内容梗概

本論文では，様々な分野で使用されている液晶用ガラスの生産過程における欠損を検出するシステムについて検討し，ハードウェアを用いた画像処理によって高速に検出する手法を提案する。液晶用ガラスの欠損を検出するシステムは，対象となる液晶用ガラスの表面を撮影した画像に対して，画像中に含まれるノイズの影響を軽減する Time Delay Integration，欠損のエッジ検出を行うラプラシアンフィルタ，輝度値を閾値によって分ける 2 値化，どのような形の欠損があるかを調べるために欠損部分に番号を振るラベリングの 4 つの画像処理アルゴリズムから構成される。

液晶用ガラス欠損検出システムの設計では，回路の書き換えが可能な FPGA（Field Programmable Gate Array）を利用して設計し，ラプラシアンフィルタ，2 値化，ラベリングを FPGA 上に実装した。欠損検出システムはまず始めに，Time Delay Integration によってノイズを軽減した液晶用ガラスの画像データを FPGA 上のメモリに書き込み，実装したラプラシアンフィルタ，2 値化，ラベリングを用いて画像処理を行って画像中に欠損があるかを判別していく。FPGA 上での処理を高速化するために，画像データが書き込まれているメモリへのアクセス回数を削減するシステムも構築し，画像処理に必要なデータを毎回メモリにアクセスして取り出す場合と比べて約 89%，シフトレジスタを使用してメモリにアクセスして取り出す場合と比べても約 67%のアクセス回数の削減を行った。

実験では，2 種類の構成を用いてラプラシアンフィルタ，2 値化，ラベリングのハードウェア化を行い，回路規模や処理時間を計測し，異なるハードウェア量で同じ画像処理を行う回路の比較を行った。実験の結果，小規模な構成のハードウェア量は大規模な構成のハードウェア量と比べて約 98%の削減となり，処理時間はソフトウェア処理と比べて，大規模な構成の回路では 26.5 倍，小規模な構成の回路では 22.23 倍の速度向上が得られた。

目次

図目次

表目次

1. はじめに

液晶用ガラスはテレビやパソコンディスプレイ，カーナビなどディジタル情報機器を中心に様々な分野で使用されており，近年ではスマートフォンの普及によりますますその需要が高まっている。これらの需要に見合う生産スピードの確保と，更なる高速化を実現するために，液晶用ガラスの製造過程で欠損が生じてないかを調べる検査に要する時間も短縮される事が望まれる。

本論文では，液晶用ガラスの欠損の有無を正確に調べるために，画像処理を用いて欠損を検出していく。しかし，画像処理によって欠損を調べるには多くの画像データが必要になり，画像データが多ければ多いほど処理に時間を要することになる。画像処理は CPU 上で動作するソフトウェアによって実行されている場合が多く，近年の CPU の処理能力は目を見張るほどの急成長を遂げている。とはいえ，処理の内容によっては十分とはいえないものもある。瞬時に結果が必要になる場合では，ソフトウェア処理では十分な性能や速度が得られないことも少なくない。また，ソフトウェア処理を行う PC のサイズがこのような検査装置の小型化の妨げになる。

そこで，画像処理をリアルタイムで行い，かつ高精度で高速なシステムを実現するために，FPGA を用いて液晶用ガラスの欠損を高速に検出するシステムの構築を目標とする。

FPGA（Field Programmable Gate Array）とは，回路の書き換えが可能な集積回路であり，近年のテクノロジーの進化により，高集積化，高性能化，低消費電力化，低コスト化が進んでいるロジックデバイスで，CPU によるソフトウェア処理と比べて高速な処理が可能である。また，信号処理部分に PC のような大きさを必要とせず，検査装置の小型化も図ることができる。このような FPGA の特徴を活かして，液晶用ガラスの欠損を検出する検査装置のシステムを構築し，欠損検出画像処理の高速化を図る。

本論文では，2 章で液晶用ガラスの欠損検出方法と，欠損検出に使用する画像処理アルゴリズムについての説明を行い，画像処理の効果と処理を行った画像の変化について述べる。3 章では，FPGA 上での欠損検出システムの構成と，データを格納するメモリや，メモリアクセス回数削減のために設計を行った 2 種類のレジスタファイルの構成について述べる。4 章では，2 章で説明した画像処理アルゴリズムを FPGA 上で実現するための手法と，欠損検出システム全体の制御について述べ，制御信号による動作モジュールの管理と処理内容について述べる。5 章では，設計した 2 種類の欠損検出システム全体の回路規模や各モジュールの回路規模，動作周波数や処理時間を計測し，ソフトウェア処理と比較して考察を述べ，6 章では本論文の成果と今後の課題について述べる。

2. 液晶用ガラスの欠損検出方法

2.1 全体の流れ

液晶用ガラスの欠損（傷）は製造過程で必ず生じるものではないが，欠損が生じないように気をつけていても何枚かに欠損は生じる。そのため欠損の検査は必ず行われ，傷がついているものを市場に出さないよう検査は正確に行われなければならない。検査をより正確に行うために，本論文では画像処理を用いて液晶用ガラスの欠損検出を行う。

画像処理に用いる画像は，あらかじめ液晶用ガラスの表面をカメラで撮影したものを使用し，画像処理には 4 つの画像処理アルゴリズムを使用した。使用した画像処理アルゴリズムは Time Delay Integration（以下 TDI），ラプラシアンフィルタ，2 値化，ラベリングの 4 つで，これらを組み合わせて欠損検出画像処理システムにする。本研究で設計した液晶用ガラスの欠損検出画像処理システムの流れを図 1 に示す。

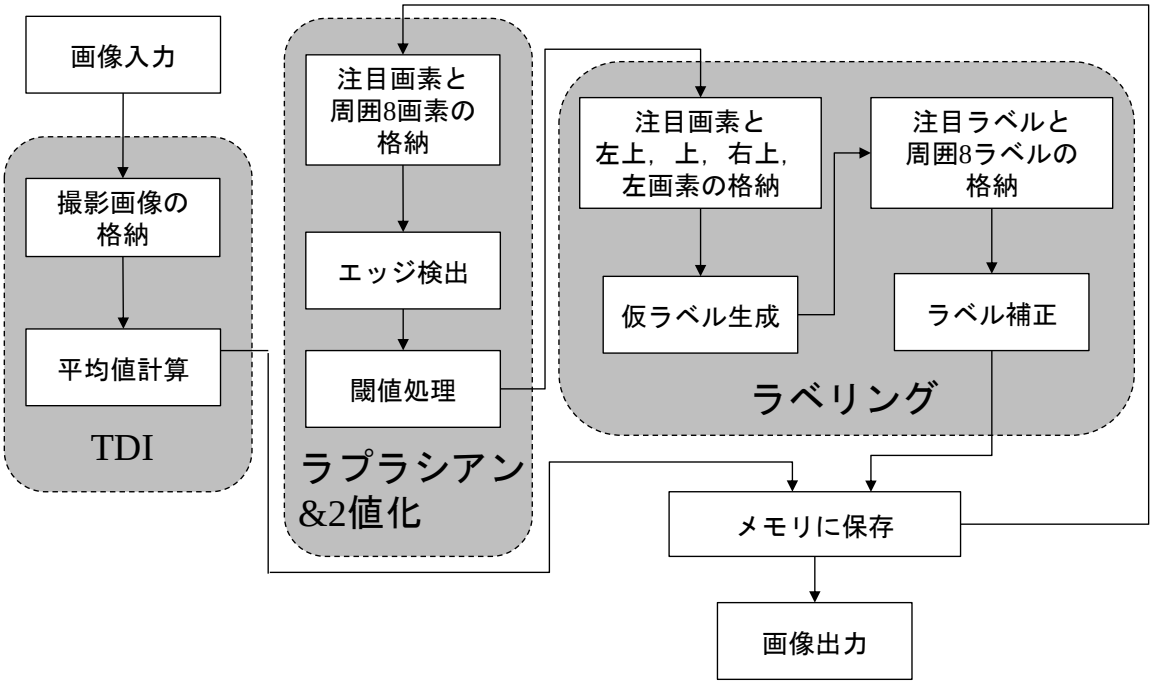


図 1. 液晶用ガラス欠損検出画像処理の流れ

まず，入力された撮影画像に TDI を用いて雑音除去を行い，後に行う画像処理で正しい結果が得られるよう画像に前処理を行う。次に，ラプラシアンフィルタを用いて画像に二次微分処理を行い，画像中に含まれる欠損のエッジ検出を行う。そして，エッジ検出された画像の輝度値を設定した閾値で分ける 2 値化を行い，最後にラベリングを用いて欠損部分に番号を振り，画像を出力する。

本論文で使用した液晶用ガラスの欠損画像の 1 つを図 2 に示す。画像の画面中央左にある点が欠損であり，周りの丸は欠損が分かりやすいように付けてある目印である。

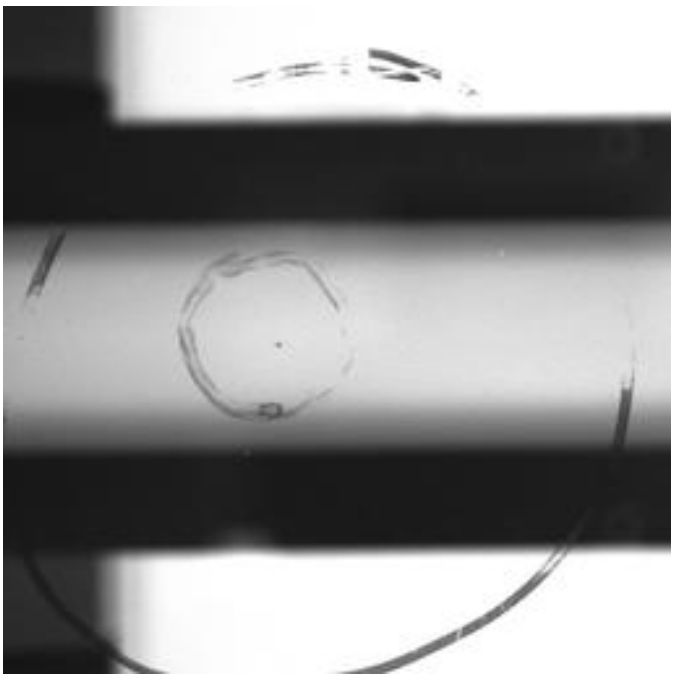


図 2. 液晶用ガラスの欠損画像

2.2 TDI（Time Delay Integration）

TDI とは，対象物を縦方向に 1 ラインずつずらして撮影を複数回行い，撮影した画像の共通部分を重ね合わせて平均値を取ることで，ノイズの影響を減少させる手法である。液晶用ガラスの表面を撮影した画像の中には画像そのものが歪んでいたり，ぼけていたりするものや，塵や埃が原因でノイズがひどいものもあるので，雑音除去を施さないまま画像処理を行うと，正しい欠損を見つけられなくなるからである。1 ラインずつずらして撮影する様子を図 3 に示す。図 3 の 1 枚目から 3 枚目を使用した場合の TDI 処理の様子を図 4 に示す。

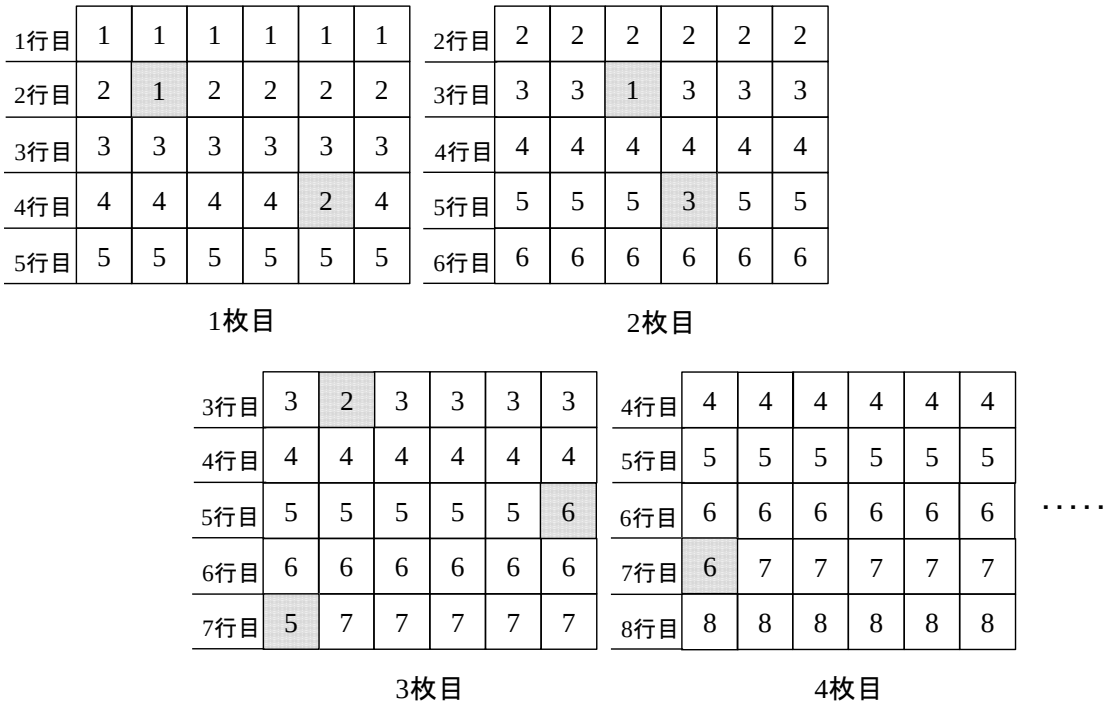


図 3. 画像撮影の仕方

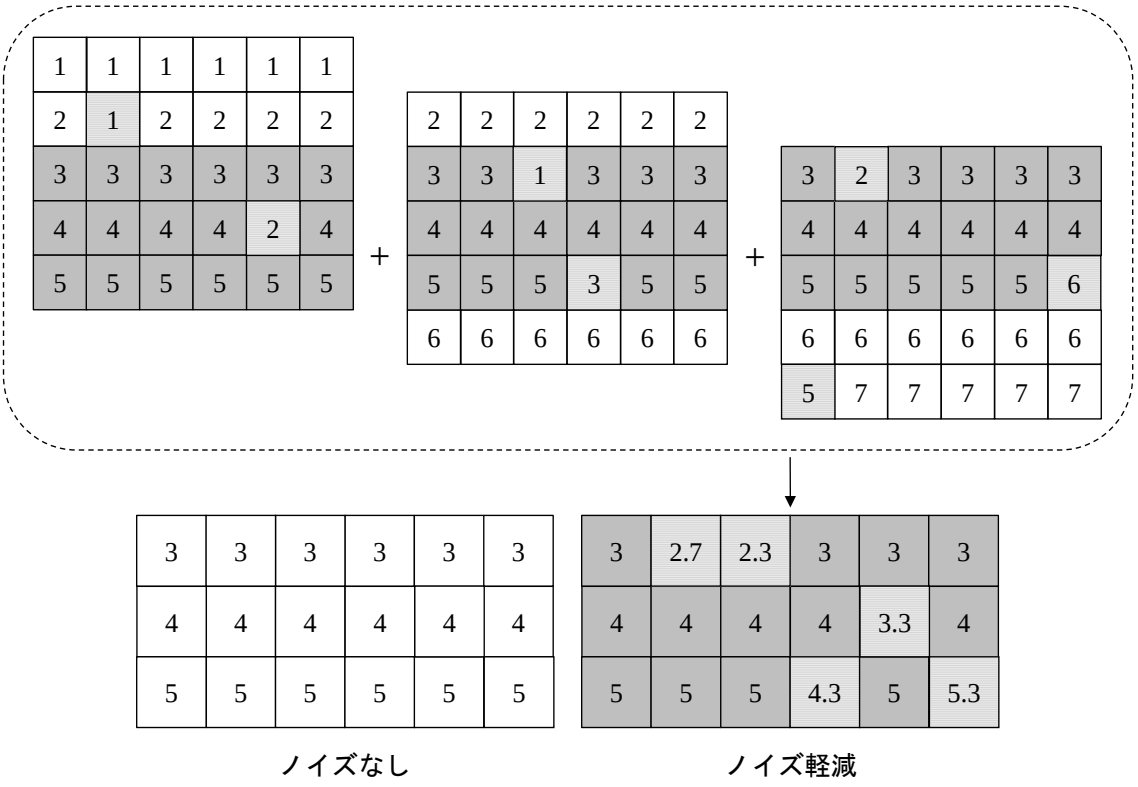
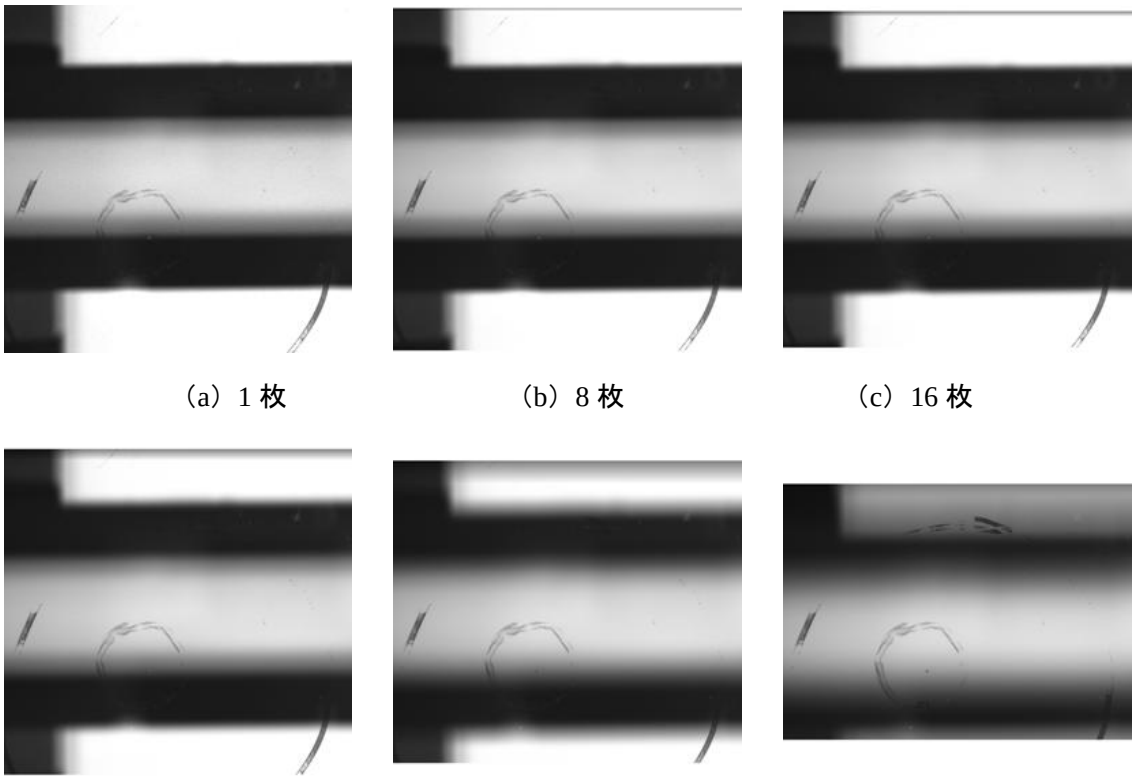


図 4. TDI を用いた雑音除去

TDI 処理は画像の共通部分を重ね合わせるので, 処理に使用する画像の枚数が多ければ多いほど画像が縦方向に小さくなっていく。図 4 では画像を 3 枚使用しての結果となるので雑音除去の影響は小さいが，使用する画像の枚数を 10 枚，20 枚と増やすほど平均値は真の値に近づいていき，よりノイズの影響が小さい画像が生成される。5 章で行う実験では，液晶用ガラスの欠損画像を 128 枚使用して TDI 処理を行っている。

TDI 処理に使用する画像の枚数による画像の変化を図 5 に，TDI 処理とノイズの数の関係を図 6 に示す。ノイズの計測には CPU 上のソフトウェア処理を用い，TDI 処理を施した画像にラプラシアンフィルタと 2 値化，ラベリング処理を行い，最後まで残ったラベル数を計測したものである。画像サイズは 5 章で行う実験に使用したものと同じで，サイズが 256*256，1 画素 8bit のものを使用して計測を行った。



（a）1 枚　（b）8 枚　（c）16 枚

（d）32 枚　（e）64 枚　（f）128 枚

図 5. TDI 処理による画像の変化

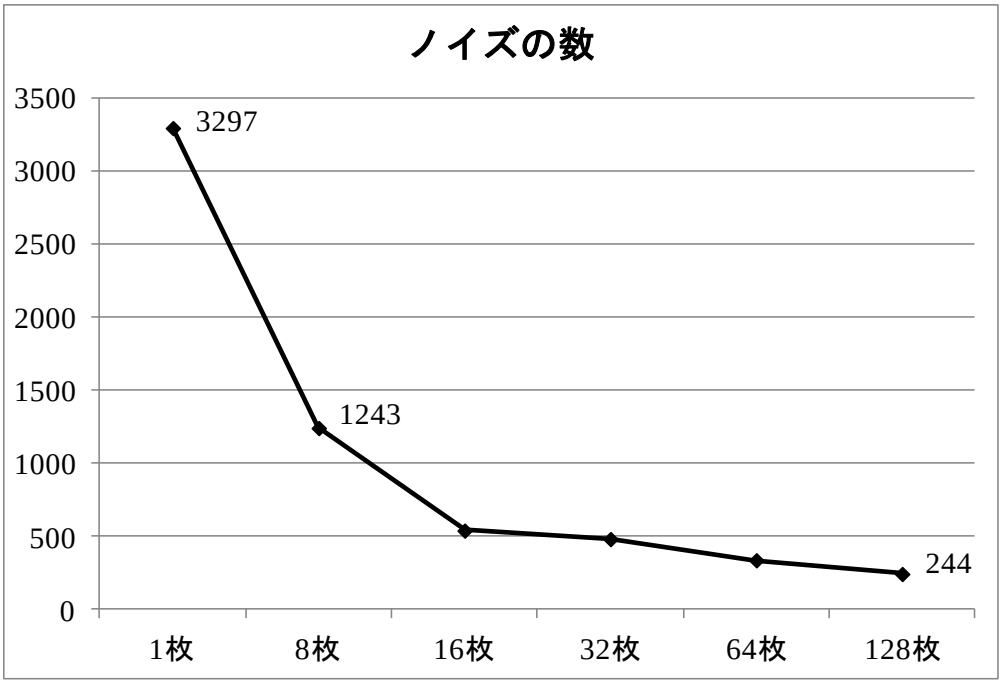


図 6. TDI に使用する画像の枚数とノイズの変化

2.3 ラプラシアンフィルタ，2 値化

ラプラシアンフィルタとは，特定の微分オペレータを使用して画素値を二次微分処理し，画像中に含まれる対象物のエッジ検出を行う手法である。ラプラシアンは一次微分であるグラディアントをさらに微分したもので，直接輪郭線を求めることができる。一次微分と二次微分との違いを図 7 に示す。

元画像の変化がある部分をエッジとすると，一次微分ではエッジの領域に山ができているのが分かる。この部分を 2 値化することにより，細線化するのが一次微分により輪郭線を求める方法である。一方，二次微分ではちょうどエッジの中心，輪郭線に相当する部分を挟んで正負の山ができている。この正から負に変化する部分を求めることにより，直接輪郭線を求めることができる。微分計算は隣接画素の差を計算するので，雑音の影響を大きく受け，二次微分であるラプラシアンは一次微分のグラディエントよりも雑音の影響を大きく受けるので，2.2 で説明した TDI を用いて十分にノイズを軽減させる必要がある。

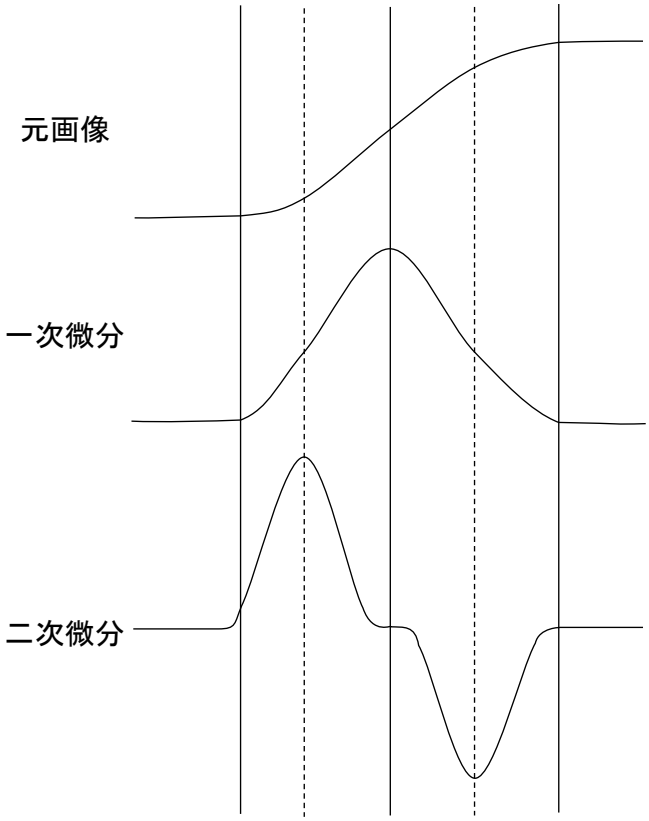


図 7. 一次微分と二次微分の違い

本論文では，上下左右と斜め方向を含めた大きさが 3×3 の 8 近傍のマスクパターンを使用しており，画像の注目画素とマスクパターンの注目画素を重ね，各係数をそれぞれ乗算し，その結果を合計した物が新しい画素値となる。注目画素の座標（x, y）の画素値を f（x, y），ラプラシアンフィルタにより求める新しい画素値を L（x, y）とすると，求める画素値は式 1 のように表される。

$$\begin{aligned} L(x,y) = {} & f(x-1,y-1) + f(x,y-1) + f(x+1,y-1) \\ & + f(x-1,y) + \{f(x,y)*(-8)\} + f(x+1,y) \\ & + f(x-1,y+1) + f(x,y+1) + f(x+1,y+1) \end{aligned} \quad \text{・・・式 1}$$

図 8 に本論文で使用したマスクパターンを示し，そのマスクパターンを用いたラプラシアンフィルタ処理の様子を図 9 に示す。

1	1	1
1	-8	1
1	1	1

図 8. マスクパターン

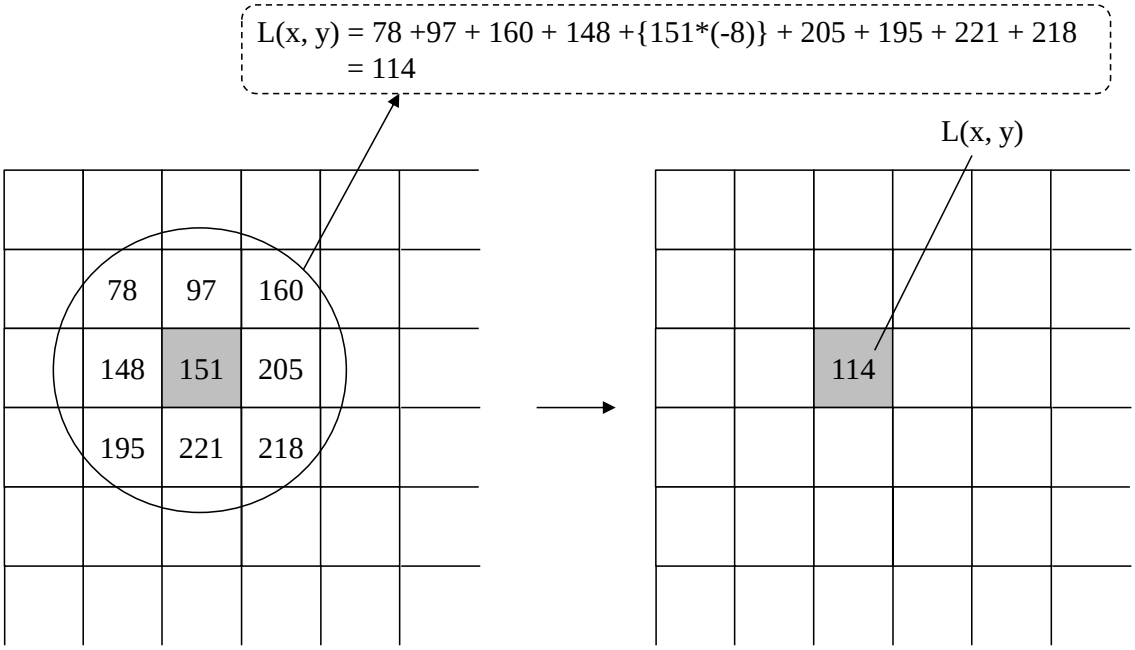
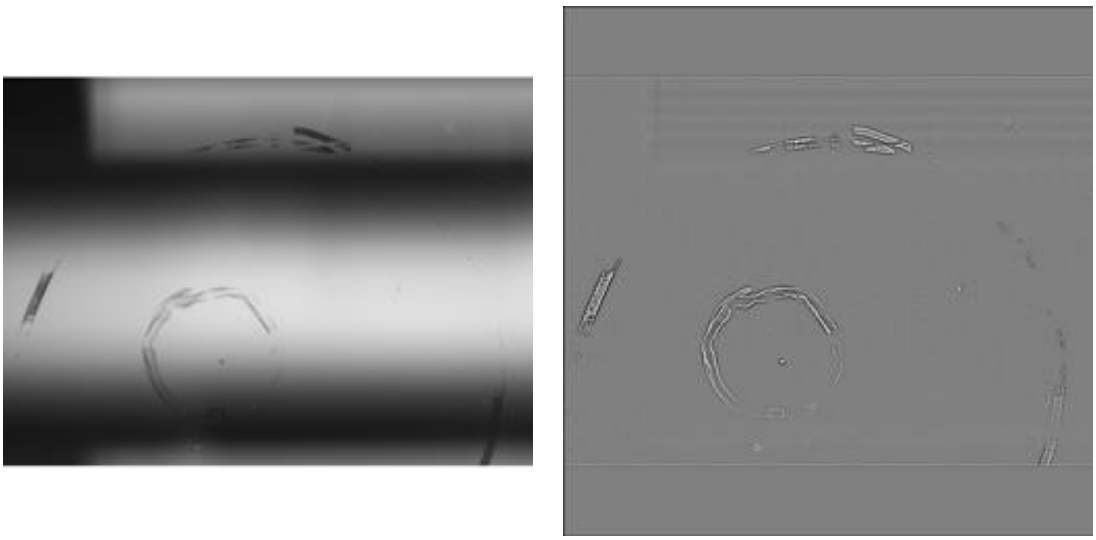


図 9. ラプラシアンフィルタを用いたエッジ検出

ラプラシアンフィルタ処理前の注目画素は“151”なので，“151”を中心とした 9 画素をマスクパターンの各係数とかけあわせ，計算結果の“114”が新しい画素値となる。

ラプラシアンフィルタによってエッジ検出された画像は，輪郭の強さに応じた濃淡画像となり，欠損部分が浮いているような画像に変化する。ラプラシアンフィルタによる画像の変化を図 10 に示す。



（a）TDI 処理後　　（b）ラプラシアンフィルタ処理後

図 10. ラプラシアンフィルタによる画像の変化

2 値化とは，濃淡のある画像をあらかじめ設定した閾値を用いて白と黒に分ける手法である。液晶用ガラスの撮影画像は 1 画素 8bit となっており，画素値は 0 から 255 の 256 階調に分かれている。ラプラシアンフィルタによって出力された新しい画素値と閾値を比べ，画素値が閾値よりも上なら“255”を，下なら“0”を出力する。ある入力画像に対して，閾値を“120”と設定した場合の 2 値化処理の様子を図 11 に示す。

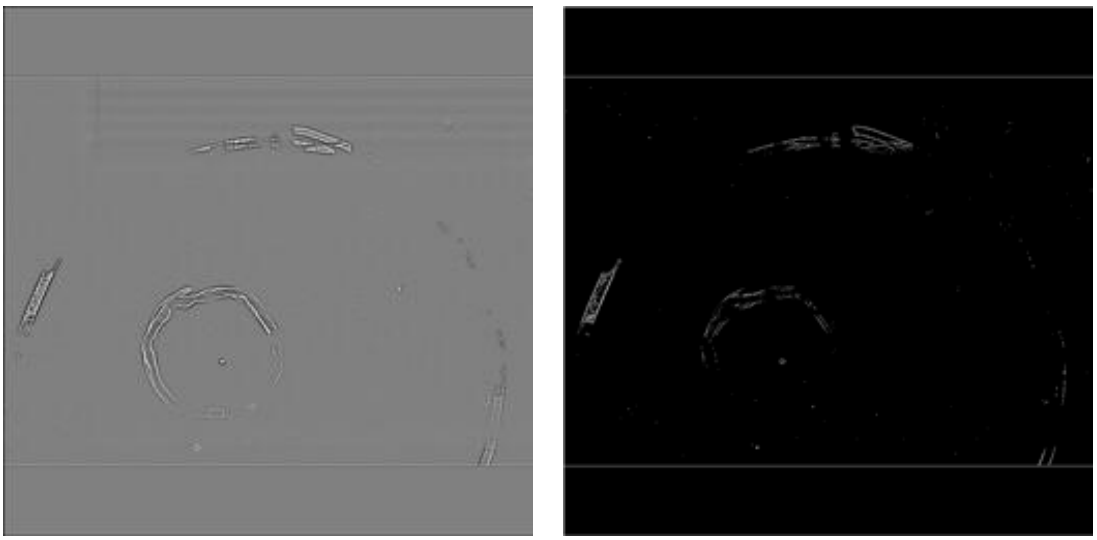
176	129	151	142	96	65	53
160	87	216	17	104	225	81
251	21	114	232	26	236	129
223	85	177	142	92	8	219
25	224	115	110	155	237	170
101	112	166	237	242	78	152
162	107	244	53	248	66	219

→

255	255	255	255	0	0	0
255	0	255	0	0	255	0
255	0	0	255	0	255	255
255	0	255	255	0	0	255
0	255	0	0	255	255	255
0	0	255	255	255	0	255
255	0	255	0	255	0	255

図 11. 2 値化を用いた閾値処理

2 値化によって閾値処理された画像は，欠損部分は白く，欠損以外の部分は黒くなった画像に変化する。2 値化による画像の変化を図 12 に示す。



（a）フィルタ処理後　　（b）2 値化処理後

図 12. 2 値化による画像の変化

2.4 ラベリング

ラベリングとは，画像中の連結成分に同じ番号（以下ラベル）をつけ，異なった連結成分には異なったラベルをつける手法である。連結成分にラベルをつけることで，画像中の様々な画素を個々の連結成分に分離することができ，同じラベルごとの面積や幅などの特徴量を求める事で，どのような欠損があるかを求める事ができる。

本論文では，ラベリングを用いて画素にラベルをつける走査を 2 回行っており，1 回目の走査では 2 値化処理後の画像に対して仮ラベルを生成し，2 回目の走査では生成した仮ラベルを元に誤ってつけられたラベルの補正を行う。

仮ラベルを生成する 1 回目の走査では，図 13 のように仮ラベルをつける注目画素の左上，上，右上，左のラベルを参照し，ラベルがあればそれをコピーして同じラベルをつけ，周囲 4 画素にラベルがない場合は新しいラベルをつける。1 回目の走査による仮ラベル生成処理の様子を図 14 に示す。

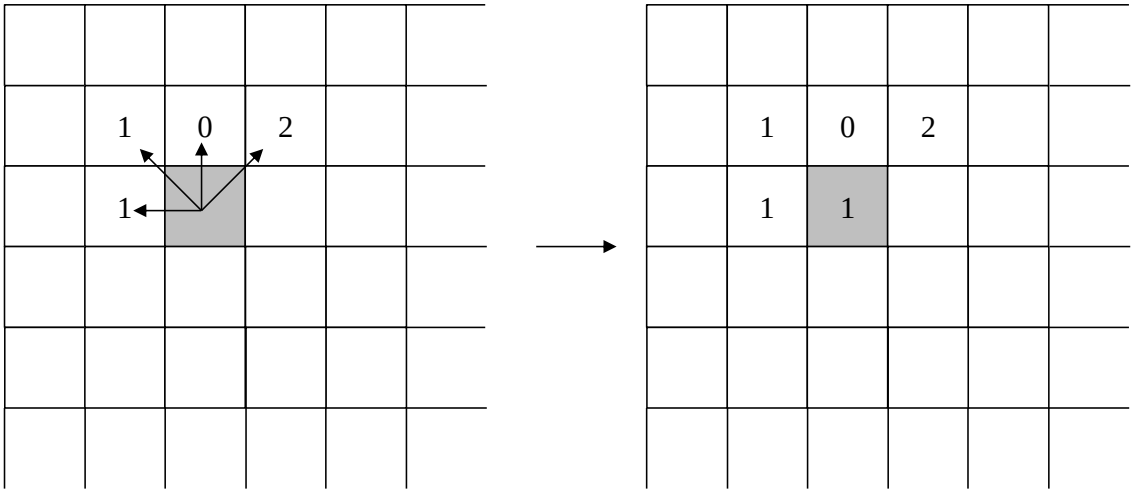


図 13. 仮ラベル生成時の参照データ

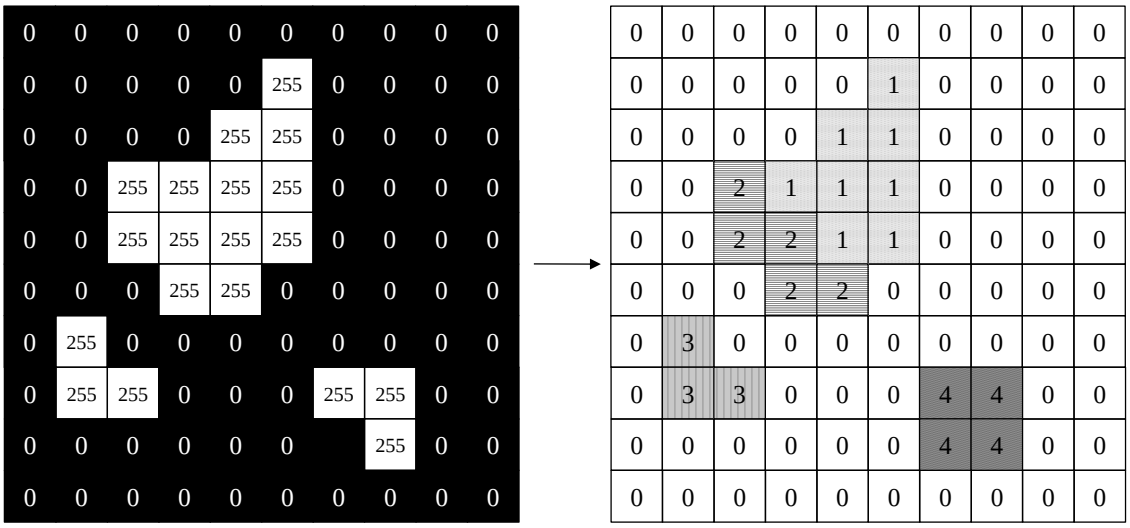


図 14. ラベリングを用いた仮ラベル生成

ラベルの補正を行う 2 回目の走査では，図 15 のように注目画素の左上，上，右上，左，右，左下，下，右下の全方位のラベルを参照し，注目画素が持つラベルよりも小さいラベルがあればそれをコピーしてラベルをつけ直し，周囲 8 画素に注目画素よりも小さいラベルがない場合は補正を行わず，注目画素が持っているラベルをそのまま出力する。2 回目の走査によるラベル補正処理の様子を図 16 に示し，1 回目と 2 回目の走査を合わせたラベリングによる画像の変化を図 17 に示す。

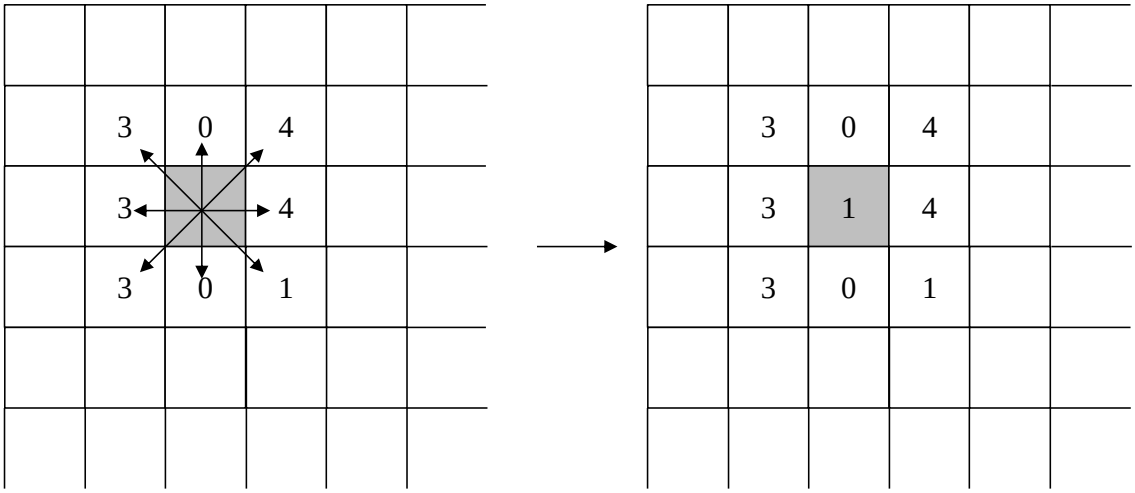


図 15. ラベル補正時の参照データ

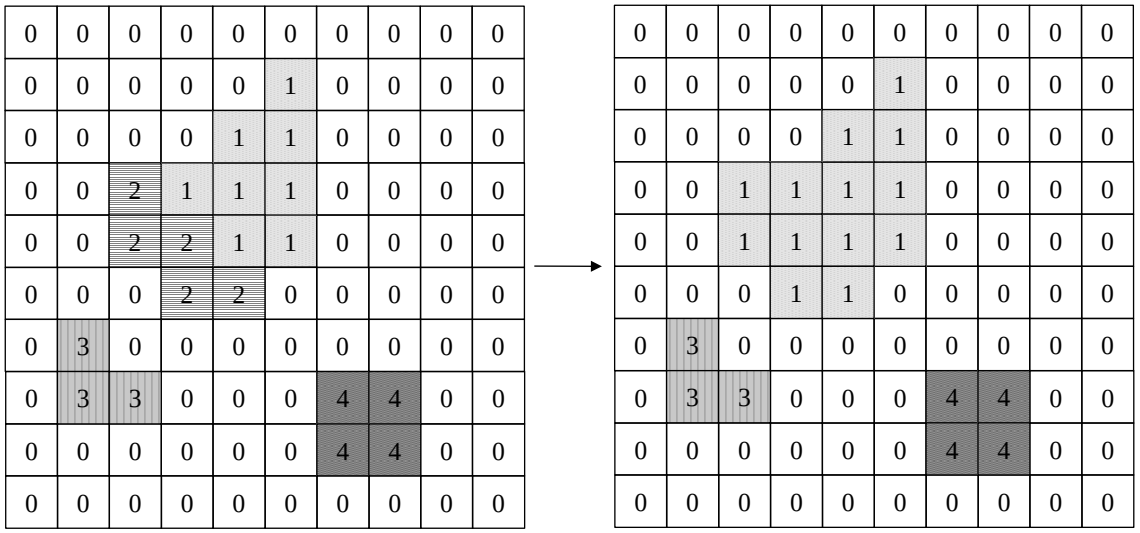
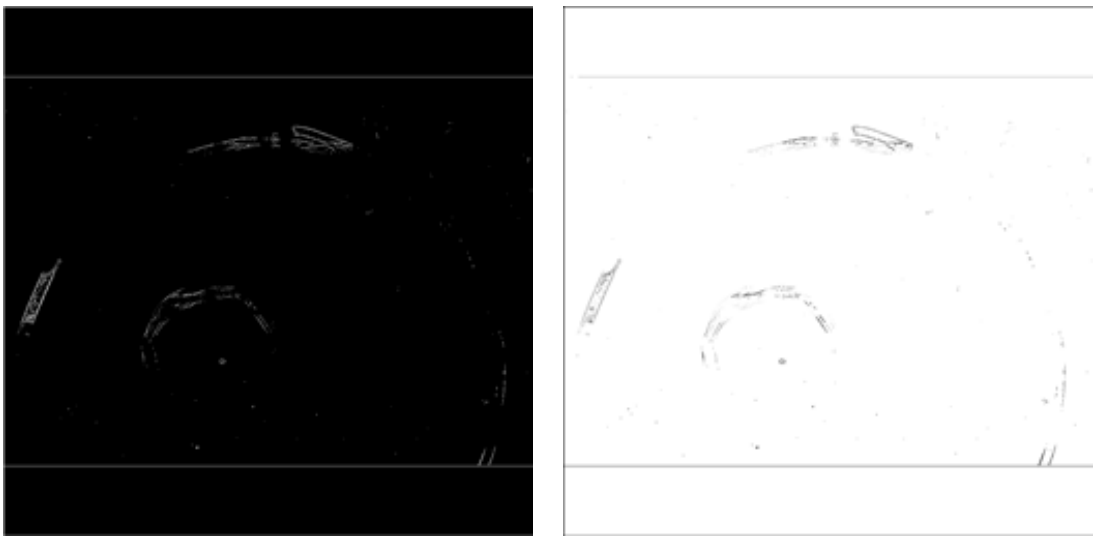


図 16. ラベリングを用いたラベル補正



（a）2 値化処理後　　（b）ラベリング処理後

図 17. ラベリングによる画像の変化

3. 画像バッファメモリの実現方法

3.1 システムの構成

本論文では，液晶用ガラスの撮影画像を FPGA ボード内の Block RAM を用いて格納し，画像処理を行う。しかし，TDI 処理では撮影画像を 128 枚（256*256*8bit/1 枚）使用するため, Block RAM では容量が不足する。従って, TDI 処理を CPU 上のソフトウェアで処理し，ラプラシアンフィルタ，2 値化，ラベリングをハードウェア化して FPGA 上に実装して処理することにした。

本論文では，FPGA 内に 256 行分のデータを保持するレジスタファイルを持った構成（以下構成 1）と，3 行分のデータを保持するレジスタファイルを持った構成（以下構成 2）の 2 つの構成を用いて画像処理のハードウェア化を行った。構成 1 を用いた欠損検出システムの構成を図 18 に示し，構成 2 を用いた欠損検出システムの構成を図 19 に示す。

（1）構成 1

Generate_ADDR は処理全体の制御とアドレスの生成などを行い，Generate_LAP はラプラシアンフィルタ&2 値化処理を行う。Generate_LAB はラベリングの 1 回目の走査となる仮ラベルの生成を行い, Generate_TLAB は 2 回目の走査となるラベルの補正を行うモジュールとなっている。DATA_Register は，Block RAM に格納されているデータや各処理後のデータを全て保持するレジスタファイルとなっており，これらを使用して入力されたデータに対して画像処理を行う。処理の手順を以下に示す。

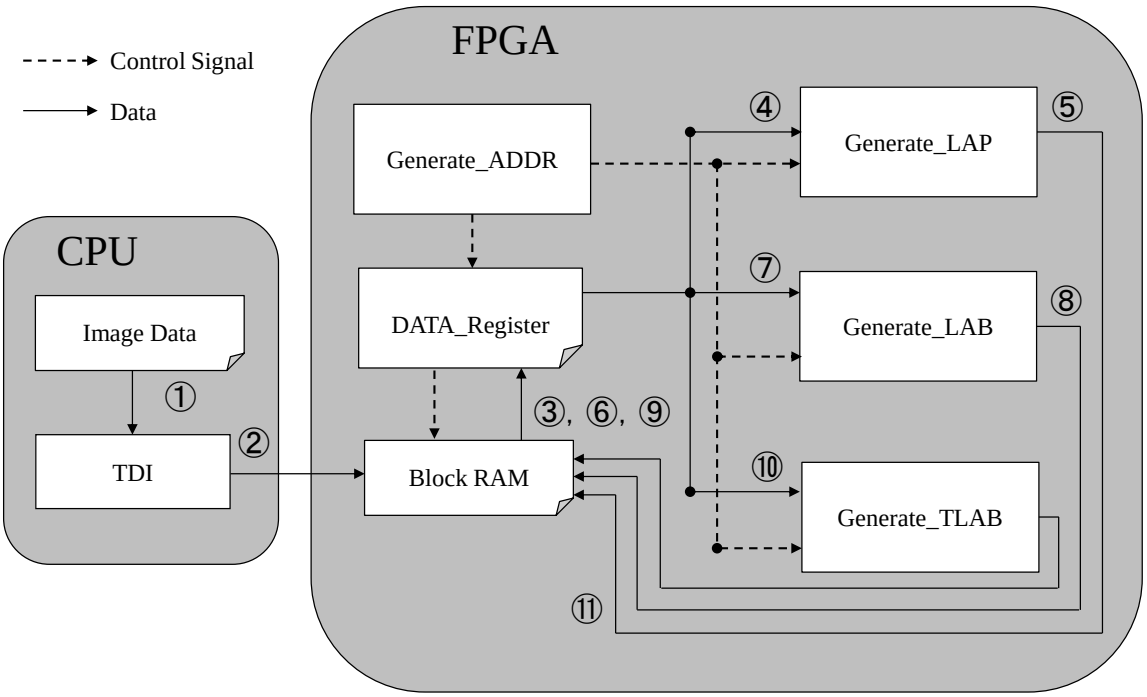


図 18. 欠損検出システムの構成 1

①画像ファイルからデータを取り出し，TDI を用いて雑音除去を行う。
②雑音除去後のデータを FPGA 上の Block RAM に書き込む。
③Block RAM から 256 行分のデータを取り出し，DATA_Register に書き込む。
④⑤DATA_Register からデータを取り出して全データに対してラプラシアンフィルタ&2 値化処理を行い，Block RAM に書き込む。
⑥Block RAM から 256 行分のデータを取り出し，DATA_Register に書き込む。
⑦⑧DATA_Register からデータを取り出して全データに対して仮ラベルの生成を行い，Block RAM に書き込む。
⑨Block RAM から 256 行分のデータを取り出し，DATA_Register に書き込む。
⑩⑪DATA_Register からデータを取り出して全データに対してラベルの補正を行い，補正した値を Block RAM に書き込む。

（2）構成 2

Generate_ADDR，Generate_LAP，Generate_LAB，Generate_TLAB は構成 1 と同じモジュールとなっており，構成 1 と異なる点は，Block RAM に格納されているデータを，BRAM_RW を用いて 1 行分ずつ取り出して PIX_Register で保持する点と，ラプラシアンフィルタ&2 値化処理後のデータを LAP_Register で，仮ラベル生成後のデータを LAB_Register で保持する点である。PIX_Register，LAP_Register，LAB_Register はそれぞれ 3 行分のデータを保持する大きさとなっており，これらを使用して入力されたデータに対して画像処理を行う。処理の手順を以下に示す。

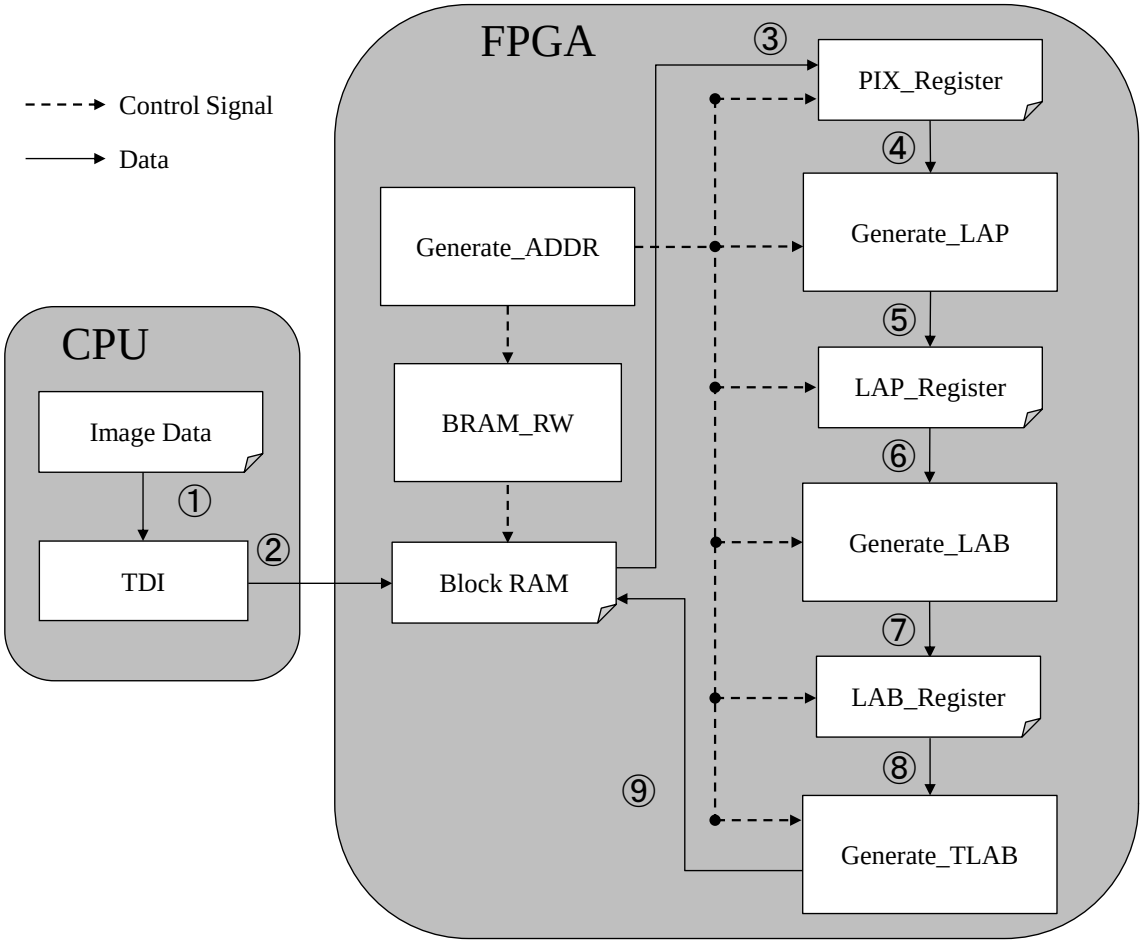


図 19. 欠損検出システムの構成 2

①画像ファイルからデータを取り出し，TDI を用いて雑音除去を行う。
②雑音除去後の画像データを FPGA 上の Block RAM に書き込む。
③Block RAM から 1 行分のデータを取り出して PIX_Register に書き込む。
④⑤PIX_Register からデータを取り出してラプラシアンフィルタ&2 値化処理を行い，LAP_Register に書き込む。
⑥⑦LAP_Register からデータを取り出して仮ラベルの生成を行い，LAB_Register に書き込む。
⑧⑨LAB_Register からデータを取り出してラベルの補正を行い，Block RAM に書き込む。

構成 1 では③から⑪までの動作を 1 回行うことで，構成 2 では③から⑨までの動作を 256 回行う事でラプラシアンフィルタ，2 値化，ラベリング処理の完了となる。本論文では主に構成 2 について述べ，構成 1 は回路規模や処理時間などの比較を行うのに使用した。

3.2 メモリの構成

TDI 処理を行った撮影画像のデータは，FPGA 内の Block RAM を使用して格納を行い，処理モジュールに渡される。Block RAM は Xilinx 社の CORE GENERATOR を使用して生成した IP を用いた。大きさは，液晶用ガラスの撮影画像のサイズが 256*256，1 画素 8bit なので 524,288bit となっており，これにクロック，ライトイネーブル，読み込み／書き込みアドレス，画像データの入出力ポートがついた Single Port RAM となっている。Block RAM の構成を図 20 に示す。

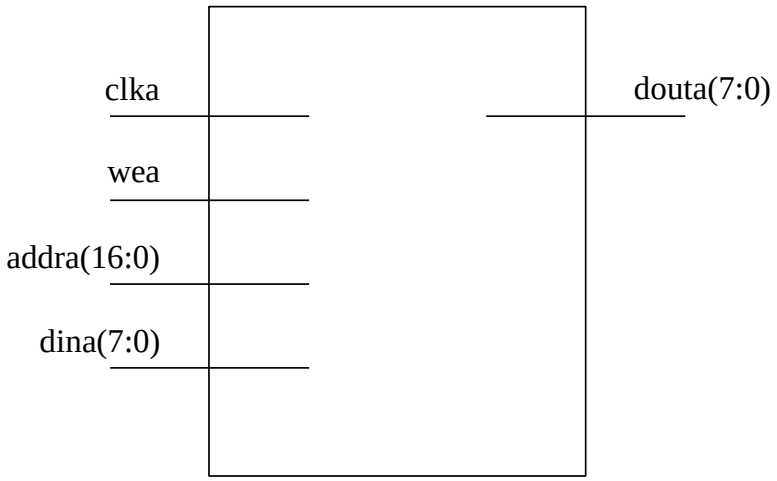


図 20. Block RAM の構成

clka には処理モジュールに用いるクロックを反転したものを入力し，wea と dina は Generate_TLAB から出力される書き込み許可信号とラベル補正後のデータを入力する。addra には BRAM_RW で生成した Block RAM 用のアドレスを入力する。

Block RAM をコントロールしているのが BRAM_RW であり, Block RAM に入力する読み込み／書き込みアドレスは，Generate_ADDR で生成した処理モジュール用の X 座標と，Y 座標を BRAM_RW で変換して入力し，BRAM_RW に入力されたステージフラグによって Block RAM のデータの入出力の制御を行う。

3.3 レジスタファイルの構成

画像処理を行うためには，さきほどの Block RAM に何度もアクセスをしてデータを取ってくる必要がある。メモリのアクセス回数は処理速度に大きく影響し，アクセス回数が少なければ処理を高速化することができる。

ラプラシアンフィルタ処理やラベリング処理では, 図 21 のように注目画素を 1 つずつずらして処理を行うため，対応する領域を毎回取り出す必要がある。

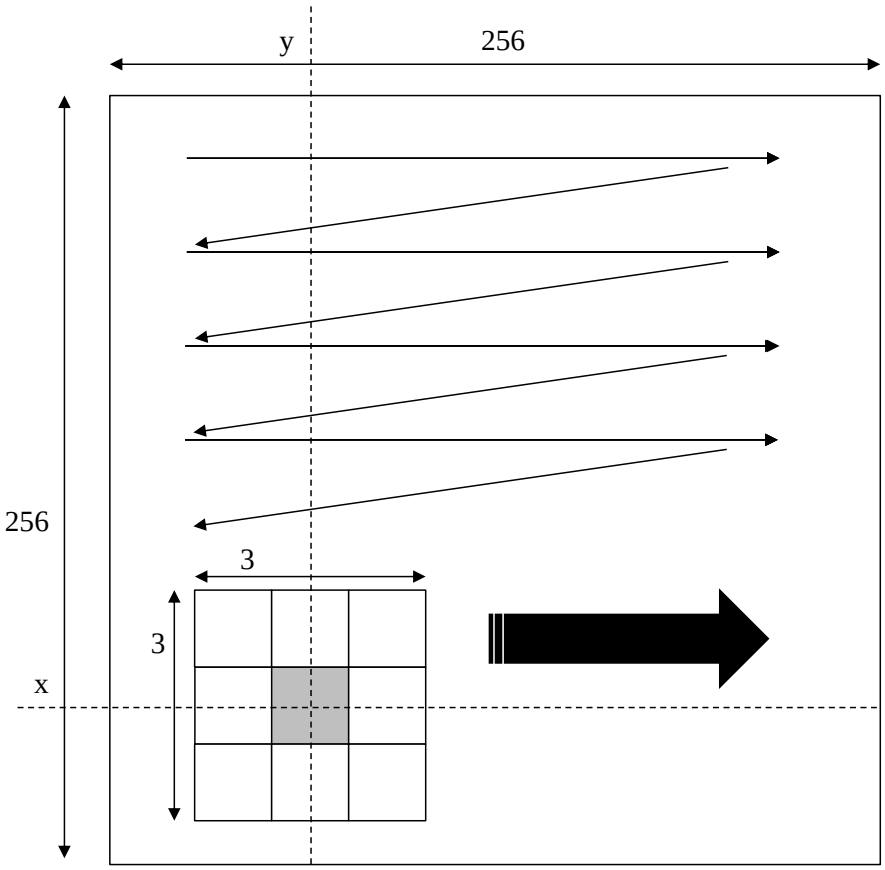


図 21. 画像処理の順番

処理を行うたび毎回メモリにアクセスしてデータを取り出す場合，本論文で使用した液晶用ガラスの画像サイズが 256*256, 処理に用いたマスクパターンなどの大きさが 3*3 なので，586,740 回メモリにアクセスする必要がある。

毎回メモリにアクセスしてデータを取り出す場合，図 22 のように取り出すことになり，メモリアクセス回数は N 回目の取り出しで 9 回，N+1 回目の取り出しで 9 回となり，注目画素を隣にずらすだけで合計 18 回メモリにアクセスしている。

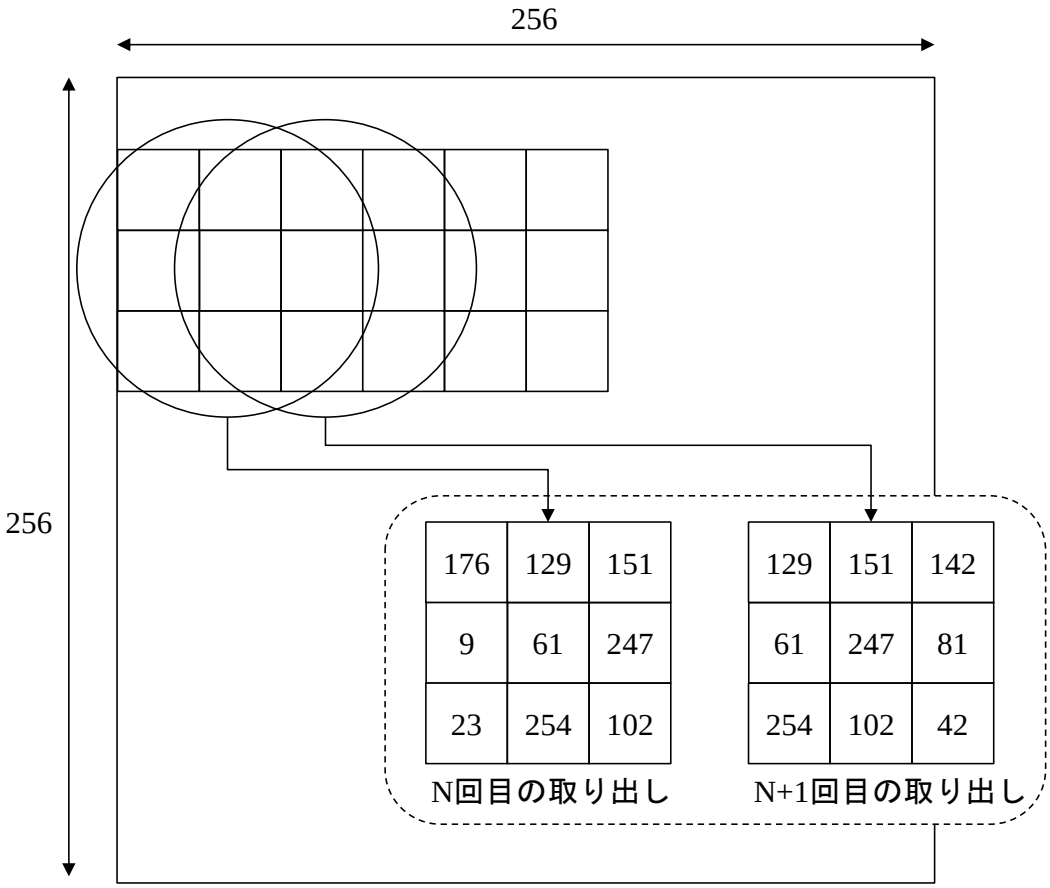


図 22. メモリアクセス

しかし，図 23 に示すように，毎回メモリにアクセスしてデータを取り出す場合，メモリから取り出してきたデータが直前に取り出したデータと重複している。重複しているデータを改めて取り出す必要はないので, 図 24 のようにシフトレジスタを使用して N 回目で取り出したデータを左にシフトし，N+1 回目の取り出しは足りないデータだけを取り出せばよい。メモリアクセス回数は N 回目の取り出しで 9 回, N+1 回目の取り出しで 3 回となり, 注目画素を隣にずらしても計 12 回のメモリアクセスですむ。このように，シフトレジスタを使用することでメモリアクセス回数を削減することができる。

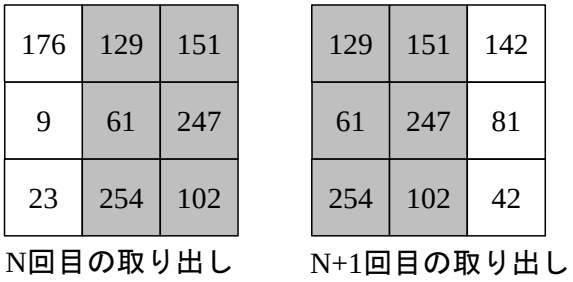


図 23. 取り出したデータの重複

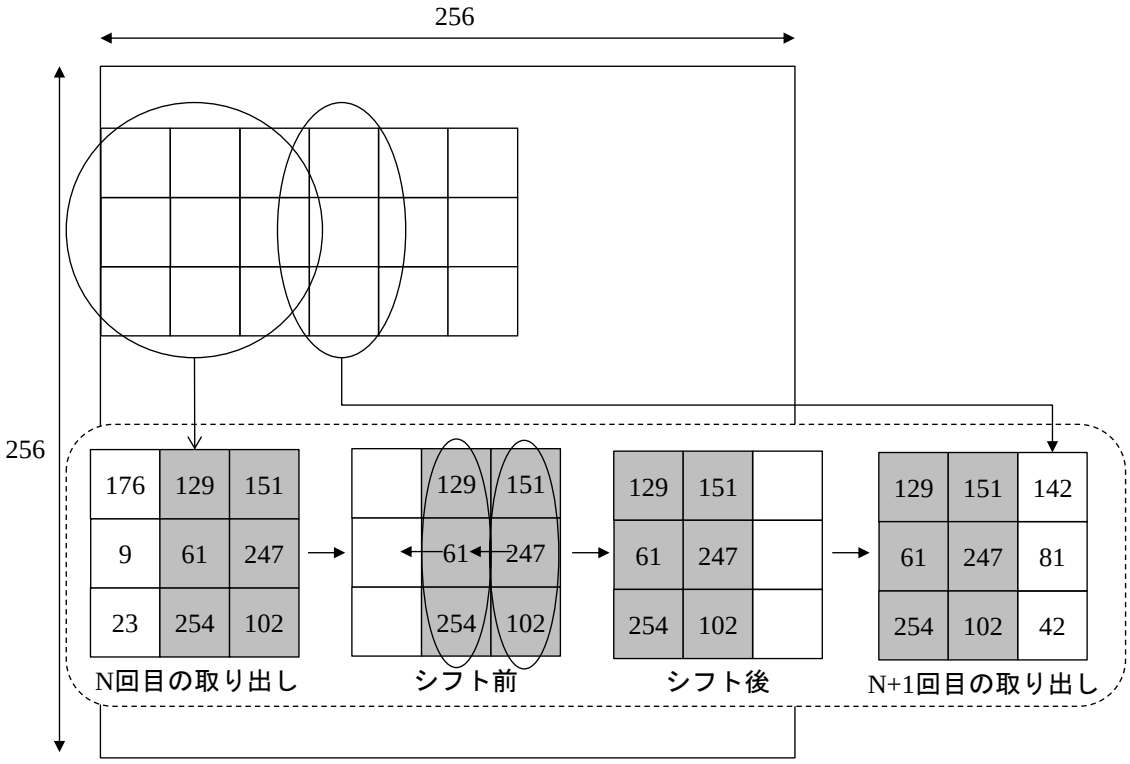


図 24. シフトレジスタを使用してデータを取り出す

シフトレジスタを使用してデータを取り出す場合，メモリアクセス回数は 196,096 回となり，毎回メモリにアクセスする場合と比べて約 67%のメモリアクセス回数の削減となり，処理速度も向上する。

しかし，さらなる処理速度の向上を図るため，本論文では Block RAM に格納されているデータを，図 25 のように液晶用ガラスの画像サイズと同じ大きさのレジスタファイルを用意して取り出す方法と，図 26 のように 3 行分のデータを保持するレジスタファイルを用意して取り出す方法の 2 種類の方法を用いてメモリアクセス回数を削減することにした。

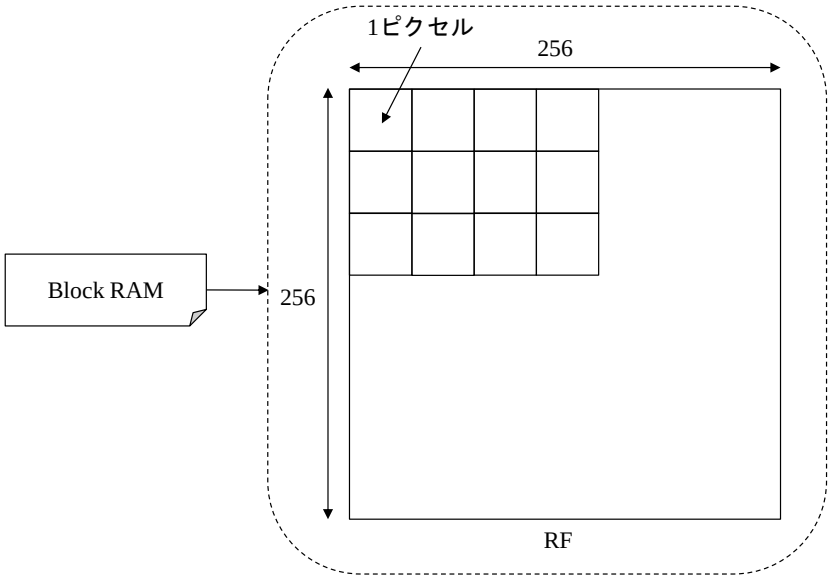


図 25. 256 行分のデータを保持するレジスタファイル

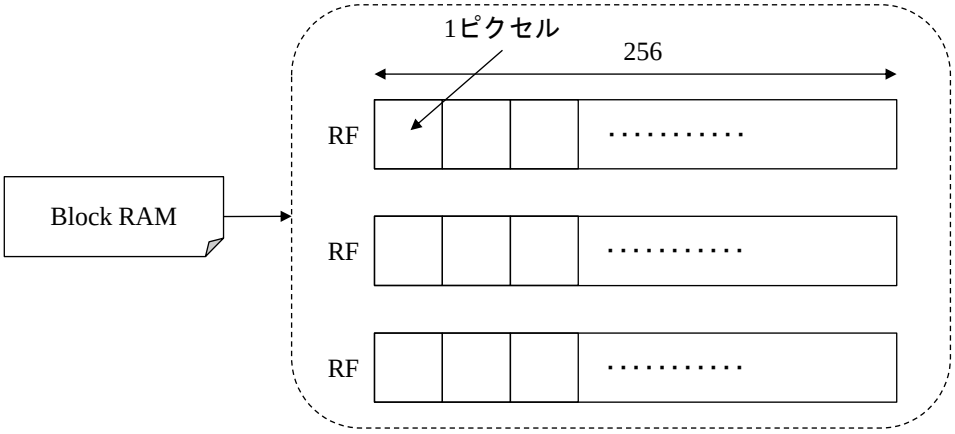


図 26. 3 行分のデータを保持するレジスタファイル

図 25 の方法では, Block RAM に格納されているデータを全てレジスタファイルに書き込むことで，処理の際に対応する領域のデータをメモリにアクセスして取り出す必要を無くした。

図 26 の方法では, 3 つのレジスタファイルを用意して Block RAM に格納されている対応する領域のデータを順番に 3 行ずつ保持し，処理の際にはこの 3 つのレジスタファイルからデータを取り出すことで，メモリにアクセスする回数を削減した。どちらの方法でもメモリのアクセス回数は 65,536 回となっており，毎回メモリにアクセスしてデータを取り出す場合と比べて約 89%, シフトレジスタを使用してデータを取り出す場合と比べても約 67% のメモリアクセス回数の削減となる。

3 行分のデータを保持するレジスタファイル, LINE_DATA0・LINE_DATA1・LINE_DATA2 のそれぞれの大きさは 256*8bit となっており，それぞれで 1 行分のデータを保持する。処

理に必要な 9 つのデータは，LINE_DATA0・LINE_DATA1・LINE_DATA2 からそれぞれ 3 つずつ取り出され，マスクパターンなどに対応する領域として出力される。レジスタファイルに保持されたデータを処理モジュールへ出力する様子を図 27 に示す。

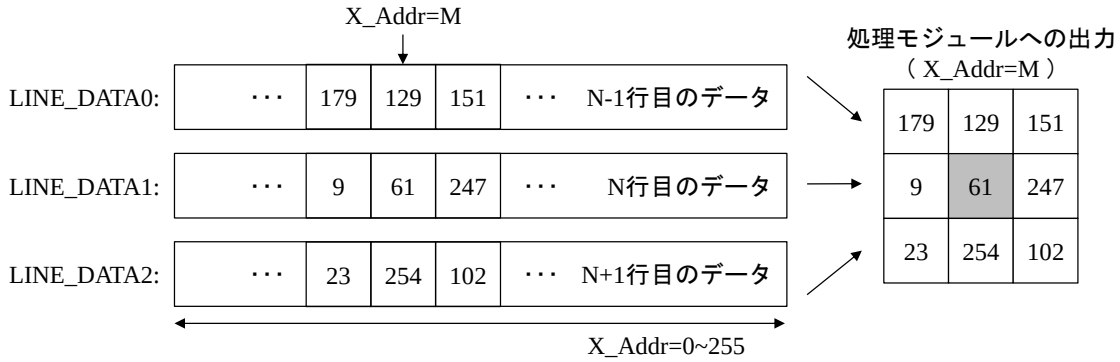


図 27. レジスタファイルの入出力 1

図 27 は，X 座標が M の時に各レジスタファイルに保持された 3 行分のデータから 9 つのデータを処理モジュールへ出力する様子で，LINE_DATA1 に保持されている N 行目のデータを注目画素として処理を行っている場合である。この時，LINE_DATA0[M-1]，LINE_DATA0[M]，LINE_DATA0[M+1]がマスクパターンなどに対応する領域の上の行として，LINE_DATA1[M-1]，LINE_DATA1[M]，LINE_DATA1[M+1]が真ん中の行として，LINE_DATA2[M-1]，LINE_DATA2[M]，LINE_DATA2[M+1]が下の行として出力される。この処理を画像の左端から右端（X=0~255）まで行い，N 行目のデータを注目画素として全て出力し終えたら，Block RAM から次の行のデータを取り出してレジスタファイルに保持し，LINE_DATA2 で保持されている N+1 行目のデータを注目画素として先のように出力する。

しかし，レジスタファイルは 3 つしかなく，全部で 3 行分のデータしか保持できないので，Block RAM にアクセスして次の行のデータを取り出して保持するときは，データのシフトを行わずに，一番古い行のデータが保持されているレジスタファイルに上書きを行って次の行のデータを保持する。

図 27 では N-1，N，N+1 行目のデータが保持されていて，この中で一番古い行のデータは N-1 行目のデータが入っている LINE_DATA0 なので, 次の行のデータを Block RAM から取り出すときは，LINE_DATA0 に上書きを行って N+2 行目のデータを保持する。N+2 行目のデータを取り出して保持し，N+1 行目のデータを注目画素として出力する様子を図 28 に示す。

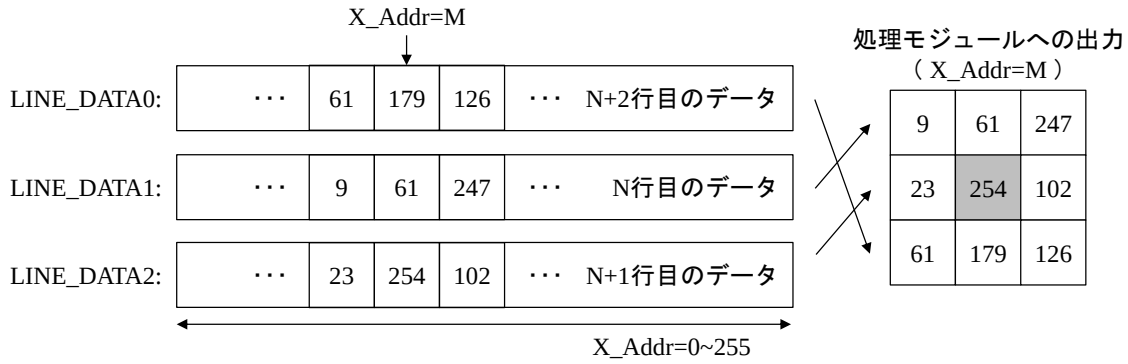


図 28. レジスタファイルの入出力 2

図 27 では，LINE_DATA0 に N-1 行目のデータが入っていたが，図 28 では N+2 行目のデータが保持されている。そして注目画素の行は N 行目から N＋1 行目のデータに変わり，処理モジュールへ出力するデータも変化する。LINE_DATA1 で保持しているデータを対応する領域の上の行として，LINE_DATA2 で保持しているデータを真ん中の行として，LINE_DATA0 で保持しているデータを下の行として出力する。こちらの処理も画像の左端から右端（X=0~255）まで行い，N+1 行目のデータを注目画素として全て出力し終えたら，Block RAM にアクセスして次の行のデータを，一番古い行のデータを保持している LINE_DATA1 に上書きを行い，N+2 行目のデータを注目画素のデータとして出力する。これを画像の上から下まで（Y=0~255）まで行う。このように，用意した 3 つのレジスタファイル，LINE_DATA0・LINE_DATA1・LINE_DATA2 に Block RAM から取り出したデータを保持し，シフトレジスタは使用せずに Block RAM からデータを 1 行ずつ受け取って処理モジュールへと出力する。レジスタファイルにデータを書き込む順番は，LINE_DATA0→LINE_DATA1→LINE_DATA2→LINE_DATA0→LINE_DATA1･･･となっており，処理モジュールへの出力の順番は Generate_ADDR で生成されるカウント信号によって制御されている。

処理モジュールへ出力する際に対応する領域のデータがない場合，つまり画像の端を処理する時は，Generate_ADDR から入力される X 座標，Y 座標を使用してレジスタファイル内で端フラグの生成を行い，これを用いてデータの値を“0”として出力する。

端フラグは X 座標／Y 座標が 0 の時に立ち上がる TCon_X_0／TCon_Y_0，X 座標／Y 座標が 255 のときに立ち上がる TCon_X_255／TCon_Y_255 の 4 つからなる。対応する領域にデータがなく，処理モジュールに“0”を出力するパターンは図 29 に示すように全部で 8 通りである。X＝1~254 かつ Y＝1~254 のときは端フラグが立っていないので，図 27 や図 28 で示したように出力を行う。表 1 に端フラグが立つタイミングを示す。

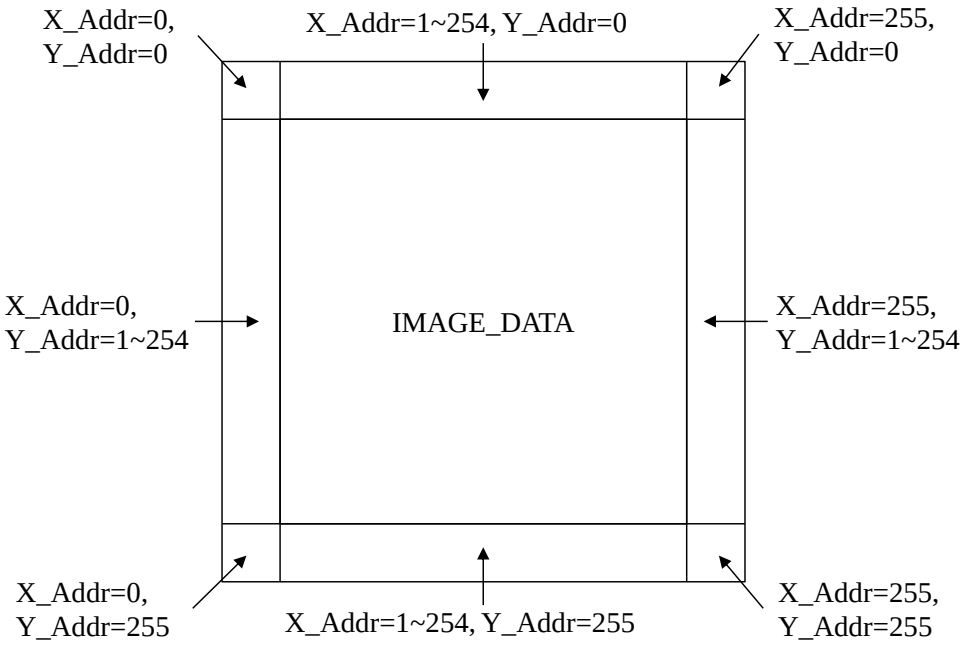


図 29. 対応する領域にデータが存在しないパターン

表 1. 端になったときのフラグ

	TCon_X_0	TCon_X_255	TCon_Y_0	TCon_Y_255
X_Addr=0, Y_Addr=0	1	0	1	0
X_Addr=0, Y_Addr=1~254	1	0	0	0
X_Addr=0, Y_Addr=255	1	0	0	1
X_Addr=1~254, Y_Addr=0	0	0	1	0
X_Addr=1~254, Y_Addr=1~254	0	0	0	0
X_Addr=1~254, Y_Addr=255	0	0	0	1
X_Addr=255, Y_Addr=0	0	1	1	0
X_Addr=255, Y_Addr=1~254	0	1	0	0
X_Addr=255, Y_Addr=255	0	1	0	1

図 30 は X＝0，Y＝0 の時の出力の様子で，注目画素は画像の一番左上の“179”となり，これを中心とした 9 つのデータを処理モジュールに出力する。注目画素の左上，上，右上，左のデータは画像の外なのでデータが存在せず，TCon_X_0=1，TCon_Y_0=1 により“0”が出力されている。

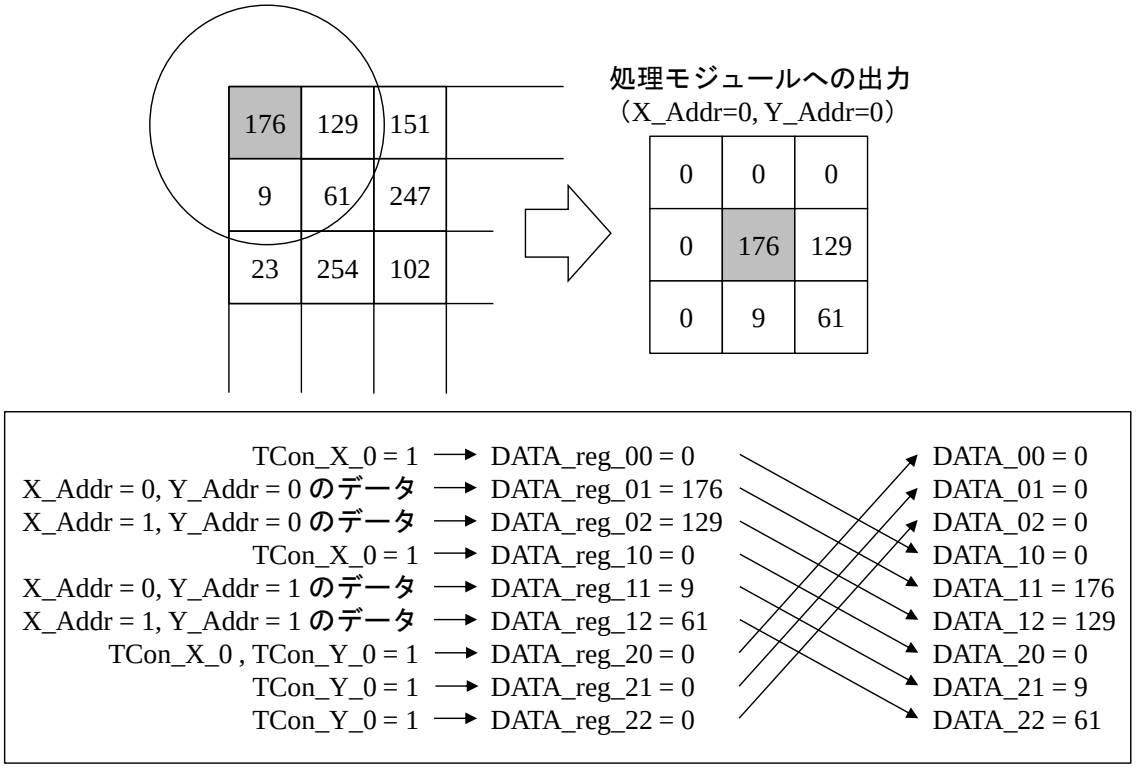


図 30. 端でのレジスタファイル出力

このような 3 行分のデータを順番に保持して処理モジュールへ出力するレジスタファイルを，Block RAM から取り出したデータを保持するのに 1 つ（PIX_Register），ラプラシアンフィルタ&2 値化処理後のデータを保持するのに 1 つ（LAP_Register），仮ラベル生成後のデータを保持するのに 1 つ（LAB_Register）で，計 3 つを作成した。

各レジスタファイルは後述する制御モジュール Generate_ADDR によって動作を制御されている。PIX_Register の構成を図 31 に示す。x と y は Generate_ADDR で生成される X 座標，Y 座標が入力され，入出力の制御信号 stg_flag とスタート信号 lap_start，データの調整信号 cnt も Generate_ADDR で生成されたものを入力する。DATA_IN は Block RAM から取り出したデータが入力され，DATA_00 から DATA_22 までの 9 つは Generate_LAP に向けての出力になり，cnt により順番を調整された 9 つのデータが対応する領域のデータとして処理される。

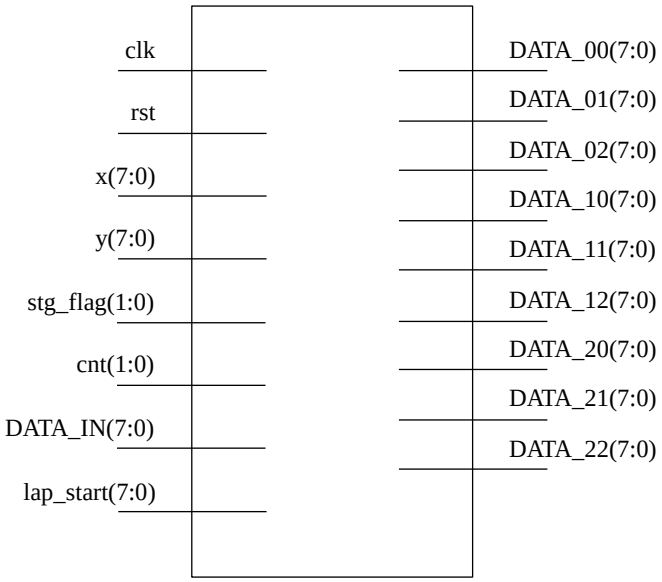


図 31. PIX_Register の構成

図 32, 図 33に示すLAP_RegisterとLAB_RegisterもPIX_Registerとほぼ同じ構成になり，異なる点として，LAP_Register はデータを 5 つしか出力しない点である。LAP_Register の出力先は Generate_LAB になり，仮ラベルの生成では注目画素以降のデータを参照する必要がないので 5 つとなっている。

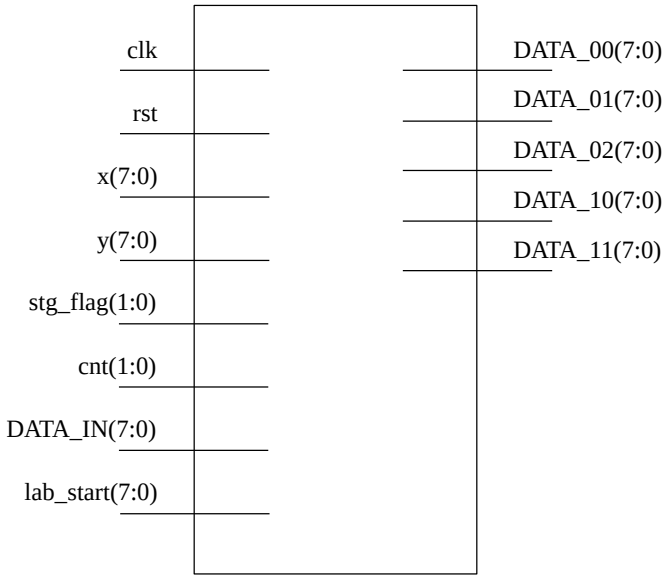


図 32. LAP_Register の構成

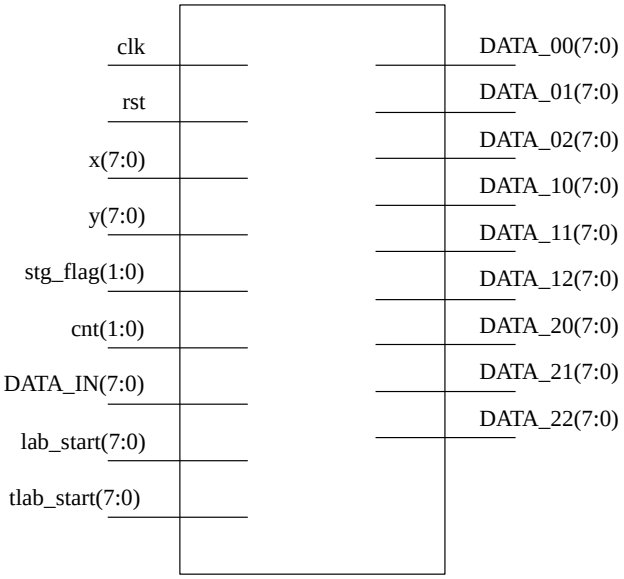


図 33. LAB_Register の構成

4. FPGA 上での欠損検出システムの設計と実現

4.1 制御モジュールとアドレス生成

FPGA 上のモジュールの制御を行っている Generate_ADDR では，処理に必要な X 座標や Y 座標，処理モジュールのスタート信号や制御信号などを生成している。本論文で生成した制御信号をステージフラグと呼び，このステージフラグにより Block RAM からデータを取り出してレジスタファイルに書き込むのか，画像処理を行うのかを制御している。表 2 にステージフラグ別の処理の内容を示す。

表 2. ステージフラグ

Stage Flag	動作モジュール	処理
0	Generate_ADDR／BRAM_RW ／PIX_Register	Block RAM から 1 行分のデータを取り出して PIX_Register に書き込む
1	Generate_ADDR／PIX_Register ／Generate_LAP／LAP_Register	ラプラシアンフィルタ&2 値化処理を行い，処理後のデータを LAP_Register に書き込む
2	Generate_ADDR／LAP_Register ／Generate_LAB／LAB_Register	仮ラベル生成処理を行い，処理後のデータを LAB_Register に書き込む
3	Generate_ADDR／LAB_Register ／Generate_TLAB／BRAM_RW	ラベル補正処理を行い，処理後のデータを Block RAM に書き込む

Generate_ADDR で生成するスタート信号は lap_start, lab_start, tlab_start の 3 つで, lap_start は PIX_Register の出力とラプラシアンフィルタ&2 値化モジュールのスタート信号になり，lab_start は LAP_Register の出力と仮ラベル生成モジュールのスタート信号になる。tlab_start は LAB_Register の出力とラベル補正モジュールのスタート信号になる。各スタート信号は立ち上がるタイミングが違っており，lap_start は，処理モジュールが対応する領域のデータとして注目画素以降のデータも必要になるので，PIX_Register に 2 行分のデータが保持されるまで待ってから立ち上げる。lab_start は，注目画素以降のデータを参照する必要がないので，LAP_Register にデータが保持され次第立ち上げる。tlab_start は注目画素以降のデータも参照する必要があるので，LAB_Register に 2 行分のデータが保持されてから立ち上げる。スタート信号が立ち上がるタイミングを図 34 に示す。

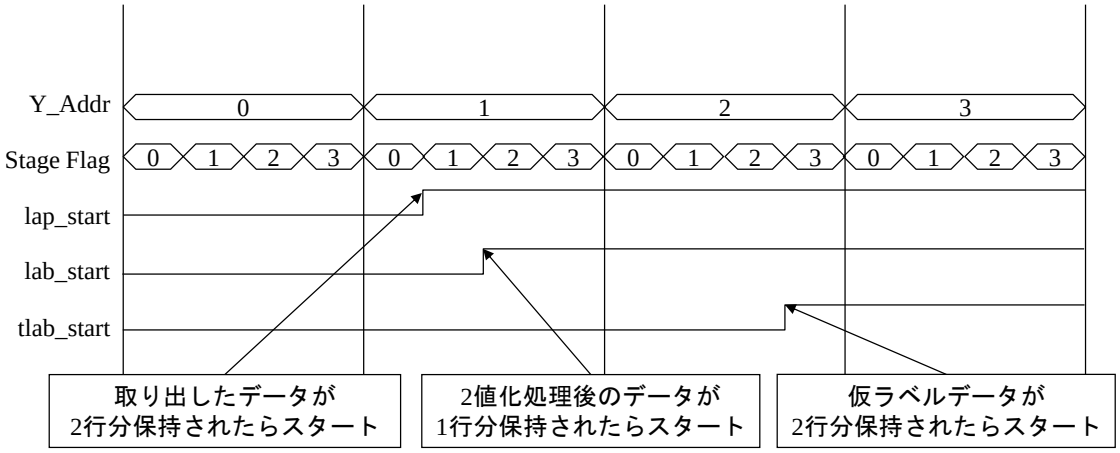


図 34. スタートのタイミング

これらのステージフラグ，処理モジュールのスタート信号に加え，処理で使用する X 座標，Y 座標などの生成をここでは行っている。

4.2 ラプラシアンフィルタ&2 値化モジュール

Generate_LAP は, PIX_Register から出力されたデータに対してラプラシアンフィルタ処理と 2 値化処理を行い，LAP_Register に書き込みを行う。Generate_LAP はステージフラグが 1 の時動作し, 処理を開始するのは PIX_Register に 2 行分のデータが保持されてから処理を開始する。入力されるデータは，すでに PIX_Register で対応する領域のデータとして順番が調整されているので，lap_start が立ち上がったら入力される 9 つのデータをもとに，式 1 の演算を行う。そして，ラプラシアンフィルタ処理後のデータを閾値と比べ，閾値よりも上なら“255”を，下なら“0”を LAP_Register に出力する。

4.3 ラベリングモジュール

ラベリング処理は 2 つのモジュールに分かれており， Generate_LAB と Generate_TLAB から構成される。

Generate_LAB は LAP_Register から入力されるデータをもとに仮ラベルを生成し，LAB_Register にデータを書き込む。ラベルをつける際，注目画素の左上，上，右上，左のラベルを参照しながら仮ラベルをつける。周囲 4 画素にラベルがあった場合はそれをコピーして出力し，ラベルがない場合は新しいラベルをつけて出力する。Generate_LAB では注目画素以降のデータは参照しないので，処理に必要なデータ数は注目画素と周囲 4 画素の 5 つということになる。図 35 に示すように，入力される 5 つのデータは LAP_Register が保持しているデータなので，保持されているデータはラベル情報ではなく，“0”か“255”という 2 値化されたデータになる。

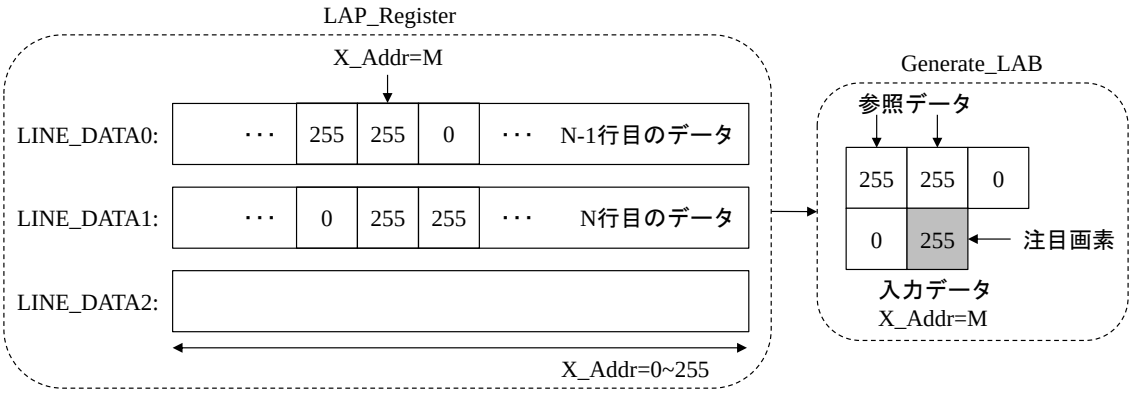
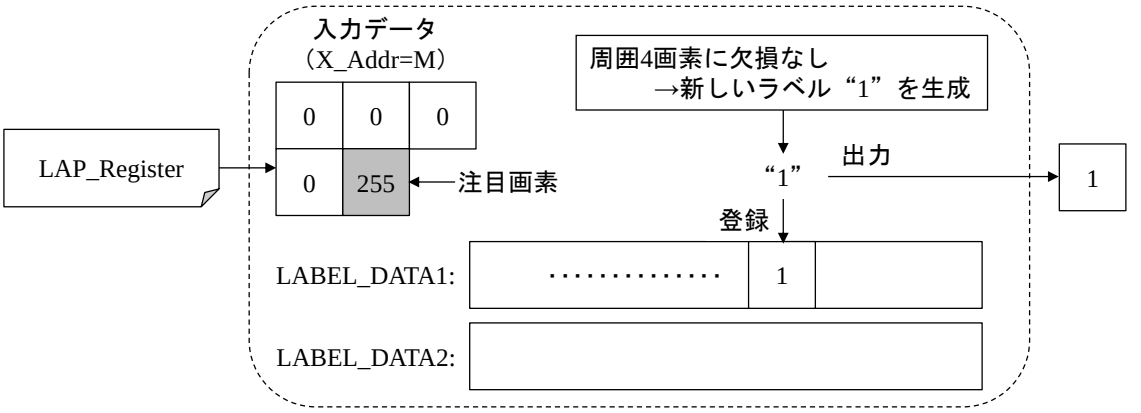
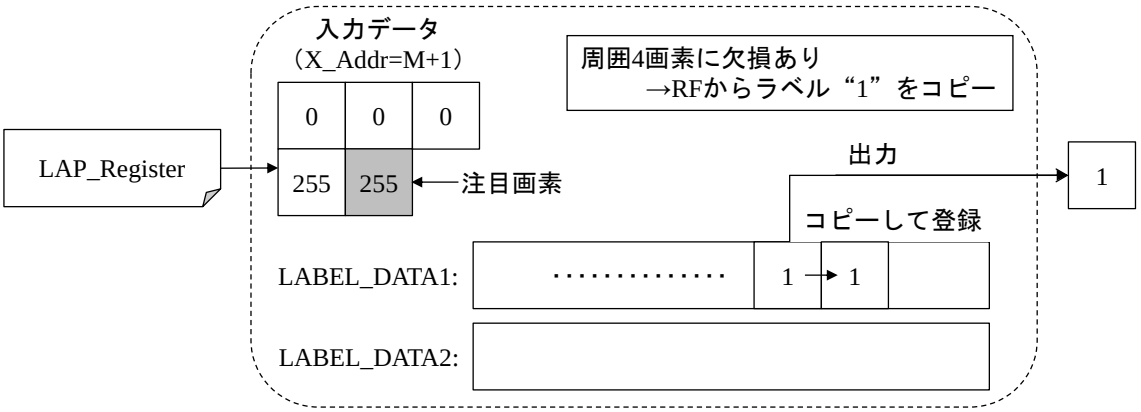


図 35. Generate_LAB に入力されるデータ

入力された注目画素のデータが“255”のとき，周囲 4 画素のラベルを参照しながら仮ラベルの生成を行うが，左上と上の“255”というデータは 2 値化による処理結果なので，注目画素に正しいラベルを付けることができない。そこで図 32 に示すように，Generate_LAB 内に生成した仮ラベルを 2 行分保持するレジスタファイルを用意し，これを用いて問題の解決に当たった。



（a）周りに欠損がない場合



（b）周りに欠損がある場合

図 36. 内部にレジスタファイルを持つ Generate_LAB

図 36 は X 座標が M と M+1 のときに入力されるデータを処理する様子を示したもので，X 座標が M のときの周囲 4 画素に欠損がないので，新しい仮ラベル“1”を生成し，内部のレジスタファイルに生成したラベルを登録し，出力をしている。このように生成したラベルを，処理モジュール内部に用意したレジスタファイルに保持することにより，X 座標が M+1 の処理を行うときに，今まで生成したラベルを参照することが可能になり，周囲に欠損がある場合，該当するデータのラベル情報をレジスタファイルから取り出し，処理を行う。内部に用意するレジスタファイルの大きさは，Generate_LAB が注目画素以降のデータを参照する必要がないので 2 行分となっている。入力された注目画素のデータが“0”の場合は，内部のレジスタファイルに“0”を登録して出力を行う。

Generate_TLAB は LAB_Register に保持された仮ラベルデータをもとに注目画素を含む 9 つのデータの中から最も小さいラベルを注目画素のラベルとして出力する。ラベルの補正を行う際，最小のラベルをつけるためにラベル同士の比較を行う。手順として，注目画素を除く 8 つのラベルをそれぞれ比較して最小ラベルを探し，その最小ラベルと注目画素のラベルを比較してより小さなラベルを Block RAM に出力する。その様子を図 37 に示す。

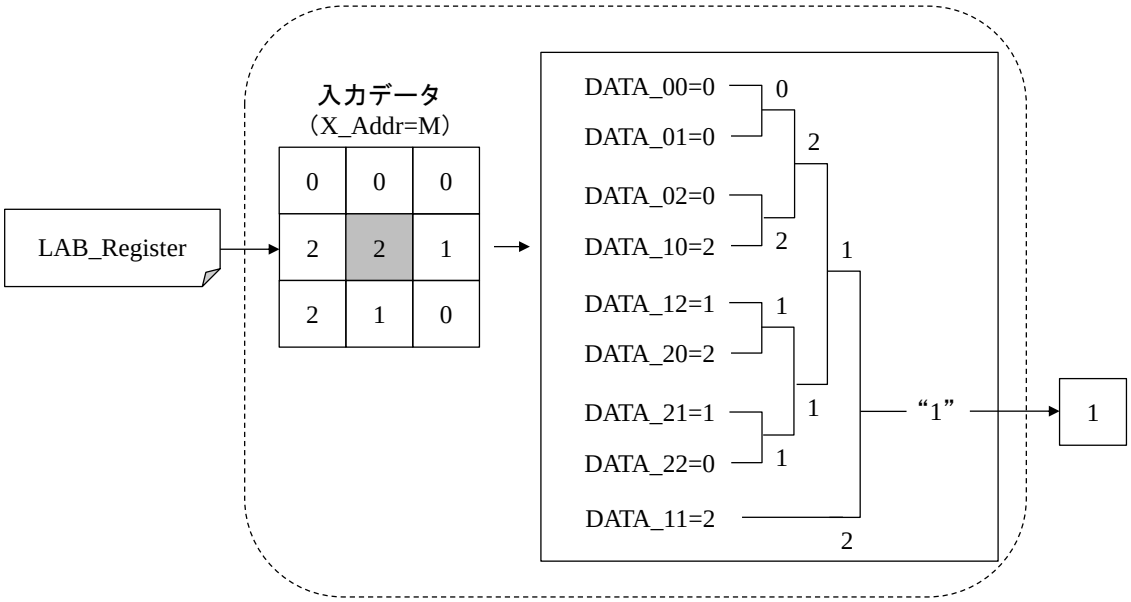


図 37. 最小ラベルの探索

Generate_LAB はステージフラグが 2 のときに動作し，Generate_TLAB は 3 のときに動作する。この 2 つをラベリングモジュールとして FPGA 上に実装していく。

5. FPGA 上での実験

5.1 実験条件

実験には TDI を用いて雑音除去をあらかじめ施したデータを使用し，データを FPGA 内の Block RAM に格納して画像処理を行う。

使用した液晶用ガラスの撮影画像のサイズは 256*256，1 画素 8bit のものを使用し，図 18 と図 19 で示した構成 1 と構成 2 の FPGA 上での回路規模や処理時間を計測する。 処理時間は，画像ファイルからデータを取り出し，全データに対して画像処理を行い，メモリに格納するまでの時間を処理時間とした。

FPGA は Xilinx 社の Virtex5 ML507 評価ボードを用い，スライス数 11200，FF（フリップフロップ）回路数 44800，LUT 数 44800，Block RAM 容量 5328Kbit となっている。デザイン設計と論理合成には同社の設計ツール ISE を用い，ISim でシミュレーションを行った。

また，処理時間の比較を行うために，同じ画像処理を CPU 上のソフトウェアで実行して処理時間を計測する。比較用に実験したソフトウェア処理の動作環境は Intel Core 2 Quad CPU Q9400 2.67GHz，実装メモリ 4.00GB，OS は Windows7 Ultimate のものを使用した。

5.2 実験結果

設計した構成 1 と構成 2 のデザインの論理合成を行い，回路規模や動作周波数，処理時間を計測した。表 3 に構成 1 の全体の回路規模と，表 4 に構成 2 の全体の回路規模を示す。

表 3. 構成 1 の全体の回路規模

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	21	44800	0%
Number of Slice LUTs	124643	44800	278%
Number of fully used LUT-FF pairs	8	2295	0%
Number of bonded IOBs	51	640	7%
Number of Block RAM/FIFO	16	128	10%
Number of BUFG/BUFGCTRLs	1	32	3%

表 4. 構成 2 の全体の回路規模

Logic Utilization	Used	Available	Utilization
Number of Slice Registers	52	44800	0%
Number of Slice LUTs	2262	44800	5%
Number of fully used LUT-FF pairs	19	2295	0%
Number of bonded IOBs	51	640	7%
Number of Block RAM/FIFO	16	128	10%
Number of BUFG/BUFGCTRLs	1	32	3%

構成 1 は全データを保持するために, LUT 数がハードウェア容量を超える 124643／44800 となった。構成 2 の LUT 数は，構成 1 と比べて約 98%減の 2262／44800 となり，レジスタ数・LUT-FF ペアが少し多くなった。これは，3 行分のデータを保持するレジスタファイルなどの制御信号の数が多くなったことが考えられる。

構成 1 と構成 2 の各モジュールの回路規模を表 5 と表 6 に示す。

表 5. 構成 1 の各回路規模

	Registers	LUTs	LUT-FF pairs
RF_Read_Write	0	119116	0
Generate_ADDR	19	47	19
Generate_LAP	0	77	0
Generate_LAB	9	404	9
Generate_TLAB	0	148	0

表 6. 構成 2 の各回路規模

	Registers	LUTs	LUT-FF pairs
PIX_Register	0	634	0
LAP_Register	0	574	0
LAB_Register	0	557	0
BRAM_RW	0	9	0
Generate_ADDR	27	60	27
Generate_LAP	0	105	0
Generate_LAB	9	397	9
Generate_TLAB	0	148	0

構成 1 の RF_Read_Write が 256 行分のデータを保持するモジュールになり，構成 2 の PIX_Register，LAP_Register，LAB_Register がそれぞれ 3 行分のデータを保持するモジュールとなっているので，他のモジュールと比べて LUT 数が多くなっている。また，構成 2 の Generate_ADDR のレジスタ数・LUT 数・LUT-FF ペアが構成 1 と比べて増えているのは，PIX_Register，LAP_Register，LAB_Register に対する制御信号の数が多くなっているからであり，構成 2 の全体の回路規模でレジスタ数・LUT-FF ペアが増えた要因となっていることが分かる。処理モジュールの回路規模は，Generate_LAP は構成 2 の方が，Generate_LAB は構成 1 の方が少し大きい結果となっている。

構成 1 と構成 2 の動作周波数と，ソフトウェア処理も含めた処理時間を表 7 に示す。

表 7. 動作周波数と処理時間

	CPU	Method 1	Method 2
Maximum Frequency	――――	69.83MHz	39.06MHz
Processing Time／pixel	2276.2ns	85.93ns	102.41ns
Processing Time	149.17ms	5.63ms	6.71ms
Speed up ratio	1	26.50	22.23

最大動作周波数は構成 1 が 69.83MHz，構成 2 が 39.06MHz という結果になり，処理時間は構成 1 が 5.63ms，構成 2 が 6.71ms という結果になった。処理時間はソフトウェア処理と比べて構成 1 で 26.5 倍，構成 2 で 22.23 倍の速度向上が得られた。

5.3 考察

実験の結果，構成 1 が一番高速に動作するという結果になり，ソフトウェア処理と比べて，26.5 倍の速度向上が得られた。しかし，ターゲットデバイスである Virtex5 ML507 評価ボードのハードウェア量を上回る結果となっている。

一方構成 2 は，処理速度は構成 1 より少し遅いものの，全体的なハードウェア量はとても小さいものになった。LUT 数では構成 1 と比べて，約 98%のハードウェア量の削減となっている。また，各モジュールの回路規模を求めることにより，全体の回路規模で構成 2 のレジスタ数・LUT-FF ペアが増えた原因が，3 行分のデータを保持する際の制御信号などであるということも分かった。しかし，使用するレジスタ数や LUT 数が増えたといっても，ターゲットデバイスに十分実装できる回路規模となっている。構成 2 の処理速度が構成 1 と比べて遅くなっている原因は，小さなハードウェア量で同じ画像処理を実現するために，レジスタファイルの入出力に関するデータの制御などで遅延が生じているからである。こちらの構成での処理時間は構成 1 と比べて 1.08ms 遅い 6.71ms という結果となったが，ソフトウェア処理と比べて 22.23 倍の速度向上が得られた。

構成 2 の設計では，端の処理を含めた入出力の順番を調整する部分を工夫して設計を行い，3 行分のデータを保持するレジスタファイルを作成することで，大幅なハードウェア量の削減ができた。データの扱い方も構成 1 と構成 2 で違い，構成 1 では Block RAM に格納されているデータを全てレジスタファイルに書き込んで処理を行ったが，構成 2 では，格納されているデータを 1 行ずつ取り出して処理を行っている。

また，構成 2 の方法では処理モジュールのパイプライン化が見込まれる。Generate_LAP・Generate_LAB・GenerateTLAB はそれぞれ独立して動作しており，PIX_Register・LAP_Register・LAB_Register にデータが保持されていれば，処理を開始することができるので，ステージごとに処理を分けることにより，処理を並列に行うことが可能になる。これにより，さらなる画像処理の高速化が見込まれる。

6. おわりに

本論文では，株式会社ケー・デー・イーとの協同研究として，CPU を用いたソフトウェア処理による画像処理や，FPGA 上に実装させるためのモジュールの設計を行い，ハードウェアを用いた画像処理によって欠損を高速に検出する手法を提案した。

欠損検出システムの設計では，2 種類の構成を用いてラプラシアンフィルタ，2 値化，ラベリングのハードウェア化を行い，ソフトウェア処理と比べて大規模な構成の回路で 26.5 倍，小規模な構成の回路で 22.23 倍の速度向上を得ることができた。また，小規模な構成の回路設計では，大規模な構成の回路と比べて約 98%のハードウェア量の削減を行い，ターゲットデバイスに十分実装できうるハードウェア量となった。

今後の課題として，液晶用ガラスのノイズを軽減させる TDI のハードウェア化と，考察で述べた処理モジュールのパイプライン化があげられる。FPGA の仕組みやデザインツールの使用方法などをさらに学び，目標とする欠損検出画像処理システム全体をハードウェア化することが課題である。

謝辞

本研究の機会を与えてくださり，ご指導をいただきました山崎勝弘教授に深く感謝いたします。本研究は，株式会社ケー･デーイーとの共同研究であり，研究に必要なデータを提供していただいた三宅淳司氏と西田洋隆氏をはじめ，様々な面で貴重な助言や励ましを下さった孟林助手や研究室の皆様に深く感謝します。

参考文献

[1] C. J. Lu and D. M. Tsai, “Defect inspection of patterned TFT-LCD panels Using a Fast Sub-image Based SVD”, Int. Journal of Production Research, vol.42, pp.4331-4351, 2004.

[2] Y. H. Liu, Y. C. Liu and Y. Z. Chen, “High-speed inline defect detection for TFT-LCD array process using a novel support vector data description”, Expert Syst. Appl., 38(5):6222-6231, 2011.

[3] Y. ITO and K. NAKANO, “Low-Latency Connected Component Labeling Using an FPGA”, International Journal of Foundations of Computer Science, pp405-425, 2010.

[4] K. Wu, E. Otoo, and A. Shoshani, “Optimizing Connected Component Labeling Algorithms”, Medical Imaging 2005 Image Processing. Proceedings of the SPIE, Volume 5747, pp.1965-1976, 2005.

[5] 松崎裕樹, 山崎勝弘, 平岡邦廣, 西田洋隆, 三宅淳司, “マハラノビス距離を用いたガラス検査用画像判別の実現”, FIT2006, I-031, 2006.

[6] 松崎裕樹, 山崎勝弘, 小柳滋, 平岡邦廣, 西田洋隆, 三宅淳司, “マハラノビス距離を用いたガラス検査用画像判別とラベリングのハードウェア化”, 情報処理学会関西支部支部大会, C-09, 2006.

[7] 松山圭輔, 孟林, 山崎勝弘, “FPGA ボードを用いた液晶用ガラスの欠損検出画像処理の高速化”, 情報処理学会関西支部支部大会, A-03, 2011.

[8] 松山圭輔, 孟林, 天井康雄, 山崎勝弘, “FPGA を用いた液晶用ガラス欠損検出システムの高速化”, IEICE Technical Report RECONF2012-37, pp.79-84, 2012.