The 19th LSI DESIGN CONTEST in OKINAWA 2016

2016年コンテストテーマ:
人物検出画像処理システムのハードウェア設計

開催日: 平成28年3月11日(金)
琉球大学 50周年記念会館

主催: LSIデザインコンテスト実行委員会

共催: 琉球大学工学部, 九州工業大学情報工学部,
電子情報通信学会 基礎・境界ソサイエティ スマートインフォメディアシステム研究会

協賛: 日本シノプシス合同会社, 株式会社産業タイムズ社, ギガファーム株式会社
株式会社アナログ・デバイセズ, 公益社団法人沖縄県情報産業協会

後援: 国立沖縄工業高等専門学校, CQ出版社

http://www.lsi-contest.com

目次

コンテストの概要・目的

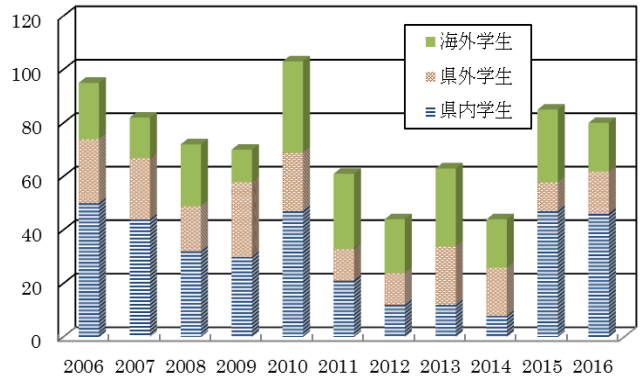
九州・沖縄さらに東南アジア地域の半導体産業および組み込み機器等のエレクトロニクス産業の振興を目的に、標題の学生向けの設計コンテストを毎年実施しております。主催は、琉球大学および九州工業大学の教員で構成するLSIデザインコンテスト実行委員会および九州経済局の外郭団体である九州半導体イノベーション協議会の協力で実施しております。2016年の応募者は、国内外を含め80名を超える盛況ぶりでした。

東シナ海および南シナ海沿岸のLSI産業拠点地域（九州、韓国、台湾、中国、シンガポール、フィリピン、マレーシア）は、世界の半導体市場の実に40%を越える高いシェアを有しており（九州以東の日本本土は含みません！）、その中心に位置する沖縄はそのビジネスチャンスが大変高いものと期待されます。そうした地理的好条件の沖縄にて、学生向けLSI設計コンテストを実施し、学生すなわち未来のエンジニアの設計スキルを上げることにより、将来的に沖縄や東南アジアでの企業誘致やベンチャー起業につなげたいと考えております。

今回は、コンテスト初の画像処理にチャレンジします。具体的には、最近の自動ブレーキシステムに使われる人物検出システムです。基本的なパターンマッチングは固より、色んなアイデアが生まれることを期待しております。

本コンテストの主旨をご理解頂き、多くの学生の皆様（高専、大学、大学院生）の参加を期待しております。

学生参加数の推移



国内

琉球大学、千葉大学、九州工業大学、会津大学、大分県立工科短期大学、沖縄職業能力開発大学校、大阪工業大学、大阪大学、京都大学、近畿大学、高知工科大学、神戸大学、芝浦工業大学、職業能力開発大学校、東海大学、東京工業大学、東京都立科学技術大学、東京理科大学、東北大学、徳島大学、豊橋技術科学大学、長崎大学、新潟大学、広島大学、法政大学、山形大学、横浜国立大学、立命館大学、早稲田大学、沖縄高専、有明高専、木更津高専、久留米高専、豊田高専

海外

Bandung Institute of Technology、Institut Teknologi Telkom、Telecommunication College Bandung（インドネシア）、Univ. of Science、Ho Chi Minh city、Hanoi Univ. of Science（ベトナム）、Chosun 大学(韓国)、Univ. of Valladolod（スペイン）、Univ. of Kaiserslautern（ドイツ）、NTI（エジプト）、McMaster 大学（カナダ）、南西大学（中国）

これまでの設計課題

- ■ 2000 年： 「リードソロモン CODEC 用のガロア体での行列乗算器の設計」
- ■ 2001 年： 「デジタル CDMA レシーバ」
- ■ 2002 年： 「差集合巡回符号エラー訂正回路」
- ■ 2003 年： 「静的ハフマン符号用の可変長デコーダ」
- ■ 2004 年： 「共通鍵暗号 AES 用 SubByte 変換回路」
- ■ 2005 年： 「デジタル FM レシーバー」
- ■ 2006 年： 「2 次元積符号繰り返しデコーダ」
- ■ 2007 年： 「64 点高速フーリエ変換」
- ■ 2008 年： 「RSA 暗号デコーダ」
- ■ 2009 年： 「Small RISC Processor」
- ■ 2010 年： 「エラー訂正：BCH 符号」
- ■ 2011 年： 「DCT」
- ■ 2012 年： 「16/64/128-point Flexible FFT」
- ■ 2013 年： 「SW・HW 協調設計を用いたノイズ除去システム」
- ■ 2014 年： 「SW・HW 協調設計を用いたノイズ除去システム」
- ■ 2015 年： 「正弦波発生回路」

■ 2016 年： 「人物検出用パターンマッチング回路」

開 催 概 要

名　　称　：「第 19 回 LSI デザインコンテスト in 沖縄 2016」
主　　催　：LSI デザインコンテスト実行委員会
　　http://www.lsi-contest.com/
共　　催　：琉球大学工学部、九州工業大学情報工学部、電子情報通信学会 基礎・境界ソサイエティ スマートインフォメディアシステム研究会
協　　賛　：日本シノプシス合同会社、株式会社産業タイムズ社、ギガファーム株式会社、株式会社アナログ・デバイセズ、公益社団法人沖縄県情報産業協会
実行事務局　：九州工業大学情報工学部電子情報工学科尾知研究室
　　LSI デザインコンテスト実行委員会事務局
日　　時　：2016 年 3 月 11 日（金）　13:00～18:00
会　　場　：〒903-0213 沖縄県西原町千原 1
　　琉球大学 50 周年記念館（下記キャンパスマップ 30 番）
　　http://www.u-ryukyu.ac.jp/univ_info/images/campus_map_large.jpg
目　　的　：実践的な課題を用いた学生対象のデジタル集積回路設計のコンテストであり、半導体集積回路設計能力の向上とともに国際的に交流することで学生の工学に関する視野を広めることを目的としている。
対　　象　：国内大学・大学院生、高専学生、アジア地域大学生
来場予定者　：100 名（入場無料）
募集方法　　：各大学・高専へのパンフレット送付、LSI 関連雑誌等へのリリース
ホームページ：http://www.LSI-contest.com/

【審査員】
審査委員長：　琉球大学　和田 知久
東京大学　藤田 昌宏
大阪大学　尾上 孝雄
九州工業大学　黒崎 正行
琉球大学　吉田 たけお
バンドン工科大学　Trio Adiono
沖縄工業高等専門学校　兼城 千波
CQ 出版社　西野 直樹
他協賛企業審査員　　（敬称省略・順不同）

連絡先
住　所：〒820-8502 福岡県飯塚市川津 680-4 九州工業大学情報工学部電子情報工学科
電　話：0948-29-7667　Email: support@LSI-contest。com
LSI デザインコンテスト実行委員会委員長
担当者　尾知　博

【設計テーマ】　人物検出画像処理システム

第19回のテーマは、最近の自動ブレーキシステムに使われる人物検出システムです。

画像認識技術の一つとして人物検出が挙げられます。人物検出は、デジタルカメラや自動車の自動ブレーキシステム、監視システム等に用いられている技術です。今回は「人物検出」をテーマとして、処理の高速化、回路規模の削減を目指したハードウェア設計を行うことを目的とします。

要求されている設計はHDL（VHDLもしくはVerilogHDL）による設計と論理合成および検証結果です。特にHPの設計資料で使ったシノプシス社の合成ツールを使用する必要はなく、FPGA等の合成ツールを利用しても構いません。

実装アルゴリズムと環境

■　テンプレートマッチングの概要

画像からの人物検出は、近年研究が盛んに行われており多くの手法が提案されている。 その中でも簡明で実装が容易なテンプレートマッチング法について説明する。

テンプレートマッチングでは、テンプレート画像と呼ばれる小さな画像を用意し、被探索画像の左上 から順番に右下まで比較を行い、テンプレート画像と被探索画像がどの程度一致しているか(類似度)を算出する。類似度の指標として一般的に画素値を用いる。類似度の算出方法はいくつか存在するが、今回は高速な演算が可 能な SAD(Sum of Absolute Difference)を用いる。

・SAD(Sum of Absolute Difference)

SAD は画素値の差の絶対値の総和によって表される。加算と減算のみで実装が可能であるため容易でかつ高速 な演算を行うことができる。被探索画像の x 行目、y 列目の画素値を I(x、y)、テンプレート画像の x 行目、y 列目の 画素値を T(x、y)とすると SAD は以下の式によって表すことができる。

$$R_{SAD}(d_x, d_y) = \sum_{j=0}^{N-1} \sum_{i=0}^{M-1} |I(d_x + i, d_y + j) - T(i,j)|$$

SAD の値が小さいほどテンプレート画像と被探索画像の類似度が高いことを示し、完全に一致している場合は SAD の 値は 0 になる。

■　実装環境

Synopsys® Synphony Model™ Compiler
Synopsys® Synplify Pro®
Synopsys® Design Compiler®
Mathworks® MATLAB® /Simulink®

または設計環境に応じて、
　Synplify Pro/Premier or その他論理合成ツール
　RTL handcoding(VHDL or Verilog-HDL)
などの設計環境が挙げられます。

課題

Level 1： 初心者向け テンプレートマッチングを用いた人物検出

Level の基本課題では、以下の主な流れに沿って画像のテンプレートマッチングを行う回路の設計を行っていただきます。

1。被探索画像、テンプレート画像の入力
2。入力された画像のグレースケール化
3。グレースケール化した画像の二値化
4。マッチング処理
5。結果画像の出力
（結果画像は色枠で囲むなどしてマッチした部分がわかるようにしてください。）

入出力については以下のとおりです。

入力：被 探 索 画 像　（サイズ：640×480）
　　テンプレート画像　（サイズ：40×100）
出力：結　果　画　像　（サイズ：640×480）

Level 2： 上級者向け さまざまなアルゴリズムを用いた人物検出

Level2 の自由課題では、様々な人物検出アルゴリズムを用いて人物検出を行う回路を作成していただきます。

入出力については以下のとおりです。
入力：被 探 索 画 像　（サイズ：任意）
　　テンプレート画像　（サイズ：任意）

出力：結　果　画　像　（サイズ：被探索画像と同じ）

入出力ともに、画像のスキャン順は問いません。また、被探索画像は静止画、動画を問いません。

審査：JUDGE

■　審査メンバーによる以下の 4 項目各 10 点で審査を実施（10 pint each）

1）アカデミック的、新奇アイデア的な観点（Academic、　New Idea）

2）実用設計、産業面応用的な観点（Used in real life、 Good for industry）

3) FPGA 等の実装レベルの観点（Good prototype by FPGA etc。）

4）プレゼンテーションの観点（Good presentation）

表彰：AWARD

■　優勝（電子情報通信学会 SIS 賞）　SIS AWARD

【アカデミック的、新奇アイデア的な観点】の BEST

■　その他、2)、3)、4)の観点からも賞を贈呈します。

FPGA を用いたリアルタイムエッジ検出回路の実装

九州工業大学 情報工学部 電子情報工学科
中村健太郎　　下留諒　　吉川奈波

はじめに

FPGA は，ハードウェア内部の構成をリソースの範囲の許す限り自由に設定できる PLD の中の１つである．また，ここでいうエッジとは，色が変化が大きい箇所における色の変わり目のことをいう．今回は XILINX 社の FPGA ボード“Spartan“およびカメラを用いて，取り込んだ映像に対して処理を行い，エッジを検出するシステムを，なるべく単純化した回路で実装する方法について報告する．

1. 設計課題

画像認識技術の一つとして人物検出が挙げられる．人物検出は，デジタルカメラや自動車の自動ブレーキシステム，監視システム等に用いられている技術である．今回は「人物検出」をテーマとして，処理の高速化，回路規模の削減を 目指したハードウェア設計を行うことを目的とする．Level1 初心者向けの基本課題は，人物検出の手法の一つであるテンプレートマッチング法を用いて被探索画像とテンプレート画像の二つを入力とする人物検出ユニットの設計とする．Level2 の課題は，画像の入力形式や検出方法等はすべて自由とし，よりユニークな人物検出ユニットの設計とする．

2. 用いた回路ブロック

今回，回路の設計には MATLAB/Simulink を用いている．このソフトウェアはあらかじめ用意されている回路ブロックを結線していき，適切なパラメータを設定することで回路を実装することができる．

2.1 Add

図 2.1 に示すこのブロックは，入力された信号に対して加算を行った結果を次のブロックへ出力する．入力は適宜増やすことができる．

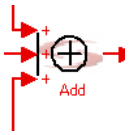


図 2.1：Add ブロック(MATLAB/Simulink 上表記)

2.2 Gain

図 2.2 に示すこのブロックは，入力された信号に対して増幅および減衰を行った結果を次のブロックへ出力する．

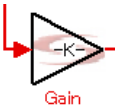


図 2.2：ブロック(MATLAB/Simulink 上表記)

2.3 Switch

図 2.3 に示すこのブロックは，上段と下段の 2 つの入力信号に対して，設定した閾値および閾値との検証に用いる入力信号を用いることで，上下どちらか片方の入力信号を次のブロックへ出力する．

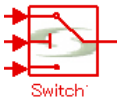


図 2.3：ブロック(MATLAB/Simulink 上表記)

2.4 RAM

図 2.4 に示すこのブロックは，din から入力された信号を wadr に示す場所に格納する．また radr に示す場所に記録された値を dout から出力し次のブロックへ出力することができる．格納と出力は双方有効にすることもできるが，どちらか片方の機能のみ有効にすることもできる．このブロックに z^{-1} と表記があるように，出力が 1 クロック遅れる．

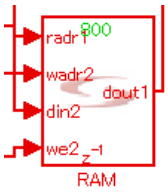


図 2.4：ブロック(MATLAB/Simulink 上表記)

2.5 Constant

図 2.5 に示すこのブロックは，設定した値を次のブロックへ出力し続ける．

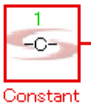


図 2.5：ブロック(MATLAB/Simulink 上表記)

2.6 Counter

図 2.6 に示すこのブロックは，1 クロックごとに指定した値を加算もしくは減算していった結果を逐次次のブロックへ出力する．デフォルトでは無限にカウントするが，指定した値になるとカウントをやめることもできる．入力信号が 1 であるときにカウントを行う．

図 2.6：ブロック(MATLAB/Simulink 上表記)

2.7 Recast

図 2.7 に示すこのブロックは，入力信号の型を変換し，変換後の値を次のブロックへ出力する．今回では出力が RGB(符号なし整数 8bit)で出力する必要があり，その変換に用いている．

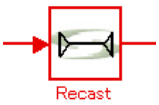


図 2.7：ブロック(MATLAB/Simulink 上表記)

3. 回路の機能説明

3.1 エッジ検出方法

エッジ検出に使用した技法は以下の 3 つである．

・グレースケール化
・二値化
・差分法によるエッジ検出

3.2 回路の説明

今回作成した回路は図 3.1 である．

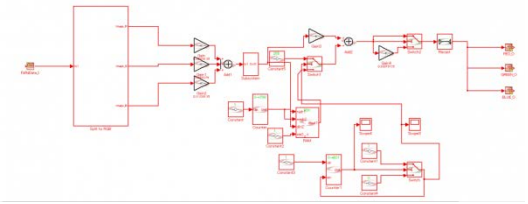


図 3.1：作成した回路

作成した回路を使用した技法に沿って説明する．

(1)R,G,B それぞれに値を乗算し，グレースケール化を行う．

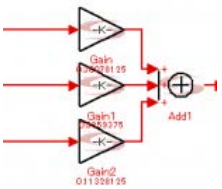


図 3.2：回路(1)

(2)グレースケール化した信号を二値化する．

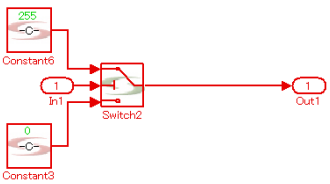


図 3.3：回路(2)

(3)RAM に二値化した信号を横 1 列分格納する．ここで今回はピクセル 8bit，横一列 800 ピクセルであるので，8×800＝6400bit よって 800Byte 実現している．

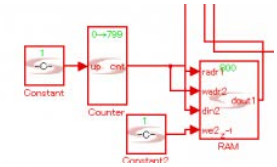


図 3.4：回路(3)

(5)図 3.5 を用いて横 1 ピクセルずらした信号と元の信号を差分法を用いてエッジを検出する．ここで縦は RAM での遅延により 1 ピクセル分ずれている．

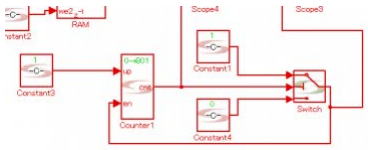


図 3.5：回路(4)

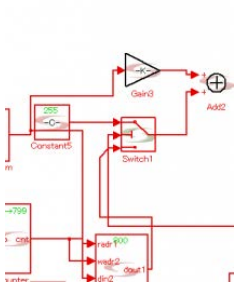


図 3.6：回路(5)

(6)符号付き整数を符号なし整数に変換したのちに出力する．

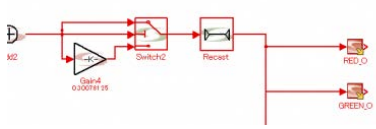


図 3.7：回路(6)

4. 工夫点

4.1 画像処理

今回，人物検出をするために検出される画像へいくつかの処理を行った．今回行った処理はグレースケール化，二値化，エッジ検出の 3 種類である．グレースケール化とは画像を白黒で表す処理であり，画像が RGB 合わせて 24 ビットであれば，グレースケール化することによって 8 ビットのデータにすることができる．今回は NTSC 加重平均法という方法でグレースケール化を実現している．次に，二値化とは画像を白か黒かの二値で表す処理であり，画像データが 0 か 255 の 2 パターンとなるため，とても単純になる．最後にエッジ検出とは，画像の色の境界を際立たせる処理である．人物の形を捉えるのに適した方法であると考え，画像処理を行った．

4.2 画像処理の順番

今回の回路では図 4.1 に示すように，入力された画像に対し，グレースケール化，二値化，エッジ検出の順に処理を行った．今回エッジ検出の処理を実現するためには ROM を使っており，ROM のメモリには制限があることと，処理で扱うデータ量を制限する目的でエッジ検出の前にグレースケール化，二値化の処理を行っている．これらの処理を予め行うことにより，実際の画像処理はほぼリアルタイムで行うことができた．

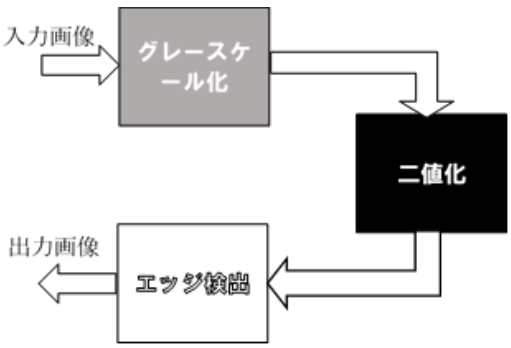


図 4.1　画像処理順

4.3 エッジ検出の方法

今回エッジ検出の方法には差分法を用いた．差分法とは，元画像の色を反転したものを縦横それぞれ数ピクセルずつ動かして，元画像と足し合わせることでエッジを検出する方法である．今回の差分法の計算は式(4.1)，式(4.2)に示すような式で行った．今回エッジ検出の方法に差分法を選んだ理由は，実装

がとても容易であり，処理が単純なので高速化が図れるためである．その結果は図 4.2 のようになる．

$$x_O = x_I - (\overline{x_I} + 1) \quad (4.1)$$
$$y_O = y_I - (\overline{y_I} + 1) \quad (4.2)$$

x_O：エッジ検出後の画像の縦成分
y_O：エッジ検出後の画像の横成分
x_I：元画像の縦成分
y_I：元画像の横成分
$\overline{x_I}$：元画像を反転させた画像の縦成分
$\overline{y_I}$：元画像を反転させた画像の横成分

図 4.2 二値化画像(左)とエッジ検出後画像(右) ※1

縦横エッジ検出の実現

エッジ検出を実現するにあたって，画像を横にずらすにはディレイを数個かけることで実現できる．しかし，縦方向へずらすのにディレイを用いると，今回の場合 800 個のディレイをかける必要があり，処理時間が大幅に増加してしまう．そこで今回は，縦横数ピクセルずつずらす処理，式(4.1)(4.2)でいうと括弧内の＋1 部分の処理の実現には ROM を使用した．ROM へ画像データの横一列を読み込んだ後，順にデータを出力することで縦方向へ 1 ビットずらしている．また，MATLAB 上での ROM のブロック図は図 4.3 に示すような図になっており，ディレイが 1 つ入っていることが分かる．このディレイによって横方向へ 1 ビットずらす処理も実現されている．

図 4.3 ROM のブロック図

5. 改善点

今回，回路を作成するにあたっての問題点に関する改善点をいくつか述べる．

5.1・グレースケール化と二値化について

始めは二値化を行った後にグレースケール化をして処理を行ったが，壁や窓と人が同化してしまったため，人物の認識ができなかった．よって，グレースケール化をして色の情報を少なくして二値化を行うことによって人物の形が見えるようになった．

5.2 縦に 1 ピクセルずらす方法について

RAM を使用して縦 1 ピクセルずらす方法について，すべての値について RAM に格納された値と元の信号と重ね合わせるだけでは 1 列ずらすことができない．それは初めの 1 行分の横方向に関しては RAM 内にデータがないためである．そのため counter と switch を使用して初めの 1 行分の横方向(今回の場合は 800 ピクセル分)は 255 を出力し，元の信号と重ね合わせることによって 1 ピクセルずらしエッジを抽出することができた．

5.3 符号付き整数について

差分法では減算しているため信号の値が負になることがある．R,G,B の値は 0～255 であるためその範囲以外の値になると出力できない．そこで recast を使用して符号なし整数に変換し，出力することができた．

6. 考察

これらのエッジ検出回路をどのように用いれば人物検出へと応用することができるかの考察を行う．

6.1 大量のサンプルデータとのマッチングを行う

ROM 内に多種多様なサンプル画像の値を書き込んでおき，SAD 値等で比較を行うことで検出することで人物検出を行うことができる．このとき二値化した値を用いることでメモリ領域を節約することができる．二値化した状態だと 1 ピクセルあたり 1 ビットで表現可能なため，横一列(800px)は 800bit = 100byte の記憶域で表現できる．DVD 相当の解像度 (プログレッシ

ブ)だと 720 x 480 = 345600bit = 43.2KB のメモリ領域があれば 1 フレームを格納することができる．

この方法の欠点としては，実装は簡易であるが大量のサンプルデータが必要となり，いくら二値化したとしてもメモリリソースに依存してしまうことが上げられる．大容量のメモリを搭載することが可能な場合に有効な手段の 1 つであると考える．

6.2 .サブピクセルに分割して画像を単純化する

図 6.1 に示すように，取り込んだ 1 フレーム分の画像データをさらにいくつかのブロック単位のサブピクセルに分割した上で，サブピクセルをいくつかのパターンに分類し，単純化することで人物の検出を行うことができる．

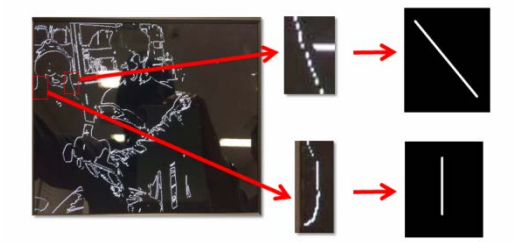


図 6.1 サブピクセル生成イメージ

具体的に説明すると図 6.2 に示すように，1 フレーム分の二値化された画像データを分割したサブピクセルに対して，横軸方向にスキャンを行う．このときに白い点であらわされている箇所を交点として検出を行う．それを数カ所行い，それぞれの交点に対して最小二乗法を適応し，直線を求める．この直線の傾きと適当に設定した閾値から，このサブピクセルのパターンを判断し単純化することができる．

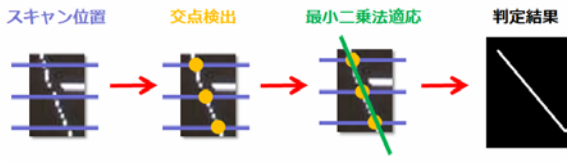


図 6.2 サブピクセル単純化イメージ

単純化した後，直線が続く部分(手足)などの人体に特徴的な部分を検出することで人物検出を行うことができる．

想定される欠点としては，実装が複雑になる点，精度がいかに適切に単純化できるかで大きく変化することがある．後者はサンプル数をある程度増やしたり，スキャンする箇所や近似の種類(二次関数など)を増やしたりすることである程度精度を確保することができると考える．

6.3 色情報とエッジの情報を合わせて使用する

エッジだけでは機械的に人物を検出することは難しいため，色情報のうち肌に近い色を残して併用することで人物検出を行うことができると考える．エッジに加えて肌色であるかどうかの情報 1bit を加え，合計 2bit を 1 ピクセルの情報とすることで検出精度を上げられると考える．

欠点は明らかに服装に左右されてしまう点が上げられるが，これは皮膚を覆わない顔などを特徴点として検出することである程度精度を上げられるのではないかと考えられる．また，色以外にもサーモカメラの温度情報などいくつかの人間の特徴的な情報を数値化できる機器を併用することで，さらに精度の向上が見込まれる．

7. まとめ

今回の目標では取り込んだ画像から人物検出を行うところまでの実装を目標としていたが，知識不足で人物検出の実装まで至らなかった．しかし，その過程で人物検出へと応用できるであろうアルゴリズムや技法を発見し，実際に回路に実装することができた．また実装の際には，論理合成にかかる時間も考慮し，RAM を使うなど精度と簡易さを両立することを目標に，工夫点を盛り込み実装することができた．考察にも記したように，今回実装した回路を応用することで十分に人物を検出することは可能であると考えるため，3 種類以外にも様々な応用方法を検討したい．

8. 出典

※1 社会環境医学講座(秋田大学)
http://www.mis.med.akita-u.ac.jp/~kata/image/sobelprew.html

テンプレートマッチングにおける類似性の検出

高江洲 慧， 武田 都子， 松田 祐希
沖縄工業高等専門学校 情報通信システム工学科

概 要

人物検出を行うため，その手法の一つであるパターンマッチング法を FPGA 上に実装することを目的としている．Level1 の課題では，類似度の比較に SAD（Sum of Absolute Difference）値を，テンプレート画像として 8bit の画素値を用いた．マッチングした箇所の色を変更することで対象物を検出する回路を実装した．Level2 では，出力結果から，類似性の度合いを視覚的に確認できるように改変し，画像データの一致度を複数段階で表現できるように改良した．シミュレーションではマッチング箇所の色が類似度により変化することが確認できた．

1 はじめに

自動車の自動ブレーキシステムなどに用いられる技術である人物検出機能を FPGA 上に実装することを目的とし，テンプレートマッチングによる人物検出回路の設計を行った．人物検出の手法としては，被探索画像を順にテンプレート画像を比較して類似度を算出するテンプレートマッチング法を用いた．類似度の算出方法には画素値から得られる SAD を用いることで高速な処理を目指した．Level2 では，類似度により描画を変更することで，類似度を視覚的に表現できるように改良した．これにより，例えばディジタルカメラの顔認識機能を使用し，複数人の顔が検出された場合や顔以外のものが検出された場合に，一致した度合いを含む曖昧性を考慮した判定が可能になると考えられる．

2 設計

実装した処理のアルゴリズムについて図 1 と図 2 にフローチャートを用いて示す．

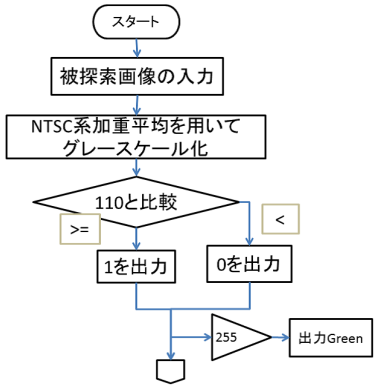


図 1: フローチャート 1

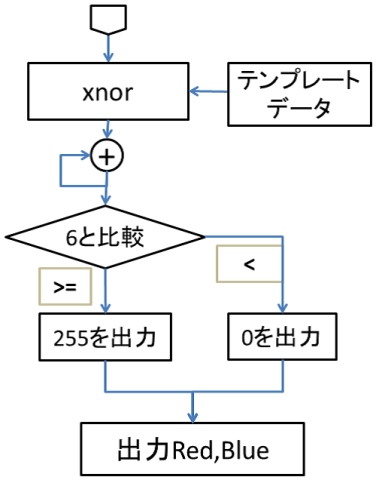


図 2: フローチャート 2

2.1 仕様

今回設計した回路の仕様について説明する．入力は被探索画像とし，テンプレートデータは設計段階で定められた 8bit のデータベクトルとする．出力は緑と黒で 2 値化された画像とし，マッチングした画素の色を変更することでマッチング箇所の描画を行った．色の変更は [緑 → 白，黒 → 赤紫] とする．Level2 では，マッチング箇所の描画パターンを増やした．より高い類似度でマッチした箇所の描画を [緑 → 白，黒 → 赤紫] と

し，これより低い類似度でマッチした箇所を [緑 → 薄青，黒 → 青] とした．これにより，類似度が視覚的にわかりやすく確認できるようになった．

2.2 Level1

設計仕様書のアルゴリズムを参考にし，回路の作成を行った．実際に作成した回路では，入力された被探索画像が緑と黒で 2 値化され，マッチした部分の画素を白と赤紫でそれぞれ出力される．シミュレーション結果は後述の図 11 で示す．この回路は実際に FPGA に実装し動作確認を行った．動作環境について表 1 に示す．この時のテンプレートには 8 ビットのデータベクトルに”00111100”を与えた．

表 1: FPGA の動作環境

Family	Spartan6
Device	XC6SLX45

被探索画像の入力から 2 値化までを行うブロック図を図 5 に示す．この処理には，rgbtoy と Binarization の 2 つの回路ブロックが用いられる．まず，被探索画像を R，G，B の 3 つのデータで読み込む．読み込まれたデータが rgbtoy に入力される．rgbtoy 内部では NTSC 系加重平均法を用いている．その計算式を式 (1) に示す．rgbtoy 内部のブロック図を図 3 に示す．

$$Y = 0.298912 * R + 0.586611 * G + 0.114478 * B \quad (1)$$

また，グレースケール化されたデータは Binarization に入力される．2 値化する際の閾値は”110”としている．閾値は, グレースケール化されたデータが 8bit であるため”0〜255”の範囲で表され，中央値である”127”を基準として考えた．今回使用した画像が暗かったため，閾値を低く設定した.Binaryzation 内部のブロック図を図 4 に示す． 2 値化されたデータはマッチング処理部と出力部に送られる．マッチング処理を行っているブロック図を図 7 に示す．マッチング処理部では Binarization の出力と ROM に格納されているテンプレートデータの xnor を 1bit ずつ計算し，Acumulator1 により 8bit の SAD の総和を求めている．ただしこの SAD は類似度が高いほど値が大きくなる．ROM1 からは 8 クロック毎

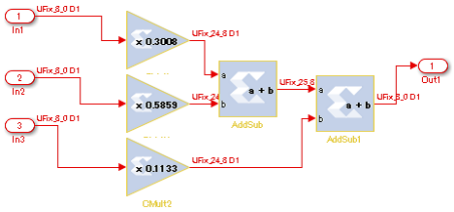


図 3: rgbtoy

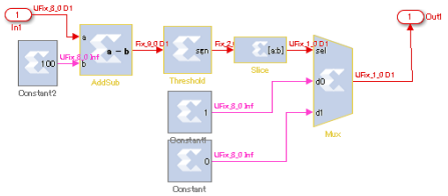


図 4: Binarization

に’1’ が出力され，Acumulator1 のリセット信号となっている．Acumulator1 の計算結果は Binarization1 に入力される．Binarization1 では，マッチング箇所の描画が行われている．Binarization1 の内部のブロック図を図 6 に示す．SAD が閾値未満であれば”0”が，閾値以上になると”255”が出力される．SAD は 8bit の xnor の総和であるため最大は”8”となる．これより，閾値は”6”と設定した． 出力部分のブロック図を図**??**に示す．出力部には 2 値化処理を行った Binarization からのデータと描画処理を行った Binarization1 の出力からのデータが入力される．Binarization のデータは Green の出力となるが，”0”と”1”で 2 値化されているため 255 倍されている．Binarization1 のデータは Red と Blue の出力となる．

2.3 Level2

Level2 では，描画処理のパターンを追加するために回路のマッチング描画部分に Binarization2 を追加した．Binarizatoin2 の内部のブロック図を図 9 に示し，Level2 のマッチング処理部分のブロック図を図**??**に示す．Level1 では Binarization1 の結果から Red と Blue の描画を行っていたが，Level2 では Binarization1 が Red の描画を，Binarization2 が Blue の描画を行う．Binarization1 の閾値を”5”とし，Binarization2 の閾値を”6”とした．これにより，Level1 のマッチング描画に加えて，類似度が劣る箇所を青と薄青で描画される．こ

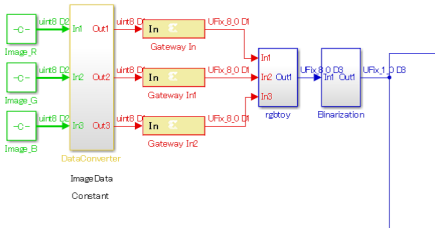


図 5: 入力 2 値化

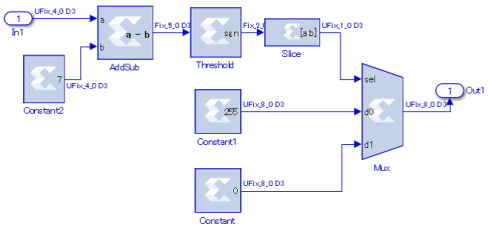


図 6: Binarization1

のときのシミュレーション結果を後述の図 12 で示す.

2.4 シミュレーション結果

lv1 と lv2 のシミュレーション結果を図 11 と図 12 にそれぞれ示す．lv1 では類似度の高い箇所に白と赤紫が描画されている．Level2 では類似度が少し低い箇所にもマッチング描画を行う処理を追加したので，Level1 で描画されていた箇所に加え，類似度がやや高い箇所に青と薄青の描画が行われている．

2.5 比較

ISE Design Tool を用いて回路合成を行った結果を表 2 に示し比較する．また，Level2 の回路を FPGA Generate を用いて回路デザインの確認を行った．これを図 13 に示す．LUT 数を比較すると，Level2 の回路の方が約 200 ほど多い．回路レイアウトを比較すると確認しづらいが，Level2 の方がやや複雑になっていることが分かった．Level2 で行った回路の変更はマッチング描画部分のみで，変更内容は Binarization2 を追加したのみであった．よって，Level1 と Level2 では差が少なかったと思われる．

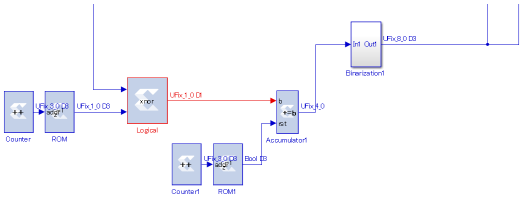


図 7: マッチング処理

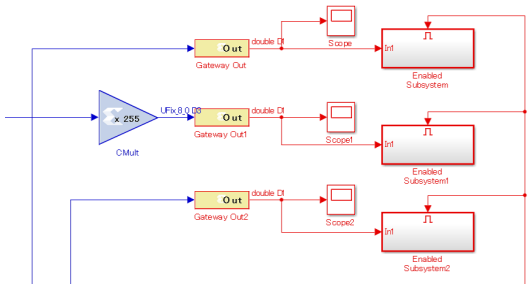


図 8: 出力部

2.6 今後の課題

今回作成した回路では，テンプレートをあらかじめ定めた 8bit データとした．しかし，実際にはテンプレートとして画像が用いられることが想定される．そのため，テンプレートに画像が入力できる回路を実装することが課題である．

3 まとめ

Level1 ではテンプレートマッチング法を用いて被探索画像から特定のデータを検出する回路の開発を行った．類似度に SAD を使用し，画像を 2 値化し比較することで，アルゴリズムの単純化と処理の高速化を目指した．シミュレーションで，マッチング箇所が白と赤紫で描画されることが確認できた．また，FPGA 上でも動作することが確認できた．Level2 では Level1 の回路をもとに，類似度を視覚的に捉えられることで，人間がわかりやすく確認できるようにすることを目的として改良した．シミュレーションでは類似度により 2 パターンのマッチング描画が行われていることが確認できた．また，回路の変更点も少なく，単純なアルゴリズムで実装することができた．回路規模の比較では，Level2 の回路の方が回路規模が大きくなっていることが確認できた．これは Level2 の回路は Level1 に回路

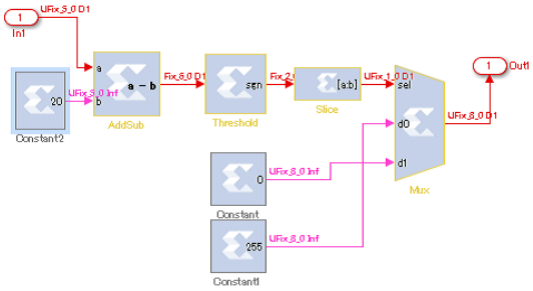


図 9: Binarization2

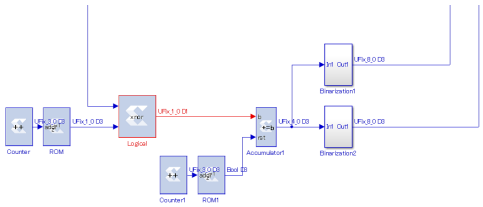


図 10: マッチング処理 (level2)

を追加という変更を行ったので回路規模も増加したと考えられる．課題としては，テンプレートデータを画像に置き換え，マッチング処理と出力をおこなうことが残った．

表 2: LUT 数

	Level1	Level2
LUT 数	1286	1481

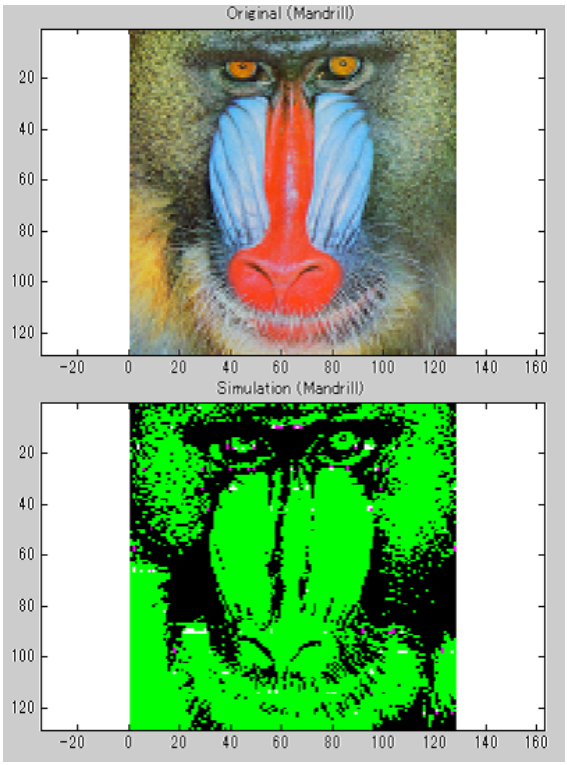


図 11: lv1 シミュレーション結果

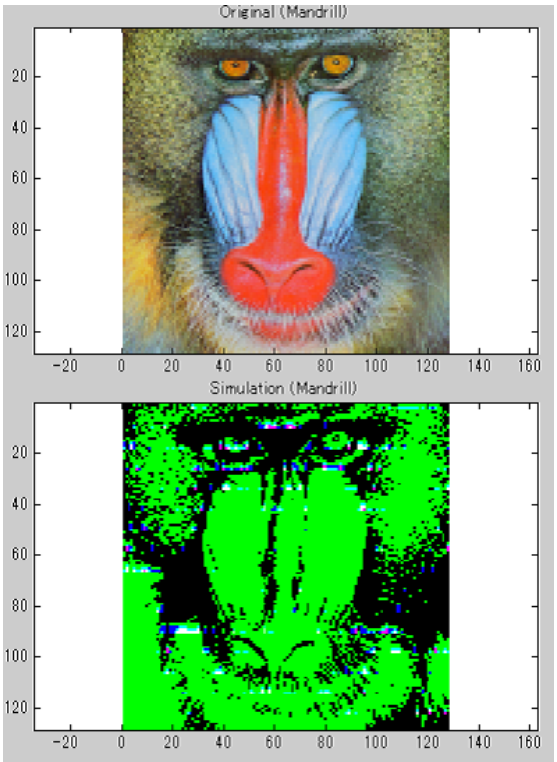


図 12: lv2 シミュレーション結果

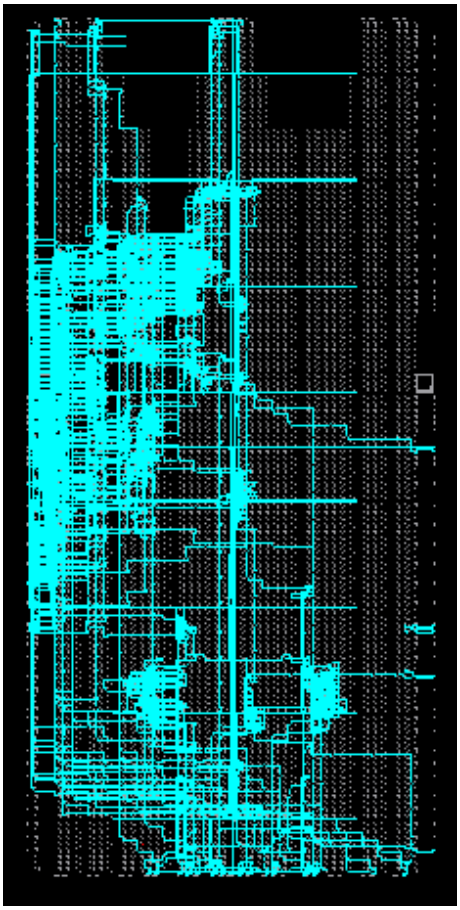


図 13: Level2

色特徴の検知による小面積人体検出回路の検討

与座章宙，菅野義貴，与座皇哉
沖縄工業高等専門学校4 年　情報通信システム工学科

概要

本論文では，テンプレートマッチング手法を用いた人物検出回路の設計を行った．**Lv1** の課題では被探索画像を二値化し，同じく二値化されたテンプレート画像と **EX-NOR** をとる事により一致度を検出する回路を構成した．しかし, 人物を特定するテンプレート画像を定義するには, 二値化された人物像と一致させたい領域を事前に抽出しなければならず，複数の人物を検出する際にはスケーラビリティに欠ける．その他, 多くの **ROM** やカウンター回路を必要とするため，回路規模が大きくなるという問題があった．そこで，**Lv2** の課題における人体検出法として, 高速動作と小面積化の両立を目的として，肌色検出回路を構成した. 取り込んだ画像の値が肌色と判定される範囲内に収まれば肌色を, それ以外は黒を出力させることで人物を検出する．また, テンプレート画像を用意しなくても人物の検出が可能となり, スケーラビリティも向上すると考えられる. さらに，回路はテンプレート画像（肌色）との比較だけで構成できるため,**ROM** やカウンター回路を削減できる．評価の結果，回路規模を約 **50**％削減することができた.

1　はじめに

近年，画像認識技術は様々な機器に搭載されており，技術の高度化に伴い応用範囲が拡大している．画像認識技術の一つとして人物検出があり，人物検出の手法の一つとしては，テンプレートマッチング法がある. 今回は小面積かつ高速な演算による人物検出を目的として，Lv1 の課題として SAD（Sum of Absolute Difference）値を用いて画像の一致度を比較し人物検出を行う回路を構成した. しかし，今回実装した Lv1 の回路では，SAD の算出やマッチング箇所の描画に多くの ROM やカウンター回路を必要とするため, 回路規模が大きくなるという課題がある. また，複数の人物を同時に検出するテンプレート画像を定義するためには, 二値化された被探索画像に対して，人物像と一致させるためのテンプレート画像を事前に抽出しなければならず，複数の人物検出の際にはスケーラビリティに欠ける. 一方，人物の特徴ベクトルを抽出することでスケーラビリティの向上を図る事も可能ではあるが，被探索画像から特徴ベクトルを抽出するための回路が必要となり，回路規模の増加や，処理性能の低下につながると考えられる. そこで Lv2 の課題では，高速化・小面積化並びに，スケーラビリティの向上の並立を目的とした回路を検討した．具体的には，人体の色は肌色が主であるという点に着目し, 肌色のみを検出する回路を構成した. 肌色検出は二値画像の比較のみで構成されるため, 回路規模を削減できると考えられる.

2　テンプレートマッチング法

画像が一致しているかを判断する手法の一つにテンプレートマッチングが存在する. テンプレートマッチングの計算には,SSD（Sum of Squared Difference）や NCC(Normalized Cross-Correlation)，ZNCC(Zero-mean Normalized Cross-Correlation) といった多くの手法があるが，本稿では，Lv1 の課題として，演算量が少なく高速な演算が可能である SAD（Sum of Absolute Difference）を用いる事とした．SAD は被探索画像をラスタスキャンし，同じ位置の画素の輝度値の差の絶対値の合計値により一致度を判定するアルゴリズムである. すなわち，ＳＡＤの値が小さいほど似ていると判断することができる. 本章では，Lv1 の課題として設計した SAD 値を用いた人体検出アルゴリズムについて述べ，次に色特徴に着目した人物検出アルゴリズムについて述べる.

2.1　Lv1：SAD 値を用いた人体検出

SAD 値とは Sum of Absolute Difference の略であり, 画素値の差の絶対値によって表される.I(x,y) を走査している範囲の x 座標と y 座標で表される画素値,T(x,y) をテンプレート画像の x 座標と y 座標で表される画素値とし式 (1) によって表される.

$$\mathrm{SAD} = \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} \mathrm{abs}(I(x,y) - T(x,y)) \qquad (1)$$

SAD 値が小さければ小さいほど, 走査している範囲の画素と, テンプレート画像の画素の差異が小さいことを意味する. すなわち，0 であれば完全に一致していると見なす事ができる．しかし，被探索画像を直接比較する場合, RGB すべての色を比較する必要が生じるため, 比較演算を三回実行しなければならず，処理性能が低下すると考えられる.

そこで, 被探索画像とテンプレート画像の RGB 値をグレースケール化し，さらに二値化する構成とした. その利点として，グレースケール化することにより比較処理が一回の演算で実現できる. さらに二値化を行うことで,SAD 値の計算が画素同士の排他的論理和の否定 (EX-NOR) で求めることができ, 計算回数を大幅に減らすことができる. グレイスケールの計算式を式 (2) に示す.

$$\mathrm{Y} = \mathrm{R} * 0.298 + \mathrm{G} * 0.586 + \mathrm{B} * 0.114 \tag{2}$$

EX-NOR による SAD 値計算の式を式 (3) に示す.

$$\mathrm{SAD} = \sum_{\mathrm{x}=0}^{\mathrm{X}-1} \sum_{\mathrm{y}=0}^{\mathrm{Y}-1} \overline{\mathrm{Gray(I(x,y))} \oplus \mathrm{Gray(T(x,y))}} \tag{3}$$

2.2 Lv2 : 色特徴に基づく人体検出

人物の特徴の一つに肌の色がある．同じ人種の肌色は色空間上である程度まとまった分布をとることが予想されるため，この色分布を予め作成して利用することで人物領域を抽出することができる. また，髪などの不要な情報を含まない顔領域のみの抽出が可能になるため，人物認識などへの発展も考えられる. 従来，肌色の検出法は画素の値である RGB をそのまま計算するのではなく,HSV 色空間に変換した後に肌色を判定するのが主だった．しかし,RGB から HSV へ変換するには除算と乗算が必須であり, 回路規模が大きくなると考えられる．そこで, 今回は RGB の比較だけで肌色を検出できるように構成した．今回用いた肌色を判定するためのモデル式を, 式 4 に示す．RGB の値が式 4 の条件をすべて満たすなら, 肌色として判定する.

$$\begin{cases} 30 < R \\ 30 < G < 180 \\ 50 < B < 220 \\ 10 < R - G \end{cases} \tag{4}$$

先ず，肌色の色分布を調べるために，肌色に見えるカラーコードサンプルを事前に調べ, それらの範囲を定義した．最初は式 5 の範囲にあるものが肌色だと定義して, シミュレーションを行った．次に，人物写真を用いて肌色なら白, 肌色でないなら黒いドットが出現するような回路でシミュレーションを行った．結果は全て黒が出力され, 白いドットは一つも見当たらなかった．これは，肌色とみなす範囲が狭すぎて検出できなかったと考えられる.

$$\begin{cases} 221 < R < 247 \\ 169 < G < 221 \\ 130 < B < 208 \end{cases} \tag{5}$$

式 5 の範囲ではなにも検出できなかったので, 新たに式 6 を定義した.

$$\begin{cases} 180 < R \\ 150 < G < 200 \\ 130 < B < 180 \\ 30 < R - G \end{cases} \tag{6}$$

そこで，肌色の範囲を式 6 のように定義した. この範囲を肌色にしたのは,WEB サービスで任意のカラーコードの色を生成する仕組みを使い, 生成した範囲の色は肌色に見えると判断できたためである．この範囲を超えた場合, 以下の 4 つのような色になってしまう.

1. 赤色が弱いと全体的に暗くなり, 肌色ではなく黄土色のような色になってしまう.
2. 赤と青が十分に明るくても, 緑が暗いと紫のような色になってしまう．しかし, 緑色が強すぎても黄緑のような色になってしまう.
3. 赤と緑が十分に明るくても, 青が暗いと黄色のようになってしまう．しかし, 青色が強すぎても紫のような色になってしまう.
4. 肌色を構成する三原色で, 赤と緑は青より強く, 肌色に見せる要因になっている．しかし, 赤と緑が同じ値に近づくと色は白っぽくなってしまう.

肌色に見えるかどうかのテストは, 自分で見て判断したのと, 友人一人に見てもらい, 肌色と呼べる範囲を探した.

式 6 の範囲でもシミュレーションした. 今度は黒一色ではなかったが, 背景の白しか検出できておらず, 肌色は検出できていなかった．理由として式 5 も式 6 の肌色も, ディスプレイで見ているカラーコード上での肌色であり, 現実の人間の肌色とはかけ離れているからだと考えた．もう一度正しい範囲を見つけるべきだと考え, 何度も調節を行って式 4 の範囲にある色が肌色と判定される事がわかった．森の中で人が 5 人いて集合写真

をとっている写真を使いシミュレーションを行った. シミュレーション結果では, 焦げ茶色の服の色と濃いめの肌の色, 日光の白と色が薄い肌の白 (肌) 色をきちんと分けて, 顔と手の肌色だけ検出できていた. 式 4 の範囲なら背景と人物の肌色を分けて, 肌色だけを識別できると考え, 今回の回路に用いることにした.

3　回路の仕様と設計

今回設計した回路のける共通部分の仕様を説明をする.

回路はカメラからの映像を取り込み, 映像処理, ディスプレイへの映像出力の三段階で構成されている.

カメラ (VGA) からの映像は一度メモリに保存されて, メモリに蓄えられたデータを映像処理部に送り, 出力部のメモリに一時的に保存し, そこからディスプレイに出力する回路となっている.

Lv1 と Lv2 では映像処理の部分をそれぞれ個別に設計した.

4　Lv1 の処理部の構成

カメラから取り込んだデータを処理する回路の動作について説明する.

全体として, 二値化・比較・出力の順で処理が行われている. 処理部には 1 ドットずつメモリから画素値が読み出される. 読み出された RGB の情報を持つ画素値は，それぞれ定数倍された後に合算され,256 階調のグレースケールとなって出力される.

今回は, 式 (2) を用いて,RGB をグレースケール化を行った.

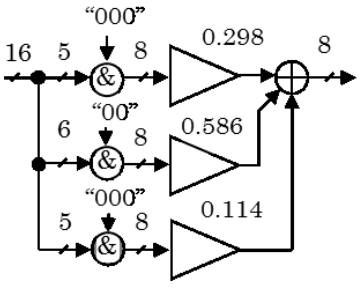


図 1: グレースケール化ブロック図

4.1　グレースケールの二値化

グレースケール化されたデータは，そこから比較器に入り, グレースケールの値が 127 より大きければ 1, 小さければ 0 となって二値化が行われる. 二値化されたデータはその先で分岐し, 片方は EX-NOR に, 他方は出力の G(緑) にそのまま接続される.

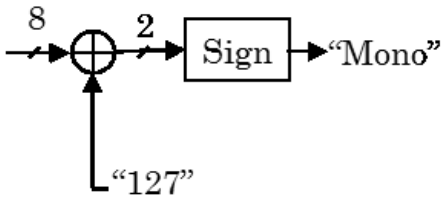


図 2: 二値化ブロック図

4.2　二値化データの SAD 計算

EX-NOR の片方に二値化したデータを入力させ, 他方には ROM1 からの値を入力する. この ROM1 にはテンプレート画像とする 1 行ベクトルを格納しておき,1 列ずつ二値化されたデータと比較する. ROM1 のアドレス入力にカウンターを設置し,ROM1 の中身を順番に繰り返し走査するようにしている. EX-NOR に ROM1 からのデータと二値化されたデータが入力されているので, 両者が一致しれば 1 を, しなければ 0 を出力するようになっている.

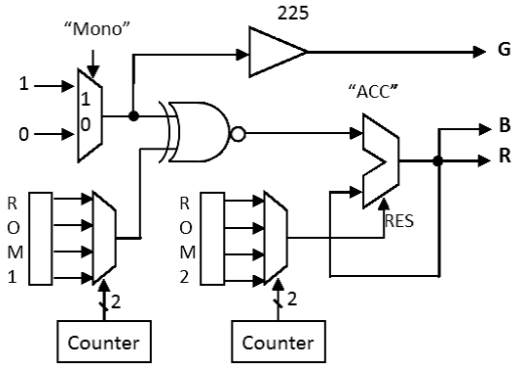


図 3: SAD 計算ブロック図

4.3 SAD 値のしきい値

EX-NOR の先にはアキュムレーターを設置し, 計算された一致数を加算していく. EX-NOR とアキュムレータで, 絶対値を取り総和を求め,SAD 計算をしている. 加算されていった値がしきい値を超えたら出力の R(赤) と B(青) に出力される. このアキュムレーターを動作させ続けていると, アキュムレータの値が設定したしきい値を飽和してしまう. しきい値を超え続け, 一致した判定ををを出力し続けてしまうので, 定期的にアキュムレーターの値を 0 にリセットする必要がある. そのリセットする周期を ROM2 に格納しておき, アドレス入力にカウンターを設置し ROM2 を繰り返し走査, 出力するようにしている.

5 Lv2 の処理部の説明

今回設計した肌色を検出する回路は, 赤 · 緑 · 青の値がそれぞれ設定した範囲内にあるかを判断するためのフラグ値を出力させ，フラグの論理積を取ることで, その結果に応じて肌色と黒を出力する構成となっている.

5.1 赤の明るさの判定

8 ビットで入力された赤の値を, 明るさが十分かを判断し,1 ビットのフラグを出力する. 今回は赤の明るさが 30 を超えないと明るさが不十分だとして 0 を, 超えたら明るさは十分だとして 1 を出力するような回路を設置した.

5.2 緑の明るさの判定

8 ビットで入力された緑の値を, 明るさが十分かを判断し,1 ビットのフラグを出力する. 今回は緑の明るさが 30 と 180 の間にあれば 1 を, そうでないなら 0 を出力するような回路を設置した.

5.3 青の明るさの判定

8 ビットで入力された緑の値を, 明るさが十分かを判断し,1 ビットのフラグを出力する. 今回は青の明るさが 50 以上あれば 1 を, そうでないなら 0 を出力するような回路を設置した.

5.4 赤と緑の差の判定

8 ビットの赤と緑を入力し, その差を判断して,1 ビットのフラグを出力する. 今回は赤と緑の明るさの差が 10 より大きくて赤が緑より強ければ 1 を, そうでなければ 0 を出力するような回路を設置した.

これまでに説明した, 赤の明るさ判定, 緑の明るさ判定, 青の明るさ判定, 赤と緑の差の判定をする回路を図 4 にまとめる. 上から, 赤の明るさ判定, 緑の明るさ判定, 青の明るさ判定, 赤と緑の差の判定をする回路の順で並んでいる.

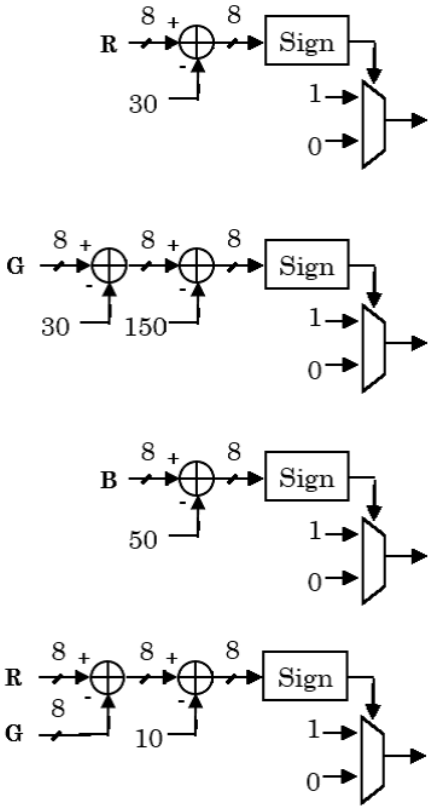


図 4: 肌色検知フラグ器

5.5 肌色かどうかの判定

4 つのフラグを入力とし, 論理積をとった値を 1 ビットの結果として出力する. 今回の肌色として判定されるためには,RGB の値が式 (4) に示した条件をすべて満たしていなければならない. 赤の明るさ判定, 緑の明る

さの判定, 青の明るさの判定, 赤と緑の差の判定する回路からのフラグを入力する. 入力された 4 つのフラグがすべて 1 となるとき, 肌色だと判断できるので, 論理積の 1 ビットの結果を出力する回路を設置した.

5.6 肌色の出力

肌色と判定された 1 ビットのフラグを入力し,8 ビットの RGB を 1 つずつ, 合計 24 ビットを出力する. 肌色かどうか判定された 1 ビットの出力を,255 倍して R へ,200 倍して G へ,180 倍して B へ出力する回路を設置した. 判定された結果を 255 倍して R,G,B に出力した場合白が出力される. 今回は肌色を出力したいので, 肌色に見えるような色を出力するようにした.

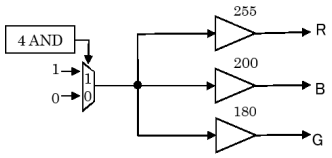


図 5: 肌色出力ブロック図

6 出力結果

作成した 2 つの回路のシミュレーション結果を示す. 比較するために, 図 6 の画像を入力した.

図 6: シミュレーション入力画像

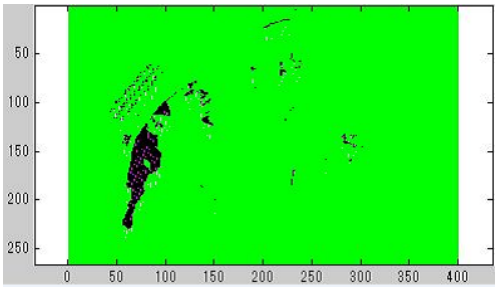


図 7: 回路 1 シミュレーション出力

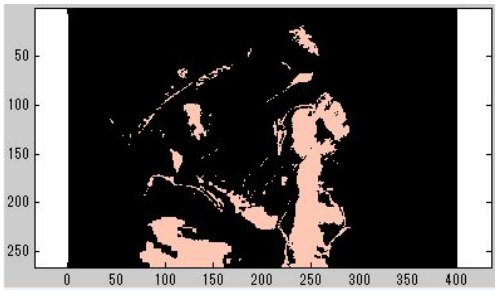


図 8: 回路 2 シミュレーション出力

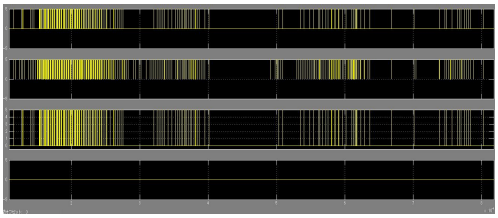


図 9: 回路 1 シミュレーション出力

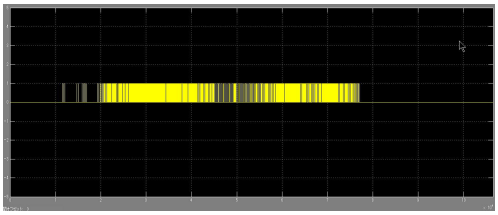


図 10: 回路 2 シミュレーション出力

図 9 の回路 1 のシミュレーションでは, 一致しなかったところが黒か緑が, 一致したところが赤紫か白が出力されている. 今回用いたテンプレートベクトルは 11000011 で, 目のあたりを判定しようとした. 探索が成功しておらず, 髪の毛の黒い部分が僅かな範囲だけ一致した様子が見られる.

図 10 回路 2 のシミュレーションでは, 全体的に暗くなり, 肌色と判定されたところだけが肌色に塗られている. 明るい肌色は全て検出できている事がわかる. 暗い肌色は茶色に近いので, 肌色としては検出できなかと考えられる. このことから, 二値化を行い検出するよりも, 肌色検出を用いた方が人物検出としては精度が高いと言える.

回路 1 のテンプレートベクトルは一度入力したら固定長なので, 静止画のごく一部の横の長さが一致した部分しか検出できない. しかし, 回路 2 では肌色なら検出できるので静止画か動画かに関わらず検出でき, 横の長さが一致してなくても検出できると考える.

今回, 回路 1 は FPGA チップへ実装した. デバイスは Spartan3,XC3S400-4PQ208 へ書き込んだ.

回路 2 は開発ツールでの配線がうまく行かず, ファイル生成がうまく行かなかったため, 実機への実装には至っていない.

7 性能比較

回路 1 と 2 を XillinxISE で合成して得られた結果を示す.

性能	回路 1	回路 2
LUTs	110	57
FlipFlop	110	57
最大クロック周波数 [MHz]	193.9	-
最小クロック周期 [ns]	5.158	-

回路 2 は回路 1 に比べて,LUT 数が 110 から 57 へと,48%減少している.FlipFlop 数は 110 個から 57 個へと,48%減少している.

今回は回路 2 の最大クロック周波数, 最小クロック周期を解析することができなかったため, 比較ができていない.

8 今後の課題

開発ツールをうまく扱うことができず, 回路 2 は最大クロック周波数などを出すことができなかった. 回路面積の比較はできたが, 時間制約を比較することができなかった. 自分が使う開発ツールをちゃんと使えるようにして, すべての項目で比較できるようにしたい.

回路の速度低下を抑えて, より高速化を目指したい.

回路 2 には, 最適化に繋がりそうな箇所が 4 つ見つかった.

AND 回路の前段にある 4 つの色を判定する回路の出力は,1 ビットの計算結果を使ってマルチプレクサから 1 ビットのデータを出力していた. マルチプレクサに入力している 1 ビットの計算結果を使えば,4 つのマルチプレクサを省略することが可能で, さらなる回路の縮小に繋がると考える.

Lv2 で作成した回路は肌色を検出するが, 人物の特定までには至らない.

Lv1 で特徴的なベクトルの参照を同時に行うことにより, 肌色かつ似た特徴の形を捉えることができ, 人体検出から顔の領域を比べる人物特定まで発展できると考える.

テンプレートマッチング法に着目した画像探索

Akiyuki Abe* , Kazuki Kondo* and Takaharu Nakao**
*Department of Advanced Production and Information Systems Engineering Course, National Institute of Technology, Ariake College
**Department of Electronics and Information Engineering, National Institute of Technology, Ariake College

***Abstract*—本レポートは LSI デザインコンテストの課題であるテンプレートマッチング法に着目した画像探索ユニットの設計について我々が行ったことを報告するものである。Level1 課題であるテンプレートマッチング法に着目した人物検出設計とその動作結果について報告する。**

I. はじめに

我々は、2014 年 4 月より FPGA をテーマとして研究を行ってきた。この研究を通して学んだ技術を実践的に身につけるために、画像処理や画像探索など複雑な処理が課題となっている本コンテストへ参加した。

本コンテストの Level1 の課題である画像のテンプレートマッチングを行う回路の設計を行った。期限内に課題に取り組むためにラピッド・プロトタイピングを採用した。ラピッド・プロトタイピングとは試作品を最初に作ることで開発時間を短縮することを目的とした開発手法である。

II. ラピッド・プロトタイピング

図 1 にラピッド・プロトタイピングの概要を示す。設計仕様書で公開されているテンプレートマッチング法で画像探索を行えるか調べるために、Arduino を用いたラピッド・プロトタイピングを行った。ラピッド・プロトタイピング後に、FPGA に Arduino と同じアルゴリズムを FPGA 上に実装した。

A. 開発工程表

FPGA で人物探索を行う為に開発工程表を作成した。表 1 の開発工程表には Arduino、Visual Studio アプリ、FPGA で実装すべき工程や予定、実際の進捗、行えなかった工程が示されている。

B. テンプレートマッチング法

公開されているテンプレートマッチング法の SAD を表す式を式(1)に示す。

$$\mathrm{SAD} = \sum_{x=0}^{X-1}\sum_{y=0}^{Y-1}|I(x,y) - T(x,y)| \quad \cdots(1)$$

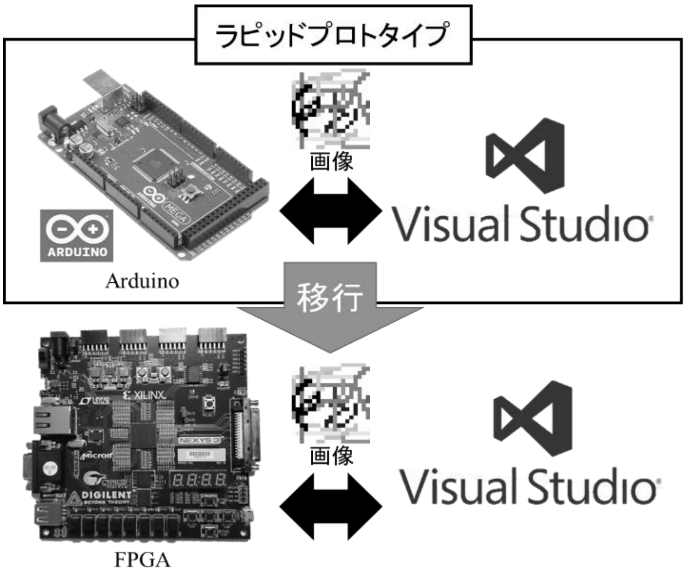


図 1：ラピッドプロトタイプの概要

表 1 ：開発工程

対象	行程内容	☆：進捗予定 ✔：実際の進捗 ○：予定通りの進捗 ×：未実行																							
		12月18日~31日								1月1日~16日								1月17日~26日							
Arduino	文字通信	○																							
	画像データの保存			✔	☆																				
	画像データの送受信				✔	☆																			
	グレースケール化					✔		☆																	
	テンプレートマッチング					✔			☆																
	枠の追加						✔	✔		☆															
Visual Studio アプリ (PC)	文字通信	○							✔		☆														
	画像の表示		○																						
	画像のRGBデータ解析		✔	☆																					
	画像データの送受信			✔	✔	☆			✔					☆											
	画像の生成					✔	☆																		
FPGA	文字通信								✔		☆														
	画像データをメモリに保存									✔	✔	☆													
	メモリから画像データを読みだし										✔	✔	☆												
	画像データの送受信								✔					☆											
	グレースケール化	×	×	×	×	×	×	×	×	×	×	×	×	×	☆	×	×	×	×	×	×	×	×	×	×
	テンプレートマッチング															☆		✔	✔	✔	✔	✔	✔	✔	✔
	枠の追加												✔	✔	✔	✔	○								

SAD は画素値の絶対値の総和によって表され、被探索画像とテンプレート画像の x 行目と y 列目の画素値を比較する。SAD が小さい程被探索画像とテンプレート画像の類似性が高い。

C. Arduino

デバックやコンパイルを素早く簡単に行うことができる Ardino を用いてラピッド・プロトタイピングを行った。Ardino はマイコンの一種である。Ardino の開発環境である Arduino IDE は C/C++ベースで開発を行える上に、USB 接続で簡単にプログラムを書き込めることから、簡単にアルゴリズムの実装を行うことが出来る。本稿で使用したのは Ardino Mega である。Arduino Mega は一般に普及している Ardino Uno よりもメモリ領域が広く画像探索に適しているボードである。しかし、コンテスト課題の画像データはサイズが大きく保存出来なかった為、自作の画像データを使用した。被探索画像のサイズは 19*20 ピクセル、テンプレート画像のサイズは 5*5 ピクセルである。

被探索画像とテンプレート画像を別々に受け取り、グレースケール化とテンプレートマッチング、枠を追加した画像を送信する Arduino のフローチャートを図 2 に示す。一部カラーで同色になっている箇所は図 2 と図 3 が同じデータを操作していることを示す。Ⓐが非探索画像、Ⓑがグレースケール化された非探索画像、Ⓒがテンプレート画像、Ⓓが枠を追加した非探索画像である。

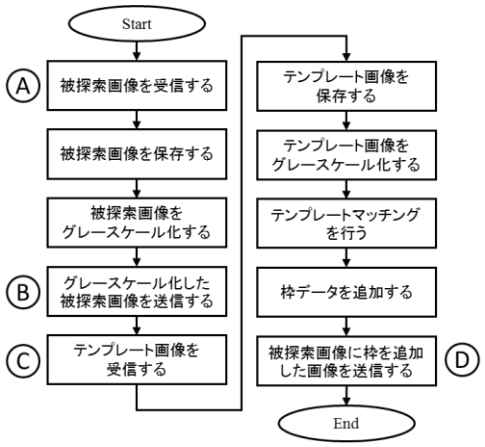


図 2：Arduino によるテンプレート
マッチングアルゴリズム

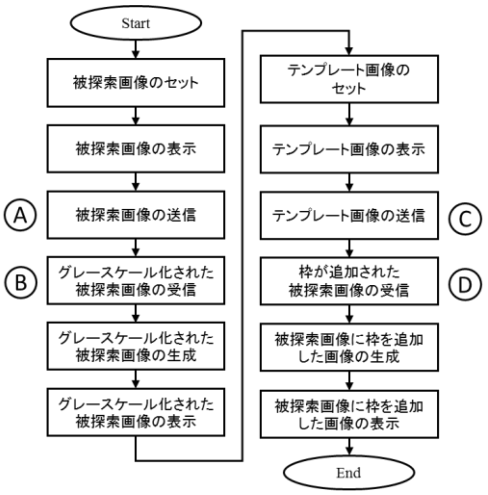


図 3：Arduino-アプリケーション間の
画像送受信アルゴリズム

D. Visual Studio

Arduino に画像データを送信する為に、Visual Studio を用いてアプリケーション開発を行った。アプリケーションに被探索画像とテンプレート画像をセットし、1 ピクセル毎に保存されている RGB データを解析し、シリアル通信で RGB データを送信する。送信後は被探索画像データ量を受信するまで待機し、受信した画像データを基に画像を生成する。画像のセットや表示、受信、生成を行う Arduino 向けのアプリケーションのフローチャートを図 3 に示す。

E. Arduino とアプリの動作結果

図 4 は Visual Studio で作成した Arduino 向けアプリケーションの動作図である。左下が被探索画像、右下がテンプレート画像、中下が受信した RGB データを基に生成した画像である。画像がぼやけているのは、Visual Studio の写真を表示するボックスである PictureBox が画像の拡大に失敗したためである。

赤枠で囲まれた部分がテンプレート画像の箇所である。テンプレート画像の箇所を外枠で囲っていることから、テンプレートマッチングが行われていることが確認出来る。

図 4：Arduino 向けアプリケーションの動作

III. FPGA への実装

図 4 で Arduino と Visual Studio で作成したアプリケーションでテンプレートマッチングによる画像探索が可能であることを確認した後、FPGA にテンプレートマッチングのアルゴリズムを実装した。

再現性を持たせる為に、対象とする被探索画像とテンプレート画像は Arduino で使用した画像とした。また HDL の記述レベルは全て RTL とした。

A. 工程の変更

2016 年 1 月 10 日頃に FPGA 上にテンプレートマッチングのアルゴリズムを実装する時間が不足すると判断して以下の工程を変更することで開発時間の短縮を図った。

1. グレースケール化を行わず赤色のみのテンプレートマッチングを行う。
2. 被探索画像データとテンプレート画像データを一括して FPGA に送信する。

FPGA に実装したアルゴリズムのフローチャートを図 5(左)に示す。図 5(左)は図 2 処理を簡略化していることが分かる。また、FPGA 用にアルゴリズムを変更したアプリケーションのフローチャートを図 5(右)に示す。図 5(左)と同じく図 5（右）は図 3 の処理を簡略化していることが分かる。

B. テンプレートマッチング法

FPGA 上で式(1)を実行するにあたり、順序回路ではなく組み合わせ回路を採用した。これは画像のカウンタ(x,y)が変わると同時に素早く、メモリに保存されている画像データのアドレスを計算する為である。これにより状態遷移による時間のロスを減らした。

C. シミュレーション

本来であれば Active-HDL によるシミュレーションを行うところである。しかし、本稿で作成した画像探索ユニットはシリアル通信も含めた設計になっているため、全体のシミュレーションを行うことが出来なかった。

ただし、本稿においては Ardino と Visual Studio を用いたラピッド・プロトタイピングによりテンプレートマッチングのアルゴリズムの再現性を保ったまま開発を行った。したがって、ラピッド・プロトタイピングが FPGA のシミュレーションの代替とすることが出来た。

D. メモリ

Arduino では使用できるメモリ量に制限があり、20*19 ピクセルの被探索画像までしかテンプレートマッチングできなかったが、FPGA ボードには外部メモリが 48MB

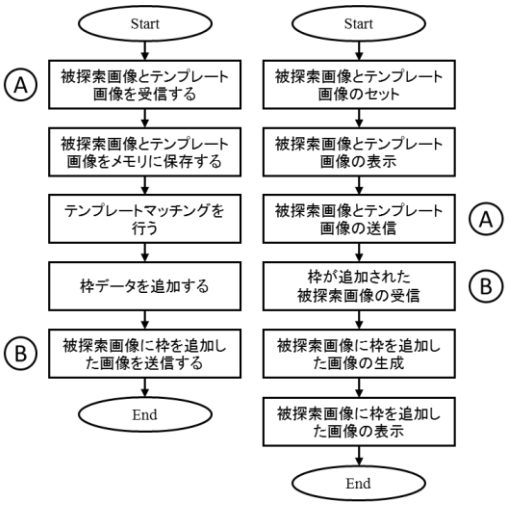


図 5：FPGA によるテンプレートマッチングアルゴリズム(左)と FPGA-アプリ間の画像送受信アルゴリズム(右)

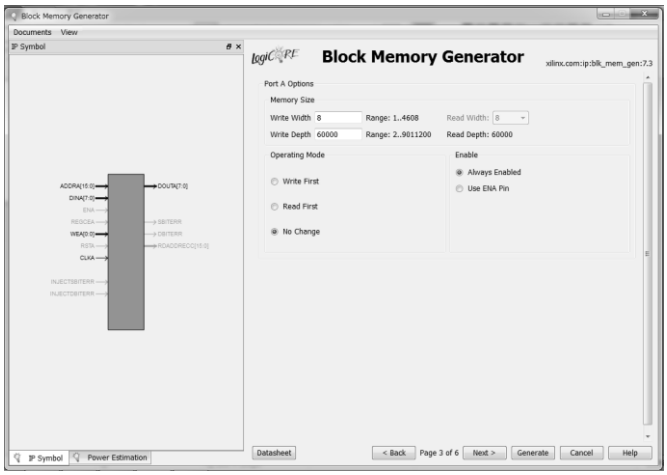


図 6：Block Memory Generator の設定画面

搭載されている為 640*480 ピクセルの画像に対しテンプレートマッチングを行うことができる。

図 6 にメモリを確保するために使用した Block Memory Generator の設定画面を示す。IC の形をした左図が Block Memory Generator によって作成されるメモリである。

CLK はメモリを動かすクロック、ADDRA は書き込み/読み込みの時に使用するメモリのアドレス、DINA はメモリにデータを入力するビット配列、DOUTA はメモリに保存されているデータを受け取るビット配列、WEA は書き込み/読み込みモードを制御するビットである。

Write Width と Write Depth でメモリの容量の大きさを指定することが出来る。Write Width はメモリに書き込み/読み込みを行う際に使用するビット配列の長さ、Write Depth はビット配列数の深さを指定する。

図 6 の設定では 60kB のメモリを確保した。

E. FPGA の動作結果

図 7 は Visual Studio で作成した FPGA 向けアプリケーションの動作図である。中下の黒枠で囲まれた部分がテンプレート画像の箇所である。テンプレート画像の箇所を内枠で囲っていることから、テンプレートマッチングが行われていることが確認出来る。本システムの論理合成には、ISE14.7 に含まれる XST を用いた。合成した結果の回路規模を表 2 に示す。

IV. 最終レポート提出後の改良点

A. FPGA の動作結果

図 8 はⅢで開発したテンプレートマッチングを行う FPGA 向けアプリケーションを改良した画面である。改良点は被探索画像とテンプレート画像の画素数を増やしたことである。

しかし、開発環境である ISE14.7 が提供する Block Memory Generator が構築するメモリ容量の上限は幅 8Bit 時に深度 60000Bit だったので 60kB しか用意出来なかった。以上の条件を踏まえ、被探索画像を 100*100 ピクセル、テンプレート画像のサイズを 10*10 ピクセルまで大きくすることが可能となった。図 8 の被探索画像は Level1 の課題に指定されているオリジナルの被探索画像をトリミングしたものである。

また、この問題はメモリ量の不足によるものであるため、更にメモリ量を確保する機能を使用することが出来れば、Level1 の課題に指定されているオリジナルの被探索画像 640*480 ピクセル、テンプレート画像 10*40 ピクセルの画像でテンプレートマッチングすることが可能であると思われる。

B. グレースケール化

公開されている NTSC 系加重平均法の式を式(2)に示す。

$$Y = 0.298912 \times R + 0.586611 \times G + 0.114478 \times B \cdots (2)$$

R は赤、G は緑、B は青の濃度を表すパラメータである。各パラメータの範囲は 0~255 である。

グレースケール化する際は Y 値を、R,G,B に全て代入し 1 ピクセルの R,G,B 値を同値にする。R,G,B 値が同値になると白~灰~黒で色を表現することができる。

Ⅲで開発したテンプレートマッチングを行う FPGA 向けアプリケーションは赤色である R のみを使用していたため、最終レポート提出後は RGB 値を全て使用するグレースケール化の実装を目指した。

図 9 はグレースケール化を行うアルゴリズムのフローチャートを示す。

まず Y 値が小数値になることを防ぐ。小数値を扱う場合、小数値の精度が原因で数値の桁落ちが発生する可能性があるためである。まず NTSC 系加重平均法を変換する。変換後の式を式(3)に示す。

図 7：FPGA 向けアプリケーション動作

表 2：回路規模

	Use(Utilization)
Slice Registers	339 (1%)
Slice LUTs	627 (6%)
bonded IOBs	32 (5%)

図 8：最終レポート提出後のアプリケーション動作

$$Y' = 298912 \times R + 586611 \times G + 114478 \times B \cdots (3)$$

次に求めた Y'を 1000000 で割った時の余りを mod 関数で求める。Y'から Y'の余りを引くと最大で左から 3 桁までが Y 値となる。

次に除算を行う。この時 Y は小数にならない。Y 'を Y に変換する式(4)に示す。

$$Y = Y' \div 1000000 \cdots (4)$$

最後に四捨五入処理を行う。余りが 500000 以上の時、Y'に 1 を足して Y 値を求める。

以上のグレースケール化のアルゴリズムを FPGA 上に実装し、FPGA ボードに搭載されている LED で動作のチェックを行った。しかし、Ⅲで作成したアプリケーションにグレースケール化のアルゴリズムを組み込むことは出来なかった。

V. まとめ

我々は FPGA の設計能力を実践的に身につけるために、Level1 の課題クリアを目標として本コンテストに取り組んできた。

まず、限られた時間の中で人物検知ユニットを作成するために、ラピッド・プロトタイピングを採用して開発期間の短縮を目指した。しかし、予想以上に FPGA の開発に時間がかかった為、Arduino に実装した一部の機能を省くことで更なる開発時間の短縮を図った。その結果、赤色に対するテンプレートマッチングを行う演算ユニットを作成することに成功した。

また、ラピッド・プロトタイピングを採用したことで、開発の初期段階でシステムの構想を掴みアルゴリズムの作成に集中することが出来た。

最終レポート提出後はメモリの増設とグレースケール化の実装に着手した。メモリは 60kB まで増設し 100*100 ピクセルの被探索画像に対しテンプレートマッチングを行うことが出来た。グレースケール化の実装では、アルゴリズムの作成と動作チェックを行ったが、単体のみの実装しか行えず、Ⅲで作成したアプリケーションに組み込んで実装するまでには至らなかった。

VI. 今後の予定

本稿では、ラピッド・プロトタイピングを用いて、VHDL で FPGA 開発を行う前に C 言語と Arduino を組み合わせたシステム開発を行った。これにより、アルゴリズムを綿密に設計した上で FPGA に実装することが出来た。したがって、ラピッド・プロトタイピングは FPGA の教育システムに 適した開発手法と言える。有明高専では来年も LSI デザインコンテストでラピッド・プロトタイピングを採用する予定である。

参考文献・Web サイト

[1] 鳥海 佳孝：『ディジタル・デザイン・テクノロジ No.13』 / CQ 出版社

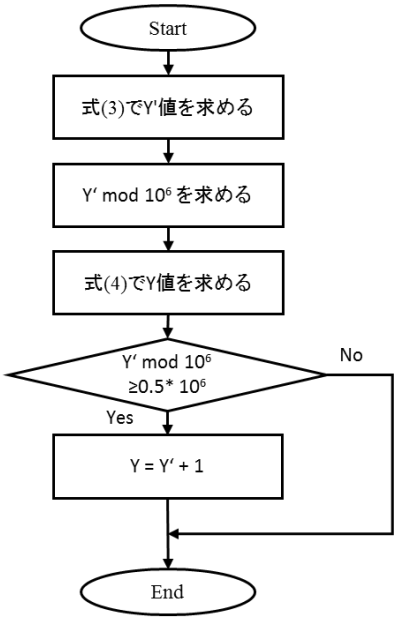


図 9：グレースケール化のアルゴリズム

色検出による人物検出回路の設計

チーム名：チェリーボーイズ

山田　賢斗　　豊川　雄大　　新垣　勇真

The design of the human detection circuit according to color detection

Kento YAMADA , Yudai TOYOKAWA , Yu-ma ARAKAKI
Department of Advanced Electronics Information Technology for Production System, Okinawa Polytechnic College, 2994-2 Ikehara, Okinawa-Shi, Okinawa,904-2141 Japan
E-mail: j1521324@okinawa-pc.ac.jp, j1521318@okinawa-pc.ac.jp
J1521301@okinawa-pc.ac.jp

１．はじめに

我々は、肌色を使用して人物検出を試みた。すでに、川島賢二ら[1]が HSV 変換を用いて肌色で人を検出するという手法と犬飼芳久ら[3]が用いて YCbCr 変換により人物の検出を行っていた。そこで我々も、HSV 変換と YCbCr 変換を用いての人を検出方法で行った。

２．処理の流れ

カメラから入力した動画像データを RGB 成分へ分解し、次に HSV 変換、YCbCr 変換を行う。その後、変換したデータ HSV、YCbCr を使って薄橙色の検出を行うための範囲の選定をする。最後にカメラからの光ノイズを除去するためにフィルタを通し、動画像として出力する。

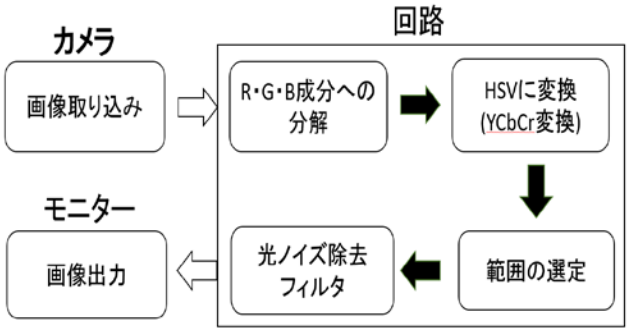


図１　処理の流れ

次に RGB-HSV 変換、YCbCr 変換式について

RGB→HSV 変換式

R が max のとき

$H = 60\times\{(B - G) / (max - min)\}$

G が max のとき

$H = 60\times\{2+ (R - B) / (max - min)\}$

B が max のとき

$H = 60\times\{4+ (G - R) / (max - min)\}$

RGB→YCbCr 変換式

$Y = 0.299\times R + 0.587\times G + 0.114\times B$

$Cb = -0.169\times R - 0.385\times G+0.500\times B$

$Cr = 0.500\times R -0.419\times G-0.081\times B$

$H = \tan^{-1} Cr/Cb$

３．HSV, YCbCr 変換のシミュレーション

MATLAB 2013b を利用してシミュレーションを行った。検知した部分を赤色で塗りつぶし、図 2 にシミュレーション結果を示す。

元画像

HSV 変換

YCbCr 変換

図２　シミュレーション結果

図 2 で示した通り、HSV 変換のほうが良好な結果となった。まずは、HSV 方式で実装することとした。

４．HSV 変換方式での実装

HSV 変換の処理を図 3 に示す。

```
if(b < r && g < r)begin↓
        if(b < g)min=b;↓
        if(g < b)min=g;
rh = 60*(b-g/r-min);↓
rv = r;↓
rs = 255*(r-min/r);↓
end↓

else if(r < g && b < g)begin↓
        if(r < b)min=r;↓
        if(b < r)min=b;↓
rh = 60*(2+(r-b/g-min));↓
rv = g;↓
rs = 255*(g-min/g);↓
end↓

else if(r < b && g < b)begin↓
        if(r < g)min=r;↓
        if(g < r)min=g;↓
rh = 60*(4+(g-r/b-min));↓
rv = b;↓
rs = 255*(g-min/g);↓
end↓
```

図 3　HSV 処理

H,S,V の各範囲は H<=30、50<=S<=255、50<=V<=255 としている。

HSV 実装結果を図 4 に示す。

元画像　　HSV 変換

図 4　HSV 実装結果

シミュレーション結果では HSV 変換の方が良好な結果となっていたが、回路に実装すると誤検出が多かった。

５．YCbCr 変換方式での実装

YCbCr 変換では、Cb,Cr を使用すればよいため、図 5 のブロック図で Cb、Cr の回路だけでよい。下のブロック図で Cr の範囲の選定を行っている。我々は、誤検出を減らすために Cb と Cr のそれぞれの選定範囲を調整した。その結果、薄橙色を検出するための範囲として川島賢二ら[1]を元に 10<=Cb<=40、220<=Cr<=245 の間とした。YCbCr 実装結果を図 6 に示す。

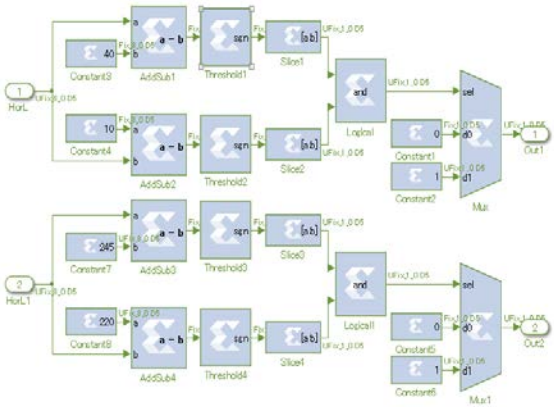


図 5　ブロック図

元画像　　YCbCr 変換

図 6　YCbCr 実装結果

HSV 変換のシミュレーション結果では浮動小数点で計算していたため良好な結果が出ていたが、実装した際に負数演算や固定小数点に変換した時に誤差が大きくなったと考えられる。

YCbCr 変換も背景の誤検出は見られたが HSV 変換と比べて背景を取ることが少なかった。H を Cb と Cr に分割し、範囲を細かく設定できること方が、HSV と比較したときに YCbCr 変換の方が回路が単純になるため、浮動演算の影響が少なかったためだと考えられる。

６．回路領域

表 1 に今回作成した YCbCr の回路と HSV 回路の回路領域を比較。

表 1 回路領域

	YCbCr回路領域		HSV回路領域	
Slice Logic Utilization	Used	Utilization	Used	Utilization
Number of Slice Rigisters	932	1%	960	1%
Number of Slice LUTs	1383	5%	8182	29%
Number of fully used LUT-FF pairs	715	46%	746	8%
Number of bonded IOBs	102	46%	102	46%
Number of BUFG/BUFGMUXs	9	56%	9	56%
Number of DSP48A1s	0	0%	9	15%

回路領域も表 1 に示したように YCbCr のほうが小規模となっているため、実装するにあたり、YCbCr 変換で作成した。

７．おわりに

YCbCr 変換による薄橙色検出では、動画を撮影した際に光の当たり方によってノイズが発生し、誤検出が見られた。しかし、光によるノイズは検出箇所が独立しているという特徴があった。そこで我々はノイズ対策として、検出した箇所が一定以上集合していれば薄橙色として判断することにした。独立している場合は検出箇所にカラー画像データを出力するようにした。イメージ図を図 7 に示す。

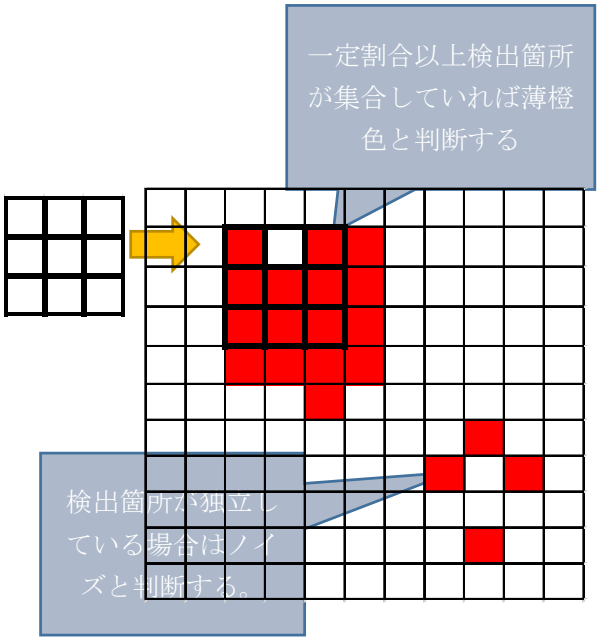


図 7　ノイズ対策イメージ図

８．〔参考文献〕

[1]川島賢二　(2004)　「顔画像認識における基礎的顔領域選択手法の検討」
<http://www.shojiro-tanaka.net/gradOutputs/2004/kawashima.pdf>

[2]「物理のかぎしっぽ」
<http://hooktail.org/computer/index.php>

[3]犬飼芳久（2004）「肌色抽出について」
<http://cafe.mis.ous.ac.jp/2004/sawasemi/Inukai/inu1.pdf>

画像の列平均化を用いたテンプレートマッチング

チーム名：OPC エンジニア

大城竜斗†　嘉数尚輝†　城間秀一†

Template matching using a column averaging of image

Ryuto Oshiro†, Naoki Kakazu†, Syuichi Shiroma†
Department of Advanced Electronics Information Technology for Production System
Okinawa Polytechnic College,2994-2 Ikehara,Okinawa-Shi,Okinawa, 904-2141 Japan
E-mail: j1521304@okinawa-pc.ac.jp, j1521305@okinawa-pc.ac.jp, j1521315@okinawa-pc.ac.jp

1. はじめに

我々のグループは、LSI デザインコンテストのテーマである「人物検出」のハードウェアを設計するにあたり、メモリの使用及び回路規模を最小限に抑えることで、処理を高速にすることを目標として開発を行った。実装した人物検出回路では、グレースケール化及び 2 値化、テンプレート画像と被探索画像のデータを平均化することにより全体のデータを削減している。まず本システムの簡易的なブロック図を図 1 に示す。また、本システムの使用機器の詳細を次項に記す。

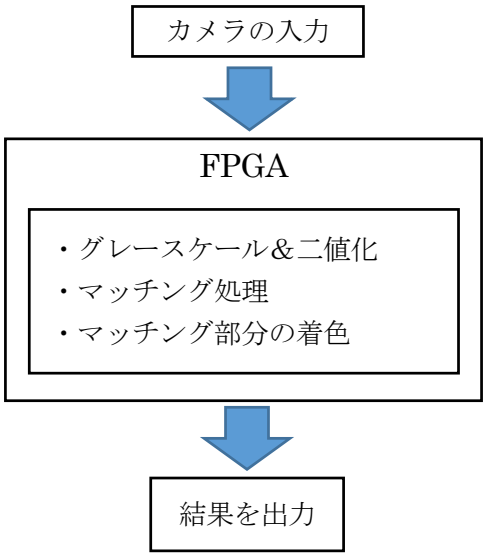


図 1　ブロック図

2. 使用機器

評価ボードは DIGILENT 社製の ATLYS Spartan-6 XC6SLX45、カメラは DIGILENT 社製の VmodCAM を使用し開発を行った。

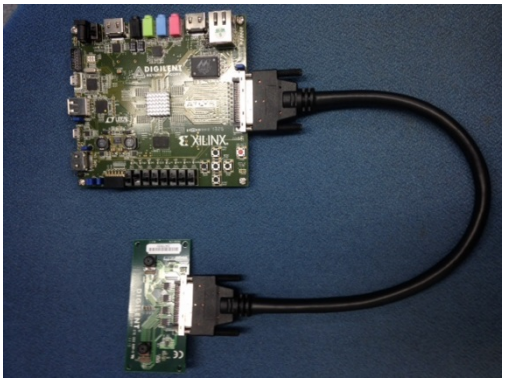


図 2　評価ボードとカメラ

この評価ボードは外部に 128MByte、内部に 64KByte のメモリが搭載されているが、外部のメモリはカメラの出力タイミング調整により使われてしまうため、外部のメモリは使用せず、CPU 内蔵の 64KByte のメモリで人物検出を試みることにした。今回使用したカメラの画像サイズは1600 × 900から640 × 480までの設定が可能で、我々は640 × 480での画像データ取り込みを使用した。カメラから入力されるデータがカラー画像で640 × 480 × 3 = 921KByte、グレー画像で640 × 480 = 307KByteとなり、入力データをメモリ内に展開することが出来ない。CPU 内蔵のメモリでは足りず以下の手法を用いた。

3. 提案手法

被探索画像はm × nピクセルのカラー画像、テンプレート画像はk × lピクセルの 2 値化された画像を使用している。2 値化したとき、白を 0、黒を 255 に振り分けている。各ピクセルの 2 値化した結果をT(x, y)、圧縮されたテンプレート画像をT_pで表すと、本手法で用いるデータ圧縮方式は

$$\mathrm{Tp(x)} = \sum_{y=0}^{l-1} \frac{T(x,y)}{l} \quad (x = 0,1,2,\dots l-1) \tag{1}$$

となる。また、被探索画像をm × nとし、各ピクセルの 2 値化した結果を I(i,j)と表すと、比較領域$I_p(x)$は(1)式より

$$\mathrm{Ip(x)} = \sum_{j=0}^{l-1} I(i+x,j) \quad (x = 0,1,2,,,k-1) \tag{2}$$

(1)、(2)より類似度を求めると

$$\mathrm{S(i,j)} = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\sum_{x=0}^{k-1} |Ip(x) - Tp(x)| \tag{3}$$

となる。th を閾値とすると、類似度の判定は

$$\mathrm{S(i,j)} = \begin{cases} 255 : if\ S(i,j) \le th \\ 0 \quad : if\ S(i,j) > th \end{cases} \tag{4}$$

となる。Matlab で行ったシミュレーション結果を図 3 に示す。

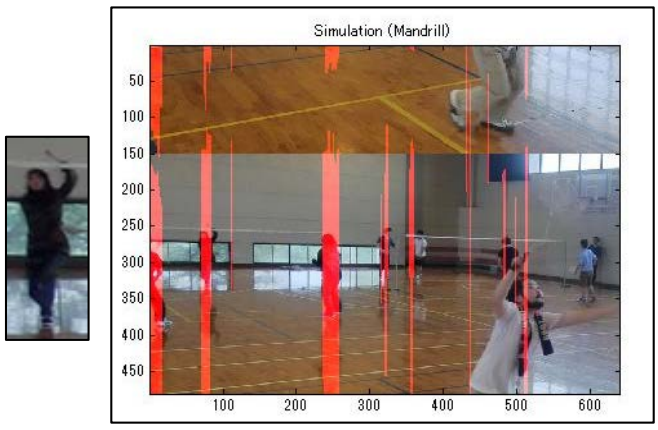


図 3　シミュレーション結果

4. 回路実装

判定結果をリアルタイムで反映させるため、遅延素子と論理和をテンプレート画像の縦 1 列のデータ幅分だけ用意し、メモリを使用せずに組み合わせ回路のみで作成した。

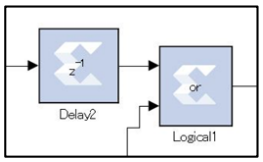


図 4　判定結果を反映する回路

また、判定部と出力を別にすることで処理速度を向上させた。

5. 実装結果

5.1. 人物検出結果

設計した回路を FPGA に搭載し、テンプレート画像と、カメラからの入力映像にマッチング処理を行い表示させた実行結果を図 5 に示す。

図 5　テンプレート画像と実行結果

実行結果からわかるように、大まかながら人物検出に成功している。人物以外にもマッチングする箇所が多いが、その理由としてピクセル単位ではなく平均化した配列との比較であるため、精度が低くなるのは当然の結果である。また、テンプレートの大きさによっても精度が変化する。

5.2. 処理速度

本システムの処理速度を評価するため、カメラモジュールのみのプロジェクトとマッチングシステムを追加実装したプロジェクトの実行時間を比較した。表 1 に比較結果を示す。表 1 から 4.7[ns]の遅延時間がわかる。

表 1　遅延時間表

遅延時間 (Data Path Delay)	[ns]
カメラモジュールのみ実装したプロジェクト	8.01ns
マッチングシステムを追加したプロジェクト	12.702ns

5.3. 回路規模

本システムの回路規模を評価するため、カメラモジュールのみのプロジェクトとマッチングシステムを追加実装したプロジェクトの回路規模を比較した。表 2 に比較結果を示す。表 2 から Registers が 3%、LUTs が 5%、LUT-FF が 9%の増加に抑えられていることから、回路の小規模化に成功していることがわかる。

表 2　回路規模

カメラモジュールのみ実装したプロジェクト

Slice Logic Utilization	used	available	utilization
Number of Slice Registers	910	54576	1%
Number of Slice LUTs	1447	27288	5%
Number of fully used LUT-FF pairs	698	1578	44%
Number of bonded IOBs	102	218	46%
Number of DSP48A1s	0	58	0%

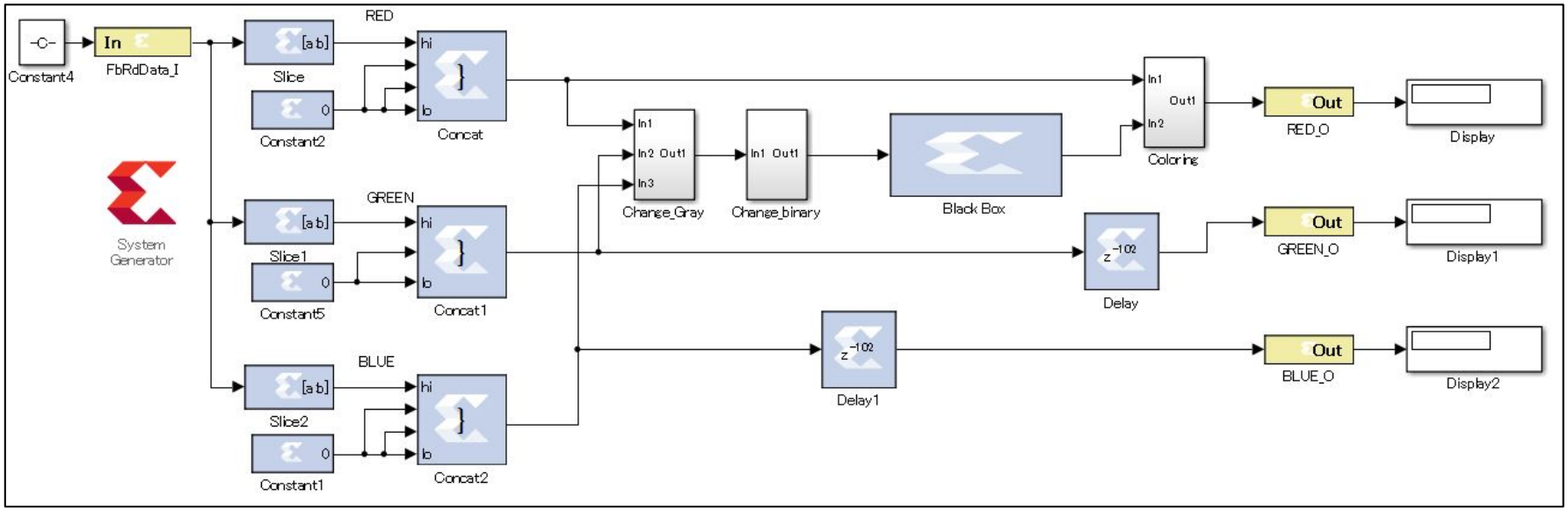
マッチングシステムを追加したプロジェクト

Slice Logic Utilization	used	available	utilization
Number of Slice Registers	2321	54576	4%
Number of Slice LUTs	2809	27288	10%
Number of fully used LUT-FF pairs	1595	958	53%
Number of bonded IOBs	102	218	46%
Number of DSP48A1s	0	58	0%

6. おわりに

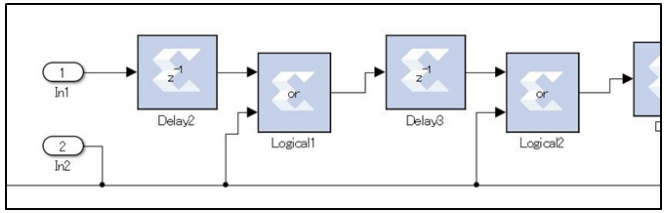
実行結果から分かるように、開発の目標である処理の高速化を達成することが出来た。しかし、データの平均値で判断するため、その性質上細かいデータを見ることができず、低い精度での動作となっている。しかし、大まかな人物検出には成功しているため、本システムと人物検出精度の高いシステムと組み合わせることで、処理を行う範囲を絞り込むことができ、無駄な処理を省くことが出来るのではないかと考える。

回路全体図



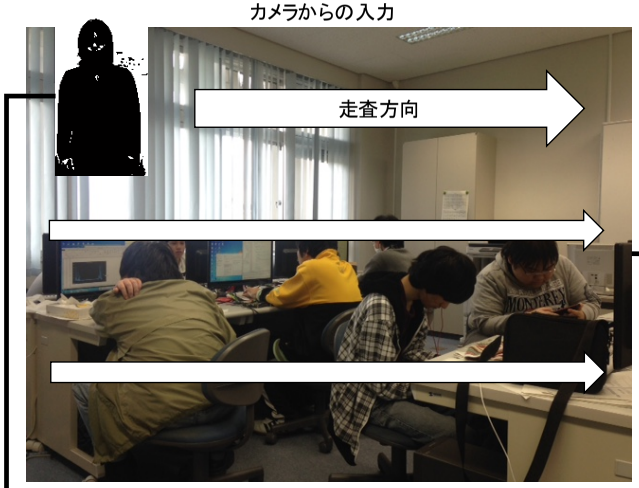
Coloring システム内部

拡大



実際の検索の流れ

カメラからの入力



FPGA内に取り込みデータ化

動画像入力データ(640*480)

	1	2	3	4	5	6	7	8	9	10	11
1	1	2	3	4	5	6	7	8	9	10	11
2	16	17	18	19	20	21	22	23	24	25	26
3	31	32	33	34	35	36	37	38	39	40	41
4	46	47	48	49	50	51	52	53	54	55	56
5	61	62	63	64	65	66	67	68	69	70	71
6	76	77	78	79	80	81	82	83	84	85	86
7	91	92	93	94	95	96	97	98	99	100	101
8	106	107	108	109	110	111	112	113	114	115	116
9	121	122	123	124	125	126	127	128	129	130	131
10	136	137	138	139	140	141	142	143	144	145	146
11	151	152	153	154	155	156	157	158	159	160	161

各列を平均化

最比較

FPGA内に取り込みデータ化

テンプレート画像データ(102*179)

	1	…	102
1	255	…	128
2	255	…	35
…	…	…	…
178	0	…	250
179	0	…	14

各列で平均化

1	…	102
127	…	85

プログラム内で呼び出し

ABS

最上位ビットのデータのみ比較

54	69	84	99	114	129	144

入力画像データ

24	57	186	230	12	145	1

テンプレート画像データ

Clockごとにデータを1bit上位シフト

69	84	99	114	129	144	90

57	186	230	12	145	1	245

30 差分の絶対値を取得

閾値(98)との比較

255 比較結果が正のとき255
負のとき0

信号を出力

超大規模シフトレジスタによる高速画像マッチングアーキテクチャ

VLSI の講義にての成果物について

琉大合同チーム 「Team of 嗚呼」
メンバー 嘉数一輝 吉田有希 當眞南 大城裕耶
沖縄 , 日本
e145704@ie.u-ryukyu.ac.jp

琉大合同チーム「チームお」
メンバー 長濱匠　衛藤凌一　林輝
沖縄 , 日本
e145734@ie.u-ryukyu.ac.jp

***Abstract*—シフトレジスタ多用により１サイクルごとに１回の画像マッチングを行う高速処理を実現した. RAM/ROM を使用しないシンプルなアーキテクチャを採用した.**

I. はじめに

近年，人物検出はデジタルカメラや自動車の自動ブレーキシステム，監視カメラなどのセキュリティシステム等に利用されており，私達の身近で利用されている．そこで今回は「人物検出」をテーマとして，処理の高速化，回路規模の削減を目指したハードウェア設計を行った．筆者らは琉球大情報工学科の２年次生であり，これまではソフトウェアの設計を主に行ってきた．今回初めてハードウェアの設計を行い，ハードウェアに対する理解を深めるとともに，実社会で要求される複数人での開発を目的として本課題に取り組んだ.

図 1 は今回の処理対象画像であり，1 ピクセル RGB24bit の 640×480 ピクセルイメージである. 図 2 は 40×100 ピクセルの RGB イメージである．図 1 中の赤枠の部分 がテンプレートとマッチングする部分である. 今回使用した 設計ツールは Xilinx ISE Design Suite であり，総合情報処理センターの実習コンピュータを用いて設計を行った.

図 1.処理対象画像

II. 設計した回路アーキテクチャ

A. 全体構成

図２に今回採用した処理の全体構成を示す．左の黄色のボックスがカラーイメージ図である．RGB 各 8bit は DEN=1 をトリガーとして 640×480 サイクルで 1bit の白黒信号 に変換され，右の白黒イメージメモリに記憶される. また，TEN =1 をトリガーとしてカラーテンプレートも同様に右の白黒テンプレートメモリに記憶される. その後, START=1 をトリガーとしてテンプレートメモリを移動させながらイメージメモリと比較し SAD（Sum Of Absolute Difference）値が計算される(式 1). SAD 値がある閾値より小さい場合，図形がマッチングしたとして SEN =1 が出力される. 同時にマッチング部のイメージ座標(X,Y)が出力され画像のマッチング位置が分かる. このマッチング座標は図 1 の赤枠の左上の座標となる.

$$SAD = \sum_{x=0}^{X-1} \sum_{y=0}^{Y-1} |I(x,y) - T(x,y)| \quad \text{(式 1)}$$

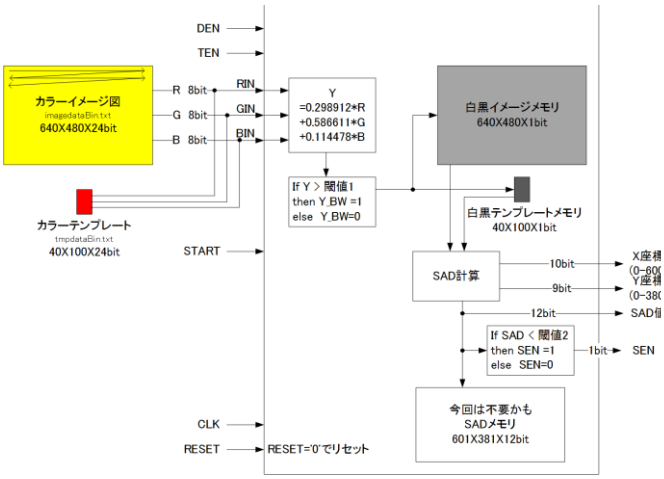


図 2.全体構成

B. RGB から BW 変換回路

図 3 は RGB(8bit)をグレースケール Y に変換し，その後ある閾値により白黒 1bit に変換する回路を示す．

Y=0.298912*R+0.586611*G+0.114478*B　（式 2）

変換式は式 2 で与えられる．実装では 8bit × 8bit の乗算器を 3 個と 8bit 加算器を 2 個用いて実現されている．図中の閾値 1 は今回は動的に計算せず，回路構成単純化のため事前調査により固定値=107 を用いた．

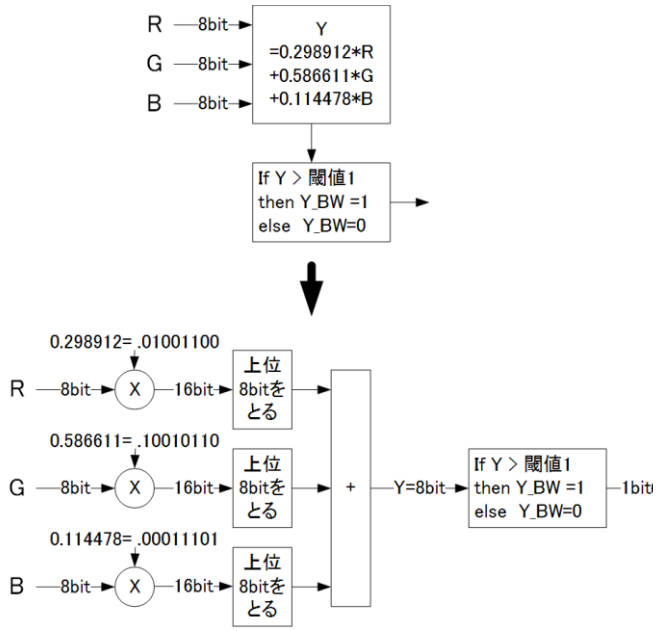


図 3. RGB から BW 変換回路

C. 画像マッチング回路

図 4 は今回の処理アーキテクチャの中核部である．通常は FPGA 内の RAM を用いて設計すると思われるが，今回は 1 サイクル 1 マッチングという高速処理を実現するために，大規模なフリップフロップによるシフトレジスタを用いるアーキテクチャとした．

左側は白黒イメージメモリであり，640×480 = 307200bit のシフトレジスタを構成している．外部からの画像入力時及び画像マッチング処理時ともデータをシフトすることで処理を行っている．MODE 信号により外部信号を取り入れ時には Y_BW が入力となり，画像比較時にはサイクリックに画像データが巡回して画像比較を行う．右のテンプレートメモリも同様にシフトレジスタで構成されている．画像の比較はテンプレートメモリのサイズ 40×100=4000bit のパラレルの EXOR で比較を行いその 4000bit の比較結果が加算され SAD 値となる．

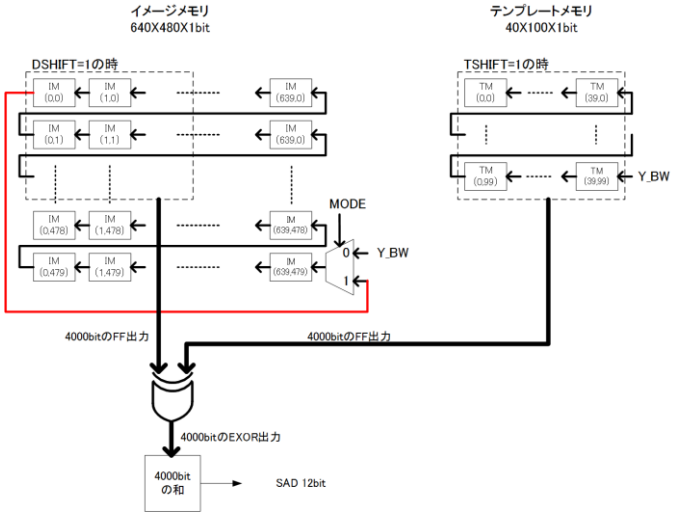


図 4. 画像マッチング回路

D. 動作波形

図 5 は概略の動作波形図である．DEN=1 をトリガーに RIN, GIN, BIN 各 8bit が取り込まれ白黒画像 Y_BW に変換されイメージメモリにシフトレジスタ動作で記憶される．次に TEN＝1 をトリガーにして同様にテンプレート画像が取り込まれる．その後 START=1 をトリガーに，イメージメモリ画像がメモリ内をサイクリックにシフトすることで画像のマッチング探索が実行される．SAD（Sum Of Absolute Difference）値は 1 サイクル毎に計算され出力される．

III. 動作シミュレーション結果

図 6a は XILINX の ISE ツールでのロジックシミュレーション波形の冒頭の部分である．DEN=1 をトリガーとして RGB 各 8bit が入力されている様子が分かる．

図 6b はロジックシミュレーション全体を縮小した波形図である．DEN, TEN, START のアサートに引き続いて画像比較が行われている．図中の黄色の矢印の部分で

SAD 値が 0 となり，SEN=1 が出力され画像マッチングが示されている．

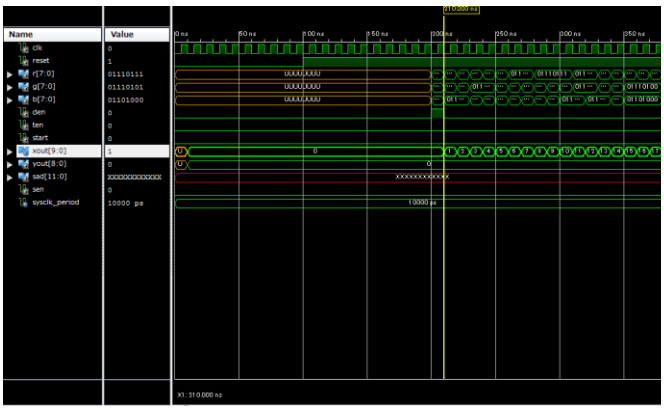


図 6a. 冒頭のシミュレーション波形

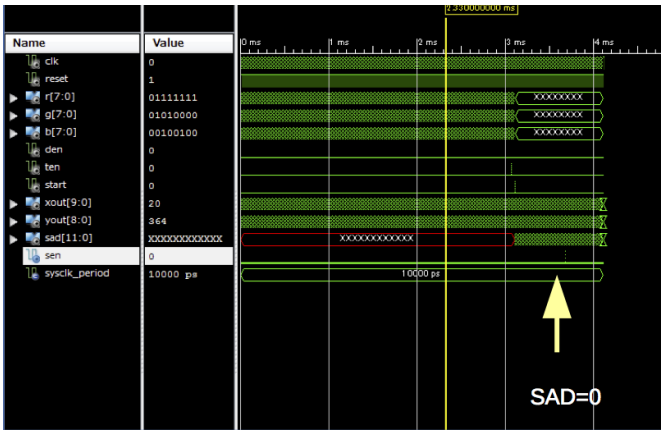


図 6b. 画像マッチングポイントを発見したときの波形

IV. 設計のまとめ

表 1 に設計のまとめを示す．処理の高速化の観点から，シフトレジスタを多用することにより 1 サイクルごとに画像マッチングを行うように設計した．また，一般に FPGA 内の RAM・ROM を使用することが多いと思われるが複雑な設計になることが多いため，設計難度の観点から初心者である筆者らは RAM・ROM を使用しないことでシンプルな設計にした．しかし，Xilinx コンパイラ，Design コンパイラでの回路合成を試みたがイメージメモリが大きすぎるようで合成できなかった．回路規模はかなり大きく，具体的には FF 数が 311.2K，EXOR が 4K，8bit 乗算器が 3 個等である．

今回の課題を通してハードウェア設計の基礎を学習することができ，普段学んでいる技術が実社会のどのような場面で使われているか知ることができた．これらのことはモチベーション向上に繋がった．また，複数人での設計を体験することでスケジュールの管理や個人の資質に合わせたタスクの割当など，プログラミング技術の他にもコミュニケーションや情報共有が大切であることを実感することができた．

表 1. 設計のまとめ

(1) シフトレジスタ多用により，1 サイクルごとに画像マッチングをする．高速また現実的．

(2) RAM・ROM 非使用でシンプルな設計．

(3) Xilinx コンパイラ，Design コンパイラでの合成をトライしたが，イメージメモリが大きすぎるようで，合成できず．

(4) 回路規模

FF 数：307200 + 4000 = 311.2K

EXOR：4K

8bit 乗算器：3

等

V. 最後に

今回，VHDL コードの見直しや複数のコンパイラでの回路合成をトライしたが，イメージメモリの合成は成功しなかった．ロジックシミュレーションでは問題なかったが，合成による正確な回路規模の値を得ることができなかった．今回日本各地及びアジアより VLSI 設計に詳しい先輩方がコンテストに参加されるので，是非ともご助言をいただきたい．

コンテストを開催していただきました各位に感謝を申し上げます．

テンプレートマッチングの高速化と実装

会津大学 コンピュータ理工学部 組み込みシステム学講座
松本 優路

1 はじめに

この回路では、被探索画像とテンプレート画像の処理の並列化、National Television System Committee (NTSC) 系加重平均法の簡略化、ヒストグラムを用いた中央値探索、閾値を越えた Sum of Absolute Difference (SAD) の計算の打ち切りを用いることによって、テンプレートマッチングの高速化を図った。また、ALTERA 社の DE2-115 への実装を可能にした。

2 設計

回路は、設計仕様書にしたがって設計したが、一部、独自のアルゴリズムを用いた。図 1 は回路モジュールの階層を表す。overall_max 下のモジュールで、histogram, matching_counter, xor_, matching_action 以外は被探索画像用とテンプレート画像用とで各々2 つ組み込まれている。

2.1 高速化

被探索画像とテンプレート画像のグレースケールと 2 値化を並列に処理することで、テンプレート画像の処理時間をなくした。さらに、グレースケール化では NTSC 系加重平均法を簡略化することによって、計算を単純にしている。2 値化のための中央値を求めるアルゴリズムはヒストグラムを用いることによって、大幅な短縮ができた。SAD の計算を打ち切ることでマッチングも高速化した。それぞれの速度の向上度は 3 に記載する。

2.2 モジュールのインスタンス化

被探索画像とテンプレート画像とで、同じ処理をする場合、同じモジュールで異なる画像サイズに対応できるように設計したことで、作成モジュールの個数を減らし、設計の負担を減らした。

2.3 実装

規程の画像サイズであれば、画像を FPGA のメモリ形式に変換し、FPGA ボードで実行後、再び画像に復元して結果を確かめることができる。

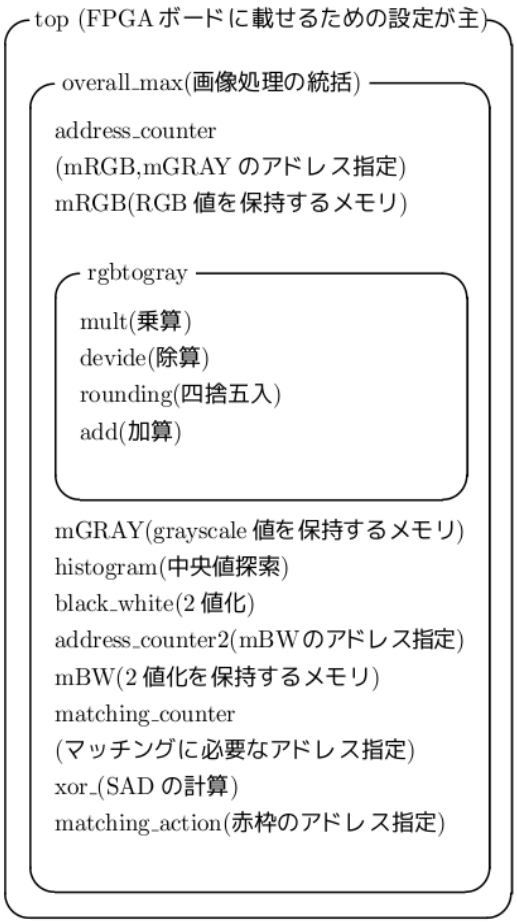


図 1: 回路モジュールの階層

2.4 top

全体のトップモジュール。FPGA ボードに載せるためのピンアサイン等の設定が主である。以下の 5 つの処理がなされた時に LED を点灯させる設定をしている。5 つの LED が全て点灯すれば処理が終わったことを目視で確認することができる。

- end_RGB：被探索画像の RGB 値の読み込み終了時
- end_center： 被探索画像のグレイスケール値の中央値の計算終了時
- end_BW：被探索画像の 2 値化終了時
- matching_start： テンプレートマッチングの開始時 ※
- end_matching：テンプレートマッチングの終了時

※マッチングの開始時は被探索画像の 1 番左上の画素の SAD を求め終わったときに立ち上がる。

2.5 overall_max

画像処理を行う事実上のトップモジュールである。以下に処理の流れに沿ってポイントを述べる。被探索画像とテンプレート画像を並列に処理を行う。また、24bit の RGB 値を、R, G, B の 8bit に 3 分割して並列に計算することによって 1 画素のグレースケール値を高速に求めることができる。計算で求まったグレースケール値をメモリに書き込むと同時に histogram モジュールにもデータを流すことによって、メモリへの書き込みとヒストグラム作成を並列に実行している。また、ヒストグラムを用いた中央値の探索は、ソートなどを用いた中央値の探索よりも高速である。テンプレートマッチングでは、1 回のマッチングに対して、SAD を最後まで求めるのではなく、SAD が閾値を越えた時点で SAD の計算を中断して、次のマッチングに移行するアルゴリズムを採用している。また、画像がマッチした場合、次のマッチングをしている間に、マッチした部分の赤枠を表示する処理を並列で処理することで、高速化している。

2.6 address_counter

メモリから RGB 値を読み取る際の mRGB のアドレスと、生成されたグレースケール値をメモリに格納する際の mGRAY のアドレス、2 値化するためにメモリからグレースケール値を読み取る際の mGRAY のアドレスを出力するモジュールである。mRGB と mGRAY の読み出し、書き込みの切り替え信号も制御している。

また、画像がマッチした時、送られてきた赤枠のアドレスを mRGB のアドレスとしてセットする。

2.6.1 mRGB,mGRAY 間でのデータのずれの防止

mRGB と mGRAY のアドレスを 1 つのモジュール内で制御することによって、この 2 つのメモリ間でのアドレスのずれが起こることはない。また、メモリの読み書き信号もこのモジュールで制御することによって、アドレスとデータ間のずれも防いでいる。

2.6.2 インスタンス化の可能

入力の pixel_number の値を変更することによって、異なる画像サイズに対応できるように設計した。どんな画像サイズでもアドレス指定することができる。

2.7 mRGB

RGB 値を保持するメモリモジュールである。今回は、FPGA ボードにのせるために ALTERA 社の QuartusⅡ 64-Bit Version 13.1.0 の MegaWizard Plug-In Manager 内の RAM:1Port を使用した。他のメモリモジュールも同様のものを使用して作成している。被探索画像のアドレス幅が 16bit に対して、テンプレート画像は 12bit で足りるため、無駄なメモリ空間を省くためにアドレス幅の異なる２つのモジュールとした。

2.7.1 アドレス幅

MegaWizard Plug-In Manager で作成できるメモリモジュールの最大のアドレス幅が 16bit なので、被探索画像用のメモリは 16bit で作成した。また、テンプレート画像は、$100 \times 40 = 4000 < 4096 = 2^{12}$ より、テンプレート画像用の mRGB は 12bit 幅で作成した。

2.7.2 データ幅

メモリのデータ幅は RGB 値の bit 数に合わせ、24bit 幅で作成した。

2.7.3 今回使用する被探索画像のサイズ

2.7.1 で記述したように、アドレス幅が 16bit に制限されるため、$2^{16} = 65536 \fallingdotseq 200 \times 300$ の被探索画像を採用した。

2.8 rgbtogray

RGB 値からグレースケール値を計算する複合モジュールである。

2.8.1 NTSC 系加重平均法の簡略化

今回、設計仕様書に記載されている NTSC 系加重平均法では、

$$Y = 0.298912 \times R + 0.586611 \times G + 0.114478 \times B$$

と計算するが、小数第 6 桁までの計算は不要であり、小数を扱う必要はないので、次のように簡略化した。

$$Y' = \frac{29 \times R}{100} + \frac{58 \times G}{100} + \frac{11 \times B}{100}$$

100 で除算後、小数第 1 位を四捨五入することで、整数のみで表すことができる。

2.9 mult

$29 \times R$, $58 \times G$, $11 \times B$ の部分を計算するモジュールである。3 つ組み込むことで R, G, B それぞれの計算を並列で処理する。このモジュールは、MegaWizard Plug-In Manegar 内の LPM_MULT を使用した。

2.10 devide

$\frac{29\times R}{100}$, $\frac{58\times G}{100}$, $\frac{11\times B}{100}$ の部分を計算するモジュールである。mult と同様に 3 つ組み込まれており、それぞれの計算を並列で処理する。100 で除算した商と剰余が出力される。このモジュールは、MegaWizard Plug-In Manegar 内の LPM_DEVIDE を使用した。

2.11 rounding

devide モジュールからの剰余を基に四捨五入を行うモジュールである。四捨五入を行うことによって、設計仕様書に記載されている NTSC 系加重平均法の計算結果により近い数値を得ることができる。

2.12 add

各々計算された R, G, B を加算して、グレースケール値を求めるモジュールである。既に R, G, B は整数で表されており、オーバーフローが起こることもないので、非常に単純なモジュールで設計することができる。

2.13 mGRAY

グレースケール値を保持するメモリモジュールである。MegaWizard Plug-In Manager 内の RAM:1Port を使用した。アドレス幅は 2.7.1 を参照。データ幅はグレースケールの bit 数に合わせて 8bit で作成した。

2.14 histogram

被探索画像のグレースケール値のヒストグラムを作成し、2 値化する際の基準値となる、中央値を求めるモジュール。

2.14.1 アルゴリズム

配列 count を用意する。配列の個数はグレースケール値、0 ～ 255 までの 256 個である。グレースケール値と同じ配列の添字 (グレースケール値が 167 ならば count[167]) の値をインクリメントして、そのグレースケール値が画像中に何個あるのかを記録することによってヒストグラムを作成する (図 2)。ヒストグラムを作成し終えたら、count[0] から順に記録されている数 (下の図での*の数) を足し合わせていき、被探索画像の画素数の半分 (今回は 3000) を越えたら、その時の count の添字の値がグレースケールの中央値となる (図 3)。

入力 : グレースケール値 ⇒ 100

count[0] = ***
count[1] = *******
count[2] = *****
⋮
count[100] = ***** + *
⋮
count[255] = **

図 2: ヒストグラム作成過程の例

count[0] = ***
count[1] = *******
count[2] = *****
⋮
count[119] = ********
⋮
count[255] = **

図 3: 中央値探索の例

図 3 の黒枠内の*の数 (=sum) が 3000 を越えるかどうか判定する。もし sum ≥ 3000 ならば、119

を中央値として出力する。もし sum < 3000 ならば、黒枠を 120 まで広げて再度判定する。

2.14.2 中央値探索の高速化

ヒストグラムの作成とメモリへの書き込みを並列に処理することで、メモリへの書き込みが終わると同時にヒストグラムも完成するのでヒストグラム作成にかかる時間はない。そこから中央値を探索するが、配列 count の 0 ～ 255 の値を足し合わせる過程で中央値が見つかるので、どんなに多く見積もっても、256clock 以内で中央値を見つけることができる。

2.15 black_white

求められた中央値を基に、2 値化をするモジュールである。

2.15.1 動作の正確性

不正確なデータを与えないような入力になっている。さらに、モジュールにも同じ制御信号を与えることによって、不正確なデータを受け取らないように設計した。このことによってこのモジュールの動作の正確性を確立している。

2.16 address_counter2

2 値化された値をメモリに書き込む際の mBW のアドレスを出力するモジュールである。mBW の読み出し、書き込みの切り替え信号も制御している。特徴は address_counter と同様なので、2.6.1、2.6.2 と同様である。

2.17 mBW

2 値化された値を保持するメモリモジュールである。MegaWizard Plug-In Manager 内の RAM:1port を使用した。アドレス幅は 2.7.1 を参照。データ幅は 2 値化の bit 数に合わせて 1bit で作成した。

2.18 matching_counter

テンプレートマッチングに必要な被探索画像とテンプレート画像のアドレスを同時に指定するモジュールである。SAD が閾値を越え、マッチングの中断信号を受けつけると、次のマッチングに移るシステムになっている。address_counter のように入力で異なる画像サイズに対応できるようにはなっていないが、条件式を変えることによって、どんな画像サイズにも対応できるようになっている。

2.19 xor_

排他的論理和を計算し、SAD を求めるモジュールである。matching_counter モジュールによって、マッチングに必要なデータは全て送られてくるので、それらに排他的論理和をとるだけの単純なモジュールで設計することができる。

2.19.1 閾値を越えることによる、SAD の打ち切り

SAD が閾値を越えた時点で計算を続ける必要がなくなるので、次のマッチングに移させる信号を出力する。また、打ち切られた SAD は出力されないように設計し、他のモジュールが誤動作を起こすことも防いでいる。テンプレートマッチングの高速具合はここで設定される閾値に大きく影響する。

2.20 match_action

画像がマッチした場合に赤枠のアドレスを指定するモジュール。マッチング処理とは別のモジュールにすることで、マッチング処理と並列に実行できるようにした。address_counter のように入力で異なる画像サイズに対応できるようにはなっていないが、条件式を変えることによって、どんな画像サイズにも対応できるようになっている。

3 速度、回路領域の評価

3.1 速度

2.4 の信号が立ち上がる時間を記載する (表 1)。なお、テンプレートマッチングの閾値は 400(テンプレート画像の画素数の 10%、つまり 90%以上の一致率)、clock cycle time は 20ns である。使用したシミュレータは ModelSim-Altera Edition 10.1d である。

信号	時間 (ns)	前信号との差 (ns)	clock 数
end_RGB	1200020	-	60001
end_center	1202000	1980	99
end_BW	2402080	1200080	60004
matching_start	2425660	23580	1179
end_matching	446192780	443767120	22188356

表 1：各信号の立ち上がり時間

3.1.1 テンプレートマッチングの前まで

end_RGB と end_BW の clock 数を見るとほぼ 60000 と、被探索画像の画素数と同じなので、グレースケール化と 2 値化は、1clock で 1 画素処理していることになる。中央値の探索は、99clock と 2.14.2 で記述したように、256 以下の clock で求まる。

3.1.2 テンプレートマッチング

2.4 の注意書きに記述したように、matching_start は 1 つ目の SAD が求まった後に立ち上がるので、表 1 の数値は 1 つ目のテンプレートマッチングが 1179 前後の画素数で計算が打ち切られていることを示す。end_matching の clock 数をテンプレートマッチングが行われる回数、(200 - 100) * (300 - 40) = 26000 で割ると、22188356 / 26000 = 853.398 であり、1 回のマッチングに約 853clock かかることになる。SAD の計算を打ち切らなかった場合、1 回のマッチングにテンプレート画像の画素数 (4000 画素) と同じ clock がかかるため、向上度は 4000 / 853 = 4.689 であり、約 4.7 倍の速度で計算する。

3.2 回路領域

QuartusII 64-Bit Version 13.1.0 での Compilation Report の一部を記載する。

Total logic elements	7,740 / 114,480 (7%)
Total combinational functions	5,832 / 114,480 (5%)
Dedicated logic registers	4,998 / 114,480 (4%)
Total registers	4,998
Total pins	50 / 529 (9%)
Total memory bits	2,301,952 / 3,981,312 (58%)

表 2 : Compilation Report の一部

4 実行結果

実際に FPGA ボードで実行した結果の画像を添付する (図 4, 5, 6, 7, 8)。テンプレートマッチングの閾値は 400 である。

5 その他、要求したいものすべて

5.1 被探索画像のサイズを大きくするために

被探索画像のサイズを大きくするには、メモリを拡張しなければならない。以下に大きな画像を扱うための方法を述べる。

図 4: 被探索画像

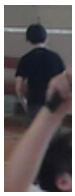


図 5: テンプレート画像

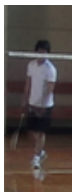


図 6: テンプレート画像 2

図 7: マッチング結果

図 8: マッチング結果 2

5.1.1 メモリモジュールの自作

1 つ目の方法としては、MegaWizard Plug-In Manager 内の RAM:1Port を使用しないで自作のメモリモジュールを使用することである。メモリのアドレス幅を 16bit を越えて設計することができ、より大きな画像のサイズを扱うことができる。しかし、FPGA ボードは MegaWizard Plug-In Manager 内で作成されたメモリモジュールのみにしか対応しないので、自作のメモリモジュールではボードでの実演は不可能となる。

5.1.2 ボード内のメモリの使用

2 つ目の方法は、FPGA ボードに積まれている、ISSI 社の 64MB SDRAM を使用する方法である。しかし、In-System Memory Content Editor が使えず、メモリを RGB 値で初期化することができないため、今回は使用しなかった。このメモリの初期化を行う回路を FPGA 内に作成することも考えたが、今後の課題である。

5.2 苦労したこと

5.2.1 matching_counter の作成

メモリのアドレスは 1 次元なので、2 次元的に表されている画像を 1 次元に落とさなければならず、条件式が複雑になってしまい、作成やデバックがしづらかった。特にデバックでは、1clock 毎に波形を細かく確認して、エラーの原因を見つける作業がとても大変だった。

5.2.2 ボードへの実装

シミュレータ上では正しい波形が出ているのに、ボードに実装するとどうしても正常に動かないことがあった。ボードに載せているので、波形を調べる方法もなく、はっきりとした原因がつかめず、原因と思われる verilog コードをしらみつぶしで改良していった。今回の作業の中で 1 番時間がかかった作業だった。

5.3 感想

この様な規模の回路を全て自分で設計したのは初めての経験だったので、達成感が大きい。また、5.2.2 の様なエラーをデバックできたときはとても嬉しかった。histogram と matching_counter モジュールは複雑なモジュールで作成が難しかった。

アルゴリズムの面でも、hitogram を導入できたことは非常に大きかった。ただ、テンプレートマッチングでの大きなアルゴリズムの改良は行うことができなかった。改良の案はいくつか考えることが出来たのだが、それを実現できる技術を身につけておらず、まだまだ技術のスキルアップが必要だと痛感した。

FPGA Implementation of Queue Counter using Template Matching with Sum of Absolute Difference Algorithm

[1]Mahendra Drajat Adhinata, [2]Novi Prihatiningrum, [3]Ricky Disastra
Department of Electrical Engineering, School of Electrical Engineering and Informatics
Bandung Institute of Technology, Jl. Ganesha No. 10 Bandung, 40132, Indonesia
Email: [1]mahendra.drajat@outlook.com, [2]n.prihatiningrum@students.itb.ac.id, [2]r.disastra@students.itb.ac.id

Abstract- ***Sum of Absolute Difference (SAD) algorithm is one of many algorithm used in template matching. In this paper, we will implement the use of SAD in template matching for the queue counter application in FPGA and evaluate its performance. The searching image is 640 x 480 pixels and the template image is 40 x 100 pixels. The searching image is stored in flash memory in gray scale value and the template image is stored in FPGA in binary value. The SAD processor array is mapped into horizontal map so there are 40 x 100 SAD processing elements used. From the compilation in ALTERA Quartus, it is shown that this design can have maximum frequency 128.35 MHz and consumes 17 411 combinational logics, 17 158 registers, and 59 899 memory bits. This design is implemented in Altera DE2-115 development board in 100 MHz frequency. The computation time needed to finish all image is 307 200 clock cycles, thus with 100 MHz frequency, it can be finished in 3.072 ms or 325 fps.***

***Keywords*— FPGA, SAD, template matching, queue counter.**

I. INTRODUCTION

Template matching is a technique used in digital image processing for finding parts of an image which match a predefined template image. Template matching technique are flexible and relatively straightforward to use, which makes it one of the most popular methods of object localization. It is widely used in manufacturing as a part of quality control, a way to navigate a mobile robot, or as a way to detect edges in images.

There are several methods which can be used for template matching. Some have high computational complexity and high accuracy but result slower calculation. The simplest and most widely used method is Sum of Absolute Difference (SAD). It is widely used because of its simplicity to be implemented in hardware systems.

In this paper, we will implement SAD algorithm with horizontal mapping to get faster computation time for queue counter application and evaluate its performance.

II. APPLICATION OVERVIEW: QUEUE COUNTER

In our daily life, queue mechanism is widely used in a system. Queue can be found in many aspects of life such as in an amusement park, bank, and other public services. In queue, we sometimes have to know how many people are in the queue. In order to do that, a tool to identify number of people in the queue is needed.

Template matching is one of many methods which can be used to identify object in an image. If we use template matching method, we can identify people in the image. After doing the identification, we can get the number of people in the queue.

Figure 1 People in Queue

III. TEMPLATE MATCHING AND SAD

The template matching technique compares the search image with the small image called template image. The comparison is done from the upper left in the order of the lower right. The comparison results the degree of similarity of the two images using the pixel value as an indicator. We will use Sum of Absolute Difference (SAD) to calculate the degree of similarity.

SAD calculate the sum of all the values of the difference between pixel values of the search image and the template image. Below is the formula to get SAD value.

$$SAD = \sum_{x=0}^{X-1}\sum_{y=0}^{Y-1}|I(x,y) - T(x,y)|$$

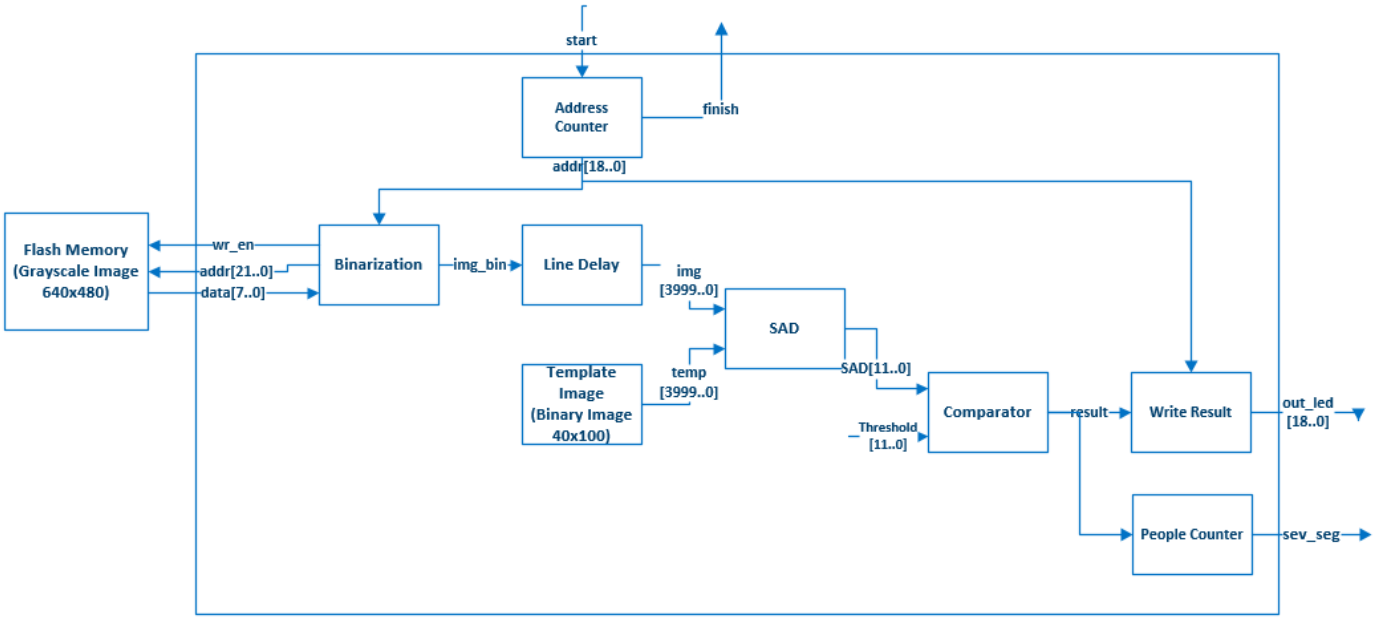


Figure 2 Proposed Design Block Diagram

I(x,y) stands for the row x column y of the search image while T(x,y) is of the template image. In this case, we will apply binary template matching so the subtraction can be implemented by using XOR operation and results in faster process.

The degree of similarity of the template image and the search image is higher as the value of the SAD gets smaller. The value of the SAD will become 0 if it is exactly matched.

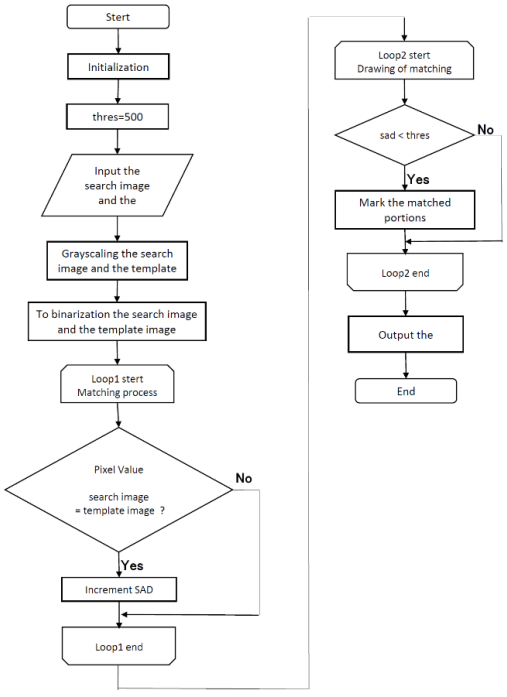


Figure 3 Template Matching Flowchart

The template matching algorithm can be done in five processes, which are input the images, gray scaling and binarization, matching method, drawing matching points, and output the resulting image. The flowchart of template matching algorithm is shown in the figure 3 below.

The first process is to get the inputs which are the search image and the template image. Both images are in form of hexadecimal file consists of R, G, and B values of every single pixel of the image. The next process is gray scaling and binarization. Gray scaling process is done using the NTSC Coefficient Method using the following equation.

$$Y = 0.298912 \times R + 0.586611 \times G + 0.114478 \times B$$

To binarize the gray scale image, the threshold has to be determined first. After getting the binarized image, the SAD calculation is done. SAD is simplified as the sum of the XOR of the template image and the search image. SAD calculation is done at each location starting from the upper left to the lower right of the searching image. The resulting SAD is compared to the threshold value. Mark the portions that SAD is lower than the threshold. The final process is to output the resulting image marked the matched portions.

IV. PROPOSED DESIGN ARCHITECTURE

In this paper, we will design an FPGA-based architecture of template matching using SAD for queue counter application. The searching image is in size 640x480 pixels and the template image is in size 40x100 pixels. The block diagram of the design architecture is shown in Figure 2.

The searching image is stored in flash memory in grayscale value. To get the grayscale value, we first do the preprocessing to the RGB image using MATLAB. The next process is binarizing the gray scale value one by one, then sending each

value into line delay. Output of the line delay is 4000 data to be the input of SAD processor array. These data will be processed with the 4000 template binary data stored as the constants. At one clock the result comes out and compared to the threshold in comparator, after that the coordinate of matched position is generated in write result block, and the people counter will count how many matches occur. The detail process of each block will be explained below.

A. *Image Preprocessing*

Each pixel of the image consists of three color channels: red (R), green (G), and blue (B). It is needed to be converted into grayscale value based on equation using the NTSC Coefficient Method. This process is done by using MATLAB. We first read the RGB value of the image by using img_read function. This results in an array with size 640x480x3. After that, the grayscale value is computed for each pixel started from the left upper to the right using the NTSC Coefficient Method.

B. *Flash Memory*

The gray scale values of the searching image generated by MATLAB are stored in flash memory. The flash memory can be accessed using DE2-115 Control Panel available in the CD. There are 307200 8-bits data to be stored started from address 0 to 307199. To read the data stored in the flash memory, there are several signals needed to be sent, such as *read_enable* signal and address of the data to be read. The read processes can only be done one by one.

C. *Binarization*

The binarization processed is done for every pixel of the image stored in the flash memory. This block sends *read_enable* signal to the flash memory and the address of the data started from 0. The threshold for binarization is chosen to be 128, so we only take one-bit MSB which is the 8th bit of the grayscale value as the image binary result. The binary result from this block is sent to the Line Delay.

D. *Line Delay*

The Line Delay block is needed to feed image binary data to the SAD processor array. The line delay size is 640x100 because the length of the searching image is 640 pixels and the width of the template image is 100 pixels. The output of this block is taken 40 data from each line, so there are 4000 data taken out at a time. These 4000 bits image data will be checked with the 4000 bits template data. The line delay is shown in the Figure 4.

This line delay works by shifting the binary image input from binarization block. It shifts the data one by one, so to get the first 4000 data output for the first comparison the latency time needed is 64000 clock cycles.

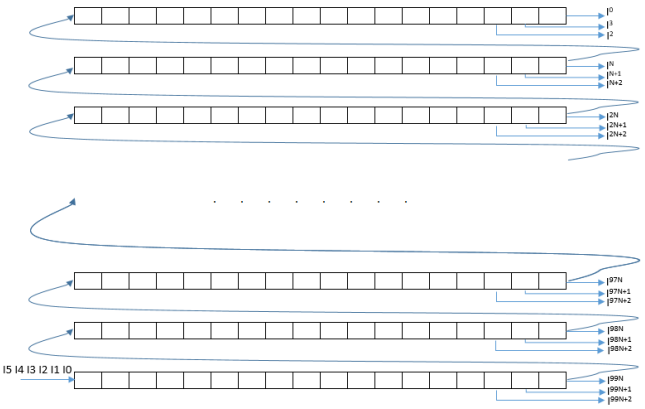


Figure 4 Line Delay Block

E. *SAD Processor Array and Comparator*

We choose to implement the horizontal mapping from the SAD Processor Array to get faster computation time. The processing element needed to implement the horizontal map is 4000 processing element. By using this architecture, we can get one SAD value at a time. To finish all calculations it is needed 307 200 clock cycles. Each SAD processing element only consists of XOR gate to compare each searching image pixel with the template. To get the SAD value, the results from all PEs have to be added. In order to add all the results, the 12-stages tree adder are implemented. The tree adder consists of 2000 1-bit adder, 1000 2-bits adder, 500 3-bits adder, 250 4-bits adder, 125 5 bits adder, 63 6-bits adder, 32 7-bits adder, 16 8-bits adder, 8-9 bits-adder, 4 10-bits adder, 2 11-bits adder, and 1 12-bits adder. This tree adder uses less logic gates than if we use 12-bits adder at each PEs.

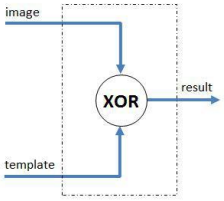


Figure 5 Processing Element

The comparator block compares the SAD result value with the threshold value defined to check whether the part of the searching image match the template image. If the SAD result value is lower than the threshold, the result of the comparator is high. This result is then fed into write result block to show the coordinate of the matching point and to people counter block to count number of people in the searching image.

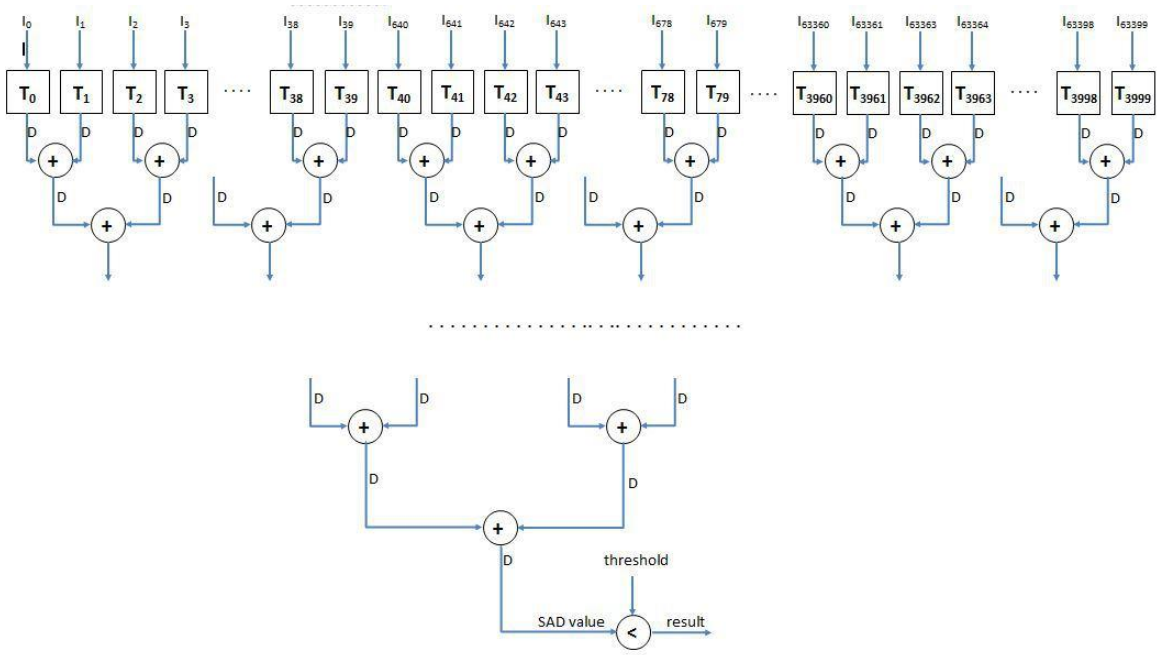


Figure 6 Processing Array with Tree Adder and Comparator

F. Write Result

This block get the result value from comparator block and data address from address counter block. If the result value is high, this block will compute the coordinate of the matching point. The address is first subtracted by 64014 because there is a latency time 64000 clock cycles from the line delay, and 14 clock cycles from the tree adder. After subtracted we get the address of the matching point. The x-y coordinate is computed by these equations:

$$x = \left\lfloor \frac{address}{640} \right\rfloor$$

$$y = address \bmod 640$$

Both x and y value are then shown in the seven segments, each uses 3 seven segments.

G. People Counter

This block will count number of matches that occur in the searching image. It get the result value from comparator block. After reset, the counter will be set to 0. Whenever a high signal comes from the comparator, the counter counts up the number of matching occurs. This value will be shown in two seven-segments.

V. RESULT AND ANALYSIS

The first step is image preprocessing using MATLAB. The MATLAB code for the image preprocessing is shown in Appendix A. To run the MATLAB script, we can easily type run tes.m in MATLAB Console. The result is hexadecimal file consists of 307 200 data. The data is stored in flash memory. The data is converted into binary by binarization block. This block uses 43 combinational logics, 29 registers, and 601 memory bits.

The binary image data is then fed into line buffer. This block uses 73 combinational logics, 1104 registers, and 59298 memory bits. The memory resource used for this block is high because the line delay size is 640 x 100. The line delay needs 64 000 clock cycles to fill all data for the first computation. The output of this block is taken 4000 at a time and fed into the SAD processor array.

The SAD processor array consists of 4000 PEs made of XOR operation and 12-stages tree adder. The comparator to the threshold is also made in this block. This block uses 15947 combinational logics and 15949 registers. The time needed for computation all image is 307 200 clock cycles.

The last step of the system is to write the result of matching coordinate and number of people in the image. To compute the result of matching coordinate, it is needed 1158 combinational logics and 19 registers. The needed resource is big because the process is dividing and modulo, and we have not optimized this operation. For the people counter, it is only needed 5 combinational logics and 4 registers. For showing in the seven-segments, we use 7 bcd to seven segments blocks, each needs seven combinational logics.

The total combinational logics needed is 17 411, the total register needed is 17 158, and the memory bits needed is 59 899. The overall result of analysis and synthesis resource is shown in the Figure 7.

Analysis & Synthesis Resource Utilization by Entity

	Compilation Hierarchy Node	LC Combinationals	LC Registers	Memory Bits
1	⊟ |top_level_v2	17411 (0)	17158 (0)	59899
2	|CLOCKDIV:clockdiv_0|	76 (76)	33 (33)	0
3	|addr_counter_v2:addr_counter_v2_0|	60 (60)	19 (19)	0
4	|bcd:bcd_0|	7 (7)	0 (0)	0
5	|bcd:bcd_1|	7 (7)	0 (0)	0
6	|bcd:bcd_2|	7 (7)	0 (0)	0
7	|bcd:bcd_3|	7 (7)	0 (0)	0
8	|bcd:bcd_4|	7 (7)	0 (0)	0
9	|bcd:bcd_5|	7 (7)	0 (0)	0
10	|bcd:bcd_6|	7 (7)	0 (0)	0
11	⊞ |compare_4000_1bit:compare_4000_1bit_0|	15947 (4)	15949 (0)	0
5159	|counter:counter_0|	5 (5)	4 (4)	0
5160	⊞ |line_buffer_640_100_4000_1bit:line_buffer_640_100_4000_1bit_0|	73 (0)	1104 (0)	59298
6315	⊞ |read_flash:read_flash_0|	43 (19)	29 (19)	601
6321	|template:template_0|	0 (0)	1 (1)	0
6322	⊞ |write_result_v2:write_result_v2_0|	1158 (25)	19 (19)	0

Figure 7 Analysis and Synthesis Resource Utilization

Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	128.35 MHz	128.35 MHz	pll_0|altpll_component|auto_generated|pll1|clk[0]	

Figure 8 Restricted Fmmax

This system maximum frequency at 85 C is restricted to 128.35 MHz as shown in Figure 8 above. For FPGA implementation, we use frequency 100 MHz generated by Altera Mega Function Wizard PLL. Using the 100 MHz frequency, the full image process which needs 307 200 clock cycles can be done in 3.072 ms or with speed 325 fps.

Simulation is done with ModelSim, the result is shown in Figure 9 and Figure 10. Figure 9 shows simulation result for only SAD processor array block. The result shows that the SAD block works well in computing the SAD value. When the part of the searching image matches the template image, the SAD value is 0. When it is not matched, the SAD value gets higher. Figure 10 shows overall simulation. It shows that when the result has high value, which means the matched part of the searching image is found, the coordinate is computed and the counter counts up the people.

For FPGA implementation, frequency 100 MHz is used. The template image is shown in Figure 11. The searching image we use is shown in Figure 12. In the searching image, there are 5 occurrences of the template image to prove the concept of the computing of the people.

Figure 10 Template Image used for FPGA Implementation

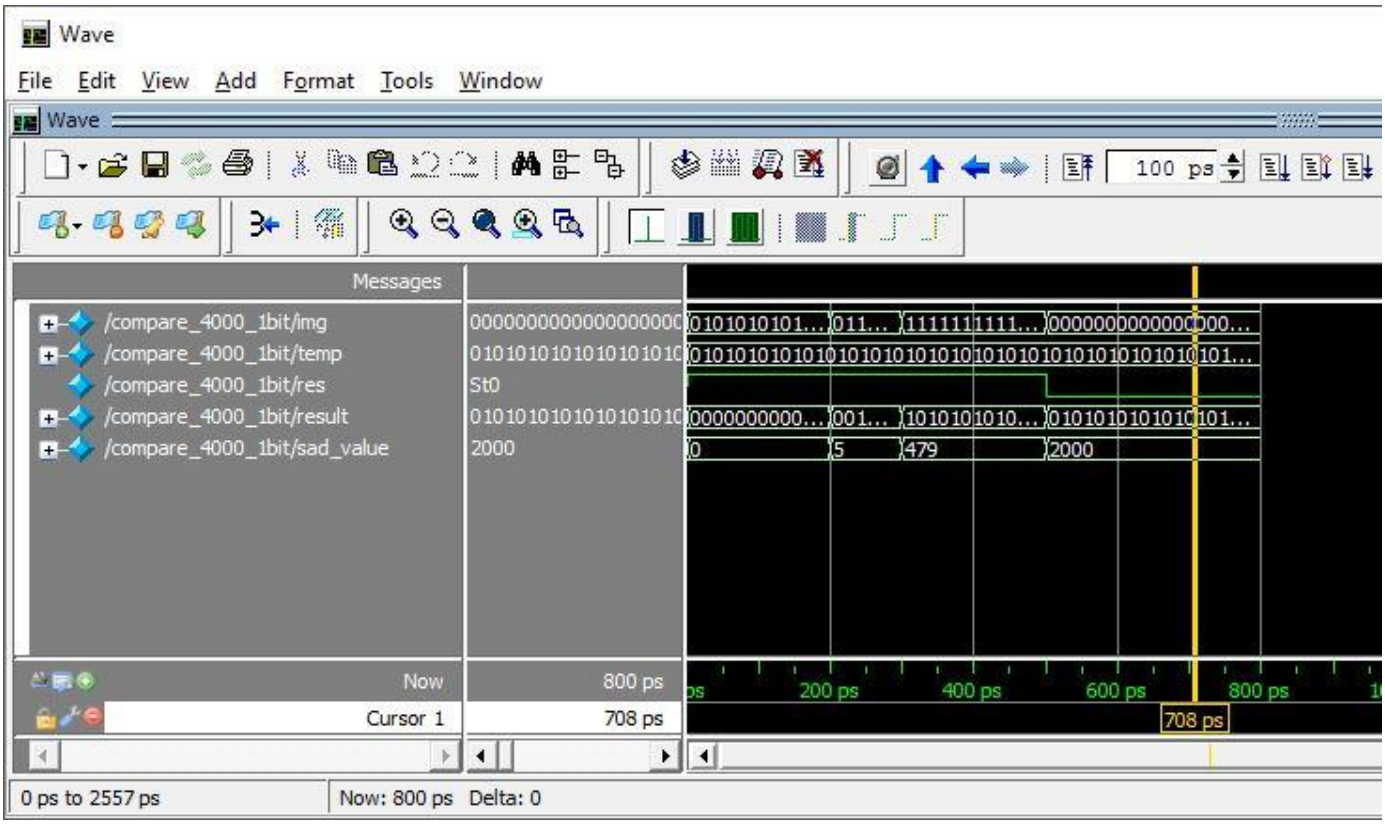


Figure 9 Simulation Result for SAD Block

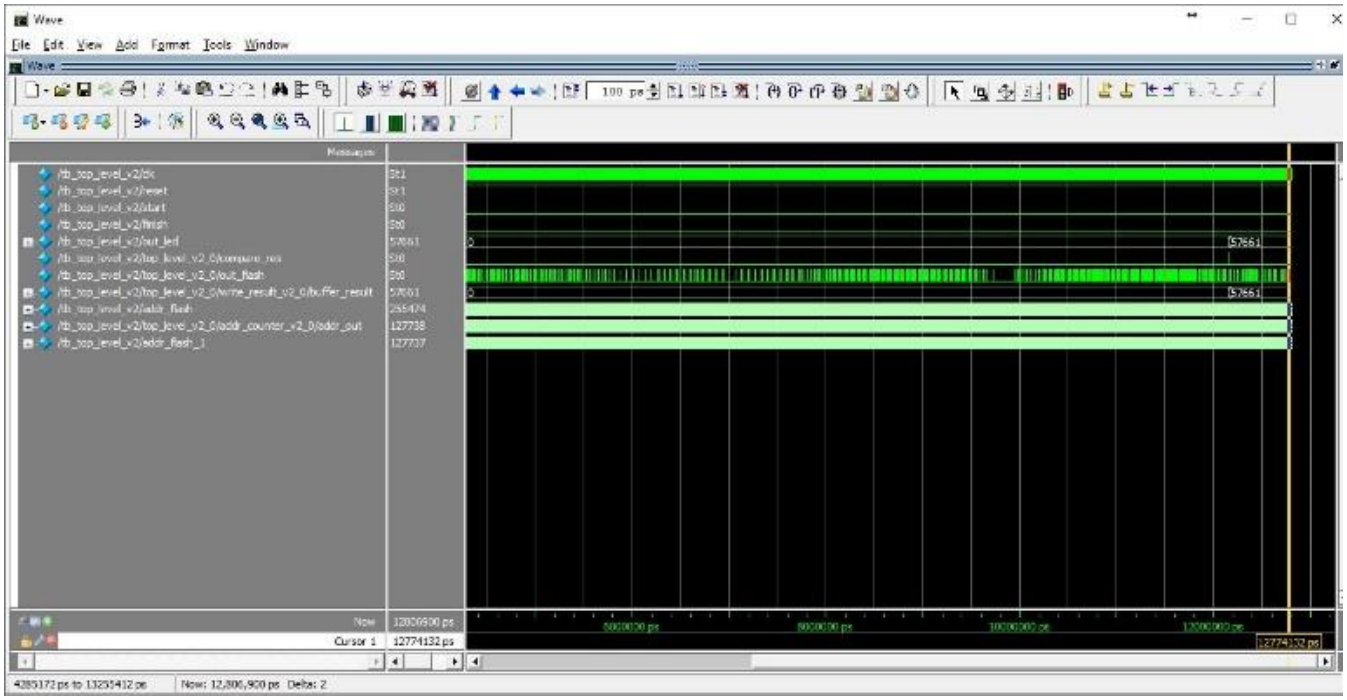


Figure 11 Overall Simulation Result

Figure 12 Searching Image used for FPGA Implementation

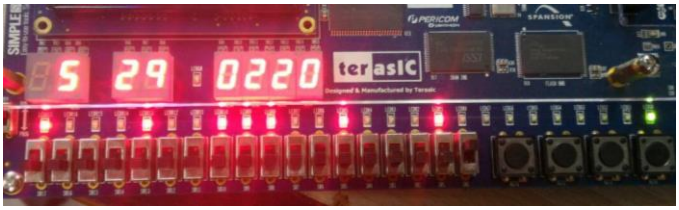


Figure 12 FPGA Implementation Result

The FPGA implementation result is shown in Figure 13 above. The number of people is shown on the most left seven segment, it shows the correct value because the occurrences of the template image in the searching image is 5. The coordinate of the last matched image is shown in the next six seven segments. The first three shows the x value: 290, and the last three shows the y value: 200 which are also correct. This shows that the FPGA implementation we do is successful.

VI. CONCLUSION

From the result, it is shown that this design can have maximum frequency 128.35 MHz and consumes 17 411 combinational logics, 17 158 registers, and 59 899 memory bits. This design is implemented in Altera DE2-115 development board in 100 MHz frequency. The computation time needed to finish all image is 307 200 clock cycles, thus with 100 MHz frequency, it can be finished in 3.072 ms.

REFERENCES

[1] DE2_user_manual.pdf

[2] Kung, S.Y.,1988. *VLSI Array Procesor*. New Jersey: Princeton University

Real-Time Human Detection Circuit by Template Matching using B-HOG Feature Amount

Naotaka Hasegawa, Yuya Kimura, and Yoji Shibuya
Graduate School of Engineering, Chiba University
Chiba, Japan
E-mail: afka3451@chiba-u.jp

***Abstract*—We have designed and implemented a human detection circuit by template matching using B-HOG (Binarized-Histograms of Oriented Gradients) feature amount. Our circuit permits real-time processing of 30fps video at 76% of precision.**

Keywords—Human detection, HOG, Real-time processing

I. INTRODUCTION

We have implemented the circuit to develop a compact and high-precision surveillance camera system. To detect humans, we have used template matching method using the HOG[1] feature amount which is strong to change in the lighting environment and easy to capture shapes of the object.

We have realized the circuit that does not require Block RAM and external memories by the raster-scan using shift registers. Moreover, by various techniques such as a down-sampling and approximation, we have succeeded in real-time processing of 640×480 sized / 30fps video.

Additionally, we have verified precision of the circuit by inputting 50 images in which people appear, and confirmed that the precision of detection is about 76%.

II. SYSTEM

Figure 1 shows the schema of devised system. First, host-PC receives the frame captured by web camera (Logicool HD Pro Webcam C920t). Second, host-PC executes down-sampling to captured frame as pre-processing. Third, host-PC sends down-sampled frame to the evaluation board (Xilinx Artix-7 XC7A100T). The evaluation board executes template matching using B-HOG feature amount and calculates the similarity between the frame and template image while scanning. Fourth, getting higher similarity than threshold, the evaluation board sends the coordinates to host-PC. Finally, host-PC surrounds each detected person in a red frame using the coordinates.

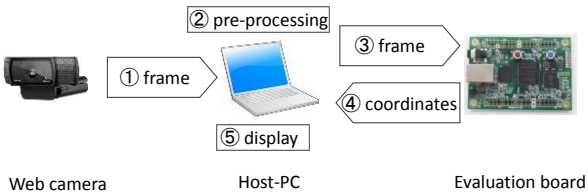


Fig. 1 Devised system

III. CIRCUIT ARCHITECTURE

Figure 2 shows the flow of the implemented system.

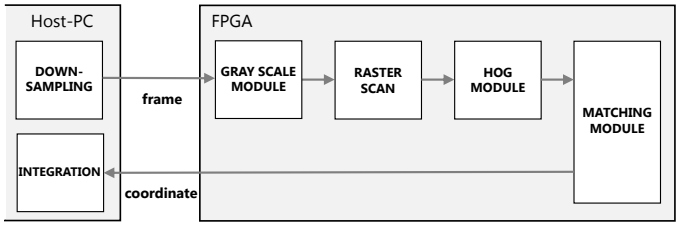


Fig. 2 Human detection system

DOWN-SAMPLING

The frame size (640×480) captured by web camera is too large for processing capacity of the evaluation board to process in real time. Therefore, host-PC executes down-sampling at first. TABLE 1 shows the resolution of down-sampled frame and template image.

Next, as shown in Figure 3, host-PC sends sequentially pixels of the down-sampled frame from the upper left to the evaluation board.

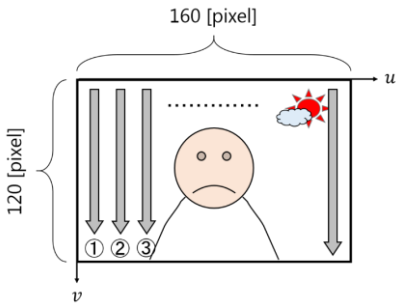


Fig. 3 Transfer order of the input frame

TABLE 1 Down-sampled and template resolution

Name	Resolution [pixels]
Down-sampled frame	160×120
Template	12×30

GRAY SCALE MODULE

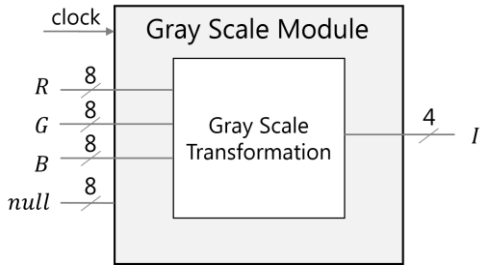


Fig. 4 Block diagram of gray scale module

This module transforms the 24bits color image into a 4 bits gray scale image as shown Eq. (1). Figure 4 shows the block diagram of gray scale module. $R(u,v)$, $G(u,v)$ and $B(u,v)$ are the red, green, and blue intensity value, and $null$ is non-value. $R(u,v)$, $G(u,v)$, $B(u,v)$ and $null$ are 8 bits, and $I(u,v)$ is 4 bits.

$$I(u,v) = \begin{cases} 0.298912 * R(u,v) \\ +\,0.586611 * G(u,v) \\ +\,0.114478 * B(u,v) \end{cases} \quad (1)$$

RASTER SCAN

By using the shift register, we have implemented raster scan of the input frame without deterioration of throughput [2]. We explain about raster scan with $W_T \times H_T$ (=12×30) sized scanning window in $W_i \times H_i$ (=160×120) sized input frame. In this case, we use the 1,350 steps shift register shown in Figure 5 and Eq. (2). In every clock, a pixel intensity value $I(u,v)$ is thrown into the shift register in order of Figure 3.

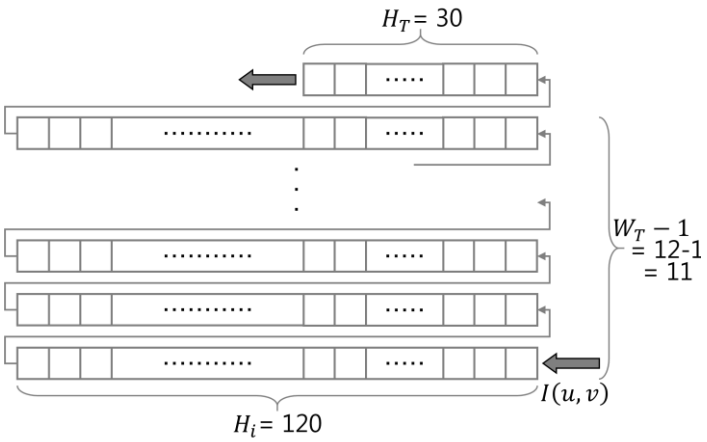


Fig. 5 Shift register of 1,350 steps

$$\text{steps} = H_i \times (W_T - 1) + H_T = 1{,}350 \quad (2)$$

When $I(u,v)$ is thrown pixel by pixel and the shift register is filled up, the relationship of gray area of the shift register and scanning window in input image becomes as Figure 6. In next clock, $I(11,30)$ is thrown into the shift register and $I(0,0)$ is thrown out of the shift register (Fig. 7). Therefore, scanning window moves one pixel in v direction. Repeating this flow every clock, we have enabled raster scan using shift register in order Figure 3. Thereby, it is possible to calculate B-HOG and the similarity in the scanning window area by extracting gray area into selector every clock.

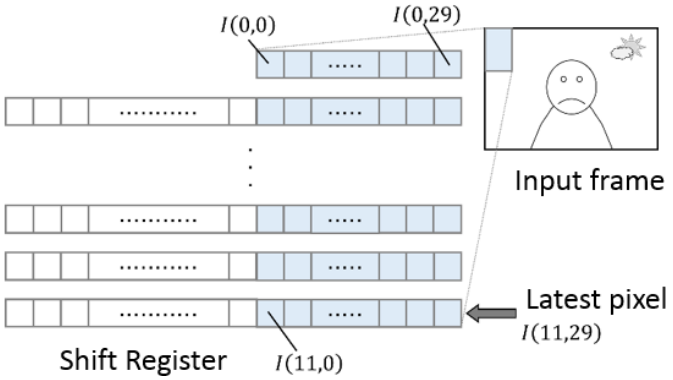


Fig. 6 When shift register is filled up

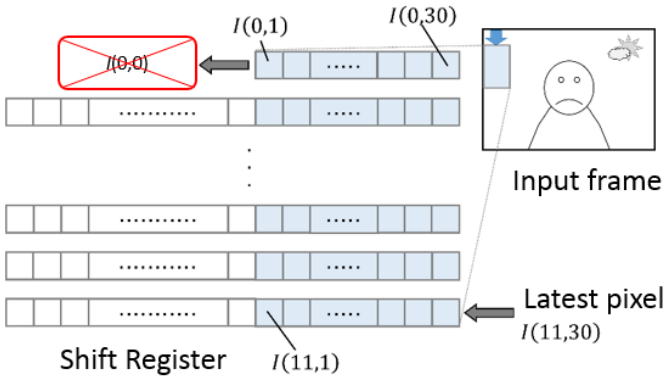


Fig. 7 Relationship between shift register and scanning window

HOG MODULE

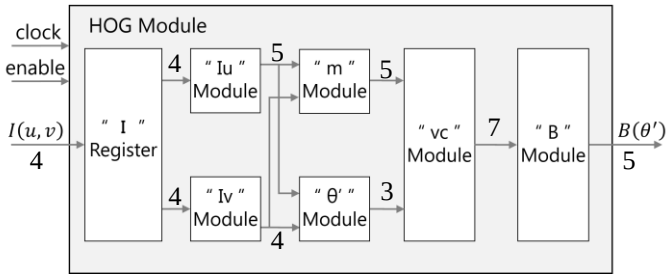


Fig. 8 Block diagram of HOG module

HOG (Histograms of Oriented Gradients) is a feature amount based on histograms of intensity gradients in local area. It is able to capture the shape of objects. However, since it has a high computational cost, we calculate HOG feature amount using some approximate calculations.

Figure 8 shows the block diagram of HOG module. First, an intensity gradient $I_u(u,v)$ of horizontal direction in scanning window area is calculated with intensity value $I(u,v)$ by Eq. (3). Similarly, an intensity gradient $I_v(u,v)$ of vertical direction is calculated by Eq. (4).

$$\begin{cases} I_u(u,v) = I(u+1,v) - I(u-1,v) & (3) \\ I_v(u,v) = I(u,v+1) - I(u,v-1) & (4) \end{cases}$$

Next, a gradient strength $m(u,v)$ is calculated. $m(u,v)$ is usually calculated by Eq.(5). However, Eq. (5) which is Euclidean distance has a high computational cost. Instead, we have adopted Manhattan distance (Eq. (6)) which had a low cost. A gradient direction $\theta(u,v)$ is calculated by Eq. (7).

$$m(u,v) = \sqrt{I_u(u,v)^2 + I_v(u,v)^2} \quad (5)$$

$$m(u,v) = |I_u(u,v)| + |I_v(u,v)| \quad (6)$$

$$\theta(u,v) = \tan^{-1}\frac{I_v(u,v)}{I_u(u,v)} \quad (0 \le \theta(u,v) < \pi) \quad (7)$$

Then, as shown in Figure 9, a gradient direction $\theta(u,v)$ is quantized by $\alpha(u,v)$ whose quantization width is $\pi/8$. When Eq. (9) is satisfied, a quantized gradient direction $\theta'(u,v)$ is generated using Eq. (8).

$$\theta'(u,v) = \frac{\alpha(u,v)}{\pi/8} \quad (8)$$

$$\left(\alpha(u,v) \le \theta(u,v) < \alpha(u,v) + \frac{\pi}{8}\right) \quad \alpha(u,v) = \frac{n\pi}{8} \quad (n = 0,1,2,\ldots,7) \quad (9)$$

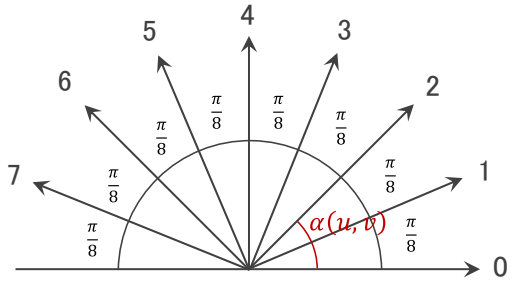


Fig. 9 Labeling

As shown in Figure 10, some pixels are combined and we call this bunch of pixels "cell". A gradient direction histogram $v_c(\theta')$ is calculated every cell by Eq. (10). δ is a delta function of Kronecker. δ becomes 1 when θ' is the same histogram element as $\theta(u,v)$. Otherwise, δ becomes 0.

$$v_c(\theta') = \sum_u \sum_v m(u,v)\delta[\theta', \theta(u,v)] \quad (10)$$

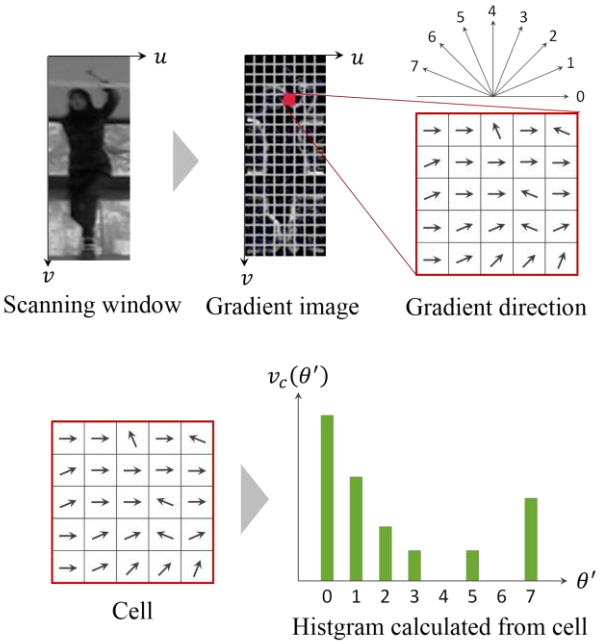


Fig. 10 HOG feature amount

Subsequently, some cells ($q \times q$ cells) are combined and we call this "block". B-HOG (Binarized-HOG) [3] is calculated by performing a threshold processing as shown Eq. (11). In this design, we have set th =0.03.

$$b_c(\theta') = \begin{cases} 1 & if\ v_c(\theta') \ge th \times \sum_{k=0}^{7q^2} v_c(k)^2 \\ 0 & otherwise \end{cases} \quad (11)$$

Last, as shown Eq. (12), the binarized histogram $B(\theta')$ is generated using $b_c(u, v, \theta')$ every block.

$$B(\theta') = \sum_u \sum_v b_c(u, v, \theta')\delta(\theta') \quad (12)$$

In this way, B-HOG of the image in the scanning window area is sequentially calculated by pipeline processing. TABLE 2 shows the size of a cell and a block in this design.

TABLE 2 Size of cell and block

Name	Size
Cell	6[pixel]×6[pixel]
Block	2[cell]×2[cell]

MATCHING_MODULE

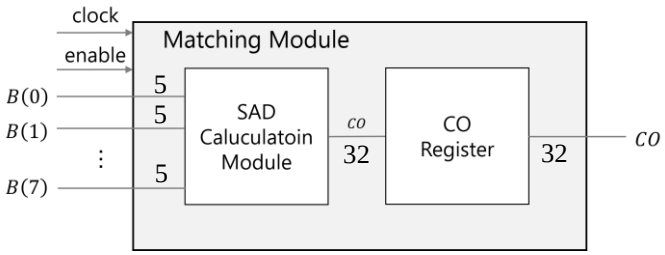


Fig. 11 Block diagram of matching module

Figure 11 shows the block diagram of matching module. Template matching is executed by using the calculated $B(\theta')$. In this design, we have adopted SAD (Sum of Absolute Difference), which has a low computational cost.

We regard $B(\theta')$ of the template as $B_T(\theta')$ and that of the image in the scanning area as $B_I(\theta')$. Then, SAD is expressed by Eq. (13). When we get smaller SAD, the similarity is high. When SAD is less than 10, the left upper coordinates CO of the scanning window are stored away at the registers.

When a calculation of the template matching for one frame is finished, the detection coordinates are transmitted to host-PC. In matching calculation, we have adopted the table reference method for a calculation of $B_T(\theta')$, and reduced calculation cost. TABLE 3 shows $B_T(\theta')$ values.

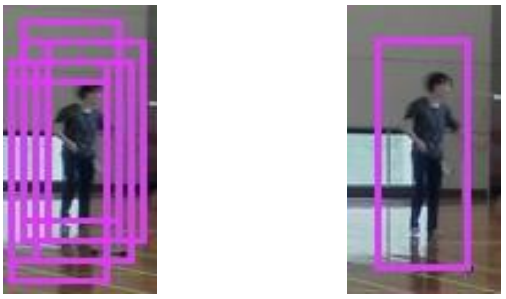
$$SAD = \sum_{\theta'} |B_T(\theta') - B_I(\theta')| \quad (13)$$

TABLE 3 $B_T(\theta')$ values

$B_T(\theta')$	Value
$B_T(0)$	8
$B_T(1)$	9
$B_T(2)$	6
$B_T(3)$	8
$B_T(4)$	4
$B_T(5)$	4
$B_T(6)$	9
$B_T(7)$	6

INTEGRATION

The coordinates transferred to host-PC are necessary to integrate as shown in Figure 12. We have adopted Mean Shift[4] and integrated coordinates.



(a) before (b) after

Fig. 12 Coordinate integration

IV. CIRCUIT EVALUATION

A. Circuit size

TABLE 4 shows the performance of the designed circuit. The maximum frequency is 124.097 MHz. Therefore, we have set the operating frequency to 100 MHz. As this table shows, this circuit does not use Block RAM at all.

TABLE 4 Performance of designed circuit

Maximum Frequency [MHz]	**124.097**
Maximum combinational Path delay [ns]	0.001
Minimum period [ns]	8.058
Number of Slice Register	12,708
Number of Slice LUTs	44,516
Number of LUT Flip Flop pairs used	45,746
Number of Block RAM/FIFO	0
Number of BUFG/BUFGVTRLs	2
Levels of Logic	19

B. Computational time

We have evaluated the computational time of this designed circuit which operate at 100 MHz.

- ***Frame transfer***
  The transfer time for down-sampled frame is 638 μs.
- ***Gray scale transformation***
  This computational time is contained in frame transfer time. Therefore, we do not consider about this time.
- ***Raster scan***
  The delay until raster scan is started is 1,350 clocks which are the steps of the shift register. However, its computational time is contained in frame transfer time.
- ***B-HOG and Similarity***
  B-HOG calculation and similarity calculation are performed by pipeline processing of 16 stages. Therefore, these time is 0.16 μs (=16×10 ns).
- ***Coordinate transfer***
  The time to transfer a detected coordinate to host-PC is 265 μs . Therefore, when n detected coordinates are transferred, the time become 265 n μs.
- ***Total***
  TABLE 5 shows the detail of the computational time. The coordinate transfer time depends on the number of coordinates. The web camera operates at 30 fps, so that the total computational time in 1 frame must be smaller than 33 ms to realize real-time processing. The real-time processing is possible when the number of detected coordinates are less than 100 ($n \leq 100$).

TABLE 5 Detail of computational time

	Function	Computational time [μs]
Hardware	Frame transfer	638
	Gray scale transformation	—
	Raster scan	—
	B-HOG and Similarity	0.16
	Coordinate transfer	265n
	Total	**638.16+265n**
Host-PC	Down-sampling	86.0
	Coordinate integration	9.69
	Total	95.69
	Grand total	**734+265n**

C. Accuracy

We have verified the accuracy of this circuit using 50 images (Fig. 13). The number of people in these images is 118.

We have calculated detected rate R_d, undetected rate R_u and false detected rate R_f. Equations (14)-(16) show each evaluation equations[5]. “Number of detections” is the number of people detected correctly. “Number of false detections” is the number of detected objects which it is not human. “Total number of detections” is a grand total of detected coordinates.

$$R_d = \frac{\text{"Number of detections"}}{\text{"Number of people for detection"}} \quad (14)$$

$$R_u = 1 - R_d \quad (15)$$

$$R_f = \frac{\text{"Number of false detections"}}{\text{"Total number of detections"}} \quad (16)$$

Fig. 13 Images used verification

Since we have aimed the reduction of the circuit size and the improvement of the computational speed, the various approximate calculations are performed at the designed circuit. We have examined whether these influence the accuracy. Then, we have compared the results of calculation including no approximation by software with that of calculation including approximation by hardware.

TABLE 6 shows both accuracies, and Figure 14 shows the output results. About detected rate R_d and undetected rate R_u, software had slightly better accuracy than hardware. However, about false detected rate, hardware had better accuracy than software. Most of the detection windows surround the people in Figure 14(a), which is the result of hardware. Thereby, we are able to confirm that the false detected rate R_f is low. On the other hand, in Figure 14(b), there are few un-detections, but there are many false detections, which is the result of software.

When we consider the balance of undetected rate R_u and false detected rate R_f, the both accuracies are nearly equal.

Therefore, we have succeeded in maintaining the accuracy, and reducing the cost of B-HOG calculation.

TABLE 6 Comparison of detection results

	Detected rate R_d	Undetected rate R_u	False detected rate R_f
Hard-ware	76.3%	23.7%	48.3%
Soft-ware	79.7%	20.3%	59.8%

(a) Hardware

(b) Software

Fig. 14 Output results

V. CONCLUSION

We have implemented the human detection system which uses template matching based on B-HOG feature amount and detects humans in pictures captured by web camera. We have resized captured images from 640×480 to 160 × 120 in host-PC. Thus, we have reduced whole calculation cost. Additionally, when we send captured frame, we have implemented fast raster scan without external memory by storing the pixel data of captured frame into shift register. Furthermore, we have implemented B-HOG calculation using approximation whose calculation cost is small, so that we have shortened calculation time more.

As a result of these techniques, we have succeeded in real-time human detection of 30fps video captured by web camera. Therefore, we have achieved development of a compact and high-precision surveillance camera system we desired.

REFERENCES

[1] Yuji Yamauchi, Hironobu Fujiyoshi, Bon-Woo Hwang and Takeo Kanade, “People Detection Based on Co-occurrence of Appearance and Spatiotemporal Features”, Pattern Recognition, 2008. ICPR 2008. 19th International Conference on, 2008.

[2] 柴田裕一郎, “メモリ・レスの画像検出回路を実現する”, Digital Design Technology, CQ 出版, 15, 88-105, 2012.

[3] 松島千佳, 山内悠嗣, 山下隆義, 藤吉弘亘, “物体検出のための Relational HOG 特徴量とワイルドカードを用いたバイナリーのマスキング”, 電子情報通信学会論文誌 D, Vol.J94-D, 8, 1172-1182, 2011.

[4] D. Comanic and P. Meer, “Mean Shift Analysis Applications”, Computer Vision, 1999, The Proceeding of the Seventh IEEE Conference, 2,1197-1203, 1999.

[5] 前渕啓材, 原田祥吾, 呉海元, 和田俊和, “ハウスドルフ距離による近赤外線画像からの夜間歩行者検出”, 画像の認識・理解シンポジウム(MIRU2011), IS2-30, 703-709, 2011.

IMPLEMENTATION OF MOVING OBJECT DETECTION ALGORITHM ON FPGA

Le Ngoc Linh, Ngoc Viet Tien
Dept. School of Electronics and Telecommunications
Hanoi University of Science and Technology
Hanoi, Vietnam
linhctqk22t@gmail.com
tiena2cva@gmail.com

***Abstract*—In this report, we demonstrate an FPGA implementation of a moving object detection algorithm. The goal of the project is to design an IP core which is capable of processing a video and detecting moving objects in it using background subtraction. Input of the core is frame data of the video, fed to a small on-chip memory attached with the module. Output is the location of each object. The operation of the core is then demonstrated by integrating it to a embedded system, with a MicroBlaze processor and a TFT Display controller to display the result on a DVI screen.**

***Keyword*— Video processing, moving object detection, background subtraction, FPGA.**

I. INTRODUCTION

Nowadays, real-time constraint has became a mandatory requirement in almost every image processing based applications. Identifying the moving objects from a video sequence is the fundamental and critical task in many systems such as robotics, gaming system, indoor security and outdoor surveillance [1]. Detecting and locating an object inside a image sequence is the prior task to do more complex actions like classifying, counting objects or even warning a suspicious event. This field of research has became a very attractive branch in computer vision.

When it comes to security or surveillance systems, the market has already had many commercial off-the-shelf products like Smart Ip camera, CCTV security camera, with built-in motion sensor which enables the camera to detect moving object in front of it.

Implementing such a system on FPGA (field-progamable gate array) has some advantages. Beside the fundamental advantages of FPGA technology when comparing with ASIC like higher performance, smaller NRE cost, shorter time to market, easy to upgrade and maintenance, ..etc, detecting moving object using pure image processing can replace the use of a motion sensor, then decrease the cost-per-unit of the product. On the other hand, FPGA is chosen due to its reconfigurable ability. Without requiring hardware change, the use of FPGA type devices expands the product life by updating the configuration bitstream remotely. Furthermore, it offers the opportunity to utilize hardware/software co-design for developing a high performance system for different applications by incorporating processors (hardware core processor or software core processor), on-chip bus, memory, and hardware accelerators for specific software functions [2]. Also, using prebuilt reconfigurable IP cores has increasingly become a trend in System on Chip (SoC) designs because of their flexibility and powerful functionality. Many modern systems are built by connecting IP cores from other manufacturers, make it faster to propose prototype to the clients.

Base on those above reasons, we decided to implement a simple moving object detection system using background subtraction algorithm on an FPGA chip. The moving object detection (MOD) logic, works alongside with a small on-chip memory (OCM) block, is packed into an IP core, which compatible with any Xilinx PLB embedded system, make it easy to integrate to bigger system, or even fabricate to a standalone LSI chip. The core achieved very high speed, only takes 2.7 ms for a VGA-resolution frame.

The rest of this report is organized as follow: section II presents the details of the algorithm we implemented. Section III describes about the architecture of the MOD logic. The implementation result on actual hardware is presented in section IV. Finally in section V, we make some conclusions and propose the improvement for our system in the future.

II. MOVING OBJECT DETECTION ALGORITHM

The algorithm is litteraly sequential. The first frame of the processed video sequence is treated as background. For the rest of the sequence, a same set of tasks is performed. The order of those tasks is as follow:

- Step 1: Read data from next frame. Calculate the absolute difference of two image: background and current frame, and then binarize the diferential result by applying a threshold.
- Step 2: Remove noise by appling a erosion filter on the binary image from step 2.
- Step 3: Scan the filtered image from top to bottom to detect objects and save the location of them.

- Step 4: Go back step 2 until the end of the video sequence.

The details about these tasks is presented in following sub-sections.

A. *Calculate and binarize the absolute difference image*

After the background image Bg(x, y) is obtained, subtract the background Bg(x, y) from the current frame $Cur_k(x, y)$. If the pixel difference is greater than threshold Th, then that pixel belongs to the moving object (white pixel), otherwise, it belongs to the background (black pixel). The expression is as follows:

$$Fg_k(x,y) = \begin{cases} 1 & if\ |Bg(x,y) \quad Cur_k(x,y)| > T \\ 0 & if\ ot\ \ erwise \end{cases}$$

Where $Fg_k(x, y)$ is the binary image of differential result. Th is grayscale threshold, its value determines the accuracy of object detection. The fixed threshold used in this algorithm is only suitable for an ideal enviroment with small light intensity variance.

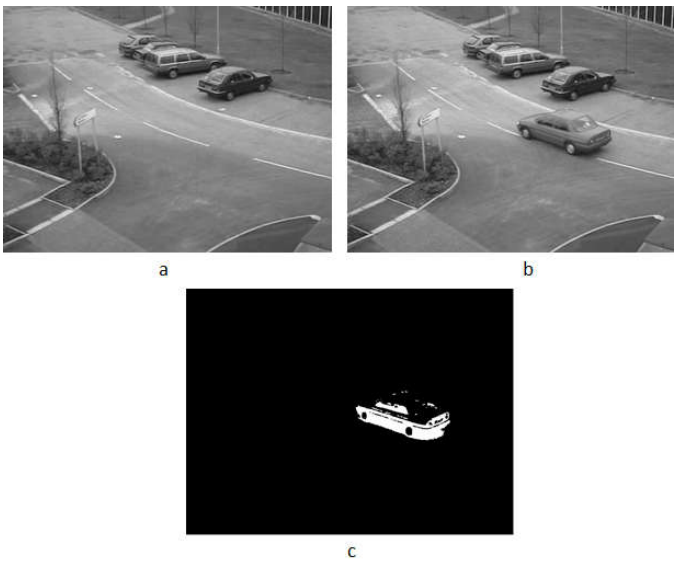


Fig 1. (a) Background image. (b) Current frame. (c) Absolute difference of two image

B. *Remove noise with erosion filter*

Noise is the main factor that makes the marking process less accurate. To reduce it, we use a filter similar to the erosion filter, with a 3x3 sliding window. If a pixel p has more than 5 white pixels around, then it is white pixel. Otherwise, it's considered as noise and removed (set to black pixel).

$$Fg'(x,y) = \begin{cases} 1 & if \sum_{i=x-1}^{x+1} \sum_{j=y-1}^{y+1} Fg(i,j) \geq 5 \\ 0 & if\ ot\ \ erwise \end{cases}$$

Fig 3 below shows an example of an binary image before and after filtered. In this image, pixel 1, 2 and 3 are set to white, while others are erased.

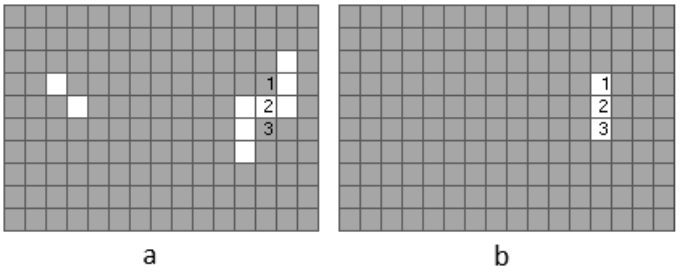


Fig 2 (a) before filtered (b) after filtered

C. *Object marking*

After filtered, the binary image is used as input for the marking process. The image is scanned from top to bottom. If a row has at least one white pixel, an objectflag is set, indicates that there's an object has been detected. The flag is set until a row with all black pixel is encountered. The region of detected object is continuously update while the image is being scanned. The flow chart for this process is shown in Fig.3.

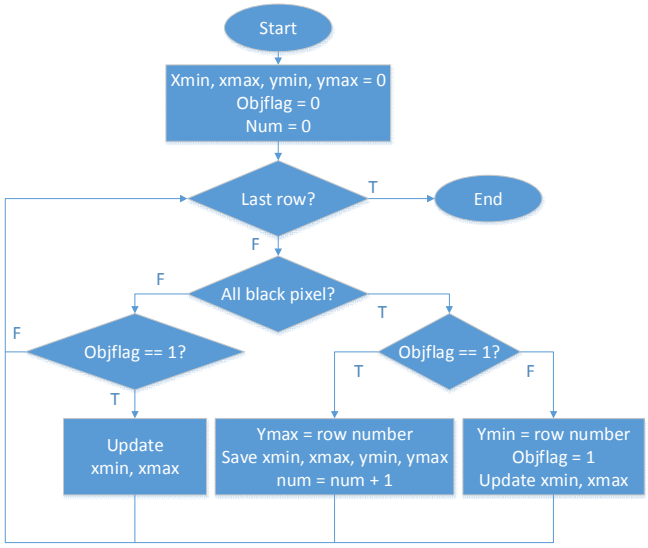


Fig 3. Object detection algorithm

Because the image is scanned in only one direction, objects with overlap heights are merged and considered as one object. Futhermore, the objectflag is set whenever a row with at least one white pixel is encountered, a small noise pixel could easily make the marking process fail.

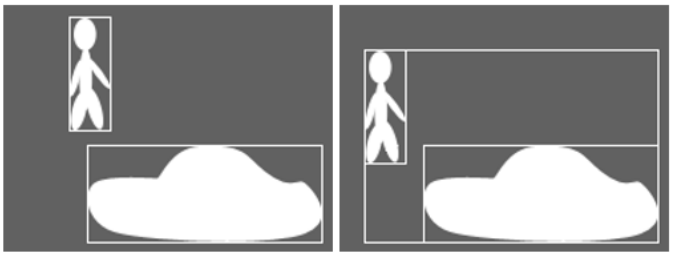


Fig 4. Objects with (a) nonoverlap heights (b) overlap heights

III. HARDWARE ARCHITECTURE

A. Top level diagram

To implement the above algorithm, we built a logic that manipulates content of an on-chip memory (OCM) block. Each task of the algorithm is realized by a separate module. The top-level diagram of the moving object detector (MOD) logic is as follow:

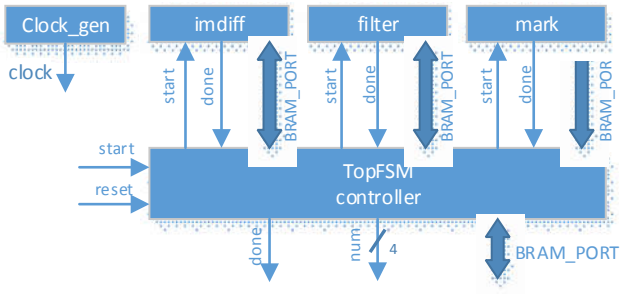


Fig 5. MOD logic top level diagram

In the above diagram, imdiff, filter and mark are the logic that performs task 1, 2 and 3 in the algorithm described before in section II. Each of the three modules takes control signals from the TopFSM controller, to generate address and data to send to the OCM. TopFSM controller is also in charge of selecting the address and data from these three processing modules to pass to the OCM, and then send back the data received from the OCM to approriate destination.

The OCM is a dual-port ram. The first port is used to get data from video input, and the other port is connected to the MOD logic. To reduce the size of the OCM, we only store grayscaled frames. The needed size is calculated as follow:

Table 1. OCM content

Background frame	640*480 = 307200 bytes
Current frame	640*480 = 307200 bytes
Binary image	640*480/8 = 38400 bytes
Object location	16*64/8 = 128 bytes (*)
Total	652928 bytes

(*) assume that 16 objects can be detected at maximum, each object needs 64bits to store 4 values: xmin, ymin, xmax, ymax

The finite state machine inside TopFSM controller is presented below:

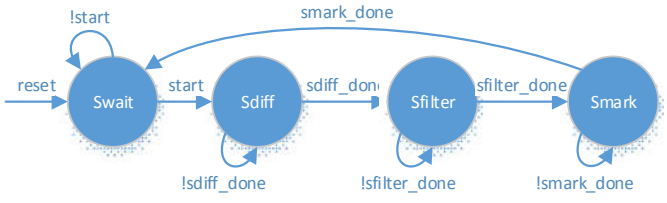
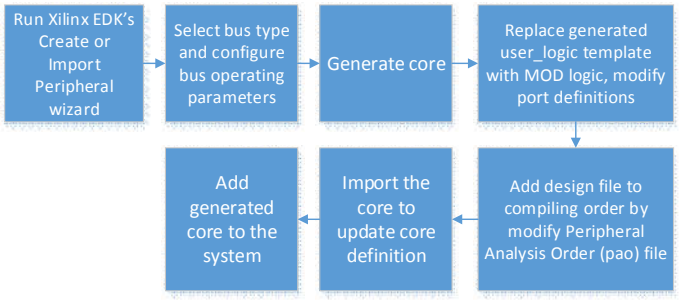


Fig 6. MOD FSM

To parallelize the computation, we organize the OCM as 128 bits width by 40808 words depth. This way, each address in the OCM can store 16 gray pixels, hence increase the processing speed by 16 times

B. IP core packing

After the simulation, the MOD logic is packed into an IP core. The core is communicate with other core throught Xilinx CoreConnect technology. To do this, we use Xilinx Platform Studio to add a IPIF (intellectual property interface) layer to the MOD logic. The design flow is as follow:



The core is packed as a slave peripheral on the PLB (processor local bus) system. We implemented three software-accesible registers to this core, so that a processor can control and monitor the operation of the core by reading and writing from/to these registers:

- reg_start: user assert a start signal to the MOD logic by writing value 1 to this register. When the start signal asserted, the MOD FSM changes from Swait to Sdiff state.
- reg_status: user observe the operating status of the MOD logic by reading this register. It has value of 0xFFFFFFFF when the FSM stay at swait, otherwise it is 0.
- reg_num is the result register. It stores the number of detected objects. reg_status and reg_num is not writeable.

IV. IMPLEMENTATION RESULT

The MOD logic is simulated using Mentor ModelSim. The background frame and a tested current frame is pre-loaded into the OCM. By observing signals waveform and memory content, we can verify that our design works as expected.

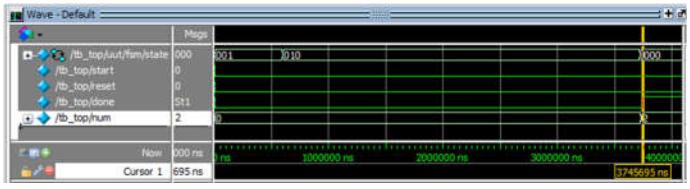


Fig 7. simulation result

The MOD logic is synthesized by Xilinx ISE design suite. The targeted FPGA device is Xilinx Virtex6 series, code name xc6vlx240t-1ff1156. Resource utilization is shown in the table below:

Table 2. Device Utilization Summary

Resources	Used	Available	Utilization
Slice Registers	5747	301440	1%
Slice LUTs	5953	150720	3%
Block RAM/FIFO	160	416	39%

The maximum frequency at which the MOD logic can operate is 139.169MHz. The number of clock cycles needed to process one frame is 374563 cycles . Put these number together, we can calculate the minimum time needed to process one frame is about 2.7 ms , meaning the maximum frame rate that the MOD logic can process is 371 frames per second.

Taking the design from Sanchez et al [1] as a reference design, we can confirm that our design has higher processing speed. Because the used algorithms are not identical and the resolution of processed videos are not the same, comparing the time needed to process one frame might not be approriate. But our design has higher maximum frequency (139,17MHz vs 23.03MHz), mostly because we used OCM instead of SRAM as memory storage. Take into account the computational parallel level, our design can produce higher throughput than theirs. Other than that, our design is able to locate and save location of multiple objects, while their algorithm is suitable for calculating just one object's gravity center.

To demonstrate the operation of our design, we used Xilinx Platform Studio to connect it with a MicroBlaze processor and some other IP cores. The block diagram of the testing system is presented in Fig 8.

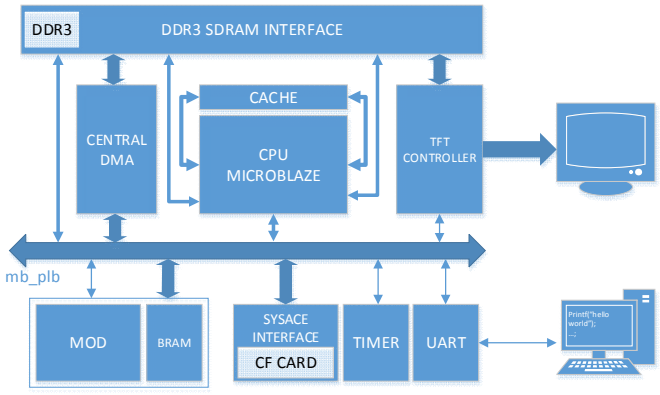


Fig 8. Overall architecture of testing system

In the system above, the video stored on CF card is read by SysACE interface. Because reading data directly from CF card is considerably slow, the whole video will then be stored on a DDR3 SDRAM, interfaced by a core named MPMC (multi-port memory controller). The video data is copied from DDR3 memory to BRAM by the CDMA – central direct memory access, to process by the MOD core. The result is exported to DVI screen by the TFT Controller. Operation of the system is monitored and controlled by the CPU. We also used a timer and uart controller to debug and communicate with the host PC.

After synthesized, the system is downloaded to an evaluation board named ML605 from Xilinx. The board consists of a Virtex6 FPGA and many peripheral devices, includes a Micron MT4JSF6464HY 512MB DDR3, a CF card slot, along with Xilinx XCCACE-TQG144I SystemACE chip, USB-UART, USB-JTAG and DVI port.

Fig 9. Running system

V. CONCLUSION AND FUTURE WORK

In this project, we have succesfully implemented a simple moving object detection algorithm. The design is capable of processing up 371 VGA-resolution frames per second, produce the number of objects and location of them. It is packed into an IP core which compatible with any system designed using Xilinx CoreConnect technology. The core can be easily reused in any security and surveillance application which in need of moving object detection.

There are many things about the algorithm can be improved. Firstly, the algorithm works on a ideal environment with small light intensity variance. To make it works in more complex condition like weather change, low light intensity, a dynamic binarization threshold must be applied. Beside that, in the marking process, we currently scan the image only in vertical direction. To increase the accuracy of the marking process, we can apply more than one scan direction. Finally, in order to make the design suitable for low-cost FPGA, we have to replace the use of on-chip memory block. One option is to read/write data directly from/to the DDR memory. Doing that also allow the design to be modified to process higher resolution video. As a trade-off, using DDR memory will decrease the process speed when comparing with OCM based design.

REFERENCES

[1] C. Sanchez-Ferreira, J. Y. Mori, C. H. Llanos, "Background Subtraction Algorithm for Moving Object Detection in FPGA" Programmable Logic (SPL), 2012 VIII Southern Conference.

[2] S. Nazeer Hussain, K. Sreenivasa Rao, S.Mohammed Ashfaq, "The Hardware Implementation of Motion Object Detection Based on Background Subtraction" - International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 10, October 2013.

[3] Y. Zhou and Y. Tan, “Gpu-based parallel particle swarm optimization”, Proc. IEEE Int. Congress on Evolutionary Computation, pp. 1493 – 1500, 2009.

[4] D. Munoz, C. Llanos, L. Coelho, and M. Ayala-Rincon, “Comparison between two FPGA implementation of the particle swarm optimization algorithm for high-performance embedded applications”, The IEEE Fifth International Conference on BioInspired Computing: Theories and Applications (BIC-TA 2010), vol. 1, pp. 1637 – 1645, 2010.

[5] Zhen Tang and Zhenjiang Miao, "Fast Background Subtraction and Shadow Elimination Using hnproved Gaussian Mixture Model", Haptic, Audio and Visual Environments and Garnes, 2007. HAVE 2007. IEEE International Workshop on 12-14 Oct. 2007, pp. 38 – 41.

[6] Niu Lianqiang and Nan Jiang, "A moving objects detection algorithm based on improved background subtraction", Intelligent Systems Design and Applications, 2008. ISDA '08. Eighth International Conference on Volume 3, 26 – 28 Nov. 2008, pp. 604 – 607.

[7] N.J.Bauer and P.N.Pathirana, "Object focused simultaneous estimation of optical flow and state dynamics", Intelligent Sensors, Sensor Networks and Information Processing, 2008. ISSNIP 2008. International Conference on 15 – 18 Dec. 2008, pp: 6l – 66.

[8] M.Dimitrijevic, "Human body pose detection using Bayesian spatiotemporal templates", 2007 International Conference on Intelligent and Advanced Systems, 2008, pp. 764 – 9.

[9] E. M. Saad, A. Hamdy and M. M. Abutaleb, “FPGA-Based Implementation of a Low Cost and Area Real-Time Motion Detection”, 15th International Conference of Mixed Design MIXDES, pp. 249 – 254, Poznan, Poland, 2008.

Parallel Morphological Template Matching for Fast Human Detection Application

Radhian Ferel Armansyah [(1)], Fadhli Dzil Ikram [(2)], Swizya Satira Nolika [(3)]
radhi.ferel@gmail.com[(1)], adzil13@gmail.com[(2)], swizya.s.n@gmail.com [(3)]
Deparment of Electrical Engineering, School of Electrical and Informatics Engineering
Bandung Institute of Technology, Jl Ganesha No. 10 Bandung, 40132, Indonesia

***Abstract*—this paper would provide the approach of human detection using parallel morphological/binary template matching method. Design of ASIC would focus on the fast template matching process with small enough area for human detection application. The approach is by calculating the SAD (Sum of Absolute Difference) between the original image (640 x 480 pixels) and template image (40 x 100 pixels). For a matched object found in the image, a boundary will be placed around the object. For every window, the difference between source and template is calculate paralel which for every clock there are 100 values of the difference. That's way the latency of this design is 40 clocks, while the throughput is 1 clock for different column in same row and 39 clock for different column in different row. This design of ASIC would be created by using Verilog code and implemented by using Cyclone II FPGA in Altera DE2 board.The performing time of the implementation is 307200 clocks. The clock frequency used in implementation is 50MHz. The time needed for every frame image is 6.144 x 10^{-3}s. So, the frame speed can reach until 162 fps for video application. The area needed for this design is 12.732 elements from total combinational functions and 4.460 from dedicated logic registers.**

***Index Terms*—Parallel, Fast Human Detection, Template matching, SAD**

I. INTRODUCTION

HUMAN detection have so diverse applications in digital image processing. There are several methods to approach the human detection algorithm. One of the simple techniques is template matching. Template matching method consists of two images, one as the template and one as the searched image. Template image consist the object that is searched in the image. After the object is found, the system will create an indicator in object image about the position of matched object.

This paper contains the architecture of the design system. First is the flowchart of template matching process that basically contains four basics which are grayscalling, binerization, matching, and drawing the border. Design of ASIC would focus on the fast template matching process for human detection. Then, the paper shows the ASIC design of matching process. Both searched image and template image are saved in ROM using ALTSYNCRAM Megafunction.The SAD calculation are done per column in the window which size are same as the template image. The time needed for one column is one clock. The array of image and template image are controlled by line buffer. This design is synthesized by Altera Quartus and simulated by Modelsim to see the performance. The image are save in .mif format and save to ROM using ALTSYNCRAM Megafunction. The design will be implemented by using Cyclone II FPGA in Altera DE2 board for prototyping the circuit.

II. TEMPLATE MATCHING

Template matching is a technique used for identifying any object in the source image that is similar to the template image. Template matching needs two primary components which are source image and template image.

A. SAD (Sum of Absolute Difference)

SAD sums the difference of gray level per pixel between source image and template image. Assume that I(x, y) is the source image, while T(x, y) is the template image, SAD can be described by the following equation. For Binary template image, the SAD value can be simplified as:

$$SAD = \sum_{x=0}^{X-1}\sum_{y=0}^{Y-1} |I(x,y)\ XOR\ T(x,y)| \quad (1)$$

, where x and y represent a pixel position. The smaller SAD value shows the higher chance of similarity between the template and source image.

B. Grayscaling

The grayscalng method based on NTSC coefficient standard can be described as

$$Y = 0.298912R + 0.58661G + 0.14478B \quad (2)$$

The verilog code is lack of the floating point support so that the coefficients are simplified by multiplier constant as following

$$Y = \frac{1}{4}R + \frac{5}{8}G + \frac{1}{8}B \quad (3)$$

C. Thresholding

Thresholding is used to binerized the grayscale image to black and white representation. This process is so essential to the SAD method that can simplify the substraction in SAD by using XOR instead. It allow the FPGA to compute faster with small number of gate array. First, the thresholding level is determined by the median gray level of image histogram. Then, the image is binerized based on whether the intensity are greater than the thresholding level.

III. SYSTEM FLOWCHART

Face detection using template matching, basically, consists of 4 major process which are grayscalling, binerization, matching process and drawing the boundary. The image is in RGB format that first must be grayscalling to make it as one dimensional image. Then, the grayscalled image is binerized based on its threshold level that has been searched by median gray level in its histogram. Binerization is needed to simplify the matching process where the SAD calculation in binary image can be achieved using XOR operation. In the matching process, the pixel, which SAD value is less than the threshold, is considered as the matched pixel. The threshold level of SAD value has been determined before as 500. This process gives out the pixel position of the matched ones. The last process is drawing the boundary of matched pixels based on the result in matching process.

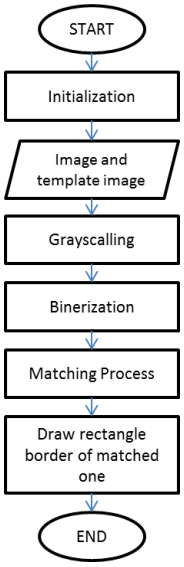


Figure 3.1 System flowchart

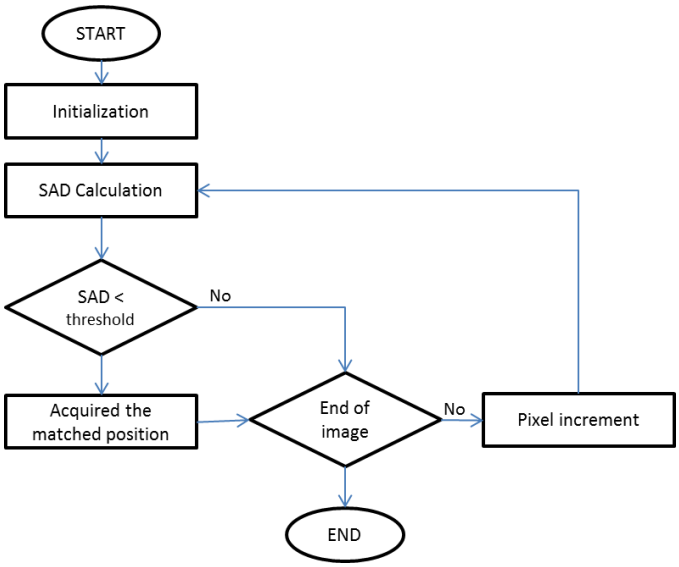


Figure 3.2 Matching process flowchart

IV. PREPROCESSING

The first two process are implemented in Verilog and simulated using Modelsim.

A. RGB Image Input

The RGB image input blocks consist of a RGB memory that loads directly from a converted image that forms a memory map so it can loaded by ModelSim.

B. RGB to Grayscale converter

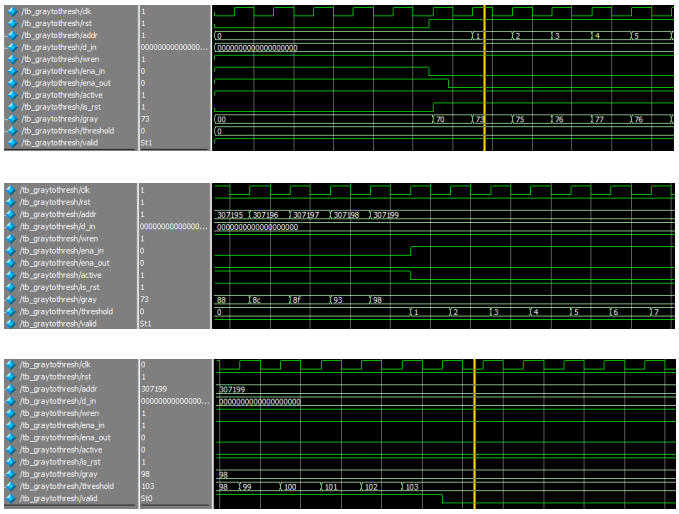
The grayscale converter takes data from RGB memory data and translates to a grayscale image using NTSC coefficient. The grayscale image data sent to a grayscale image memory and histogram median detection module to be further processed to a binary image.

Figure 4.1 Grayscalling process

The above image is processed by the ModelSim simulation that has been converted back from memory map data to a bitmap data and analyzed by a image editor program on the computer.

C. Grayscale Thresholding with Histogram

By putting the grayscale data on histogram, the histogram module will calculate the threshold for the image binarization.



The thresholding process is divided by three main process. The first one will fill the histogram memory with data, calculate the median, and output the threshold value as seen on above image.

D. *Image Binarization (Grayscale to BW converter)*

The grayscale image stored on the grayscale memory will be compared with a threshold from the histogram blocks. A value that is bigger than the threshold value will be assumed as ones, and vice versa. The resulting binary image stored on a binary image memory.

Figure 4.3 Binerization

The above image is binarization process taken from ModelSim simulation and converted back from memory map file to a bitmap file so it can be analyzed with image editor.

V. ARCHITECTURE DESCRIPTION

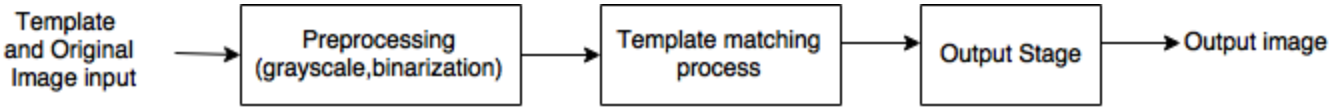


Figure 5.1 General block diagram

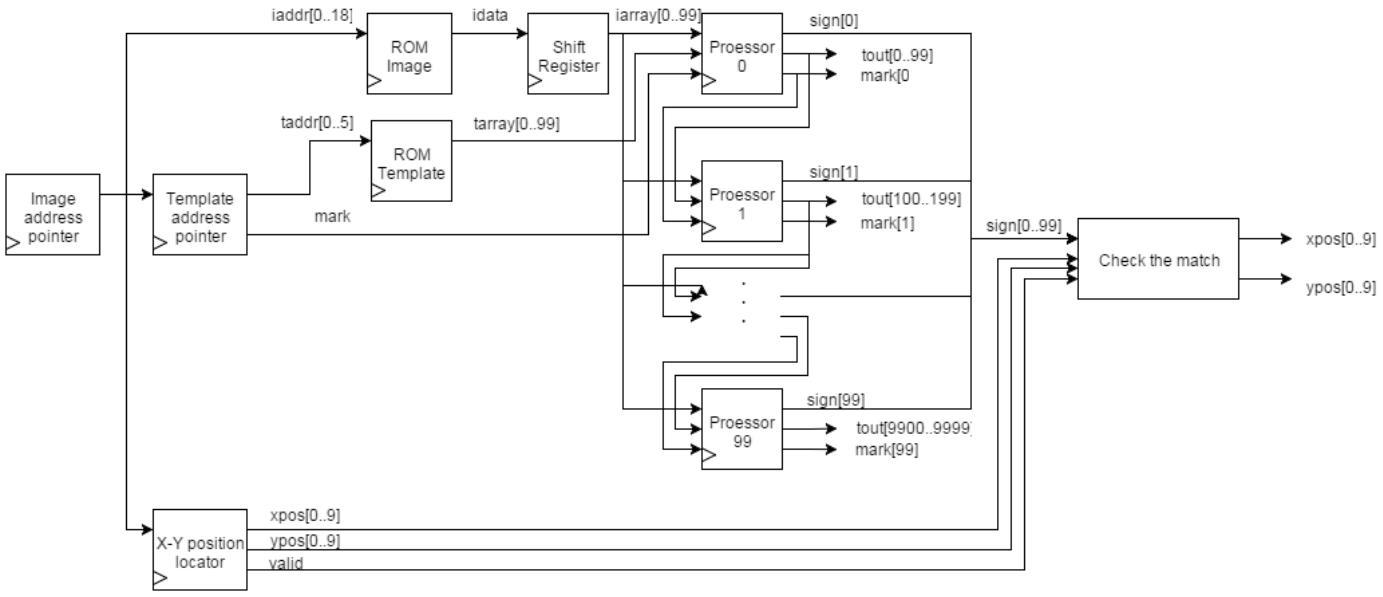


Figure 5.2 Parallel template matching processor architecture (use 40 processing element)

Based on those process, the algorithm in digital circuit design is made with constrain of speed and area. The focus of design is in the matching process, therefore the inputs, both of image and template image, are in binary image. To reduce area used in design, the maximum SAD value is limited to the threshold value (500). The greater value of SAD will remain the same as the threshold value and the sign will signify whether the SAD value is overflow.

This design calculate the SAD value until the end of image which means that the design can detect more than one object matched, but still the marked one is the last one that is matched.

The inputs which are image and template image will use ROM to restore the data pixel. The ROMs use the ALTSYNCRAM Megafunction to implement block of memory. The design is in dual port mode that can do both a read and write operation. The memories are implemented using the M4K memory blocks. The address is used for addressing input to ROM. The data from ROM image first enters the shift register to get the array of 100 datas. The arrays that will be process further are controlled by line buffer that will issue the array data of image, the array data of template image, valid, pixel position, and mark to mark the end of template.

The SAD calculation between those array datas given out by line buffer block is simplified by XOR operation. The SAD process calculates per column and shift the template array to next column each clock. The SAD value as described before are restricted to the threshold value and will give out the sign for overflow condition. If it is matched or SAD value is below the threshold, the position will change to the current position from line buffer.

The pixel in which the position is in the end of image will be a reference for match drawing. This unit consists of ROM which contains the data pixel of boundary match drawing in zero position. This posisition will be adjusted with acquired matched posisition so that boundary match drawing will follow the match posisition. To produce the drawing match boundary intensity, it also use counter for input address at this ROM.

For the output stage, the result picture will be restore in RAM. In the beginning, the RAM is already stored with original image data. For the process, the address and intensity data for boundary match drawing will be stored from drawing unit. The end condition for this detection is after a whole pixel in original image has been accessed.

All those process are done if the enable is low (active low) and the reset is high (active low) during positif edge clock or negative edge reset.

VI. LINE BUFFER

The line buffer unit issues the array of image and template per column and status line output. Status line output consists of valid, pixel position in image (x,y) and mark. Valid is used with the sign of overflow to detect whether the pixel matched or not. Valid is determined by x position and y position where for the matched one the x position will not below 40 and the y position will not below 100 . Mark is used to indicate whether the template address pointer reached the last column of template image.

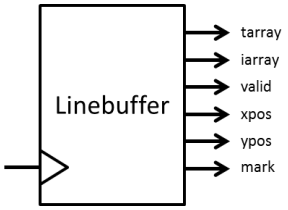


Figure 6.1 Line buffer

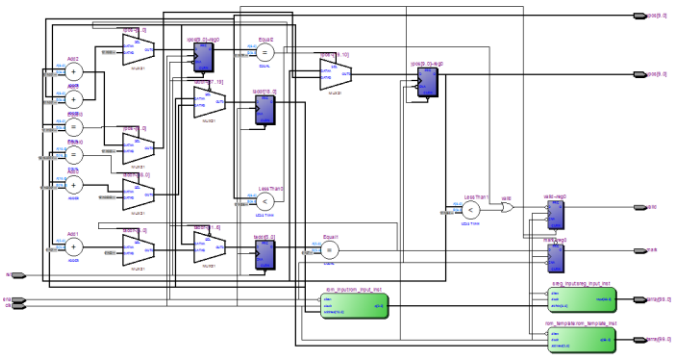


Figure 6.2 Line buffer architecture

The array data of template image is given out by ROM template unit, while the array data of image is given out by shift register that has input from the output of ROM input unit. To access what array that will be taken from the ROM is indicated by the address pointer. The address pointer of image is valued from 0 to number of all pixels in image minus one, which is 307199. The address pointer of the template image is stated from 0 to the number of column minus one, which is 39. The address pointer of template is stated like that because the SAD is calculated per every column. For every clock both of address pointer are increased one. If the address pointer have reached the maximum value stated above, the address back to first condition (0).

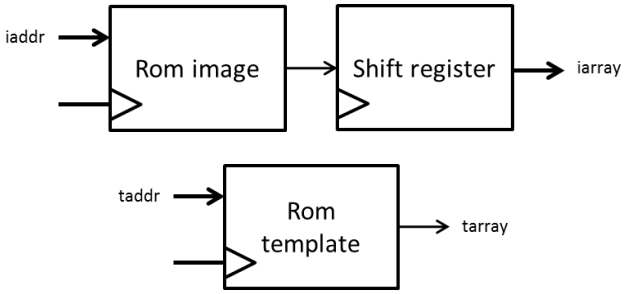


Figure 6.2 ROM

VII. INPUT STAGE

The input stage consists of ROM unit for template and original image. The ROM units have input address of the data that will be fetched, clock, and clock enable input. The enable in ROM is active high while in design is active low, so the enable of row is negation of the enable of design. The address is controlled in line buffer unit. ROMs are used internal memory that is created using MegaWizard Plug-In Manager.

The ALTSYNCRAM parameter is specified so that the operation mode is ROM and the read operation is enabled, while the write is disabled. The width data of image is determined as one, while the template is 100. The width of byteena is 1 and the byteena are defined 1 bit that described both data image are in binary. Number of word parameter is defined by number of pixel minus one for ROM image and number of column for ROM template.

Both ROMs are given input from .mif file. The file contains the data intensity per pixel that has been binerized. For image file, the data contains data stated per row then column. While

for template file, the data contains data stated per column then per row.

The output from ROM image enters the shift register unit to give out 100 data array of image based on the address desired. The shift register use RAM Block Type of M4K.

VIII. TEMPLATE MATCHING PROCESSING ARRAY

The processor array does calculate for the SAD value. This unit has inputs which are clock, reset and enable, and gives output such as x and y position of the matched pixel. For the matched pixels that are more than one, only the last one's postion that is issued by this unit. The data of image and template image are taken from line buffer unit that is called inside this unit.

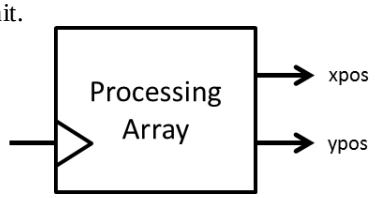


Figure 8.1 Processing array

The Datapath is shown in figure 5.2. The processing array consists of 40 processors that do SAD calculation per column. For the first time all the processor calculate the image with template column 1.

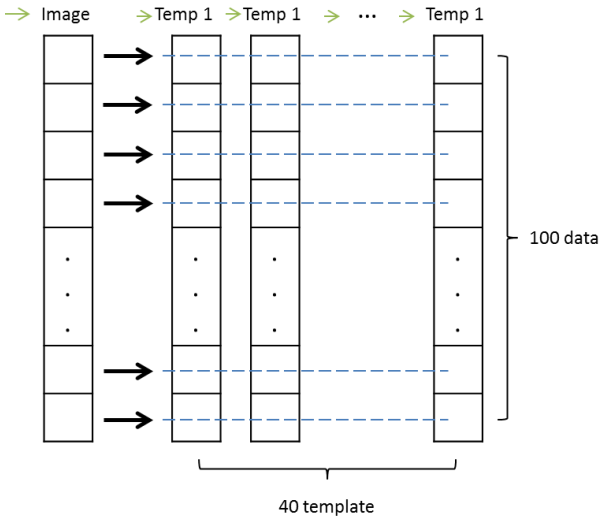


Figure 8.2 SAD calculation for the first time after enabled

After a clock the next column of template and image enter the process

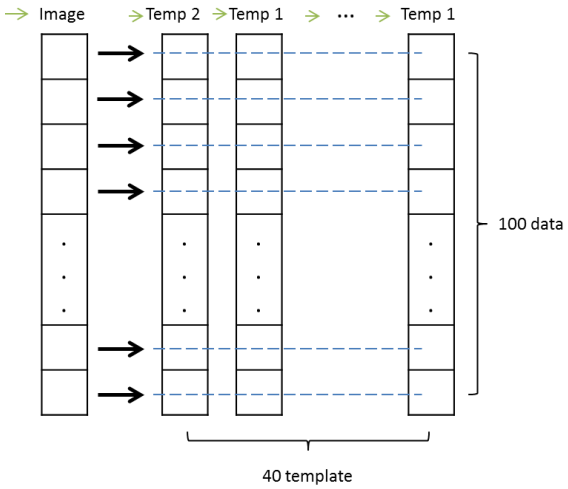


Figure 8.3 SAD calculation for the second time after enabled

After all the template enter, the process will become

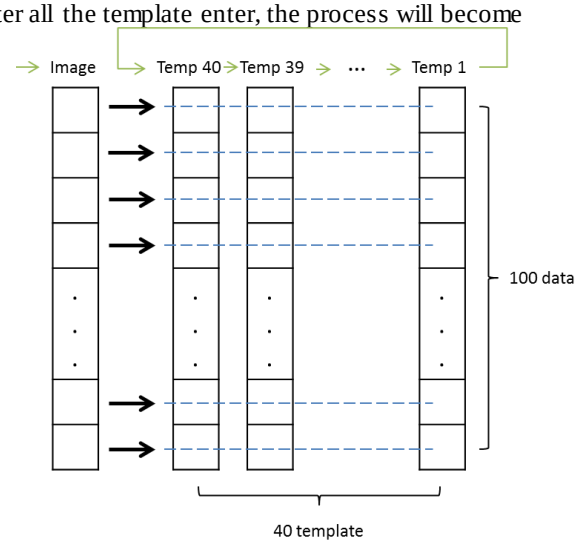


Figure 8.3 SAD calculation

For every clock the template will shift and the new image array and the new template array will enter. The SAD calculation performs between the template and image using processor unit. Processor unit give out tout that is the template shifted, markout that is the mark shifted, and the sign indicating whether the SAD is more than threshold. If the template given does not have address at 39 (the end of column), the mark has value low. On the other hand, if the template given has address at 39, the mark has value high. These mark value is controlled by line buffer unit. If the mark is high, it means that all SAD value of the 40 columns has been calculated.

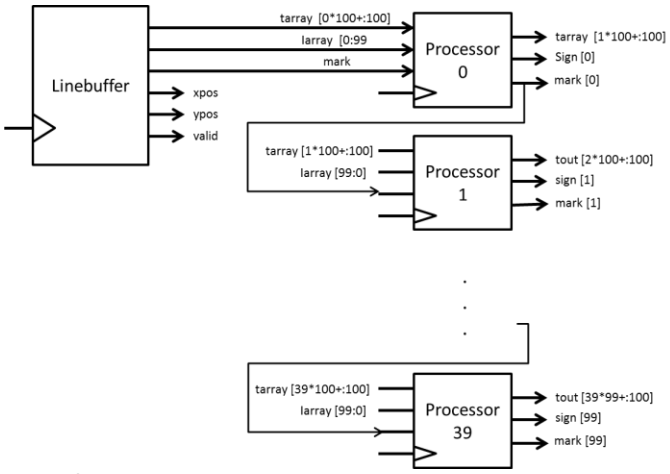


Figure 8.4

IX. PROCESSOR ELEMENT

The process calculates SAD using XOR operation, then every bit in XOR result is added using the tree adder. The overflow is detected by the carry of total SAD. If it is overflow, the total SAD will be threshold value. Otherwise, the total SAD is the addition of SAD calculated and the total SAD before. When the mark value is high, if the total SAD is not overflow, the sign will be high. Otherwise, the sign will be low. If mark value is high, the last result is omitted and result will be the result of tree adder.

The result of XOR operation is sum using tree adder. The adder works by iteration of adding the two neighbourhood data array until there are only one data. To understand more about the adder, let's see the picture below

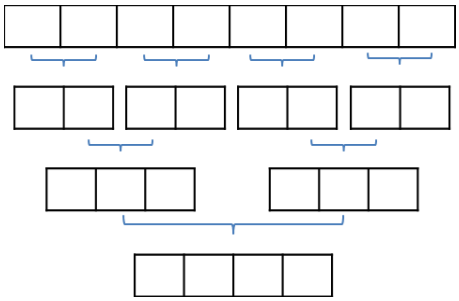


Figure 9.1 Tree adder process

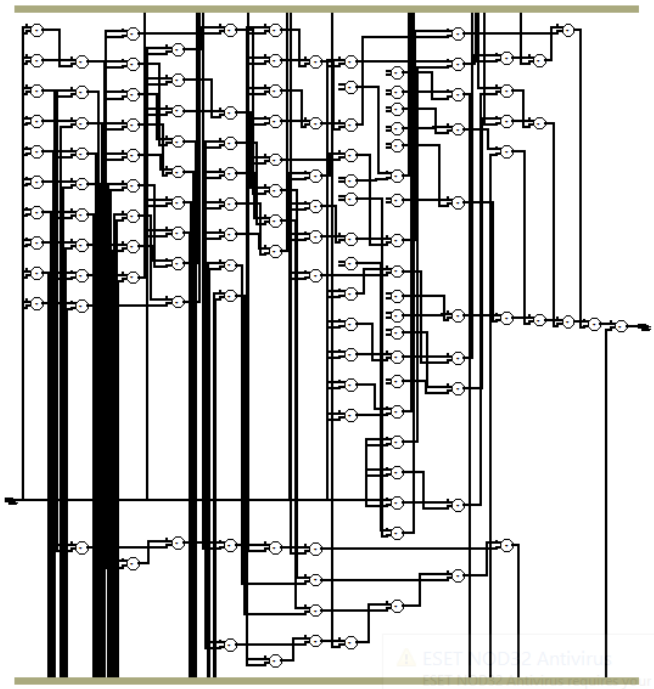


Figure 9.2 Tree adder process architecture

Figure 8.1 shows the process of tree adder where the size of array is simplified to understand easily. The two neighborhood array (size data is 1 bit) is added and give out the result of 2 bits data. Then the two neigbourhood result is added and become 3 bits data. The process goes on and goes on until the bits result are $\log_2 NDATA + 1$.

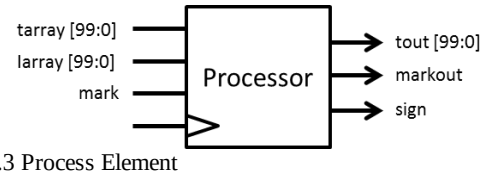
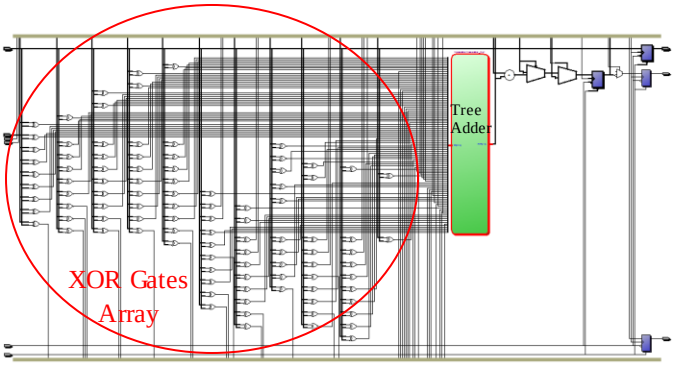


Figure 9.3 Process Element



X. MATCH DRAWING

The match drawing unit will draw a border in which the pixel is matched. The border's size is same as the template image. The input of this unit is address of the matched pixel. Based on

the address or position of the matched pixel, iterations are done to draw the boundary. First iteration is the row then the column which sizes are the row and column of template image. For the row which value is 0 (upper bound) or 99 (lower bound), the red channel is determined as 255, while the green and blue channel are 0. For the column which value is 0 (left bound) or 39 (right bound), the red channel is determined as 255, while others are 0. For other pixels in the image, the value is same as before.

The data intensity in red, green, and blue channel are given by memory.

XI. Output Process

There are 2 kinds of output stage such as match position and VGA output. For the simple match position, the x and y position then enter output processor to give out the result in seven segments. The datas input which is position first entered binary counter decimal module to count in decimal forms. Then the three decimal that represent the data input is showed using seven segment by convert the 4 bits that represent [0..9] in decimal to 7 bits in seven segments.

Notice that the seven segments used in FPGA is common cathode which means that it is active low.

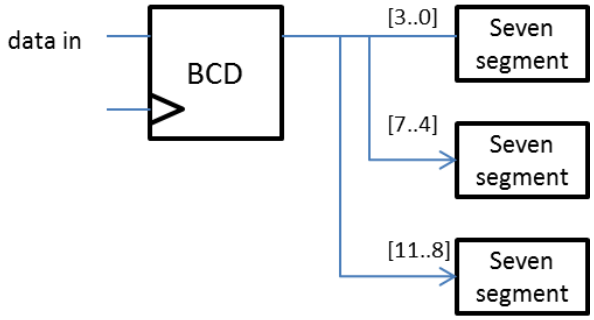


Figure 11.1 Output process in 7-s

XII. Result and Analysis

This design is implemented in verilog and simulated using Modelsim. The result of its simulation can be seen as below

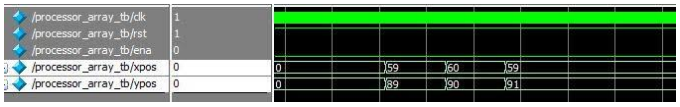


Figure 12.1 Result of simulation in Modelsim

To know how precise the result is, the result is compared by the one that simulated using MATLAB. The MATLAB result can be seen as following

```
>> Template_matching_en
SAD threshold is 511
Original image size is 640x480
Template image size is 40x100
Part1
Elapsed time is 2.207662 seconds.
Part2
Elapsed time is 138.147171 seconds.
Part3
x position = 60
y position = 93
Elapsed time is 0.063520 seconds.
>>
```

Figure 12.2 Result of simulation using MATLAB

Based on those simulations, the result of this design is similar with the MATLAB one. The x position is 59, while the x position in MATLAB is 60. The y position is 91, while the x position in MATLAB is 93. The result of implementation can be seen as following

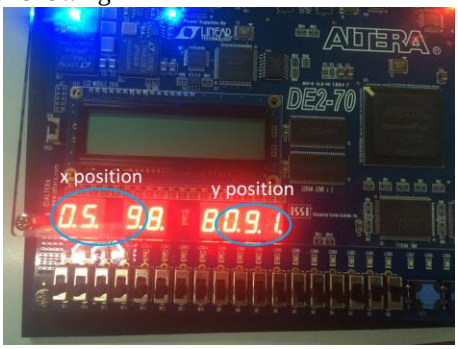


Figure 10.3 Implementation using FPGA

Based on the position given by the design, the image will be

Figure 12.4 (a) Result, (b) Template

The image inside the border is the image that matched with the template image. The difference between the design and result by MATLAB is not noticeable. To prove it again, we test

the design with two other images and the results are as following

Position	Image 1		Image 2	
	MATLAB	Design	MATLAB	Design
x	267	268	60	60
y	110	110	285	285

Tabel 12.1 Result of implementation and MATLAB

Figure 12.5 Image 1 (a) Result, (b) Template; Image 2 (a) Result, (b) Template

The position of the matched pixel in both image 1 and image 2 is similar between the implementation in Verilog and MATLAB simulation. It means that the design proposed is functioned.

The time needed to get the matched pixel position is 307200 clocks. The time needed for every pixel to calculate the SAD in that pixel is 40 clocks. While the time needed between results is 1 clock, except the one that is in different row. The performing time can be describe as

$$Performing\ time = 601 * 480 + 39 * 480 = 640 * 480 = \mathbf{307200\ clocks}$$

The clock frequency used in this design is 50 MHz, so that for every frame image, the clock needed is $\mathbf{6.144 \times 10^{-3}s}$. The frame speed can reach until 162 fps for video application.

The design is implemented using Cyclone II EP2C70F896C6N. Total logic elements used in this design are 16.689 elements including 12.732 elements from total combinational functions and 4.460 from dedicated logic registers. Total register used is 4460, while total pin used is 45.

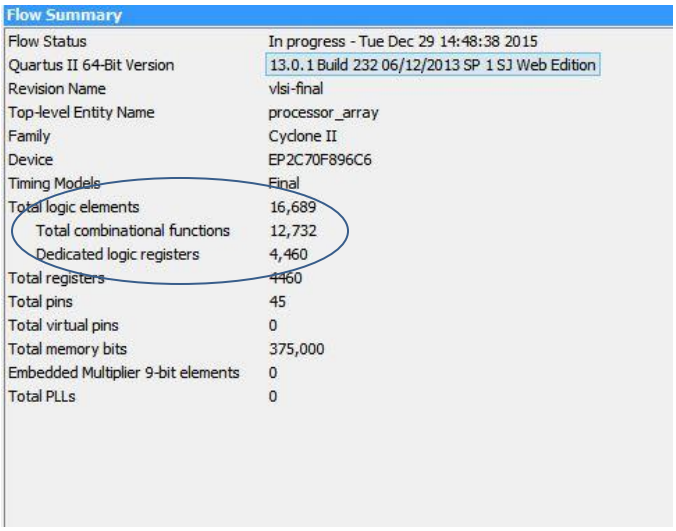


Figure 12.6. Syntesized logic element result

Figure 12.7. Output with VGA

XIII. CONCLUSION

- The SAD calculations are performed for every pixel in the image.
- The matched position output from the design is the last position that matched.
- Memory internal is used to save the data of image and template image.
- Latency of 1 pixel SAD is 40 clocks, while the thoughput between SADs is 1 clock.
- The processing time for image which size is 480 x 640 is 307200 clocks.
- The position of the matched pixel by this design is

slightly similar with the simulation in MATLAB. The difference is only about 1 or 2 pixel.

REFERENCE

[1] Patrick Elvert, Tobias Tiemerding, Class diederich Sergej Fatikow. 2014, Advanced Method for High Speed Template Matching, targeting FPGA. Presented in International Symposium on Ophotomechatronics Technologies, 2014.

[2] S. Hazel, A. kugel, R. Manner, and D.M Gavrila.. 2010, FPGA based Template Matching using Distance Transform. Available: ieeexplore.ieee.org

[3] Kaoru hasimoto, Yasuaki Ito, Koji Nakano. 2013, Template Matching Using DSP-slices on the FPGA. Presented in First International Symposium of Computing and Networking, 2013

LSI Design Contest 2016 Report

Luu Trung Tin

LEVEL 1:

In the basic challenge, along the following main flow, it will be carried out the design of the circuit for performing the template matching of image.

1. Input the search image and the template image.
2. Converting input image to the grayscale image.
3. Converting the grayscale image to the binary image.
4. Matching method.
5. Output the resulting image.

About input and output:

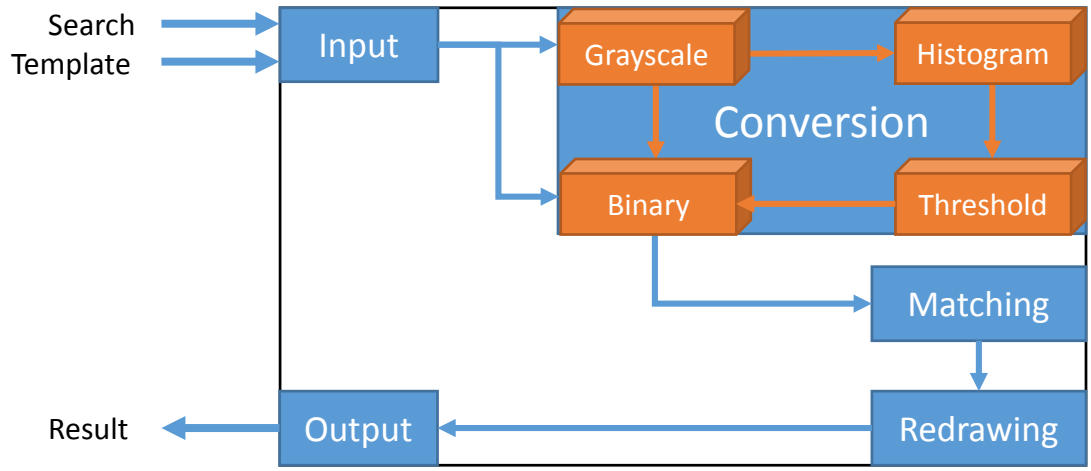
Input: Searching image (Size: 640x480)

Template image (Size: 40x100)

Output: Resulting image (Size: 640x480)

In both input and output, the image scan in order from upper left to lower right.

- **Circuit Block:**

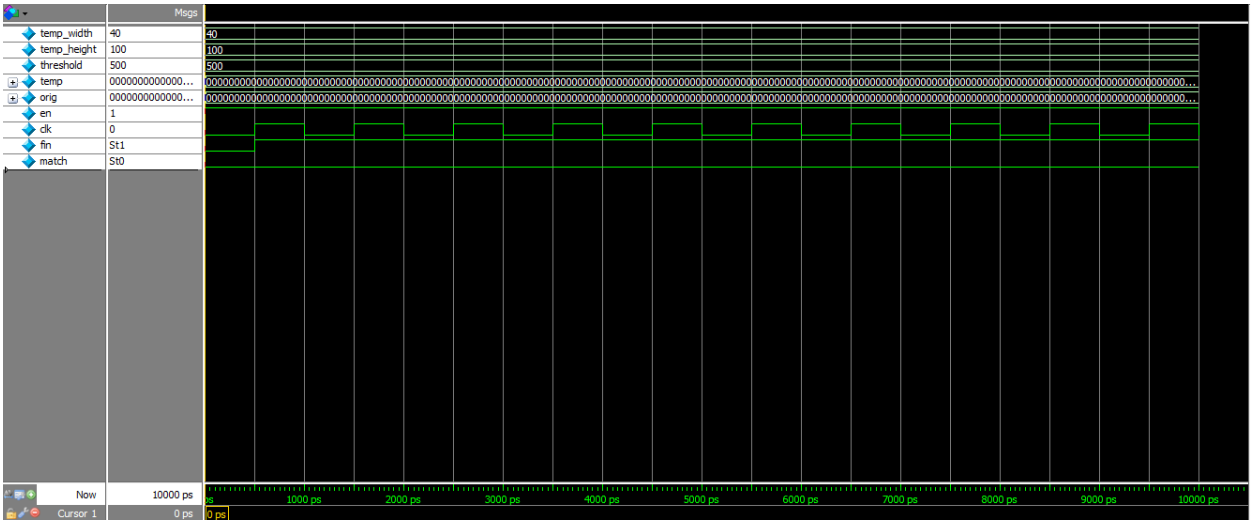


- **Functions:**
  - **Input and Output Block:** take care of the inputting and outputting processes of the whole system.
  - **Conversion Block:** converts the Search and Template images to their respective binary (Grayscale) version.
  - **Matching Block:** calculates the total difference between the Template image and each portion of the Searching image.
  - **Redrawing Block:** is used only when the total difference in the previous block is equal or lesser than 500 and to draw 4 red lines covering the portion calculated.
- **Appeal Points and Originality:**
  - Instead of creating a look-up table, our team directly compared the grayscale pixel value with the threshold to determine its Black-White status. This will remove the need for another block in the verilog code.
  - Instead of loading the whole image, every pixel got processed right after being loaded. This will decrease the time consumption due to simulatneous operations.
  - Along with some minor improvements, e.g. rounding the RGB coefficients.

- **Simulation Waveforms:**
  - Not Match:

Temp = 0xff458721325862154875421547563215478563215475632157896 2abfdce

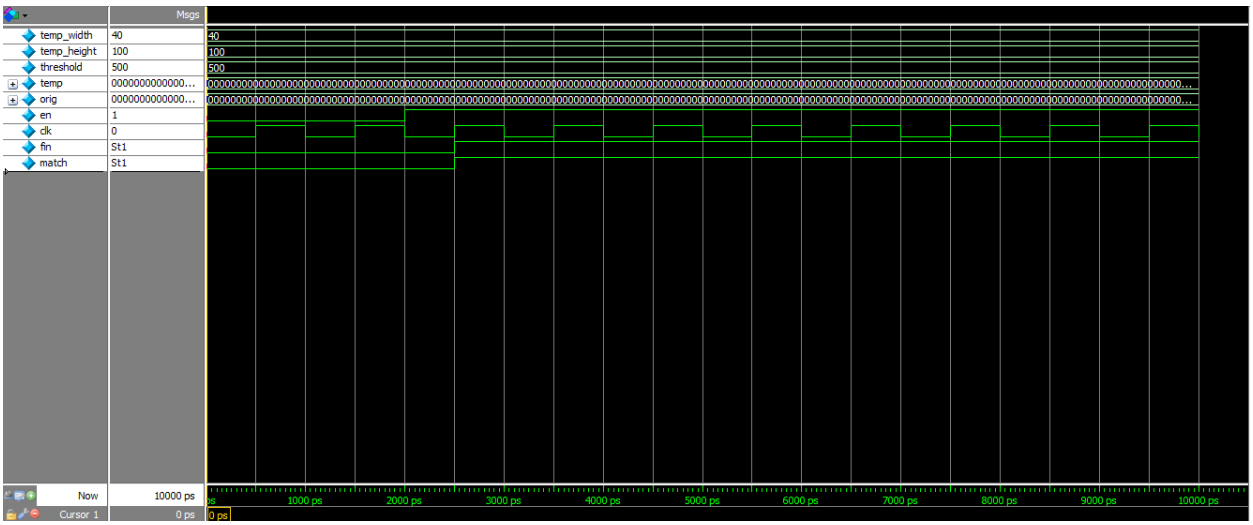
Orig = 0x21756321547856321547563215789 62abfdc1547215789632157458758962124587 54



  - Match:

Temp = 0xff458721325862154875421547563215478563215475632157896 2abfdce

Orig = 0xff458721325862154875421547563215478563215475632157896 2abfdc1



Level 2:

In free challenge, have them create a circuit for human detection using a variety of algorithms. About input and output.

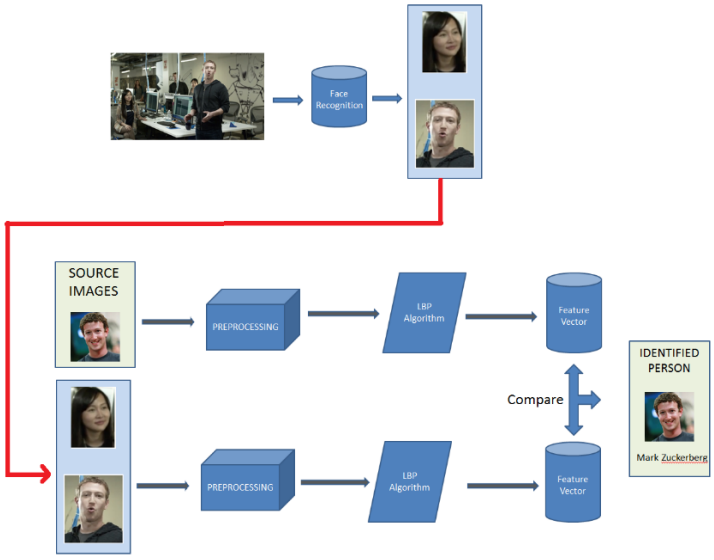
Input：Searching image (Size: 320x240)

Template image (Size: 320x240)

Output：Resulting image (Size: 320x240)

In both input and output, the order in which to scan the image does not limit. In addition, the searching target does not matter image or movie.

- **Block Diagram:**



- **Functions:**
  - **Face Recognition Process:** crops faces from the whole picture. Our method employes the rejection based classification.

    After receiving and organizing the RGB data in the Input Block, the Searching image data is converted to HSV data. According to the histogram of human skin, all pixels that fall outside H and S thresholds are rejected (marked black).

    The resulting image is non-skin rejected but there is quite a bit of noise and clutter. Rejection based on Geometry and Rejection based on Euler number are used to eliminate this problem. All remaining non-face

objects are vanished.

Final images (human faces) are cropped.

- **Face Identification Process:** identifies whether a person is in the picture by using Local Binary Pattern (LBP) algorithm.

Outputs from previous process (human faces) and image from database (template image) are converted to gray scale.

Resize images to 320x240.

Feature vectors are obtained with each image by using LBP algorithm. Then, feature vector of template image is compared with feature vector of input images (human faces) by calculating the difference. The smallest result is the one matches template image and the person is identified.

- **Simulation Waveform:**

また来年沖縄で、お会いしましょう。

©OCVB

コンテストに関してのお問合せ：
九州工業大学情報工学部電子情報工学科尾知研究室内・LSIデザインコンテスト実行委員会事務局

TEL：0948-29-7667
http://www.lsi-contest.com