度の値を更新している.

このプログラムの計算量は $O((n+m-1)^2m^2)$ である. 画像 I がテンプレート T に対し非常に大きなものである場合, 言い換えれば $n \gg m$ である場合, およそ $O(n^2m^2)$ となる.

3.2 部分計算を用いた相違度の計算

テンプレート T と画像 I の相違度 $D(T,I)$ を, より短い時間で計算するために部分計算の概念を導入する. 与えられたテンプレート T に対し, 大量のデータベース画像 $I_1, I_2, \ldots$ の各画像との相違度 $D(T,I_i)$ $(i=1,2,\ldots)$ を計算する場合を想定する. このような場合, T を固定したときの相違度 $D_T(I)$ を評価すると計算時間を抑えることができる.

最初に, 前処理としてテンプレート T の有効画素 (ドントケアでない画素) のインデックスを作成しておく. これにかかる計算時間は $O(m^2)$ である.

```
d=-1; /*ドントケア*/

/*T の有効画素のインデックスを作成*/
e=0; /*有効画素数*/
for(i=1;i<=m:i++)
    for(j=1;j<=m;j++)
        if(T[i][j]!=d){
            e++;
            ix[e]=i; iy[e]=j;
        }
```

ここで, 固定された T と画像 I との相違度 $D_T(I)$ を計算するプログラムは次のように書くことができる.

```
/*T を固定したときの画像 I との相違度 DT_I の計算*/
/*画像が複数あるときは各画像について繰り返す*/
DT_I=INT_MAX; px=1; py=1;
for(x=1;x<=n+m-1;x++)
    for(y=1;y<=n+m-1;y++){
        dt_i=0;
        for(i=1;i<=e;i++) /*有効画素について繰り返す*/
            dt_i+=(T[ix[i]][iy[i]]>I[x+ix[i]-1][y+iy[i]-1])?
                  (T[ix[i]][iy[i]]-I[x+ix[i]-1][y+iy[i]-1]):
                  (I[x+ix[i]-1][y+iy[i]-1]-T[ix[i]][iy[i]]);
        if(d_ti<D_ti){
            DT_I=dt_i; px=x; py=y;
        }
    }
```

このプログラムの計算量は $O((n+m-1)^2 e)$ である. $n \gg m$ の場合は, T を固定したときの相違度 $D_T(I)$ の計算量は $O(n^2 e)$ となり, e の値に比例して計算時間が増加する.

第4章　画像検索システムの全体図

本章では, 画像検索システムの概要を説明する. テンプレート画像 T が与えられたとき, データベース画像 $I_1, I_2, I_3, \ldots$ から T に類似する画像を検索する流れを図 4.1 に示す.

まず最初に, テンプレート画像 T とデータベース画像 I との相違度 $D_T(I)$ を高速に計算するため, FPGA 上にハード ウェア回路を設計する. 以降, この回路を相違度計算回路と呼ぶことにする.

1. 与えられたテンプレート画像 T に対し, 本研究で開発したハード ウェアジェネレータは, テンプレート画像を T に固定したときの入力画像 I との相違度 $D_T(I)(= D(T, I))$ を計算する回路記述を Verilog HDL ファイルで出力する. ここで部分計算の概念に基づき, テンプレート画像 T を固定している. そうすることによって計算量を減らし, 回路面積をより小さく, 回路遅延をより短くすることができる.

2. FPGA ベンダが提供するデザインツールにより Verilog HDL ソースから回路を合成し, 回路データを PCI 接続の FPGA にダウンロード する.

このようにして FPGA に構築された相違度計算回路は, PCI bus を経由してホスト PC とデータの送受信を行いながら動作する. 大量の画像データベースからテンプレート画像 T に類似する画像を検索する流れは次のようである.

1. ホスト PC は大量のデータベースを蓄えており, 画像データ I_k $(k = 1, 2, 3, \ldots)$ を FPGA に送信する. FPGA は画像データを受信しながらテンプレート T との相違度 $D_T(I_k)$ を計算し, ホスト PC に返す.

2. ホスト PC は FPGA から受信した相違度 $D_T(I_k)$ $(k = 1, 2, 3, \ldots)$ について, 設定された閾値以下である画像を全て列挙する. これによって T に類似した画像を得ることができる.

Verilog HDL ソースから回路を合成する時間は規模の大きな回路では数時間以上を必要とするが, このように大量のデータベースに対して同じ処理を繰り返し行う場合では, 各データに対する処理を高速に行うことができるため, 全体での計算時間を大幅に抑えることができる.

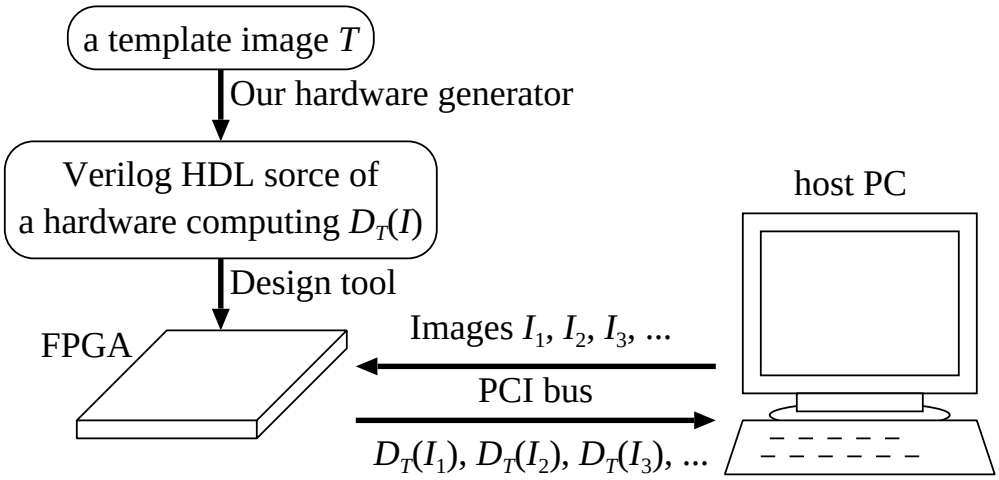


図 4.1: FPGA を用いた画像検索システム

第5章　相違度計算回路

ここでは, ハードウェアジェネレータによって設計される, 相違度計算回路について詳細を述べる.

設計される回路は完全同期式である. 全てのレジスタは, 共通するクロックが与えられており, クロックのタイミングで同時に動作する.

5.1　基本回路構成

相違度計算回路は, 図 5.1 のように基本要素として, 入力画像 I の部分画像を蓄えるためのシフトレジスタと, テンプレート画像 T と I の部分画像との相違度 $D_T(I[x,y])$ を計算するための組み合わせ回路を持つ.

図はテンプレート T の一辺が $m=4$ 画素, $L=4$ 階調の場合である. 回路の入力にはクロック毎に画像 I の縦 m 画素が PCI bus 経由で順次与えられ, シフトレジスタに送られる. 1 クロック目には, $I_{1,1}, I_{1,2}, \ldots, I_{1,m}$ が入力される. 同様に, 2 クロック目には $I_{2,1}, I_{2,2}, \ldots, I_{2,m}$ が, n クロック目には $I_{n,1}, I_{n,2}, \ldots, I_{n,m}$ が入力される. そして $n+1$ クロック目には $I_{1,2}, I_{1,3}, \ldots, I_{1,m+1}$, $n+2$ クロック目は $I_{2,2}, I_{2,3}, \ldots, I_{2,m+1}$ というように入力される.

シフトレジスタは画像 I の $m \times m$ 画素の部分画像を蓄えており, クロック毎に各レジスタの値を左に一つずつシフトすることによって部分画像を更新する.

テンプレート T と画像 I の部分画像との相違度を計算するための, $D_T(I[x,y])$ 計算回路には, シフトレジスタに蓄えられている部分画像のうち, 有効画素のみが入力として与えられる. そして各有効画素について, テンプレート画素との差の絶対値を計算し, それらを 2 分木構造に構成された $\log e$ 段の加算器で足し合わせる.

相違度計算回路の出力には現在までの入力から計算される $D_T(I[x,y])$ の最小値が現れ, クロック毎に更新される. 画像 I を全ての画素を入力し終えると, テンプレート T と画像 I との相違度 $D_T(I)$ が得られる.

5.2　並列化相違度計算回路

さらなる高速化のために, テンプレート T と画像 I の部分画像との相違度 $D_T(I[x,y])$ を k 並列に計算することを考える. これは図 5.2 のように, シフトレジスタを拡張し,

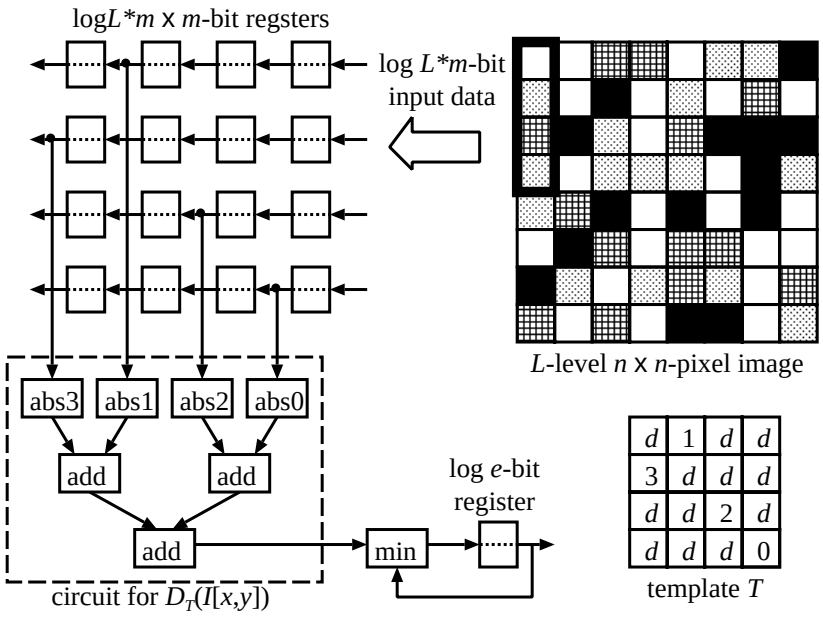


図 5.1: 相違度計算回路

$D_T(I[x,y])$ 計算回路を k 個に増やすことで実現できる. 回路の入力には, 1 クロック目には $I_{1,1}, I_{1,2}, \ldots, I_{1,m+k-1}$, 2 クロック目には $I_{2,1}, I_{2,2}, \ldots, I_{2,m+k-1}$ が与えられ, 同様にして n クロック目には $I_{n,1}, I_{n,2}, \ldots, I_{n,m+k-1}$ が与えられる. そして, $n+1$ クロック目には $I_{1,k+1}, I_{1,k+2}, \ldots, I_{1,m+2k-1}$, $n+2$ クロック目には $I_{2,k+1}, I_{2,k+2}, \ldots, I_{2,m+2k-1}$ というように入力される. 図は $k=4$ 並列の場合であり, 縦に 1 画素ずつずれた 4 つの部分画像について, 同時に相違度を計算することができる.

並列化相違度計算回路では, シフトレジスタは画像 I の $(m+k-1)\times m$ 画素分のデータを蓄える. テンプレート T および画像 I が L 階調の場合, 1 画素のデータを保持するのに, $\log L$ bit 必要なので, シフトレジスタのサイズは $(m+k-1)m\log L$ bit となる.

また, k 並列に計算する場合, k 個の部分画像と T との相違度の最小値 $\min_{0\le i<k} D_T(I[x, y+i])$ を求めるための $\log k$ 段の比較器が新たに加わる.

5.3 キャッシュ回路

回路に入力されるデータベース画像 I は, ホストマシンから PCI bus 経由で与えられる. このため, 回路の入力幅が PCI bus 幅 N_{PCI} より大きいときは, 図 5.3 のように入力したデータをキャッシュに蓄えておくための回路を前段に追加する.

図は $m=4, L=4, k=2$ で, PCI bus 幅が $N_{PCI}=4$ bit, 即ち $2(=N_{PCI}/\log L)$ 画素分の場合である. 後段に続くシフトレジスタの入力幅は $(m+k-1)=5$ 画素分である. PCI bus から供給される 2 画素分で足りない残りの 3 画素分をキャッシュで補う. キャッ

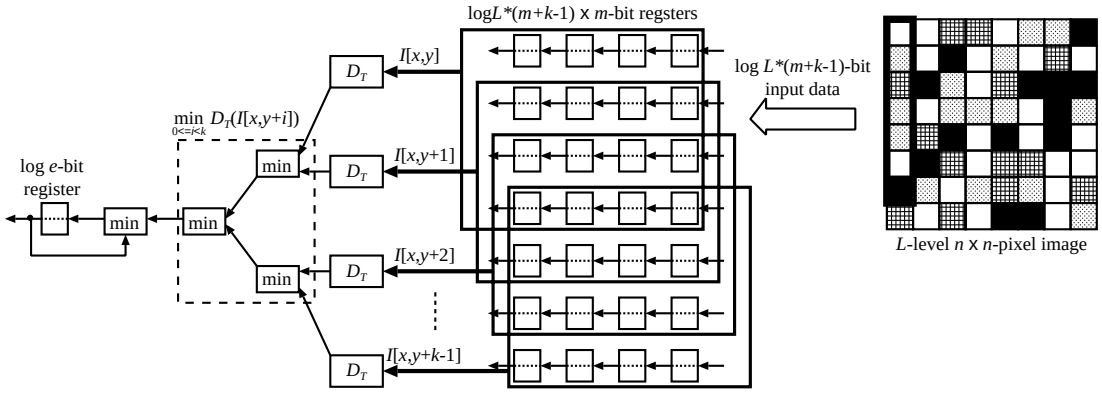


図 5.2: 並列化相違度計算回路

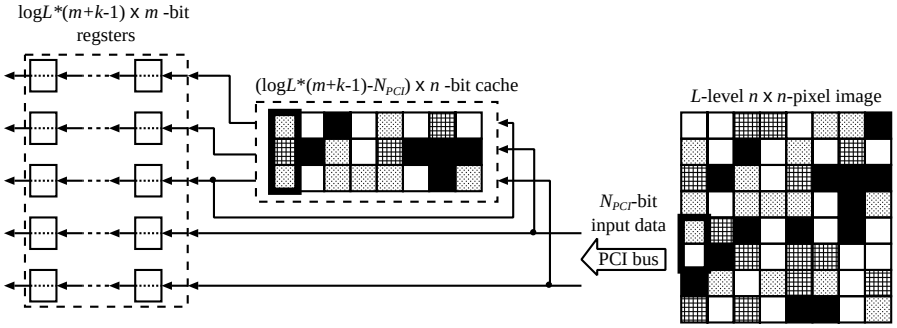


図 5.3: キャッシュ回路

シュは $3 \times n$ 画素分のデータを蓄えており, FIFO 方式で 3 画素分のデータを出力し, その一部と PCI bus から供給される 2 画素とで合わせて 3 画素となるように入力する.

5.4 相違度計算回路のコストと計算時間

相違度計算回路のコスト (ゲート数) と計算時間 (クロック数) について考える.

PCI bus 幅が N_{PCI} bit ならば, 1 クロック毎に $\lfloor \frac{N_{PCI}}{\log L} \rfloor$ 画素ずつ入力される. これより, 最大で $k = \lfloor \frac{N_{PCI}}{\log L} \rfloor$ 並列相違度計算回路にすることができる.

並列数 $k = \lfloor \frac{N_{PCI}}{\log L} \rfloor$ とするとき, k 並列相違度計算回路に必要なコストは次の通りである.

- シフトレジスタに $(m + k - 1)m \log L$ 個の Flip Flop
- k 個の $D_T(I[x, y])$ 計算回路に $O(ke \log L)$ 個のゲート
- k 個の比較器に $O(k \log L)$ 個のゲート

- キャッシュに $(m-1)n\log L$ bit の Block RAM

画像 I の相違度 $D_T(I)$ を計算するのに必要なサイクル数は, I を送信する分だけ必要であるから, 画像 I のデータサイズを PCI bus 幅 N_{PCI} で割ったものとして求めることができる.

$$\begin{aligned}(\text{サイクル数}) &= (\text{画像}\,I\,\text{の送信サイクル数}) \\ &= n^2\log L/N_{PCI} = n^2/k \end{aligned} \tag{5.1}$$

PCI busが十分な帯域幅を持っていると仮定するとき, 即ち PCI よりデータが滞りなく与えられる場合での, テンプレート T と入力画像 I との相違度 $D_T(I)$ の計算時間は, 回路の動作周波数より,

$$(\text{計算時間}) = (\text{サイクル数})/(\text{動作周波数}) \tag{5.2}$$

として求めることができる.

第6章　性能測定

ここでは, 相違度計算回路の性能評価を行うために, 回路の合成シミュレーションを行い, ソフトウェアによる方法と比較する.

6.1　実験方法

テンプレート画像 T は 32×32 画素 ($m = 32$), 階調数は $L = 2, 4, 16$ の 3 通り, 有効画素数を $e = 128, 256, 512, 768$ と変化させたときのランダムパターンとする. 各 L, e について, ランダムパターンは 10 パターンずつ用意する.

この実験では, PCI bus 幅が $N_{PCI} = 32$ bit であると想定する. 合成される回路は並列計算を行うもので, $L = 2$ のとき $k = 32$ 並列で, T と入力画像 I の部分画像との相違度 $D_T(I[x, y])$ を計算する. $L = 4, 16$ についても同様に, それぞれ $k = 16, 8$ 並列で $D_T(I[x, y])$ を計算する.

FPGA には 800 万ゲート相当である Xilinx 社 Virtex-II XC2V8000 を使用する.

デザインツールとして, Xilinx ISE ロジックデザインツールを使用し, 各テンプレートについて相違度計算回路を合成する. 各 L, e について, 回路の動作周波数, 使用したスライス数の平均を取る.

相違度の計算時間は, タイミング解析結果から得られる動作周波数をもとに, 式 (5.1),(5.2) より計算する. 入力画像 I のサイズが 1024×1024 画素 ($n = 1024$) とするときの, テンプレート T と画像 I との $D_T(I)$ の計算時間は, 次式の通りである.

$$\begin{aligned} (\text{計算時間}) &= (n^2 \log L / N_{PCI}) / (\text{回路の動作周波数}) \\ &= 32768 \log L / (\text{回路の動作周波数}) \end{aligned} \tag{6.1}$$

そして, Pentium4(2.4GHz) の Windows2000 PC でのソフトウェアアプローチによる相違度 $D_T(I)$ の計算時間と比較する.

6.2　ソフトウェアアプローチによる相違度 $D_T(I)$ の計算時間

ソフトウェアアプローチによる相違度 $D_T(I)$ の計算時間 [ms] を表 6.1 に示す.

階調数 $L = 2$ のときは, アルゴリズムを少し工夫しているため, $L = 4, 16$ の場合に比べて計算時間が短くなっている. $L = 4, 16$ での相違度 $D_T(I)$ の計算量は $O(n^2 e)$ なので,

表 6.1: ソフトウェアアプローチでの相違度計算時間 [ms]

L \ e	128	256	512	768
2	1256	1345	1652	1797
4	1495	2958	5812	8667
16	1530	3042	5970	8918

e の値に比例して計算時間が大きくなっている. ここで, $L = 4$ と $L = 16$ で計算時間にほとんど差がないのは, どちらも 1 画素を一つの整数型で表しているためで, プログラム中の繰り返しの回数は同じだからである.

$L = 2$ のとき, すなわち T が白黒 2 値画像の場合は, T と I の部分画像との相違度 $D_T(I[x, y])$ は,

$$\begin{aligned} D_T(I[x,y]) &= \sum_{T_{i,j} \neq d} |T_{i,j} - I_{x+i-1,y+j-1}| \\ &= \sum_{T_{i,j} \neq d} T_{i,j} \oplus I_{x+i-1,y+j-1} \end{aligned} \tag{6.2}$$

となる. これは XOR 演算を用いて m^2 bit の排他的論理和を求め, 1 の数を数えることで計算することができる. m^2 bit の 1 の数を数えることは, m^2 bit を例えば 16bit 毎に分割して, あらかじめ用意しておいた 2^{16} 通りの全ビットパターンの 1 の数を表にしたものを参照することによって, より速く計算することができる.

6.3 相違度計算回路の性能測定結果

Virtex-II XC2V8000 による相違度計算回路は表 6.2 の通りである. 表には各 L, e の値に対する回路動作周波数, 相違度計算時間, ソフトウェアに対しどれだけ速くなっているかを表すためのソフトウェアとの計算時間の比が記されている.

また XC2V8000 は, ロジックブロックをスライスを単位として 46,592 個持っており, 合成に必要なスライス数とその使用率も記している. スライス数は, 有効画素数 e の値におおよそ比例して増えており, $L = 16, e = 768$ のときは, 全体のスライス数のほぼ 9 割を使用している.

また, テンプレート画像 T にドントケアがない場合, 即ち有効画素数が $e = 1024 (= m \times m)$ のときについては, データを取ることができなかった. それは, デザインツールを使用して Verilog HDL コードから論理合成する際に, プログラムの使用メモリが大きくなりすぎた (2GB) ためである.

この表より, 最も効果が高いのは $L = 4, e = 768$ の場合で, ソフトウェアより 3000 倍以

上速くなっている. 最も効果の低い $L = 16, e = 128$ の場合でもソフトウェアより 350 倍速いことがわかる.

表 6.2: Virtex-II XC2V8000 による相違度計算回路

L (k)	e	Freq. [MHz]	Time [ms]	Speed -up	Slices (%)
2 (32)	128	28.0	1.17	1074	6188 (13.3)
	256	26.2	1.25	1076	11307 (24.3)
	512	24.6	1.33	1242	21765 (46.7)
	768	23.1	1.42	1265	32310 (69.3)
4 (16)	128	29.3	2.24	667	6457 (13.9)
	256	27.3	2.40	1233	11403 (24.5)
	512	25.6	2.57	2261	21659 (46.5)
	768	24.2	2.71	3198	32056 (68.8)
16 (8)	128	29.9	4.39	349	8860 (19.0)
	256	27.8	4.72	644	15021 (32.2)
	512	26.1	5.03	1187	27146 (58.3)
	768	24.6	5.31	1679	42251 (90.7)

第7章　地図記号検索システム

本研究で開発した相違度計算回路の応用として, 地図記号検索システムを試作した (図 7.1). 本システムは, 大量の地図データベースの中から, 入力された地図記号に類似する部分画像を列挙する. 地図記号と地図画像中の部分画像の相違度が設定した閾値以下であるときに, 類似するとしてその部分画像が出力される. 閾値の設定は自由であり, ユーザによって適宜決定される.

システムには, あらかじめいくつかの地図記号が用意されており, 各地図記号専用の相違度計算回路がすでに設計されている. テンプレート地図記号 T は, 32×32 画素 ($m = 32$), 白黒 2 値画像であり, ドントケアを含むことができる. 今回は図 7.2 に示す, 郵便局, 発電所, 高校, 寺, 神社の 5 つの地図記号を取り上げ, その地図記号専用の相違度計算回路を設計した. これらのテンプレート地図記号の有効画素数は, およそ 128 から 256 である. 今回挙げた 5 つの地図記号以外のものを検索したいときは, あらかじめ, その地図記号専用の回路を設計しておく必要がある.

システムは, 大量の地図データベースを持っており, 今回使用した地図画像は一枚の大きさが 2000×1544 画素と, テンプレート地図画像に対して非常に大きなもので, 全部で 52 枚用意した.

このシステムでは, FPGA に約 300 万ゲートの Xilinx Virtex-II XC2V3000 を使用した. この FPGA では, 32×32 画素 ($m = 32$), 白黒 2 値画像 ($L = 2$) のテンプレートの場合, $k = 16$ 並列で部分画像との相違度を計算することができる.

システムは次のように動作する.

- 最初に, 検索したい地図記号の相違度計算回路を開き, FPGA にダウンロードする. FPGA には回路が瞬時に構築される.
- 次に, テンプレート地図記号と地図画像中の部分画像との類似性を評価するための, 相違度の閾値を設定する. これは, ユーザが適宜決定する必要がある.
- そして, 地図画像を選択し, 検索ボタンを押すと FPGA に地図画像が送信され, 設定された閾値以下の相違度を持つ地図画像中の部分画像が全て列挙される. 地図データベース中の全ての地図画像について検索したいときのために, 全地図画像検索のボタンもついている.

またシステムは, 本論文で提案する FPGA を用いた検索の有効性を示すため, ソフトウェアによる方法でも検索することができるようになっている.

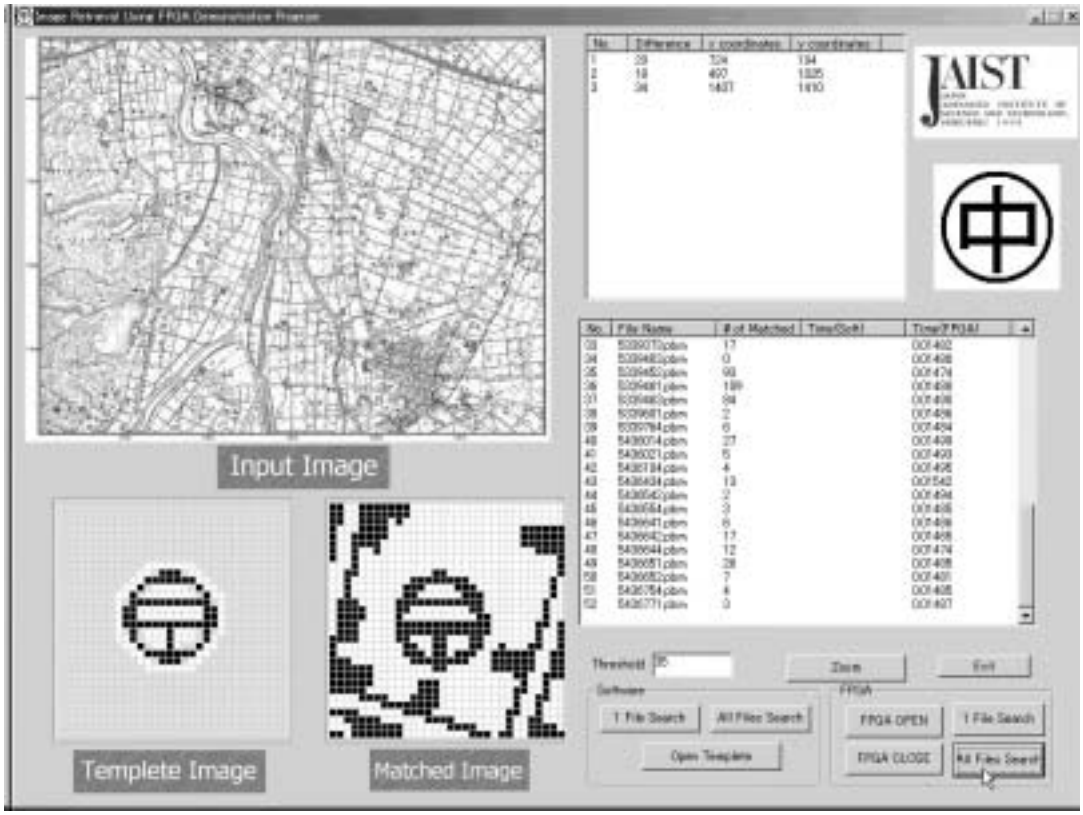


図 7.1: 地図記号検索システム

7.1 実験:地図記号の検索

本システムを使用して, 実際に地図記号を検索してみた.

7.1.1 FPGA とソフトウェアとの比較

FPGA による地図記号の検索結果と, ソフトウェアによる検索結果を比較して同じ結果が得られたことから, 回路が正しく動作していることが確認された.

FPGA を用いた地図記号の検索は, ある地図記号に対して, 一枚の地図データベースを検索するのに必要な計算時間はおよそ 0.014 秒であった. 実際に FPGA に相違度計算回路を実装し, 計算時間を測定してみると, PCI bus の動作周波数がボトルネックとなっていることがわかった. PCI bus の動作周波数は 33MHz であるが, 実際のところは 16MHz 程で動作していると思われる.

それに比べて, ソフトウェアによる計算時間はおよそ 1.94 秒となった. これより FPGA を用いた提案手法は, ソフトウェアアプローチに対し, 140 倍の高速化に成功した. この実験では, PCI bus がボトルネックとなっているため, PCI bus の動作周波数が上がると, 更

なる高速化を期待することができる.

7.1.2 閾値の設定による検索精度

現状では, 相違度の閾値はユーザによって適宜決定される. 各地図記号で適切な閾値の値は, 何度か閾値を変更して検索するうちに経験的に得ることができる. ここでは, 閾値の設定によって検索精度がどのように変わるかを調べてみた.

効率的な検索には, 探したい地図記号のテンプレートが正確 (または最適) であることが望まれる. そこで, 今回用意した 5 つの地図記号のうち, 形状が単純であり, 地図画像中でも歪みが現れにくいと思われる, 寺 (図 7.2(d)) の地図記号について調べてみた.

まず, 3 枚の地図を無作為に選び, 手作業で寺の地図記号を探しだし, これを正解とする (図 7.3 〜図 7.5). 図は国土地理院の地図である. これらの地図には, 寺の地図記号が合計で 48 個あった.

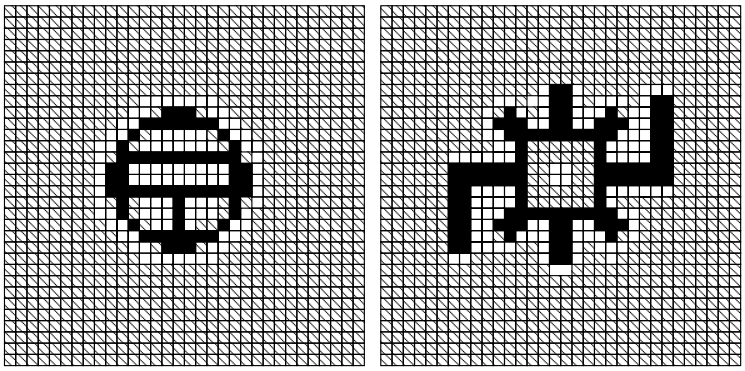
そして本システム使用して, 地図画像の中から寺の地図記号との相違度が設定された閾値以下である部分画像を検索する. 本システムでは, わずかに 1, 2 画素だけずれた 2 つの部分画像が両方ともテンプレート地図記号に類似するとして出力されることがある. そこで重複を避けるために, 地図記号との相違度が小さいものを選択することにした.

評価の方法には, 次の二つがある.

$$\text{precision} = (\text{検索出力中の正解数})/(\text{検索出力数}) \tag{7.1}$$

$$\text{recall} = (\text{検索出力中の正解数})/(\text{地図中の正解数}) \tag{7.2}$$

この 3 枚の地図画像について, 相違度の閾値と検索精度の関係を表 7.1 に示す. 表より, 閾値が 23 以下のときは検索出力には寺の地図記号以外のものを含まないが, 検索漏れが出ている. 閾値が 24 以上では, 検索出力には寺以外の誤りの画像も含んでいるが, 閾値が 26 になると全ての地図中の寺を検索することができた. これより, 寺の地図記号の場合, 閾値を 24 辺りにすると precision も recall もほどよい結果が得られそうだ. あるいは, 閾値を 26 より少し大きめに取って, 地図中の全ての寺を漏れなく検索し, あとはソフトウェアによる後処理で寺だけを取り出すという方法も考えられる. このように, 最初に FPGA を用いて候補の画像を選出しておけば, 全てソフトウェアで検索するよりも十分高速に計算することができる.



(a) 郵便局　　(b) 発電所

(c) 高校　　(d) 寺

(e) 神社

図 7.2: 地図記号

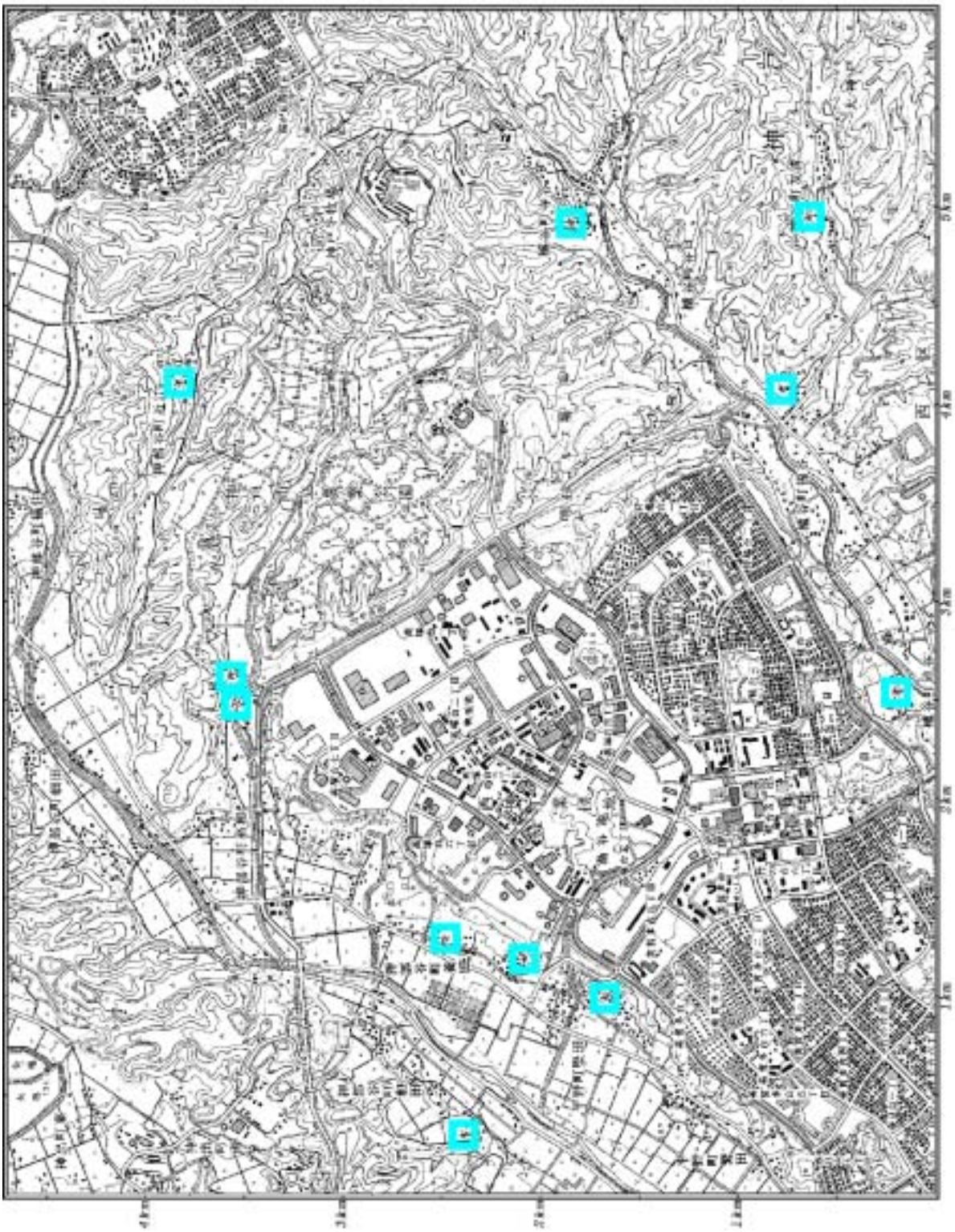


図 7.3: 地図 1(正解数 11 個)

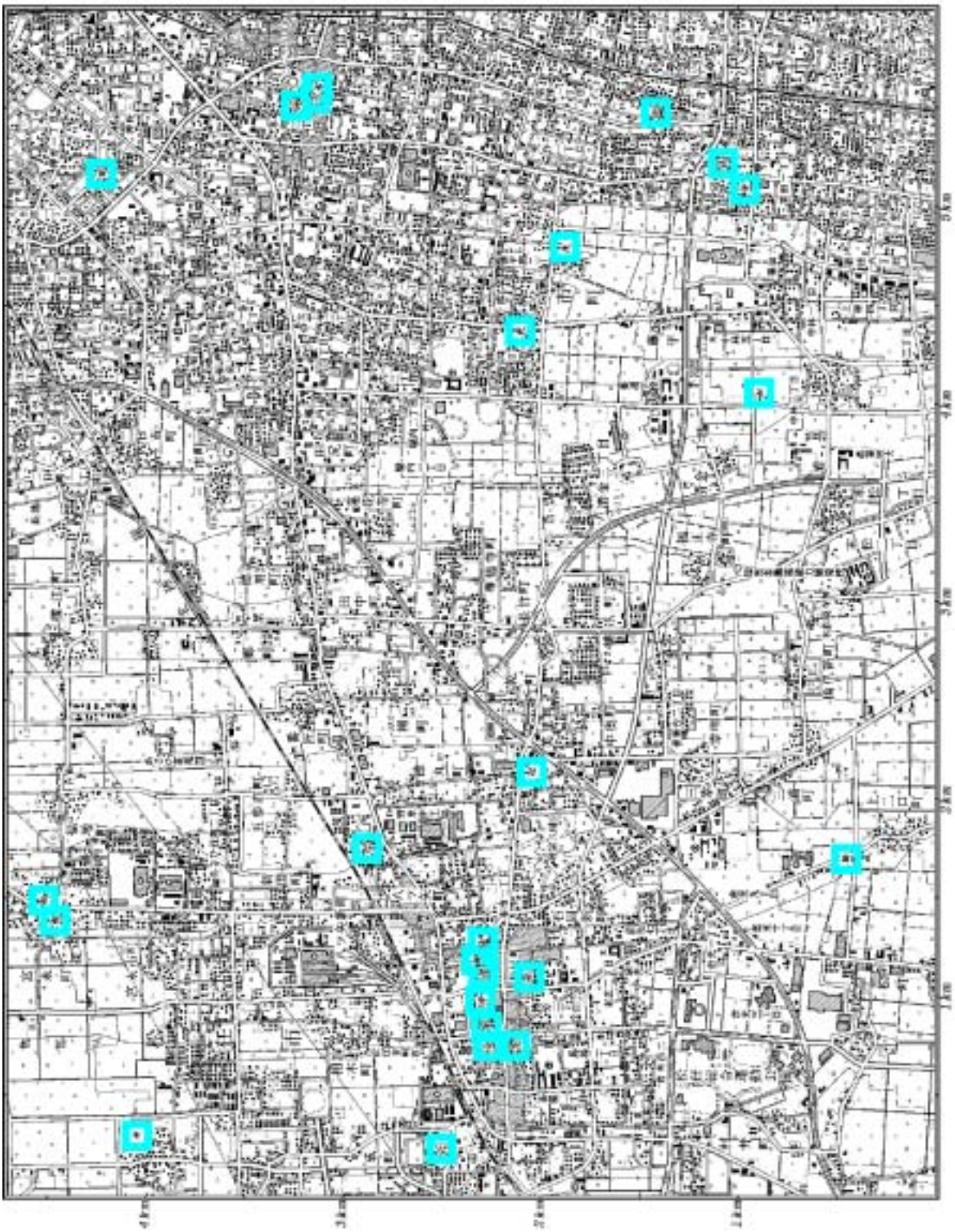


図 7.4: 地図 2(正解数 25 個)

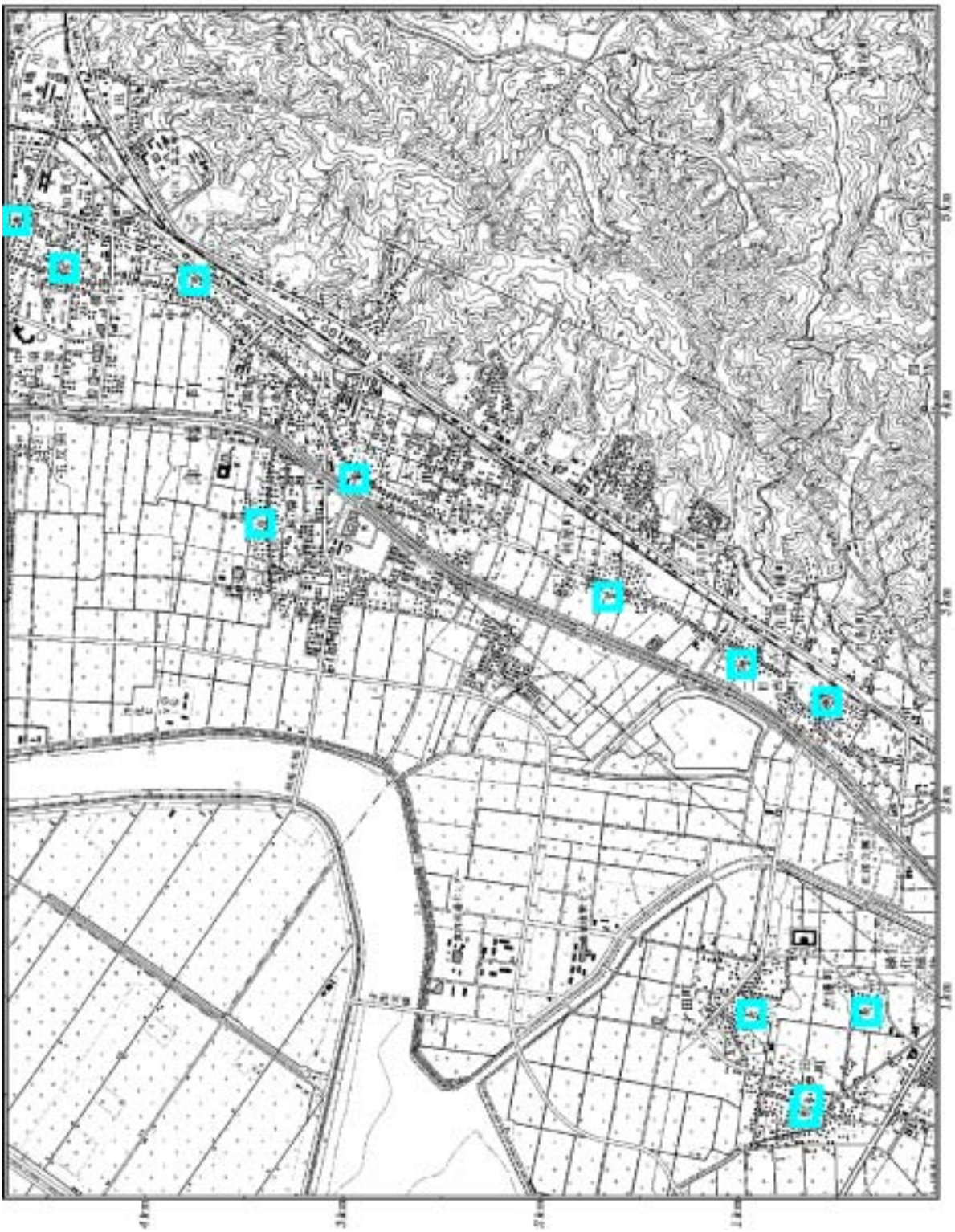


図 7.5: 地図 3(正解数 12 個)

表 7.1: 寺の地図記号における相違度の閾値と検索精度

閾値	相違度が Thr の部分画像 (うち正解数)				Thr 以下の	precision	recall
Thr	地図 1	地図 2	地図 3	合計	部分画像		
0	0(0)	1(1)	0(0)	1(1)	1(1)	1.00	0.021
1	0(0)	0(0)	0(0)	0(0)	1(1)	1.00	0.021
2	0(0)	0(0)	0(0)	0(0)	1(1)	1.00	0.021
3	0(0)	0(0)	0(0)	0(0)	1(1)	1.00	0.021
4	0(0)	0(0)	0(0)	0(0)	1(1)	1.00	0.021
5	0(0)	0(0)	0(0)	0(0)	1(1)	1.00	0.021
6	0(0)	1(1)	0(0)	1(1)	2(2)	1.00	0,042
7	0(0)	0(0)	1(1)	1(1)	3(3)	1.00	0.063
8	0(0)	0(0)	1(1)	1(1)	4(4)	1.00	0.083
9	0(0)	0(0)	1(1)	1(1)	5(5)	1.00	0.104
10	1(1)	0(0)	0(0)	1(1)	6(6)	1.00	0.125
11	0(0)	0(0)	0(0)	0(0)	6(6)	1.00	0.125
12	0(0)	1(1)	0(0)	1(1)	7(7)	1.00	0.146
13	1(1)	1(1)	2(2)	4(4)	11(11)	1.00	0.229
14	2(2)	2(2)	2(2)	6(6)	17(17)	1.00	0.354
15	1(1)	4(4)	1(1)	6(6)	23(23)	1.00	0.479
16	2(2)	3(3)	1(1)	6(6)	29(29)	1.00	0.604
17	0(0)	0(0)	1(1)	1(1)	30(30)	1.00	0.625
18	0(0)	3(3)	0(0)	3(3)	33(33)	1.00	0.688
19	2(2)	2(2)	0(0)	4(4)	37(37)	1.00	0.771
20	0(0)	1(1)	0(0)	1(1)	38(38)	1.00	0.792
21	1(1)	5(5)	0(0)	6(6)	44(44)	1.00	0.917
22	0(0)	0(0)	0(0)	0(0)	44(44)	1.00	0.917
23	0(0)	1(1)	3(1)	4(2)	48(46)	0.95	0.958
24	2(0)	4(0)	1(0)	7(0)	53(46)	0.86	0.958
25	1(0)	2(0)	2(1)	5(1)	58(47)	0.81	0.979
26	4(1)	2(0)	0(0)	6(1)	64(48)	0.75	1.00

7.2 バッファ回路

地図記号検索システムで実装した相違度計算回路は, PCI bus の動作周波数がボトルネックとなっていた. PCI bus の動作周波数は 33MHz であるが, 実際にはおよそ 16MHz で動作しているものと思われる.

このボトルネックによる損失を軽減するために, より効率のよいデータ転送が望まれる. 今回使用した PCI はその bus 幅が $N_{PCI} = 32$ bit であった. これより白黒 2 値画像の場合, 1 クロックに供給できる画素数は N_{PCI} 画素であり, 最大で $N_{PCI}/\log L$ 並列に計算することができる. しかし実装した相違度計算回路は $k = 16$ 並列であったため, 32bit のうち 16bit しか有効に使われていなかった. そこで, 効率よく画像データを送信できるようにバッファ回路を作成した.

相違度計算回路の前段にバッファ回路を挿入した例を図 7.6 に示す. 図は, テンプレートの一辺が $m = 4$ 画素, 階調数 $L = 4$, 並列数 $k = 2$ で,PCI bus 幅が $N_{PCI} = 8$ bit, 即ち $4(= N_{PCI}/\log L)$ 画素分の場合である. バッファ回路は, PCI クロックと FPGA 内部で動作するローカルクロックの 2 つのクロックで動作する. ローカルクロックは PCI クロックよりも高い周波数で動作するように設計されている.

バッファ回路は, $4 \times n$ 画素のデータを蓄える RAM を 2 個持っている. 一方は PCI からデータを入力するための入力 RAM, もう一方は相違度計算回路にデータを出力するための出力 RAM である. 入力 RAM は PCI クロックで動作し, PCI から入力された縦 4 画素分のデータを順次蓄えていく. 出力 RAM はローカルクロックで動作し, 蓄えているデータから縦 2 画素分のデータを順次出力していく. もし, 入力 RAM が一杯になる前に出力 RAM が空になったら, 出力 RAM が待つ. また, 出力 RAM が空になる前に入力 RAM が一杯になったら, 入力 RAM が待つ. そして, 入力 RAM が一杯になり出力 RAM が空になった時点で, 入力 RAM は出力 RAM に, 出力 RAM は入力 RAM に役割を交代する. より詳細に説明すると, 入力 RAM には, PCI の 1 クロック目は入力画像 I の画素 $I_{1,1}, I_{1,2}, \ldots, I_{1,m}$ が, 同様に 2 クロック目は $I_{2,1}, I_{2,2}, \ldots, I_{2,m}$ が蓄えられる. そして, n クロック目には, $I_{n,1}, I_{n,2}, \ldots, I_{n,m}$ が入力され, 入力 RAM は一杯になる. その間出力 RAM はローカルクロックで動作し, 初期状態で保持しているデータを出力する. 入力 RAM が一杯になり, 出力 RAM が空になった時点で, 役割を交代する.

そして新しい入力 RAM は, 次の n 回の PCI クロックで, $I_{1,m+1}, I_{1,m+2}, \ldots, I_{1,2m}$, $I_{2,m+1}, I_{2,m+2}, \ldots, I_{2,2m}$, ..., $I_{n,m+1}, I_{n,m+2}, \ldots, I_{n,2m}$ を蓄える. このように PCI は入力画像を縦 m 画素毎に重複なく入力する. それと同時に新しい出力 RAM は, 次の $2n$ 回のローカルクロックで, $I_{1,1}, I_{1,2}$, $I_{2,1}, I_{2,2}$, ..., $I_{n,1}, I_{n,2}$, $I_{1,3}, I_{1,m}$, $I_{2,3}, I_{2,m}$, ..., $I_{n,3}, I_{n,m}$ を出力する. 同じように入力 RAM が一杯になり, 出力 RAM が空になった時点で, 役割を交代する. この動作を繰り返していく.

このように, PCI クロックと FPGA 内部で動作するローカルクロックの 2 つの独立したクロックを用いることによって, PCI から入力されたデータを無駄なく効率よく取り出すことができる.

地図記号検索システムの相違度計算回路にバッファ回路を挿入したときの, 地図 1 枚に

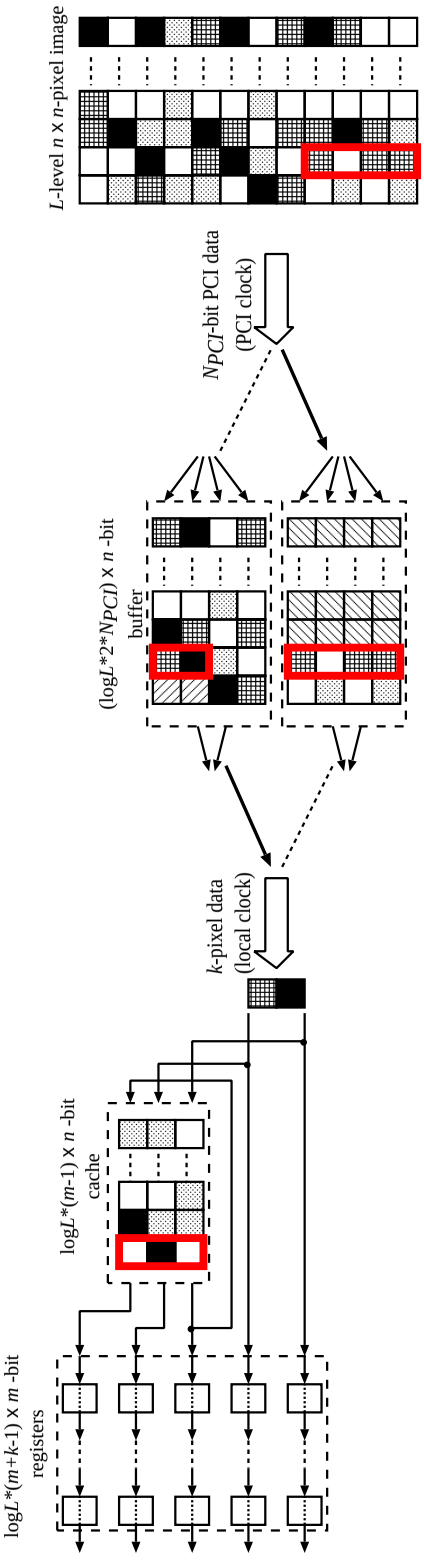

L-level n x n-pixel image
NPCI-bit PCI data
(PCI clock)
(logL*2*NPCI) x n -bit
buffer
k-pixel data
(local clock)
logL*(m-1) x n -bit
cache
logL*(m+k-1) x m -bit
registers

図 7.6: バッファ回路

かかる計算時間は, 0.008 秒となった. これは, バッファ回路を使用しないときの 0.014 秒に比べて 1.75 倍の高速化である. バッファ回路を使用しないときは, PCI bus 幅 32bit のうち 16bit しか有効に使われていなかったのが, バッファを挿入することによって 32bit 無駄なく活用できるようになった. それによって 2 倍にまでは届かなかったが 1.75 倍高速になった. しかし, この回路の動作周波数はおよそ 50MHz であり, それから求められる理論値は 0.004 秒であるため, PCI にボトルネックがあることには代わりなかった. また, 従来通りのソフトウェアによる計算時間の 1.94 秒に比較すると, 240 倍の高速化となった.

第8章　まとめ

本論文では, 画像データベース $I_1, I_2, \ldots$ の中からテンプレート画像 T に類似する画像を全て列挙するという画像検索問題について考え, FPGA を使用するインスタンスに特化した手法を提案した.

本研究で提案した相違度計算回路はテンプレート画像 T が L 階調白黒濃淡画像のときに, $n \times n$ $(n \geq m)$ 画素の入力画像 I に対し, $n^2 \log L / N_{PCI}$ クロックサイクルで動作する. FPGA に PCI 接続の Xilinx 社 Virtex-II XC2V8000 を使用するときの画像検索回路を設計し, タイミング解析ツールを用いて回路の動作周波数を解析した. そして, 得られた回路の動作周波数から画像検索の計算時間を求め, ソフトウェアアプローチによる計算時間と比較した. その結果, 本手法は従来のソフトウェアを用いて検索する手法に比べ, 最大で 3000 倍の高速化に達した.

また, 画像検索回路の応用として, 地図記号検索システムを試作した. テンプレート T は 32×32 画素の白黒 2 値の地図記号の画像であり, ドントケアを含むことができる. FPGA に PCI 接続の Xilinx 社 Virtex-II XC2V3000 を使用する場合, $k = 16$ 並列で地図記号を検索する相違度計算回路を構築することができた. そして, 実際に地図記号を検索し, 検索にかかる時間を測定したところ, ソフトウェアアプローチに比較しておよそ 140 倍高速に計算することができた. このシステムでは, PCI の動作周波数にボトルネックがあったため, バッファ回路を新たに設計し, 挿入した. バッファ回路を挿入した相違度計算回路は, バッファ回路を使用しないときに比べて 1.75 倍高速になり, ソフトウェアアプローチに比較すると 240 倍の高速化に達した. それでも PCI にボトルネックがあるために, PCI の動作周波数が上がるとより高速に計算できると期待される.

謝辞

本研究を進めるにあたり, 終始ご指導頂きました中野浩嗣助教授に心より感謝致します.

また, 日頃からたくさんのアドバイスをくださいました浅野哲夫教授, 小保方幸次助手, 元木光雄助手, Jacir Luiz Bordim さんにはとてもお世話になりました. ここにお礼申し上げます.

最後に, 共に研究活動を行ってきた情報基礎学講座のみなさんに感謝の気持を贈ります. ありがとうございました.

参考文献

[1] J. L. Bordim, Y. Ito, and K. Nakano. Accelerating the CKY parsing using fpgas. In *Proc. of International Conference on High Performance Computing, to appear*, 2002.

[2] Y. Futamura, K. Nogi, and A. Takano. Essence of generalized partial computation. *Theoretical Computer Science*, Vol. 90, pp. 61–79, 1991.

[3] N. D. Jones, C.K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall, 1993.

[4] T. Kean and A. Duncan. A 800Mpixel/sec reconfigurable image correlator on XC6216. In *Proc. of International Conference on Field Programmable Logic and Applications (FPL)*, pp. 382–391, 1997.

[5] M. J. Wirthlin and B. McMurtrey. Efficient constant coefficient multiplication using advanced fpga architecture. In *Proc. of International Conference on Field Programmable Logic and Applications (FPL)*, pp. 555–564, 2001.

論文リスト

1. K. Nakano, E. Takamichi, "An image Retrieval System Using FPGAs", to appear in the IEICE Transactions on Information and Systems

2. Koji Nakano, Etsuko Takamichi, "An image Retrieval System Using FPGAs", Proc. of Asia and South Pacific Design Automation Conference (ASP-DAC 2003), pp. 370-373, Jan. 21-24, Kitakyushu, Japan.

3. Etsuko TAKAMICHI and Koji NAKANO, "An Image Retrieval System Using FPGAs", Technical Report of IEICE, CAS2002-83〜96 , pp. 37-42, November 2002.

4. 高道 悦子, "FPGA による画像検索の高速化", 平成 14 年度 電気関係学会北陸支部連合大会 講演論文集, pp. 215, September 2002.