JAIST Repository
https://dspace.jaist.ac.jp/

Title	オーディオ電子透かしのハードウェア高速検出に関する研究
Author(s)	榊原，憲宏
Citation	
Issue Date	2005-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/1933
Rights	
Description	Supervisor:井口　寧，情報科学研究科，修士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

修 士 論 文

オーディオ電子透かしのハードウェア高速検出に関する研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

榊原憲宏

2005年3月

修 士 論 文

オーディオ電子透かしのハードウェア高速検出に関する研究

指導教官　井口寧 助教授

審査委員主査　井口寧 助教授
審査委員　松澤照男 教授
審査委員　田中清史 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

310043 榊原憲宏

提出年月: 2005 年 2 月

Copyright © 2005 by Kazuhiro Sakakibara

概 要

近年，オーディオコンテンツの不正なネットワーク配布が問題となっており，これを防ぐための技術として電子透かし技術が注目されている．電子透かし技術とは，人間が気づかれないように著作者の情報をコンテンツに埋め込む技術であり，この技術を用いて著作権の主張が可能となる．しかしながら，従来は透かしの検出をソフトウェアで行っており検出速度が非常に遅かった．そのため，ネットワークで不正に配布されるコンテンツそのものの流通を防ぐことに利用することができなかった．そこで，本研究ではハードウェアによりオーディオ電子透かしを高速に検出する手法を提案し，その回路構築法を示す．構築したハードウェアでは，検出回路の効率化や並列化を検討することで，ソフトウェアでは実現が難しい高速な複数の透かし検出を実現した．構築したハードウェアの評価を行った結果，ソフトウェアによる透かし検出の 23 倍の検出速度を達成し，最大で 82 個の透かしを同時に検出可能であることを示した．また，ネットワーク中で交換されるオーディオコンテンツから，電子透かしを高速に検出することで不正配信を防ぐ手法を提案する．

目 次

第 1 章　序論 **1**
1.1 研究の背景と目的 1
1.2 本文の構成 2

第 2 章　オーディオ電子透かし **3**
2.1 はじめに 3
2.2 オーディオ電子透かし 3
2.2.1 周波数領域を利用したオーディオ電子透かし 3
2.2.2 時間領域を利用したオーディオ電子透かし 4
2.2.3 検出方法 5
2.3 ハードウェア化に適したアルゴリズムの検討 6
2.3.1 ハードウェア化の考慮点 6
2.3.2 アルゴリズムの検討 6
2.4 wav ファイルフォーマット 7
2.5 採用アルゴリズム 8
2.5.1 埋め込みアルゴリズム 8
2.5.2 透かしデータ 9
2.5.3 検出アルゴリズム 9
2.6 ソフトウェア実装による予備実験 10
2.6.1 埋め込みフローチャート 10
2.6.2 検出フローチャート 11
2.6.3 検出速度と検出精度 12
2.7 まとめ 14

第 3 章　透かし検出アルゴリズムのハードウェア化 **15**
3.1 はじめに 15
3.2 実装環境 15
3.3 処理の分担 16
3.4 FPGA 内の回路構築方針 17
3.5 アルゴリズムの考察による乗除算の削減と並列化 18
3.6 複数の透かしデータの保存 20
3.6.1 問題と解決法 20

3.6.2 メモリアクセスによる並列化の制限 . . . 20
3.7 透かし検出法 . . . 22
3.7.1 問題と解決法 . . . 22
3.7.2 平均値を求めない透かしの検出法 . . . 22
3.7.3 閾値の決定 . . . 23
3.8 ハードウェアの構成 . . . 26
3.8.1 透かしデータ転送ユニット . . . 26
3.8.2 S 演算ユニット，F 演算ユニット . . . 27
3.8.3 比較演算ユニット . . . 29
3.9 全体の構成 . . . 31
3.9.1 1 つの透かし検出回路 . . . 31
3.9.2 全体の回路構成 . . . 32
3.9.3 データ処理と回路全体の流れ . . . 33
3.10 まとめ . . . 34

第 4 章 評価 **35**
4.1 はじめに . . . 35
4.2 評価環境 . . . 35
4.3 検出時間評価 . . . 36
4.4 ファイルサイズの増加による検出時間の推移 . . . 39
4.5 検出回路量とクリティカルパス . . . 39
4.6 まとめ . . . 40

第 5 章 不正配布監視システムへの応用 **41**
5.1 はじめに . . . 41
5.2 概要 . . . 41
5.3 実用性 . . . 43
5.3.1 検出速度 . . . 43
5.3.2 検出可能な透かし数 . . . 43
5.3.3 暗号化された転送 . . . 44
5.4 まとめ . . . 44

第 6 章 結論 **46**
6.1 まとめ . . . 46
6.2 今後の課題 . . . 46

第1章　序論

1.1　研究の背景と目的

近年，高速ネットワークの普及に伴い，さまざまなデジタルコンテンツがネットワークを介して配信されている．デジタルコンテンツの場合，品質を劣化させることなく簡単にコピーができるため，コンテンツのコピーが不正に利用されたり，不正に配信されることが問題となっている．そのため，デジタルコンテンツの著作権の主張と保護，不正コピーの防止が必要であり，その一手法として電子透かし技術が研究されてる [1][2]．電子透かし技術とは，人に気づかれないようにIDや著作者名など，著作者を特定できる情報をデジタルコンテンツに埋め込む技術であり，静止画，動画，音，ドキュメントなど，それぞれのデジタルコンテンツに対応した電子透かし技術が提案されている．この技術を用いることで，デジタルコンテンツの著作権の主張と保護が可能となる．現在では，不正コピーを防ぐコピーコントロールやコンテンツの改ざん検知などにも電子透かしが応用されている．さらに，ネットワークで問題となっているオーディオコンテンツの不正配布に対し，電子透かしが有用であることを示すために，JASRACとRIAJは電子透かしを埋め込んだオーディオファイルの実用試験を行った [3]．この実験は，現在実用されているオーディオファイル検索ロボットプログラムに，電子透かし検出プログラムを組み込むことで，ネットワーク上に意図的に公開した電子透かしの埋め込まれたオーディオファイルを探し出すというものである．この実験により，公開されたオーディオファイルをすべて発見し，公開していたサイトの情報を収集することが可能であることを示した．

しかしながら，この方法では，オーディオコンテンツの不正配布の抑制は可能であるが，不正配布を未然に防ぐことはできない．そこで，ネットワーク中で交換されるオーディオコンテンツから電子透かしを高速に検出し，転送が許可されていない透かしの入ったオーディオコンテンツの転送は中断するという方法が効果的であると考える．しかしこの方法では，ネットワークのボトルネックとならない高速な電子透かしの検出が必要となる．また，ネットワーク上では複数のデジタルコンテンツが交換されるため，特定の透かしだけではなく複数の透かし検出を高速に試みなければならない．現在，電子透かし技術は検出速度に着目した研究があまり行われていないため，ソフトウェアでの高速な透かしの検出は非常に困難であり，複数の透かしを検出するにはさらに多くの時間が必要となってしまう．クラスタなどの並列計算機やスーパーコンピュータを利用することで高速に複数の透かし検出が可能となるが，膨大なコストがかかり実用的でない．

そこで，高速に複数の透かしを検出するためには，ハードウェアを利用した透かし検出

が非常に有効であると考える．ハードウェアで透かし検出のための専用回路を構築することで高速な検出が可能となり，ソフトウェアで実現が困難な複数の透かしの高速検出も，検出回路を並列に構築することで可能となる．また，開発時間の短縮や低コスト化のために，FPGA(Field Programmable Gate Array) を用いてハードウェア回路を構築することが有用である．FPGA は，利用者が自由に回路を構築することが可能で，検出する透かしの変更や高速化のための回路の再構成など利用状況にあった自由な回路構築が可能となる．また，並列計算機やスーパーコンピュータを利用した場合と比べコストを 10 分の 1 以下に抑えることができ実用的なものとなる．

本研究では，オーディオファイルから高速に複数の電子すかしを検出する検出回路を FPGA 上に構築する手法を提案する．ハードウェアはさまざまな特性や制約を持つため，これらを考慮し高速に複数の透かし検出が可能なアルゴリズムの検討を行う．そして，そのアルゴリズムにおける高速な回路構築手法と，複数の透かし検出のための検出回路の並列化手法について提案する．さらに，提案したハードウェアの性能評価を行い，提案したハードウェアを用いた不正配布を監視する手法を提案する．

1.2 本文の構成

本論文は，以下の構成からなる．第 2 章では，オーディオ電子透かしの概要について説明し，本研究で利用するオーディオ電子透かしのアルゴリズムについて説明する．第 3 章では，利用するオーディオ電子透かしの検出アルゴリズムをハードウェア化する手法を考察し，実際の FPGA ボード上に実装する方法を示す．第 4 章では，実装したハードウェアの性能を検出時間と回路量の点で評価を行う．第 5 章では，提案したハードウェアを利用し，ネットワーク上を流れる不正配布を防ぐ方法を提案する．第 6 章で，全体のまとめを行い提案ハードウェアの問題点と今後の予定を議論する．

第2章　オーディオ電子透かし

2.1　はじめに

オーディオ電子透かしとは，著作者固有の透かしデータをオーディオデータの中に埋め込む技術である．その埋め込み手法は，耳の生理学的特性を利用して透かしデータを埋め込む手法が一般的である．耳は，特定の周波数の前後の小さな音は聞こえなくなることや，大きな音の前後では小さな音が聞こえなくなるなどの生理学的特性を持つ．この特性などを利用して，音の周波数領域に透かしを埋め込む方法 [4][5][6] と，時間領域へ透かしを埋め込む方法 [7][8][9] がある．現在提案されている多くのオーディオ電子透かしアルゴリズムは，透かしの強度や聞こえにくさに着目しているため高速な透かし検出が非常に困難である．

本研究では，アルゴリズムをハードウェア化することで電子透かしの検出を高速に行う検出回路を構築する．また，検出回路を並列化することで複数の透かしの高速検出を目指す．しかし，すべてのオーディオ電子透かしアルゴリズムがハードウェア化に適しているわけではない．ハードウェアではさまざまな特性を持つため，その特性を考慮しハードウェア化に適したオーディオ電子透かしアルゴリズムについてを検討する必要がある．

本章では，まずオーディオ電子透かしの代表的な手法である周波数領域を利用した手法と，時間領域を利用した手法についての概要を説明しその特徴を明らかにする．次に，ハードウェア化での考慮すべきことを明らかにし，ハードウェア化に適したアルゴリズムを検討する．最後に，本研究で利用するオーディオ電子透かしアルゴリズムを明確に説明する．

2.2　オーディオ電子透かし

2.2.1　周波数領域を利用したオーディオ電子透かし

周波数領域を利用したオーディオ電子透かしは，現在では一般的な手法でありさまざまなアルゴリズムが提案されている．これらの手法の一つに，周波数マスキングを利用する方法がある [4][5]．周波数マスキングとは，2 種類の音が同時に入力されたとき，その周波数が非常に接近しているときに，大きい音が微弱な音を吸収して聞こえなくしてしまうという現象である．この現象を利用することで，雑音となってしまう透かしデータを聞こえなくすることができる．文献 [4] では，透かしデータを周波数マスキングにより聞こえな

いデータを生成し，オーディオデータに埋め込むことを行っている．

また，DWT(Discrete Wavelet Transform) とパッチワーク法を利用したアルゴリズムも提案されている [6]．パッチワーク法とは，まずある領域内で任意の 2 点のサンプルの信号レベルの差分を領域内すべてで求める．この差分を $s(i)$ とすると式 (2.1) となる．

$$s(i) = a(i) - b(i) \tag{2.1}$$

$s(i)$ の総和を $S(i)$ とすると式 (2.2) となる．

$$S(i) = (a(1) - b(1)) + (a(2) - b(2)) + \cdots + (a(i) - b(i)) \tag{2.2}$$

$S(i)$ はおよそ 0 となることがわかっている．そこで，任意の 2 点のどちらかに δ を加え，もう一方は δ を引くことで式 (2.2) は式 (2.3) となる．

$$S = \sum_{i=1}^{N}(a(i) + \delta) - (b(i) - \delta)) = \sum_{i=1}^{N}\{(a(i) - b(i)) + 2\delta\} \tag{2.3}$$

つまり，この 2δ が透かしデータの 1 ビットと対応するため，著作者を識別するデータを埋め込むには領域をいくつも分割することで可能となる．ここで示したパッチワーク法は，周波数領域だけでなく時間領域に対しても利用可能であるが，周波数領域を利用することが望ましいとされている．文献 [6] で提案されているアルゴリズムは DWT の変換を行ったデータに対してパッチワーク法を適用している．ここでは，PentiumⅢ，866MHz CPU，メモリが 512MB の PC を利用し，2.77MByte のファイルから透かしを検出する時間を計測した結果は 2.9s であった．つまり，0.95Mbps の検出速度である．現在の CPU 性能で換算すると，約 4Mbps 程度の検出速度であると推測される．

このように，一般的な周波数領域を利用した電子透かしアルゴリズムは，オーディオデータを周波数領域へ変化し，そのデータに対して透かしデータを埋め込む．透かしデータは周波数マスキングなどで聞こえなくして埋め込まれる．透かしの検出時も，周波数領域へ変換が必要となり，比較的複雑な処理となってしまう．しかし，時間領域を利用した場合に比べ，透かしデータは聞こえにくく圧縮などの操作が行われても消えにくいという特徴を持つ．オーディオ電子透かしの研究は，現在のところ透かしの聞こえにくさや，透かしの強さに着目して研究が行われているため，周波数領域を利用するオーディオ電子透かしが一般的な手法である．

2.2.2 時間領域を利用したオーディオ電子透かし

時間領域を利用したオーディオ電子透かしは，比較的容易な手法であり計算資源を多く利用しない手法である．これらの手法には，エコーを利用したアルゴリズムがある [9]．この手法は，原音に対してわずかなエコーをかけ，その長さを透かしデータとして利用する手法である．エコーの長さがあまり長くなりすぎると聴覚で判別できてしまうため，エコーの長さの調節が重要なポイントとなる．

また，時間軸マスキングを利用した手法もある [7]．時間軸マスキングとは，ある大きな音の前後の小さな音は聞こえにくいという生理学的特長を利用した手法である．具体的には，大きな音の 5-20ms 前の小さな音は聞こえなくなり，逆に大きな音の 50-200ms 後の小さな音も聞こえなくなる現象を利用したものである．この手法により透かしデータを聞こえない音に変化し透かしの埋め込みが行われる．

時間領域を利用した場合でも，周波数マスキングに似た手法がある．それは，SN 比を用いた手法である [7]．この手法は，透かしデータを雑音と捉え，元のデータとの SN 比が閾値以下になるように透かしデータを調整し，データを埋め込む手法である．周波数マスキングでは，周波数が接近した音の場合，小さな音は大きな音に吸収されるという特徴を利用したが，この手法では，周波数の接近は考えず単純に小さな音は大きな音に吸収される特徴を利用した手法である．

これらの方法は，いずれも透かしデータを時間軸マスキングなどで聞こえない音に変換してオーディオデータにそのまま埋め込みを行う．そのため，周波数領域を利用した場合と違い，フーリエ変換などを必要としないため計算は比較的簡単になる．しかし，透かしデータの聞こえにくさや圧縮などの操作に比較的弱いという特徴を持つ．

2.2.3 検出方法

周波数領域を利用した場合も，時間領域を利用した場合でも検出時に元のデータを参照する方法とそうでない方法がある．

検出時に元のデータを参照する方法 [4] では，その著作権を強く主張することが可能である．しかし，元のデータと透かしの入ったデータを必要とするためメモリが多く必要となり，さらに通信帯域が大きくなってしまうという欠点を持つ．そのため，検出が頻繁になるとあまり有用な方法ではなくなる．

逆に，検出時に元のデータを必要としない方法 [5] は，著作権の主張は元のデータを利用した場合と比べ強力ではないが，元のデータを必要としないため，多くのメモリを必要とせず，通信帯域も比較的小さくできる．そのため，検出が頻繁になると有用な方法である．

また，元のデータを必要としない方法でも透かしデータを参照する手法も多く提案されている．透かしデータを参照すると著作権の主張は強く主張でき，さらに必要なメモリや通信帯域も小さくできるという利点がある．一般的には，この手法を用いた検出が多く提案されている．

2.3 ハードウェア化に適したアルゴリズムの検討

2.3.1 ハードウェア化の考慮点

本研究では，オーディオ電子透かしアルゴリズムをハードウェア化することで高速に複数の透かし検出を目指す．しかし，ハードウェアではソフトウェアでは考慮しないようなさまざまな制約や特性があるため，すべてのアルゴリズムがハードウェア化に適しているわけではない．そこで，ハードウェア化で考慮する点について明らかにし，ハードウェアに適しているアルゴリズムを考察する．

ハードウェアでは，演算回路の効率化や並列化を行うことで高速な処理が可能となるため，演算回路の効率化や並列化が見込めるアルゴリズムがハードウェア化に適していると考える．また，浮動小数点演算や乗除算の演算回路は演算に数サイクル必要となるため，整数演算や加減算の演算回路と比べ演算速度が遅くなることが知られている．そのため，浮動小数点演算や乗除算を多用しないか，別の演算で処理ができるアルゴリズムが望ましいとされている．

本研究では開発時間の短縮と低コスト化のため，FPGA を用いた回路構築を行う．FPGA を用いることで自由な回路構成が可能となるという利点を持つが，利用できる回路量や利用可能なメモリ容量に制限が有りこの制限を無視することができない．浮動小数点演算や乗除算の演算回路は比較的回路量が多くなるため，回路量の制限がある FPGA ではこれらの演算を多用しないことが望ましい．本研究では 1 つの透かしを検出する回路を並列に構築することで，複数の透かしの高速検出を目指す．つまり，多くの透かし検出に対応するためには，1 つの検出回路をなるべく小さくすることが必要となる．さらに，利用可能なメモリ量は現在利用されている PC のメモリ量と比べると非常に小さいため，あまりメモリを消費しないことが求められる．

これらの特徴を考慮すると次のようなアルゴリズムがハードウェア化に適していると本研究では考える．

- 並列化や処理の効率化が見込めるアルゴリズム
- 浮動小数点演算や乗算除算を多用しないアルゴリズム
- 利用するメモリが小さいアルゴリズム

2.3.2 アルゴリズムの検討

本研究では，オーディオ電子透かしの検出アルゴリズムをハードウェア化することで，高速に複数の透かし検出を目指す．そこで，ハードウェア化に適したアルゴリズムを検討し，そのアルゴリズムをハードウェア化する．

まず，周波数領域を利用したアルゴリズムと，時間領域を利用したアルゴリズムのどちらがハードウェアに適しているかを検討する．周波数領域を利用したアルゴリズムの場

合，一度データをフーリエ変換しなければならない．フーリエ変換では，浮動小数点演算や乗除算を多用するため比較的回路量が多くなり演算速度も遅くなる．一方，時間領域を利用した場合，透かしデータを聞こえなくするための変換は必要だが，フーリエ変換をしないという点から周波数領域を利用した場合より比較的回路量が小さくり高速な演算が期待できる．そこで，本研究では浮動小数点演算や乗除算演算を多用しないという点から，時間領域を利用した電子透かしアルゴリズムがハードウェア化に適していると考える．透かしの聞こえにくさや透かしの強度は，時間領域を利用するより周波数領域を利用するほうがよいが，本研究では透かしの聞こえにくさや強度に着目せず高速な透かし検出を目指すため，より処理の簡単な時間領域を利用したアルゴリズムがハードウェア化に適していると考える．

次に，検出時には参照するデータについて検討する．本研究では，透かし検出回路を並列化することで複数の透かし検出に対応する．しかし，検出時には元のデータか，透かしデータのどちらかを参照する必要がある．検出時に元のデータを参照する場合，オーディオファイルをいくつも保有する必要があり，非常に大きなメモリが必要となる．ハードウェアでは，非常に小さなメモリしか利用することができないため，元のデータを参照する方法はハードウェア化に適していない．これに対して，検出時に透かしデータを必要とする場合，1 つの透かしデータは著作者を特定するためのデータなのでたかだか数十キロビット程度である．元のデータと透かしデータを比べた場合，はるかに透かしデータのほうが小さい．よって，ハードウェアのメモリ制限を考慮し多くの透かしを検出するためには，検出時に透かしデータ利用するアルゴリズムがハードウェア化に適していると考える．

以上の検討事項から，本研究では時間領域を利用し検出時に透かしデータを利用するオーディオ電子透かしアルゴリズムがハードウェア化に適しているとする．本研究では，さまざまなオーディオ電子透かしアルゴリズムの中から，[7] で提案されているアルゴリズムを採用した．本研究では透かしの強度や透かしの聞こえにくさに着目せず，ハードウェアにより高速な検出を行うためアルゴリズムに改良を加えハードウェア実装する．

2.4 wav ファイルフォーマット

現在，さまざま形式のオーディオファイルが存在が，最も基本となるオーディオファイルは，非圧縮の PCM 方式で録音されたものである．これは，CD と同じ録音方式である．本研究では，最も基本となる PCM 方式のオーディオファイルに対応した電子透かしを対象とする．

wav ファイルは Windows 標準の音楽, 音声フォーマットであり，圧縮，非圧縮の両方に対応したファイル形式である．本研究では，非圧縮の PCM 方式で録音された wav ファイルを利用する．利用する wav ファイルフォーマットを表 2.1 に示す．

ビット数は 1 サンプルを示すビット幅であり，チャンネル数はステレオかモノラルかを示す．サンプリングレートが 44.1kHz とは 1 秒間に 44100 個のサンプルがあることを示す．

表 2.1: wav ファイルフォーマット

フォーマット ID	PCM の ID
ビット数	16bit
チャンネル数	2(ステレオ)
サンプリングレート	44.1kHz

次に，wav ファイルの構造について説明する．wav ファイルは，録音形式やサンプリングレートなどファイルのフォーマットについて記述された部分と，実際の音声データの部分に分かれている．チャンネル数が 2 の場合，音声データ部分は，左チャンネルと右チャンネルのサンプルが交互に記録されている．

2.5 採用アルゴリズム

2.5.1 埋め込みアルゴリズム

本研究では，電子透かしの埋め込みは，オーディオデータの左チャンネルのサンプルだけに埋め込むとする．まず，オーディオデータの左チャンネルをNサンプルづつセグメントに区切り，それぞれのセグメントで式 (2.6) を計算することで透かしデータを埋め込みとなる．

$$y(i) = x(i) + f(x(i), w(i)) \tag{2.4}$$

$x(i)$ は，セグメントの先頭から i 番目の透かしを埋め込む前のサンプルとする．同様に，$y(i)$ は，セグメントの先頭から i 番目の透かしが埋め込まれたサンプルとする．また，先頭から i 番目の透かしデータを $w(i)$ とし，透かしデータの要素は N 個とする．透かしデータは，各セグメントで同じデータを利用する．式 (2.4) の $f(x(i), w(i))$ は，$w(i)$ のデータを聞こえない音に変換する関数である．この $f(x(i), w(i))$ の定義により，透かしデータの聞こえにくさや強さが決定される．本研究では，透かしデータの聞こえにくさや強さに着目していないため，$f(x(i), w(i))$ を式 (2.5) とする．

$$f(x(i), w(i)) = \alpha|x(i)|w(i) \tag{2.5}$$

式 (2.4)，(2.5) より式 (2.6) が導き出される．

$$y(i) = x(i) + \alpha|x(i)|w(i) \tag{2.6}$$

本研究では，$\alpha = 1/64$，$N = 44100$ とする．すべてのセグメントで式 (2.6) を計算することで透かしの埋め込みとなる．しかし，すべてのオーディオデータがセグメントに分けられるわけではなく，最後の部分は端数となってしまう．この端数となる部分は，透かしの埋め込みは行われず，元のデータがそのまま利用される．

2.5.2 透かしデータ

本研究で利用する透かしデータは，著作者固有の乱数配列とし，要素の値は-1 か+1 である (2.7)．また，透かしデータの要素の合計値を Δw とし，Δw は 0 でないとする (2.8)．つまり，透かしデータの-1 と 1 の数が等しくないとする．

$$w(i) = \begin{cases} +1 \\ -1 \end{cases} \tag{2.7}$$

$$\Delta w = \sum_{i=0}^{N} w(i) \neq 0 \tag{2.8}$$

本研究では，M 系列 [10] を利用して生成された乱数配列を利用し透かしデータとした．M 系列は，1 と 0 の乱数配列であり 1 と 0 の出現確立は等しいため，透かしデータとしてそのまま利用できない．そこで，要素 0 を-1 とし，長さが m の M 系列の先頭から N 番目までを透かしデータとして利用する ($m > N, N = 44100$ とする)．M 系列の一部分を取り出すことで，固有の乱数配列ができ $\Delta w \neq 0$ となる．これにより，式 (2.7)，(2.8) を満たす透かしデータができる．

本研究では，透かしの生成は研究対象ではないため，単純な乱数生成の M 系列を利用した．透かしデータは，すでに生成されているデータとして利用する．そのため，透かしデータは別の方法で作成されたものでも利用可能である．

2.5.3 検出アルゴリズム

検出には，透かしの埋め込まれたオーディオデータと検出したい透かしデータを利用する．透かしが埋め込まれたオーディオデータを N サンプルずつセグメントにわけ，セグメントごとに計算を行う．まず，S を次のように定義する．

$$S = \sum_{i=1}^{N} y(i)w(i) \tag{2.9}$$

式 (2.4)，(2.9) より，式 (2.10) を得る．

$$S = \sum_{i=1}^{N} [x(i)w(i) + f(x(i), w(i))] \tag{2.10}$$

ここで，$\Delta w \neq 0$ であるため，次のような式が得られる．

$$S = \sum_{i=1}^{N-\Delta w} x(i)w(i) + \sum_{i=1}^{\Delta w} x(i)w(i) + \sum_{i=1}^{N} f(x(i), w(i))w(i) \tag{2.11}$$

Δw は透かしの要素である-1 と+1 の偏りを示しているため，式 (2.11) の第 1 項は透かしデータの-1 と+1 の数が等しくなるためおよそ 0 となる．もし，透かしが入ってない場合は，S の値は式 (2.11) の第 2 項の値とおよそ等しくなり，S は $\frac{\Delta w}{N}\sum_{i=1}^{N} x(i)w(i)$ にほぼ等しくなる．透かしが入っている場合，S の値は式 (2.11) の第 2 項と第 3 項の合計値とおよそ等しくなり，S は $\frac{\Delta w}{N}\sum_{i=1}^{N} x(i)w(i) - \sum_{i=1}^{N}(f(x(i), w(i)))w(i)$ にほぼ等しくなる．検出時には，元のデータである $x(i)$ の代わりに $y(i)$ を利用するが，あまり大きな誤差は生じない．そのため，式 (2.11) の第 2 項は $\sum_{i=1}^{\Delta w} y(i)w(i)$ と置き換えることができ，ほぼ $\frac{\Delta w}{N}S$ の値と等しくなる．そして，S と $\frac{\Delta w}{N}|S|$ の差分は，ほぼ $\sum_{i=1}^{N}\alpha|y(i)|w(i)$ となる．ここで r を式 (2.12) で与える．

$$r \triangleq \frac{S - \frac{\Delta w}{N}|S|}{\sum_{i=1}^{N}(f(x(i), w(i)))w(i)} \tag{2.12}$$

ここで，式 (2.5) を式 (2.12) に代入すると，

$$r \triangleq \frac{S - \frac{\Delta w}{N}|S|}{\sum_{i=1}^{N}(\alpha|y(i)|w(i))w(i)} \tag{2.13}$$

という式になる．しかし，分母は $\sum_{i=1}^{N}\alpha|y(i)|w(i)^2$ となり，$w(i)$ は-1 か+1 であるため式 (2.14) となる．

$$r \triangleq \frac{S - \frac{\Delta w}{N}|S|}{\alpha\sum_{i=1}^{N}|y(i)|} \tag{2.14}$$

もし，検出したい透かしデータが存在する場合， r の値はほぼ 1 になる．また，検出したい透かしデータが存在しない場合，r の値はほぼ 0 になる．検出時には $y(i)$ を利用したことによる誤差があるため，それぞれのセグメントで得られた r の平均値を検出値とし，検出値が 0.5 以上でその透かしデータが検出できたことにする．

次節で，本研究で利用するアルゴリズムの検出精度と検出速度がどの程度かを見積もるために実装したソフトウェアについて説明する．

2.6 ソフトウェア実装による予備実験

2.6.1 埋め込みフローチャート

図 2.1 は埋め込みフローチャートである．はじめに埋め込みたい透かしデータを生成する．この透かしデータは，M 系列を利用して発生させた乱数配列をメモリ上に保存する．M 系列では 0 と 1 の乱数配列を生成するため，0 を-1 の値に変更しメモリ上に保存する．透かしデータは 44100 個の要素とする．

透かしの生成が終わると，wav ファイルの形式を判別する．本研究で利用するファイル形式以外の場合は，プロセスを終了する．ファイル形式が合っている場合，左チャンネルのデータを抽出しメモリ上に保存する．メモリ上には，44100 の整数倍のサンプルを保存し，端数となる部分はメモリ上に保存しない．

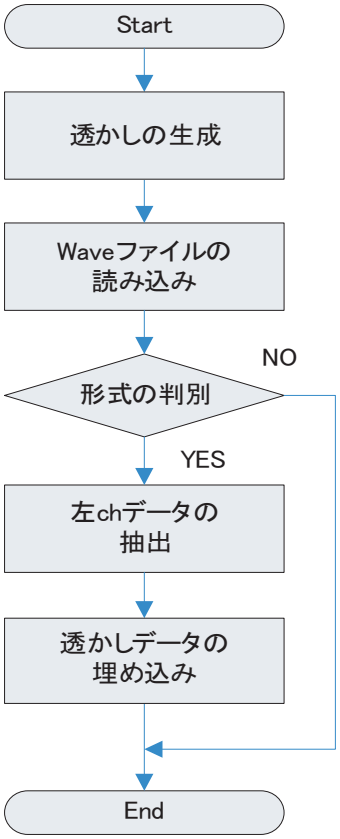

図 2.1: 埋め込みフローチャート

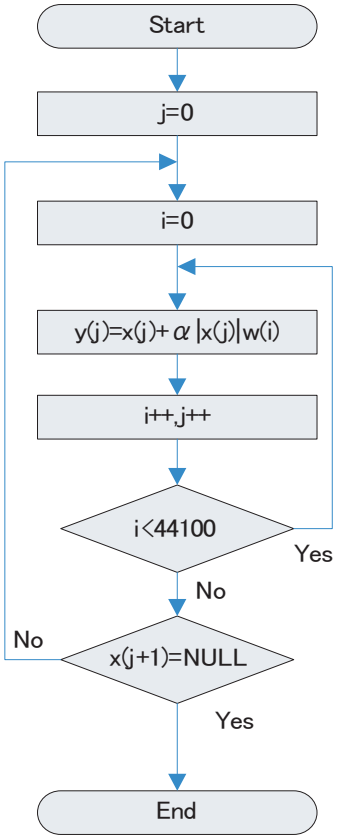

図 2.2: 透かし埋め込み

透かしの埋め込み処理は図 2.2 に示す．セグメントごとに式 (2.6) の式を計算し透かしを埋め込む．j は先頭からの位置で，i はセグメントの先頭からの位置を示す．各セグメントは 44100 サンプルであるため，ループを 44100 回行う．メモリ上には，セグメントの倍数のサンプルしか保存されていないため，各セグメントでの処理が終わったとき，次のセグメントのはじめのデータが存在するかを終了条件とする．

2.6.2 検出フローチャート

図 2.3 は検出フローチャートである．検出時も，埋め込み時と同様に，透かしの生成，ファイル形式の判別，左チャンネルの抽出を行い，透かしの検出を行なう．左チャンネルのデータは，44100 の倍数のサンプルをメモリに保存する．透かしの生成時に，Δw を計算しておく．

透かしの検出処理は，図 2.4 で示す．処理内容として，式 (2.14) を行う． j は先頭からの位置で，i はセグメントの先頭からの位置を示す．各セグメントで得られた r の平均値を計算するために，R に r を加算する．全セグメントの処理が終わったら，R を全セグメ

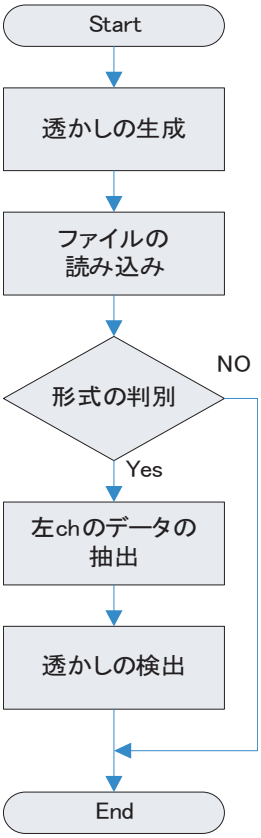

図 2.3: 検出フローチャート

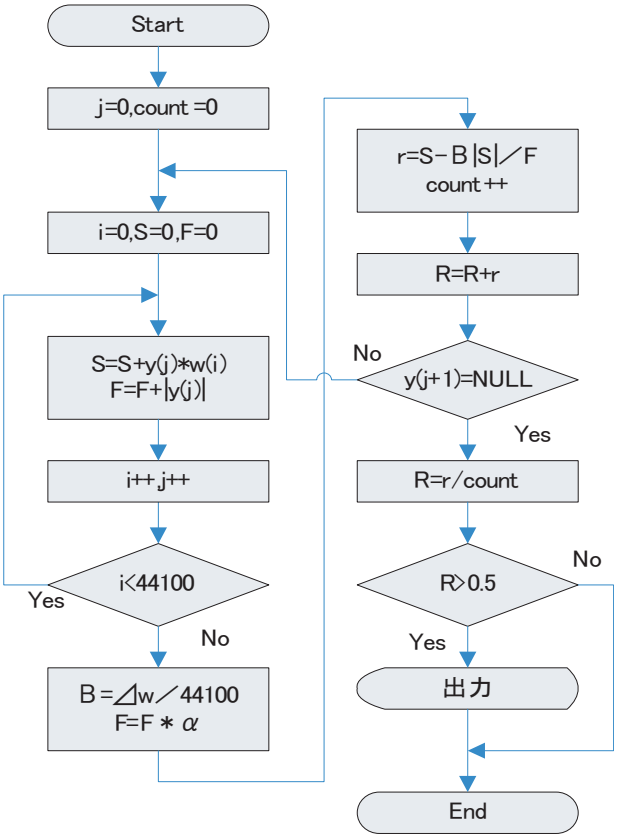

図 2.4: 透かしの検出

ント数で割ることで平均値を求める．全セグメント数は $count$ という変数で数える．平均値が 0.5 以上で透かしの検出とし，結果を表示する．

2.6.3 検出速度と検出精度

検出速度　本研究では，複数の透かしを高速に検出することが目的である．そこで，ソフトウェアにより複数の透かしを検出する場合の検出速度を計測する．今回は，25 個の透かしを検出する場合と，50 個の透かしを検出する場合を計測し，その結果を表 2.2 に示す．透かしの埋め込まれたファイルは，左チャンネルが 20.4MByte のファイルを利用した．

表 2.2: 検出速度

	検出時間	検出速度
25 個の透かし検出	5807ms	28.10Mbps
50 個の透かし検出	11341ms	14.39Mbps

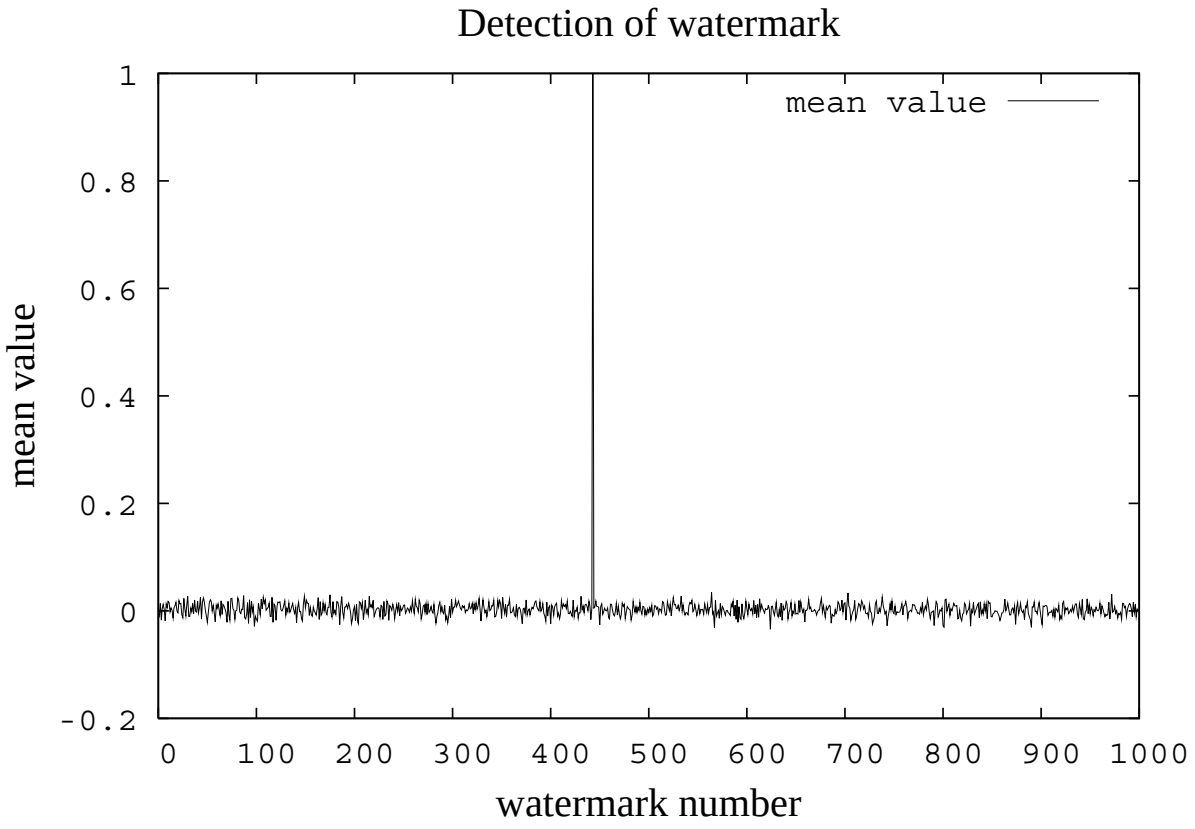

図 2.5: 検出精度

表 2.2 からわかるように，ソフトウェアにより複数の透かしを検出する場合，検出時間は透かしの数に比例して増加する．そのため，高速に複数の透かしを検出することは，ソフトウェアでは非常に困難である．

検出精度　本研究で利用するアルゴリズムの検出精度について評価する．複数の透かしデータを準備し，その中の 1 つの透かしデータが埋め込まれたファイルを準備する．そして，準備したすべての透かしデータで，透かしの検出を試み，誤検出の有無をチェックすることで透かしの精度を示す．

図 2.5 では，1000 個の透かしデータとその中の 444 番の透かしデータが埋め込まれたファイルを利用し，検出を試みた時の r の平均値を示したグラフである．グラフでは，0.5 以上の平均値は透かしデータ 444 番だけである．よって，ほぼ誤検出することなく透かしを検出できる．

2.7 まとめ

本章では，まず一般的なオーディオ電子透かしの概要を説明した．次に，これらのアルゴリズムの中でハードウェア化に適したアルゴリズムの検討を行った．本論文では，FPGAを用いてハードウェア化するため，浮動小数点演算や乗除算を多用せず，回路量やメモリ量が小さくできるアルゴリズムがハードウェア化に適していると判断した．この検討結果より，本論文では時間領域を利用し透かしの検出時に透かしデータを参照するアルゴリズムがハードウェア化に適していると考え，[7] で提案されているアルゴリズムを改良して利用する．最後に，採用したアルゴリズムを明確にしそのアルゴリズムをソフトウェアで実装を行った．採用したアルゴリズムは，ほぼ誤検出することなく検出が可能であるが，複数の透かしを検出を行うと透かしの数に比例して検出時間が増加することを示した．

第3章　透かし検出アルゴリズムのハードウェア化

3.1　はじめに

前章では，ハードウェア化に適したオーディオ電子透かしアルゴリズムについて検討を行い，本研究で利用するオーディオ電子透かしアルゴリズムを決定した．本研究では，ハードウェアにより電子透かしを検出することで，複数の透かしを高速に検出を行う．

本章では，前章で示した透かし検出アルゴリズムをハードウェア化するための方針を示し，開発過程で生じた問題点を解決する方法を示す．さらに，電子透かし検出ハードウェアの構築を示す．

3.2　実装環境

本研究で利用する実装環境を表3.1に示す．

表 3.1: 実装環境

FPGA ボード	celoxica 製　RC2000
搭載 FPGA チップ	Xilinx 製　xc2v8000-4×1
ホストと FPGA ボード間通信	PCI バス
ホスト CPU	Pentium4 2.8GHz
ホストメモリ	1 GB
回路開発環境	DK-version3
回路記述言語	Handel-C

本研究では，開発の短縮のため，FPGA ボードとして celoxica 製の RC2000[11] を利用する．FPGA ボードとは，FPGA とホストコンピュータをつなぐためのデバイスや，外部メモリなどが用意されているものである．RC2000 には，Xilinx 製 xc2v8000-4[12] と，SRAM が 4MB×6 バンク搭載されている．また，回路開発環境は DK-version3[13] を利用し，回路記述言語は Handel-C[14] を利用した．これは，RC2000 に対応した API[15] があり，開発期間の短縮のためである．

Procedure 1 ホストコンピュータの処理

Require: オーディオデータ (wav)
Ensure: ホストコンピュータの処理

```
1: if ファイル形式=wav then
2:    左チャンネルのデータ抽出 (抽出データは符号付 16bit)
3:    抽出データの符号無 32bit へ拡張
4:    while 左チャンネルのデータ ≠ EOF do
5:       if 44100 個のサンプルを抽出 then
6:          FPGA ボードへ抽出データ転送 (32bit×44100 サンプル)
7:          転送が正常に終了したことを確認
8:       else
9:          処理結果を FPGA ボードに要求
10:         受信した結果を表示
11:         処理の終了
12:      end if
13:   end while
14: else
15:   処理の終了
16: end if
```

3.3 処理の分担

本研究では，オーディオファイルから電子透かしを検出する場合，すべての処理を FPGA 行わず，ホストコンピュータと FPGA ボードで処理を分担する．それぞれの処理の流れを Procedure1,2 に示す.

ホストコンピュータ

ホストコンピュータでは，電子透かしを検出したいオーディオファイルのファイル形式のチェックを行う．本研究では，wav ファイル形式で PCM で録音されたファイルのみを扱い，それ以外のファイルの場合はその時点で処理を中断する．次に，データ部分から左チャンネルのデータを取り出し，PCI を利用してデータを FPGA ボードへ転送する．データの転送には，RC2000 用の API[15] を利用する．この API では，データは 32bit 単位で転送されるため，オーディオデータの 1 サンプルを 32bit に拡張する．また，1 セグメント単位である 44100 サンプルのデータをまとめて転送する．転送が成功したら次のセグメントのデータを転送する．すべてのセグメントデータを転送したとき，結果をハードウェアに要求する．FPGA ボードで処理された結果を受信し出力する．これらの処理は，C 言語によるプログラムで実装する．

Procedure 2 FPGA ボードの処理

Require: 符号無 32bit データ ×44100 サンプル受信，透かしデータはボード上に保持
Ensure: FPGA ボードの処理

```
1: if ホストからの左チャンネルのデータを受信 then
2:     for all i such that 1 ≤ i ≤ 44100 do
3:         y(i) ⇐ i 番目の受信サンプル
4:         透かしの検出処理 ⇐ y(i)
5:     end for
6: else if ホストから結果の要求 then
7:     検出結果をホストへ転送
8: else
9:     wait
10: end if
```

FPGA ボード

FPGA ボードでは，受信した左チャンネルのデータより透かしの検出を行う．検出したい透かしのデータは予め FPGA ボード上に保持しておき，検出時はそのデータと受信した左チャンネルのデータより検出を行なう．検出された透かしに対応する番号を検出結果とし，ホストコンピュータへ結果を転送する．

3.4 FPGA 内の回路構築方針

FPGA 上では，前章で示された検出アルゴリズムの式 (2.14) を計算し透かしの検出を行なう演算回路を構築する．回路構築は，次の点に着目して回路を構築する．

- 乗除算の削減
- 検出回路の並列化

乗除算の削減は，処理の高速化と回路量の削減のために行う．乗除算回路は，処理に数サイクル必要となるため，乗除算の削減もしくは別の演算で処理することで高速な演算が可能となる．さらに，乗除算回路は加減算回路に比べ回路量が大きい．乗除算を削減もしくは別の演算で処理することで，回路量が小さく高速な演算回路を構築することができる．

検出回路の並列化は，処理の高速化と複数の透かし検出のため行なう．検出アルゴリズムより，並列に演算が可能な部分を検討し，並列に動作させることで高速な処理が可能となる．さらに，透かしを検出する回路を並列に構築することで複数の透かし検出を同時に行うことが可能になる．

3.5 アルゴリズムの考察による乗除算の削減と並列化

回路構築方針に従い，アルゴリズムを考察することで，乗除算の削減と並列化の手法を検討する．

乗除算の削減

検出には，式 (2.9) の S を求め，式 (2.14) を計算する必要がある．これらの式を考察し，乗除算の削減方法を検討する．

$$S = \sum_{i=1}^{N} y(i)w(i) \tag{2.9}$$

$$r \triangleq \frac{S - \frac{\Delta w}{N}|S|}{\alpha \sum_{i=1}^{N} |y(i)|} \tag{2.14}$$

まず，式 (2.9) で $y(i)w(i)$ の乗算を削減する．$w(i)$ は $\{-1|+1\}$ の値であるため，この乗算を条件式と加減算により演算することができる．そこで，$w(i)$ の -1 を 0 に対応付け，$w(i) = \{1|0\}$ として計算をする．これを式で表すと次のような漸化式で表すことができる．

$$S_0 = 0 \tag{3.1}$$

$$S_i = \begin{cases} S_{i-1} + y(i) & \text{if } w(i) = 1 \\ S_{i-1} - y(i) & \text{if } w(i) = 0 \end{cases} \tag{3.2}$$
$$(i = 1, 2, \ldots, 44100)$$

Procedure 3 Calculate $S = \sum_{i=1}^{N} y(i)w(i)$

Require: $w(i) = 1 \vee w(i) = 0$
Ensure: $S = \sum_{i=1}^{N} y(i)w(i)$

```
S ⇐ 0
for all i such that 0 ≤ i ≤ 44100 do
    if i = 0 then
        S ⇐ 0
    else
        if w(i) = 1 then
            S ⇐ S + y(i)
        else
            S ⇐ S − y(i)
        end if
    end if
end for
```

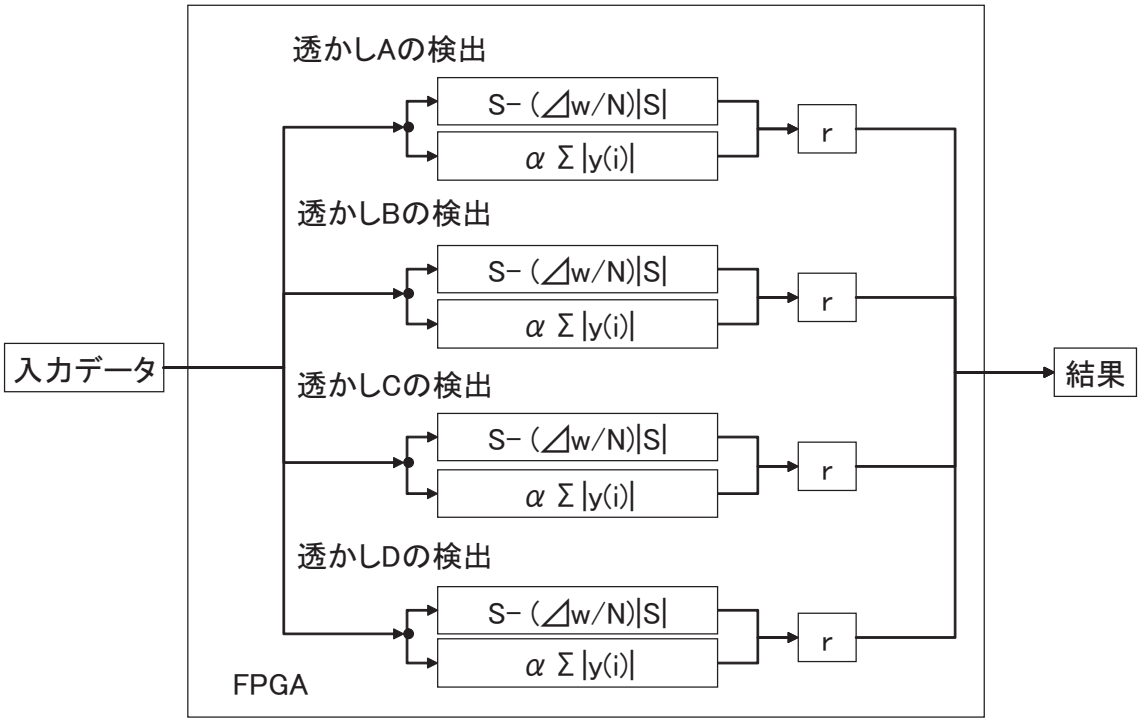

図 3.1: 検出回路の並列化

式 (3.1)，(3.2) より Procedure3 が得られ，これにより式 (2.9) の S を求めることができる．i の値が 44100 になるまで計算することで，S を求めることができる．$w(i)$ の値が 1 のときは加算を行い 0 のときは減算を行うことで，乗算を利用しないで計算が可能となる．

次に，式 (2.14) での α の乗算を削減する．α の値は，透かしを聞こえなくするための定数であり実測に基づき値を定義した．本研究では，透かしの聞こえにくさは考慮していないため，ある程度聞こえなくできる定数を選択した．ハードウェアでは，2^n の乗算はシフト演算で処理が可能であるため，本研究では $\alpha = 1/2^6$ と定義した．これにより，乗算をシフト演算ができ，透かしをある程度聞こえなくすることができる．

また，式 (2.14) より $\Delta w/N$ の除算の削減を行う．$\Delta w/N$ の分母は定数で 44100 という値を持ち，分子は透かしデータ固有の定数である．それぞれ変数でないため，回路構築時に計算しておき結果を与えておくことが可能である．そこで，透かしデータごとに $\Delta w/N$ の計算を行い，その結果を定数として FPGA 上に与えておくことで，除算を削減することができる．

並列化

並列化の考察結果を図 3.1 に示す．まず，高速処理のための並列化について述べる．式 (2.14) の計算では，分子である $S - \frac{\Delta w}{N}|S|$ の計算と，分母の $\alpha \sum_{i=1}^{N} |y(i)|$ の計算が並列に実行可能である．そのため，これらの演算を並列に動作させることで高速な演算が可能に

なる．

次に，複数の透かし検出のための並列化について述べる．利用する検出アルゴリズムでは，1 度の検出処理で 1 つの透かししか検出できない．そこで，1 つの透かしを検出する回路を複数構築し並列に動作させることで，複数の透かしを検出することが出来る．このように，並列に動作させることで，1 つの透かしを検出する時間内に複数の透かしが同時に検出できるため，ソフトウェアでは実現が難しい複数の透かしの同時検出が可能となる．

3.6 複数の透かしデータの保存

3.6.1 問題と解決法

複数の透かし検出のためには，FPGA 上に検出したい透かしのデータを保存しておく必要がある．1 つの透かしデータの要素数は 44100 個存在し，前節で透かしの値を {0|1} としたため，44100bit で透かしデータを表すことができる．m 個の透かしを検出する場合，44100 × $m$bit のデータを FPGA 上に保存しておく必要がある．しかし，FPGA 上にあるメモリは高速なメモリアクセスが可能であるが非常に小さなメモリしか利用することができない．1 つの透かしを検出する場合は，透かしデータを FPGA 上に記憶しておくことが可能であるが，より多くの透かし検出に対応する場合 FPGA 上のメモリでは不足してしまう．

そこで，本研究では利用する RC2000 に搭載されている外部メモリ (SRAM) に透かしデータを保存することでメモリ不足を解消する．SRAM は，4Mbyte×6 バンクあり，1 つの透かしデータは 44100bit であるため，約 4500 個の透かしデータの保存が可能となる．ただし，SRAM から FPGA へデータを転送するためには 3 サイクル必要となり，1 度のアクセスでは 1 アドレス分である 32bit のデータしか転送できないという制限がある．しかし，6 バンクへの同時アクセスと同時転送は可能である．この SRAM の制限を考慮し，より多くの透かし検出回路を構築する．

3.6.2 メモリアクセスによる並列化の制限

SRAM を利用することで，複数の透かしデータを保存しておくことが可能となる．しかし，SRAM 上のデータは一度 FPGA 上に転送しなければ利用可能とならない．SRAM のデータ転送は次のようになっている．

1. 要求するアドレスの指定
2. 遅延
3. 要求したアドレスのデータ受信

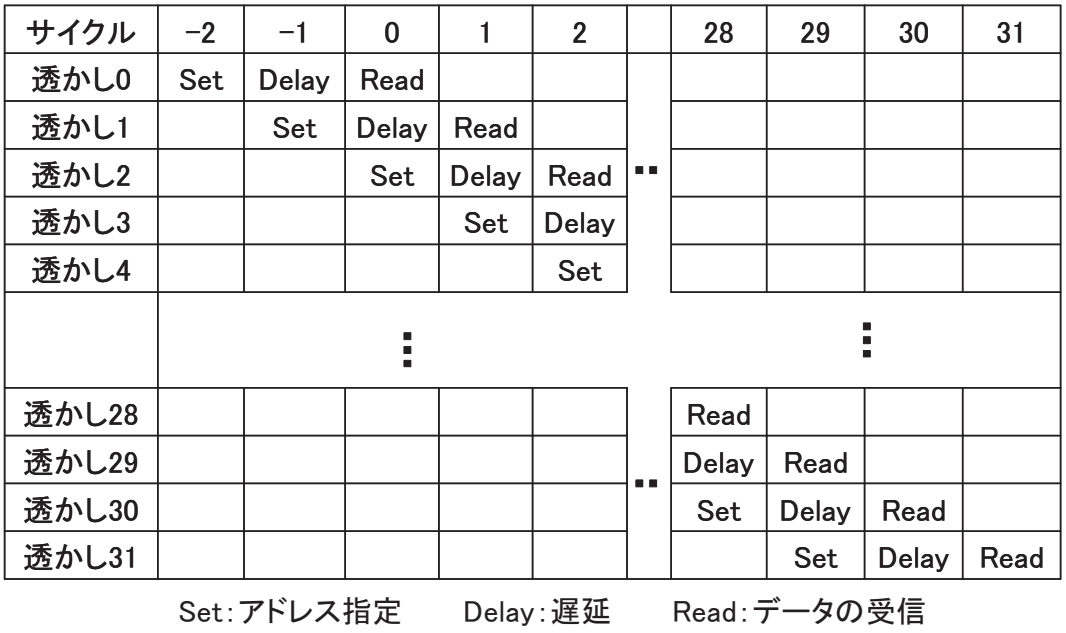

図 3.2: メモリアクセスのスケジューリング

Procedure 4 透かしデータの転送

Require: SRAM を利用

Ensure: 透かしデータ転送

1: **while** 透かしデータの受信要求 **do**
2: **for all** i such that $1 \leq i \leq 32$ **do**
3: 要求アドレスの指定 (address(i))
4: 2 サイクル前に要求したアドレスのデータ受信
5: **end for**
6: **end while**

1 度のデータ受信で 32bit のデータが受信可能である．また，これらの処理は 6 バンク並列に行うことができる．

本研究で利用するアルゴリズムでは，S を計算する場合 1 サイクルで 1 ビットの透かしデータを利用する．そのため，1 度透かしデータをロードした後，32 サイクル以内に次のアドレスのデータをロードしてくる必要がある．これらを考慮した場合，最適なメモリアクセスのスケジューリングは図 3.2 となり流れは Procedure4 となる．よって，1 バンクからは最大 32 個の透かし検出に対応できる．6 バンク並列に実行できるため，回路量の制限を考慮しなければ最大で 32×6 個の透かし検出回路が構築可能である．

3.7 透かし検出法

3.7.1 問題と解決法

本研究で利用する検出アルゴリズム 15 は，式 (2.14) の r を各セグメントで計算し，全セグメントの平均値を求めることで透かしの有無を検出する．この検出手法では r の計算での除算や，平均値計算の除算は削減することができない．これらの除算を含めた回路を構築した場合，回路合成がうまくできなかった．原因として，除算回路による遅延が大きくなり，FPGA の最低動作速度を下回ったためだと考えられる．そのため，これらの除算を削減することが必要不可欠となった．本研究では，これらの除算を削減するために，検出に必要な平均値計算を行わない透かし検出方法を提案する．

3.7.2 平均値を求めない透かしの検出法

本研究のアルゴリズムで得られる r の値の度数分布は図 3.3 に示す．図 3.3 は，透かし w が埋め込まれたオーディオファイルと透かし p が埋め込まれたオーディオファイルの 2 つのファイルから，透かし w の検出をソフトウェアで行った例である．ちなみに，$w \neq p$ であり，図 3.3 の watermark のグラフは透かし w が埋め込まれたファイルで，non watermark のグラフは透かし p が埋め込まれたファイルを示している．

図 3.3 より透かし w が入っているファイルでは，約 80%以上のセグメントで r の値が 0.5 を超えていることがわかる．また，透かし p が入っているファイルでは，r の値が 0.5 以上のセグメント数は 20%以下である．この事実を利用すると，r の値が 0.5 以上のセグメントを数えることで透かしの検出が可能であると考える．さらに，セグメント全体が無音である場合，透かしの有無に関わらず式 (2.14) の分母は 0 で r の値は無限になる．そのため，r の値が無限になるものを数えないように上限を定める．そこで，式 (3.3) の条件を満たすセグメントを数え，その数がある閾値を超えるかどうかでその透かしの有無を判別する．これにより，平均値を求めずに検出が可能となる．

$$0.5 \leq r \leq 2 \tag{3.3}$$

また，r の値を得るためには除算が必要である．この除算を削減する手法について示す．まず，式 (2.14) の分母を $F = \sum_{i=1}^{N} |y(i)|$ とし，式 (3.3) に代入した結果を式 (3.4) とする．

$$\frac{1}{2} \leq \frac{S - \frac{\Delta w}{N}|S|}{\alpha F} \leq 2 \tag{3.4}$$

式 (3.4) を式 (3.5) のように変形することで除算を削減することができる．

$$\frac{1}{2}\alpha F \leq S - \frac{\Delta w}{N}|S| \leq 2\alpha F \tag{3.5}$$

検出には，式 (3.5) の条件を満たすセグメントを数え，閾値と比較することで透かしの有無が判別できる．式 (3.5) では 2^{-1} と 2^{1} の乗算が存在するが，ハードウェアではシフト演算で処理できるため，乗算の増加にはならない．

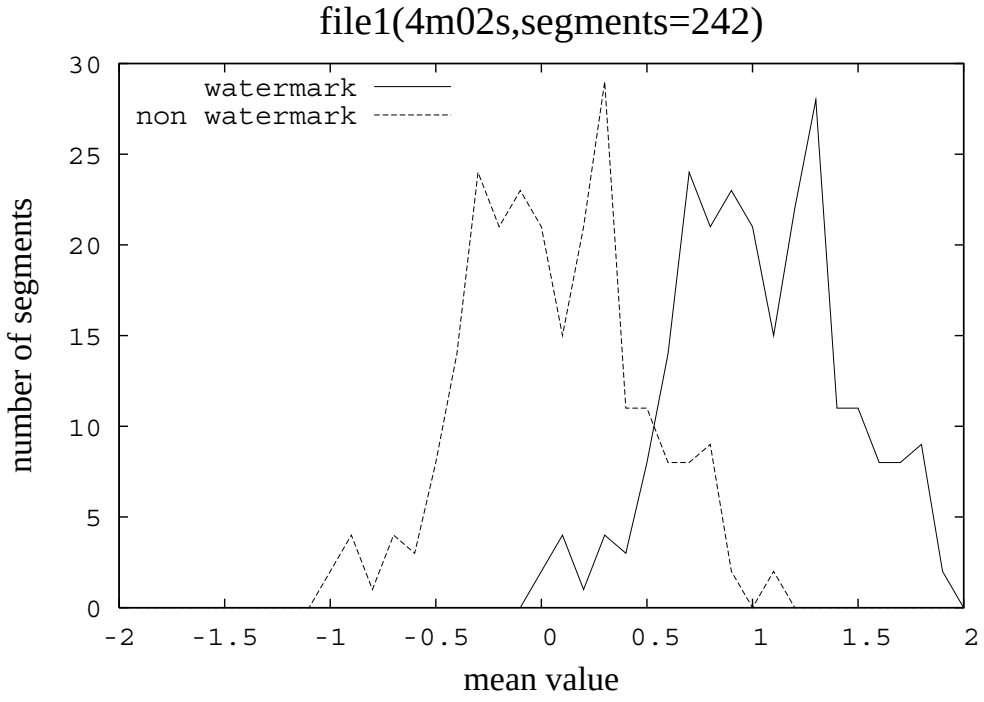

図 3.3: 透かし w の検出時の r の度数分布

3.7.3 閾値の決定

透かし検出に必要な閾値を定める．図 3.3 で示すように，検出したい透かしが入っている場合，全体の 80%以上のセグメントで r の値が 0.5 を超え，透かしが入っていない場合，全体の 20%以下のセグメントしか 0.5 を越えなかった．同様に他のファイルでも検証する．

検証方法として，図 3.3 で利用したオーディオファイルとは別のファイルを 4 つ利用して行う．利用するファイルの曲の長さは，2 分から 6 分程度のファイルを選んだ．これらのファイルを選んだ理由は，著者が保有するオーディオファイルの長さが 2 分から 6 分程度であるためだ．それぞれのファイルより，透かし w を埋め込んだファイルと，透かし p を埋め込んだファイルを生成し，図 3.3 と同様の処理を行った結果を図 3.4-3.7 に示す．

図 3.4-3.7 の結果でも図 3.3 と同様に，透かしが入っている場合，全体の約 80%以上のセグメントで r の値が 0.5 を超え，透かしが入っていない場合，約 20%以下のセグメントで r の値が 0.5 を越えるという結果を得た．そこで，本研究では r の値が 0.5 を超えるセグメントが全体の 50%を超えた場合透かしが存在すると判別する．これは，透かしが入っていない場合， r の値が 0.5 を超えるセグメントは全体の 20%以下であることを利用した．また，閾値を全体の 50%とすることで，全セグメント数を 1 ビットシフトすることで求めることができる．よって，本研究では検出で用いる閾値は全体の 50%と定める．

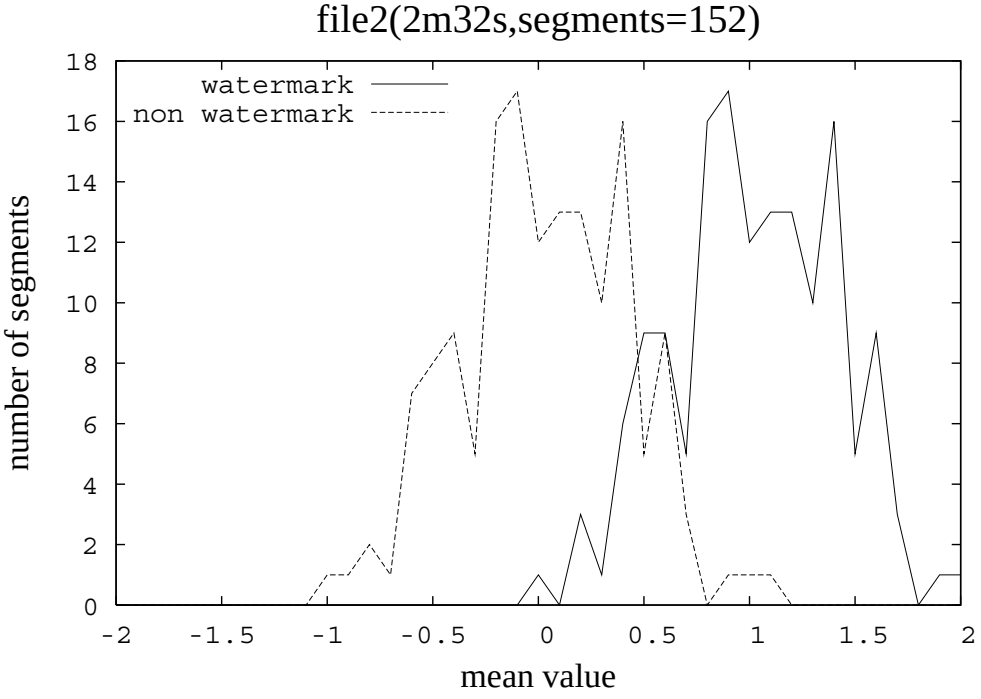

図 3.4: ファイル 2 の r の度数分布

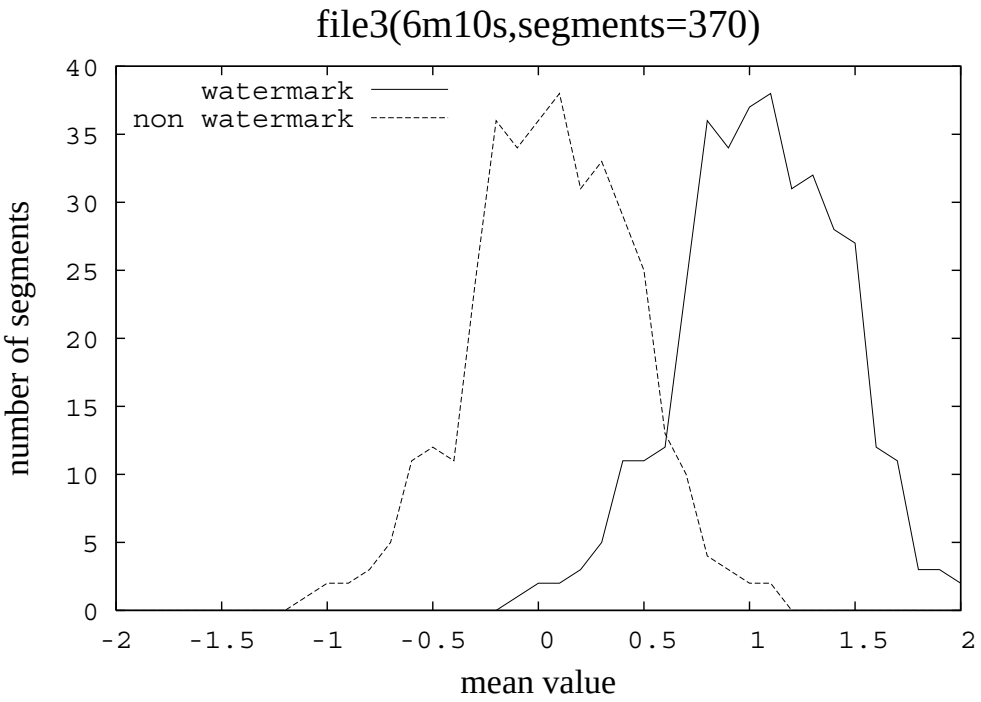

図 3.5: ファイル 3 の r の度数分布

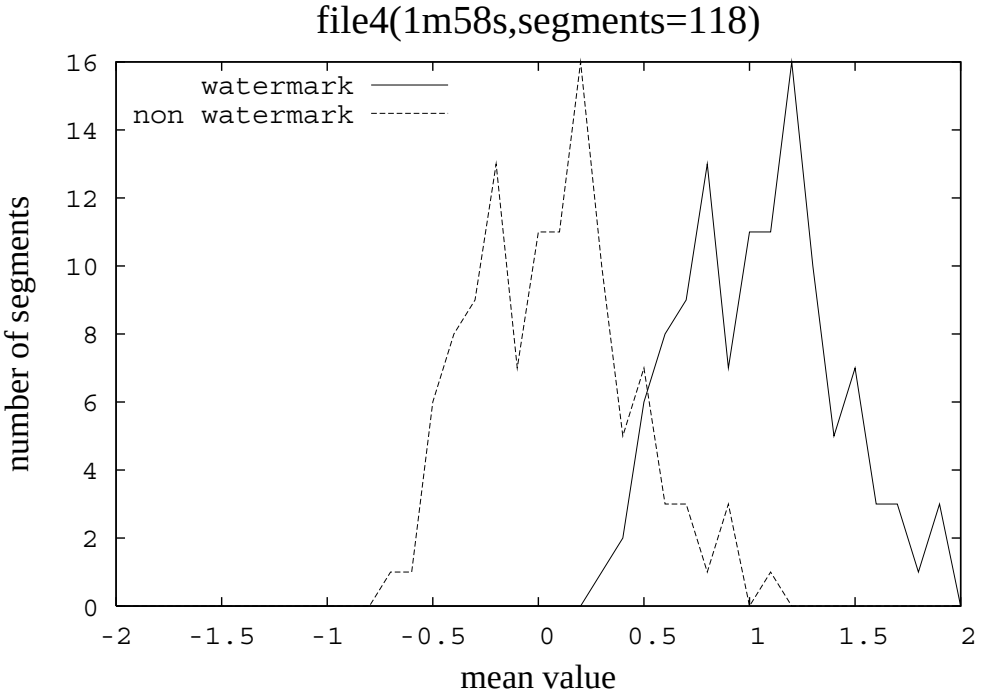

図 3.6: ファイル 4 の r の度数分布

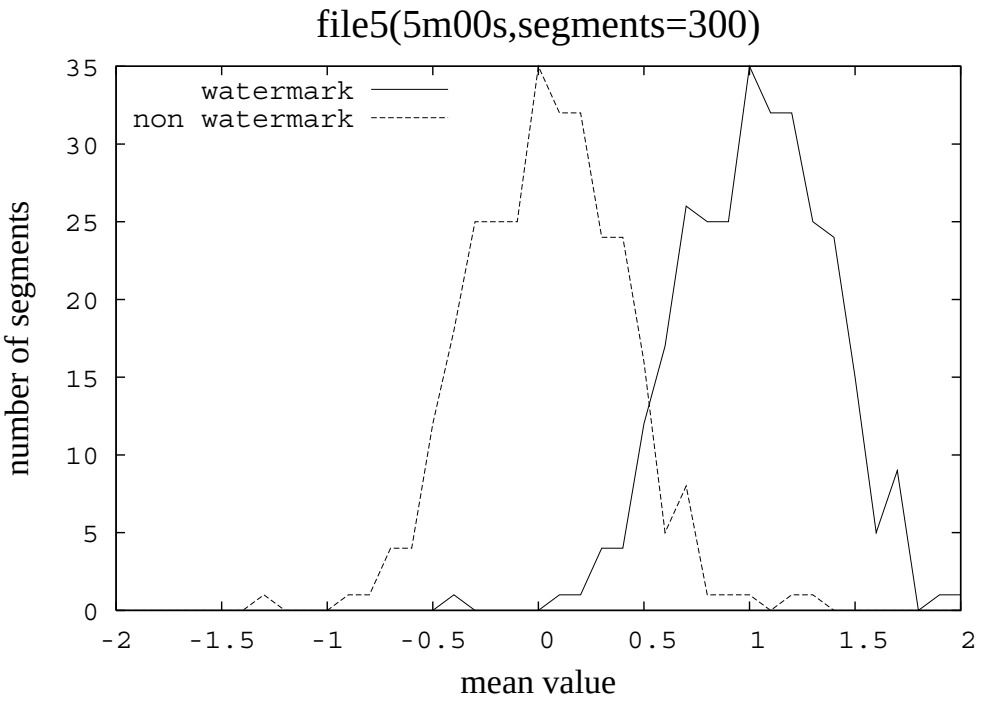

図 3.7: ファイル 5 の r の度数分布

3.8 ハードウェアの構成

前節まで示した，乗除算の削減手法と並列化手法，そして平均値を求めない検出法を利用して透かしを検出するハードウェアを構築する．ハードウェアは，次のような構成となっている．

- SRAM から透かしデータを転送する透かしデータ転送ユニット
- 式 (3.5) での S と F を計算する演算ユニット
- 式 (3.5) の比較と閾値の比較を行う比較ユニット

各ユニットの詳細は以下で示す．最後に，全体の構成を示し，ホストとハードウェア間でのデータ処理を示す．

3.8.1 透かしデータ転送ユニット

このユニットでは，SRAM に保存されたデータを FPGA 上に転送する．SRAM のデータ転送は，RC2000 ボード固有の API を利用する．API の関数と処理の流れを次に示す．

1. 透かしデータのアドレス指定 (ADMXRC2SSRAM*SetReadAddress(address))
2. 遅延 (delay)
3. tmp register へデータを転送 (ADMXRC2SSRAM*ReadData(tmp))
4. 透かしレジスタへ転送

透かしデータ転送ユニットを図 3.8 に示し，処理の流れを Procedure5 に示す．それぞれの透かしの初期アドレスは offset に保存しておく．offset は FPGA 上のメモリを利用する．Address Controller では，それぞれの透かしデータの次のアドレスを保存する next address を持つ．controller により，next address のデータを SRAM Interface へ送る．SRAM Interface は API 関数となっており，関数にアドレスを指定するだけでよい．利用するバンクの指定は，API 関数の”*”部分にバンクの番号を入れることで指定できる．SRAM から転送されたデータは 1 度 tmp register に保存する．controller の制御により，32 サイクルに 1 度すべての tmp register のデータが register に転送される．SRAM から FPGA へのデータ転送は 32 ビット幅で，それぞれのレジスタは 32 ビット保存できる．それぞれのレジスタは，検出する透かしの数だけ存在する．本研究では，25 個の透かしデータの転送を行うユニットを構築した．また，FPGA の回路量を考慮しない場合，このユニットは最大 6 個並列に構築でき，SRAM の 6 バンクから同時にデータ転送が可能である．

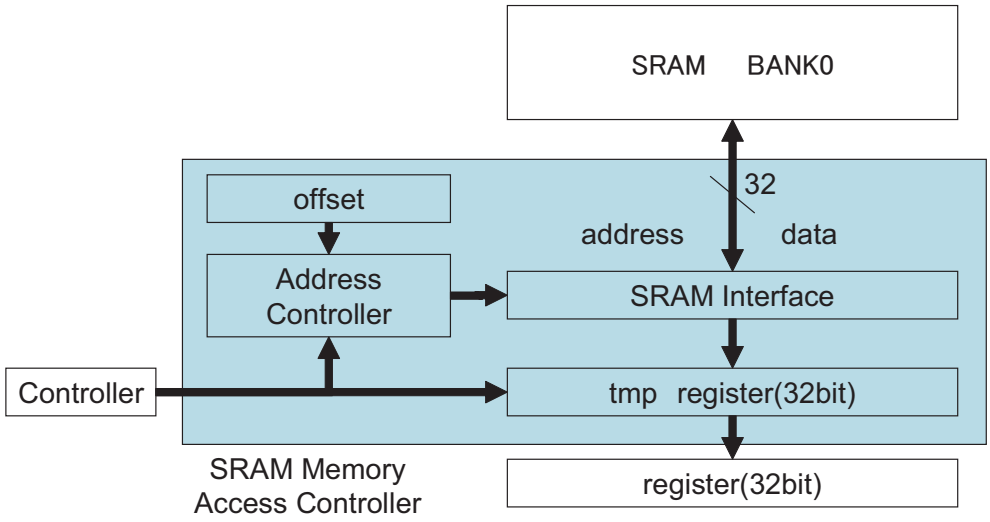

図 3.8: 透かしデータ転送ユニット

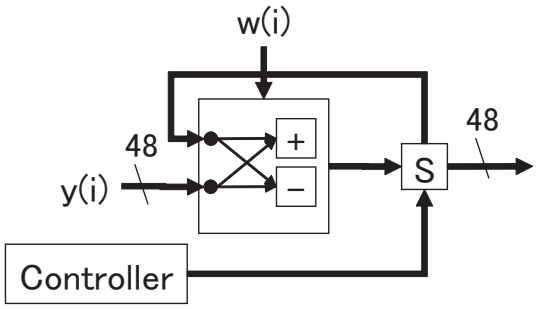

図 3.9: S 演算ユニット

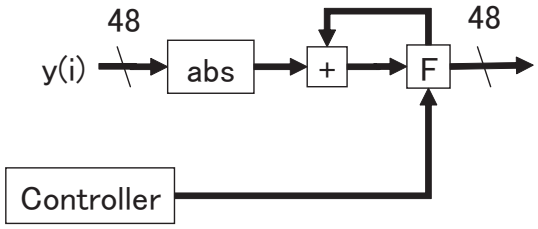

図 3.10: F 演算ユニット

3.8.2 S 演算ユニット，F 演算ユニット

式 (3.5) の S 及び F を演算する，S 演算ユニットと F 演算ユニットを図 3.9，図 3.10 に示す．入力データは符号付 48 ビットの $y(i)$ を，S 演算ユニットと F 演算ユニットで利用する．

S 演算ユニットの処理の流れは，Procedure3 と同じである．S 演算ユニットでは，$y(i)$ と透かしデータ $w(i)$ が入力値となる．透かしデータが 1 なら加算を行い，透かしデータが 0 なら減算を行う．controller により 44100 回計算した結果を出力する．出力値は S である．

F 演算ユニットの処理の流れを Procedure6 に示す．F 演算ユニットでは，$y(i)$ の絶対値を加算している．abs は絶対値演算を行う Handle-C の関数である．controller は 44100 回計算した時結果を出力する．出力値は F である．

Procedure 5 透かしデータ転送ユニット

Require: SRAM の 0 バンクを利用した場合

Ensure: 透かしデータ転送

```
1: next address ⇐ offset
2: for all i such that 1 ≤ i ≤ 32 do
3:    if i = 32 then
4:       すべての tmp register を register へ
5:       i ⇐ 0
6:    else
7:       ADMXRC2SSRAM0SetReadAddress(next address[i])
8:       ADMXRC2SSRAM0ReadData(tmp register[i-2])
9:       next address[i] ⇐ next address[i]+1
10:   end if
11: end for
```

Procedure 6 Calculate $F = \sum_{i=1}^{N} |y(i)|$

Require:

Ensure: $F = \sum_{i=1}^{N} |y(i)|$

```
1: F ⇐ 0
2: for all i such that 1 ≤ i ≤ 44100 do
3:    if i = 1 then
4:       Y ⇐ abs(y(1))
5:       F ⇐ Y
6:    else
7:       Y ⇐ abs(y(i))
8:       F ⇐ F + Y
9:    end if
10: end for
```

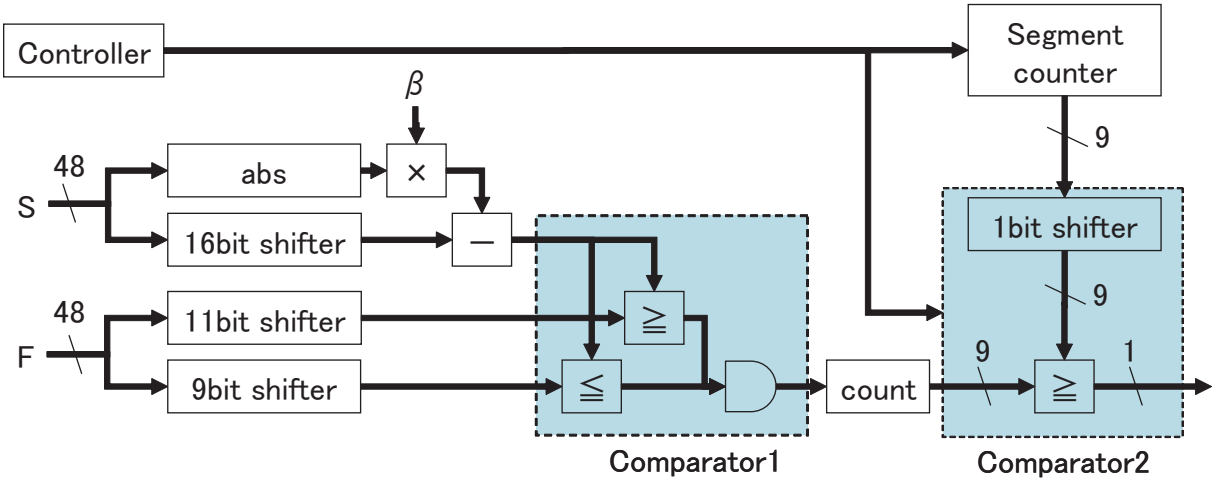

図 3.11: 比較演算ユニット

3.8.3 比較演算ユニット

このユニットでは，式 (3.5) の比較を行うユニットである．本研究では，$\frac{\Delta w}{N}$ の値は予め計算して定数を与えておくがこの値は小数となってしまう．そこで，この値を整数で与えるために全体を 2^{16} 倍して計算する．その結果，$\frac{\Delta w * 2^{16}}{N}$ の定数を β とすると，式 (3.5) は式 (3.6) となる．

$$\frac{1}{2} * 2^{16} * \alpha F \leq S * 2^{16} - \beta|S| \leq 2 * 2^{16} * \alpha F \tag{3.6}$$

また，α は 2^{-6} であるため，式 (3.7) となる．

$$2^{9}F \leq 2^{16}S - \beta|S| \leq 2^{11}F \tag{3.7}$$

S と F の値を利用して透かし検出を行なう回路を図 3.11 に示し，その流れを Procedure7 に示す． S の値より $2^{16}S - \beta|S|$ を計算し，F の値より $2^{9}F$ と $2^{11}F$ を求める．それぞれの結果を用いて，comparator1 で比較された結果がすべて真なら count を増加させる．それ以外は，count は増加させない．

segment counter は，セグメント数を数えるカウンターである．ホストより結果の要求が来たとき，count と segment counter の値を利用して透かしの検出を行なう．全セグメント数は，6 分程度のファイルでは 300 から 400 セグメントであるため，segment counter は 9 ビットカウンタとする．segment counter の値を 1bit シフトさせ，その値より count が大きい場合透かし検出と判断する．この比較は，comparator2 で行う．comparator2 の出力値は，1bit とし透かしの有無のみを出力する．

Procedure 7 比較演算ユニット

Require: S と F が計算されている．β が与えられている．

Ensure: 比較演算

1: $S1 \Leftarrow S$ shift left $16bit$
2: $S2 \Leftarrow abs(S)$
3: $F1 \Leftarrow F$ shift left $11bit$
4: $F2 \Leftarrow F$ shift left $9bit$
5: $S2 \Leftarrow S2 \times \beta$
6: $S1 \Leftarrow S1 - S2$
7: **if** $F2 \leq S1 \leq F1$ **then**
8: $\quad count++$
9: **end if**
10: $X \Leftarrow Segment\ counter$
11: **if** ホストからの結果要求 **then**
12: $\quad X \Leftarrow X$ shift right $1bit$
13: $\quad$ **if** $X \leq count$ **then**
14: $\quad\quad result \Leftarrow 1$
15: $\quad$ **else**
16: $\quad\quad result \Leftarrow 0$
17: $\quad$ **end if**
18: $\quad$ 結果の転送
19: **end if**

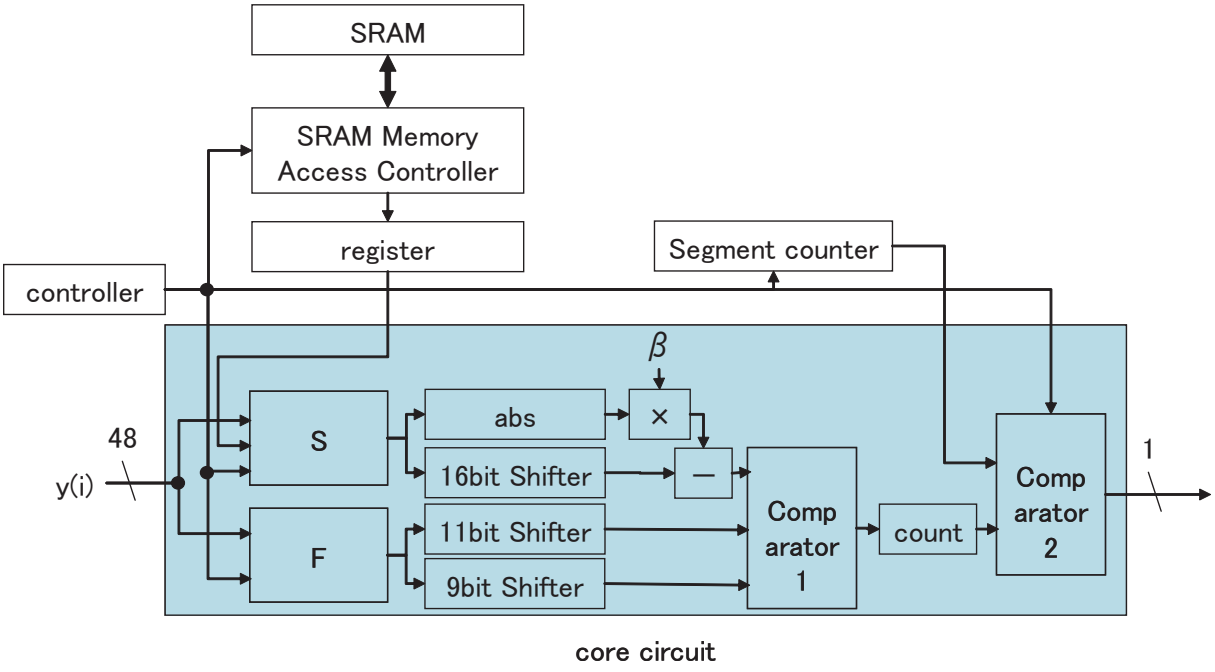

図 3.12: 1 つの透かし検出回路

3.9 全体の構成

前節までに示した，各ユニットを利用した検出回路の構成を示す.

3.9.1 1 つの透かし検出回路

1 つの透かし検出のための回路構成を図 3.12 に示す．図 3.12 は，透かしデータ転送ユニット，S 演算，F 演算ユニット，比較演算ユニットを組み合わせた回路である．入力データは符号付 48 ビットとする．これは，演算の途中でオーバフローが起きなための処置である．また，Handle-C ではビット幅の指定は厳密であるため，回路構築のミスを防ぐための処置でもある．”SRAM”-”SRAM Memory Access Controller”と”SRAM Memory Access Controller”-”register”は32 ビットである．これは，RC2000 ボードの制約により SRAM のデータ転送幅は 32 ビットとしてされているためである．グレイ部分は，core circuit であり透かしごとに β の値が異なっている．結果は，透かしが存在するときに result=1，透かしが存在しないときは result=0 とする．本研究では，β の値を回路構成時に定数として定義しているため，別の透かしを検出するときはその透かしに対応した β を与えなければならない．そのため，検出する透かしの変更では，透かしデータだけでなく回路全体も再構築させる必要がある．

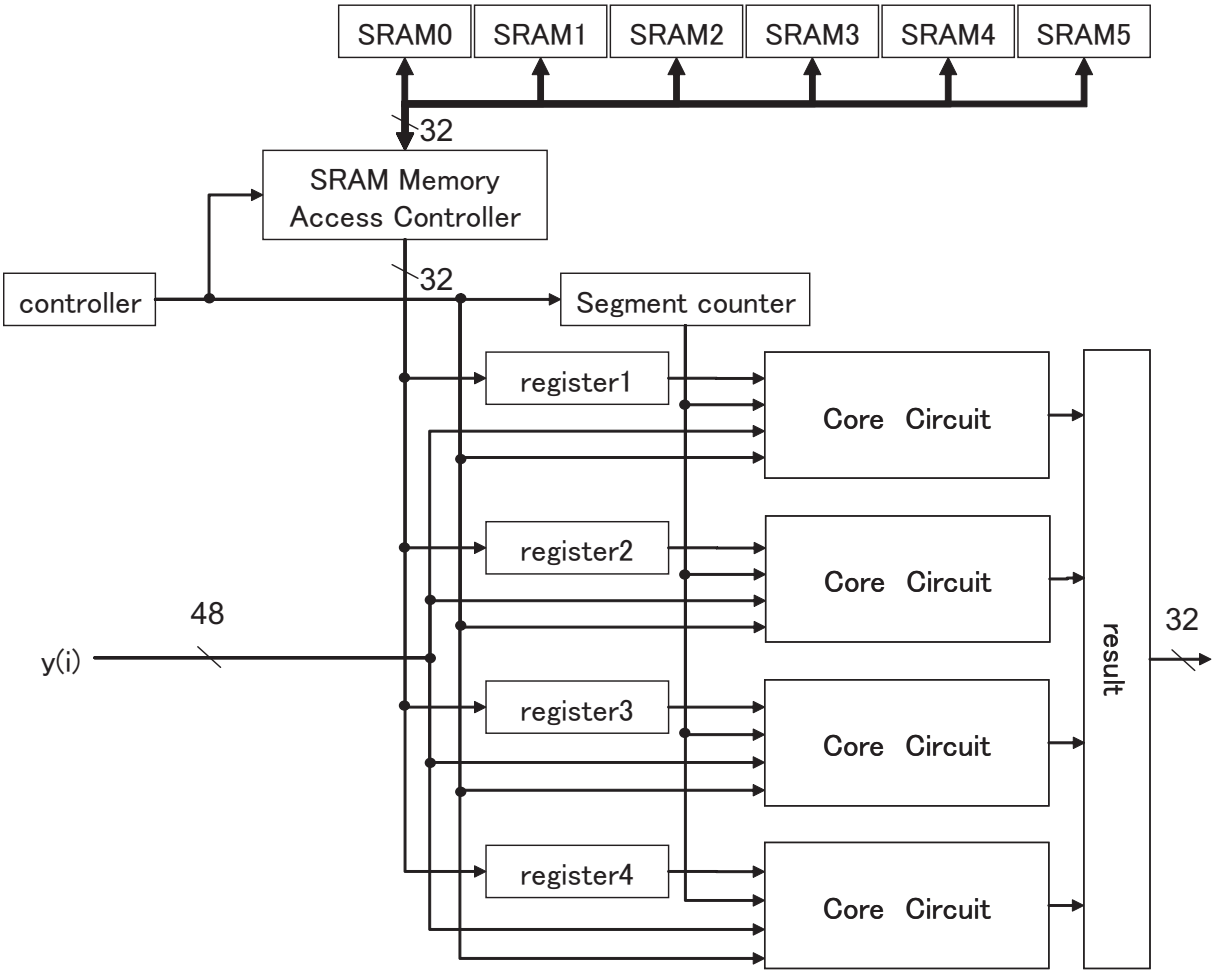

図 3.13: 複数の透かし検出回路

3.9.2 全体の回路構成

次に，複数の透かし検出回路の全体の構成を図 3.13 示す．複数の透かし検出のために，図 3.12 で示した core circuit 部分を複数並列に構築する．各 register には，検出する透かしデータが格納される．本研究では，実際には 25 個の透かしを並列に検出する回路と，50 個の透かしを同時に検出する回路を構築した．25 個の透かし検出回路では，SRAM を 1 つ利用して透かしデータを保存した．また，50 個の透かし検出回路では，SRAM を 2 バンク利用し 25 個の透かし検出回路を並列に構築した．result では，各 core circuit の結果より検出された透かしデータの番号を出力する．透かしが 1 つも検出されなかった場合は 0 を出力する．FPGA ボードからホストへのデータは符号無 32 ビットである．

3.9.3 データ処理と回路全体の流れ

FPGA ボード内のデータ処理と検出回路全体の流れを Procedure8 に示す.

Procedure 8 データ処理と検出回路の流れ

Require: 入力データは符号無し 32bit×44100 サンプル
Ensure: ハードウェア処理の流れ

if 入力データの受信 **then**
  for all *counter* such that $1 \leq counter \leq 44100$ **do**
    $y(i) \Leftarrow$ 受信データの拡張 $(48bit \rightarrow 32bit)$
    $y(i)$ をキューへ
    if $counter < 32$ **then**
      透かしデータ転送処理 (Procedure5)
    else
      透かしデータ転送処理 (Procedure5)
      S 演算処理 (Procedure3)
      F 演算処理 (Procedure6)
    end if
  end for
  Segment counter ++
  for all i such that $1 \leq i \leq 32$ { キューの中の処理 } **do**
    透かしデータ転送処理 (Procedure5)
    S 演算処理 (Procedure3)
    F 演算処理 (Procedure6)
  end for
  比較演算処理 (Procedure7 の一部)
else if ホストからの結果要求 **then**
  結果の出力 (Procedure7 の一部)
else
  wait
end if

ホストからのデータは API を用いて受信する．受信したデータは，符号無 32 ビットのデータであり，ホストから連続して 44100 サンプルのデータが転送される．各サンプルは，ハードウェアで符号付 48 ビットに拡張され，32 個の待ち行列であるキューに入る．データを受信するタイミングで，counter を増加させる．この counter が，各ユニットを制御する controller となる．始めの 32 サンプルのデータを受信している間に，初期値となる透かしデータを転送してくる．キューのデータを利用し，S 演算処理と F 演算処理を行う．1 セグメント分のデータを受信したあと，Segment counter を増やし，キューに残っているデータを処理する．すべてのキューのデータを処理した後，比較演算処理を行う．すべ

てのセグメントのデータに対して処理が行われた後，ホストからの結果要求によりホストへ結果を転送する．

3.10 まとめ

本章では，オーディオ電子透かしアルゴリズムを RC2000 ボードで構築する手法を提案し実装した．効率的な回路構築には乗除算の削減が不可欠であるため，乗除算の削減方法や別の演算で処理する方法を示し効率的な回路の構築を行なった．また，複数の透かし検出のための透かしデータの記憶に SRAM を利用した結果，回路量の制限を考慮しなければ最大 192 個の透かしを同時に検出する回路が構築可能であることがわかった．さらに，演算の遅延と回路量の増加につながる除算を利用しない透かしの検出法を提案した．最後に，透かし検出ハードウェアの回路の詳細と処理の流れを示し，25 個の透かしを検出する回路と，50 個の透かしを検出する回路を構築した．

第4章　評価

4.1　はじめに

前章までで，オーディオ電子透かしアルゴリズムの中から，ハードウェア化に適したアルゴリズムを検討し，採用したアルゴリズムをハードウェア化した．ハードウェア化では，乗除算を削減することで高速な動作と回路量の削減を行い，複数の透かしの高速検出のための並列化について検討を行ないハードウェア化を行った．

本章では，まず構築したハードウェアの透かし検出時間について評価を行う．同様に，第2章で実装したソフトウェアによる透かし検出時間と，第3章で提案した平均値を求めない検出法をソフトウェアで実装した検出時間を計測し，ハードウェアとの比較を行う．次に，構築したハードウェアのクリティカルパスより実際の動作速度を算出し，ハードウェアの回路量より最大の検出可能な透かしの数を算出する．

4.2　評価環境

評価環境として，Pentium4 2.8GHz，メモリ512MB，OSはWindowsXPを利用する．利用するハードウェアは，RC2000ボードを利用した．RC2000ボードは，Xilinx製xc2v8000のFPGAを搭載し，外部メモリにSRAMを4Mbyteを6バンク搭載している．ホストとRC2000ボードはPCIバスで接続されている．評価環境を図4.1に示す．

利用するオーディオファイルは，wavファイル，PCM，44.1Hz，ステレオとし，左チャンネルに透かしが埋め込こまれたファイルを利用する．

検出時間の計測方法は，ホスト側のプログラム中に時間計測の関数を埋め込むことで計測している．計測範囲として，左チャンネルのデータを利用して透かしの検出処理を行い出力表示までを計測範囲とする．ファイル形式の判別や，左チャンネルの抽出などの処理は計測範囲外とする．また，ハードウェアの検出時間測定時に，回路データをFPGAにロードする必要があるがこのロード時間なども測定範囲外とする．

ハードウェアのクリティカルパスの計測と回路量の測定は，回路設計ツールにより得られる値を利用する．

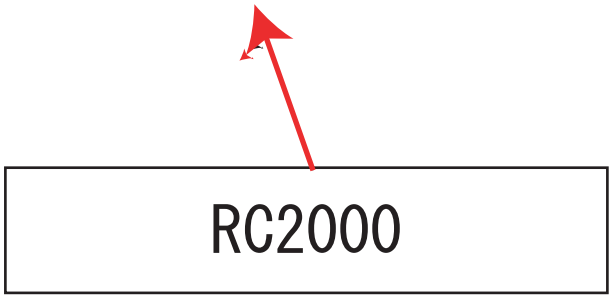

図 4.1: 評価環境

4.3 検出時間評価

本節では，第 2 章で示したアルゴリズムと，第 3 章で提案した平均値を求めない透かし検出法をソフトウェアにより実装を行い検出時間を比較した．また，構築した透かし検出ハードウェアの検出時間を測定し，ソフトウェアによる検出との比較により，どの程度検出時間が短縮されたかを明らかにする．

まず，ソフトウェアによる透かしの検出時間を測定する．第 2 章で示したアルゴリズムについては，予備実験としてソフトウェアで実装を行い検出時間を測定している．この結果と，第 3 章で提案した平均値を求めない透かし検出法のソフトウェアで実装し計測した検出時間を比較する．結果を表 4.1 に示す．表 4.1 で示す従来手法とは第 2 章で示したアルゴリズムであり，提案手法とは第 3 章で示した平均値を求めない透かしの検出法である．

表 4.1 の結果より，提案手法による検出時間の向上はソフトウェアでは現れなかった．提案手法では，演算に 64 ビット型である”long long”を利用したため，検出時間の向上につながらなかったと考えられる．しかし，提案手法はハードウェア化を前提としているた

表 4.1: ソフトウェアでの検出時間

透かしの数	従来手法 (ms)	提案手法 (ms)
5	1149	1178
10	2411	2345
15	3580	3668
20	4545	4826
25	5807	5853
30	6817	7070
35	8204	8232
40	9206	9540
45	10251	10738
50	11341	11875

め，回路量の削減や動作速度の向上には適していると考えられる．ちなみに，50 個の透かしを検出する速度は，従来手法では 14.4Mbps，提案手法では 13.7Mbps という結果を得られた．

次に，ハードウェアによる検出時間を測定し，ソフトウェアとの検出時間の比較を行なう．今回は，25 個の透かしを検出する回路と，50 個の透かしを検出する回路を構築し検出時間を測定した．ハードウェアの動作速度として 30MHz で動作させ計測を行った．検出時間を表 4.2 に示す．

表 4.2: ハードウェアでの検出時間

透かしの数	検出時間 (ms)	検出速度
25	500	326Mbps
50	500	326Mbps

検出速度に違いが生じないのはすべての透かしを同時に検出しているためであり，25 個の透かし検出も 50 個の透かし検出も同じ時間で処理が可能である．今回利用したオーディオファイルは，左チャンネルが 20.4Mbyte であるため，検出速度として 326Mbps 程度の速度を得ることができた．

今回測定した検出時間は，PCI によるデータ転送の時間も含まれているため，ハードウェアによる実際の検出時間とは異なる．そこで，計測した時間より，PCI の転送時間を引いたハードウェアのみの処理時間を計算し表 4.3 に示す．PCI は 32bit/33MHz であるため，データ転送速度は約 1Gbps である．理想的にデータが転送されたと仮定すると，20.4MByte のデータを転送するときの時間は 159ms である．つまり，計測時間から PCI のデータ転送を引いた時間は 341ms である．ハードウェアの実際の処理時間は 341ms で

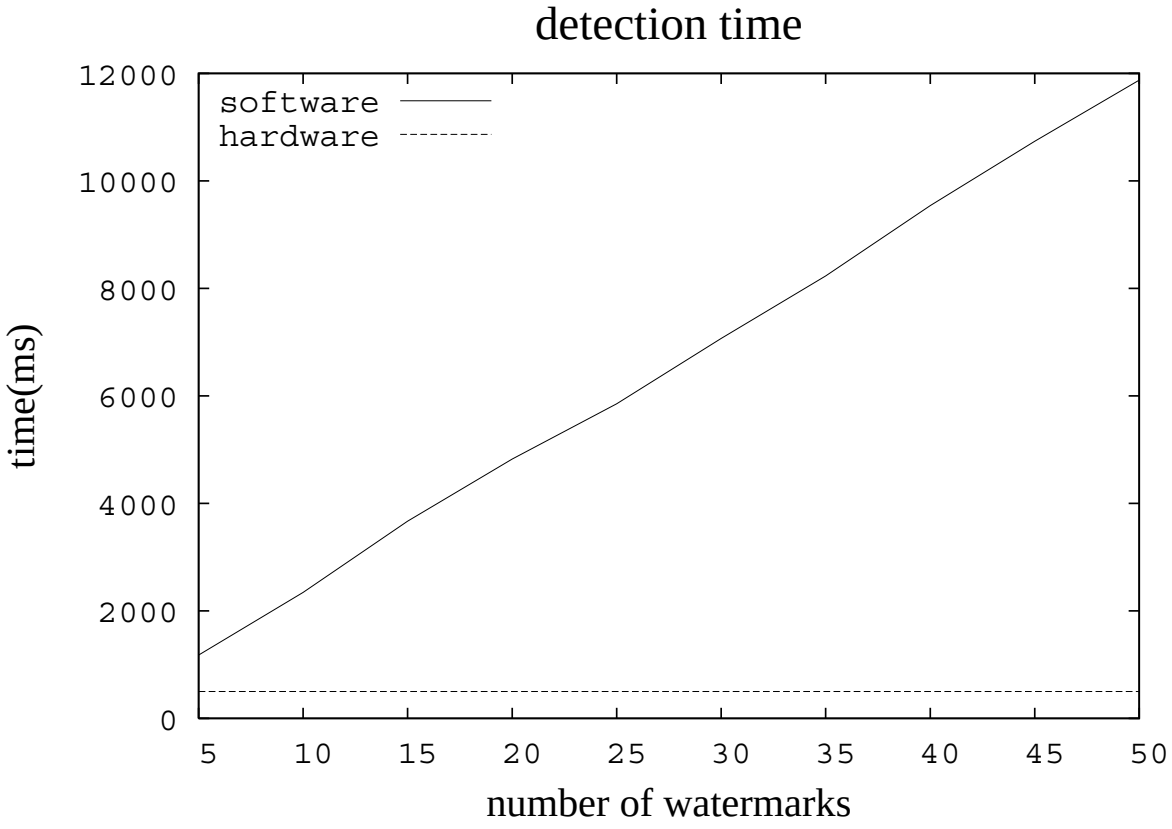

図 4.2: 検出時間の比較

あると推測され，処理速度は 476Mbps である．

表 4.3: ハードウェア単独の処理速度

検出時間 (ms)	PCI の転送時間 (ms)	ハードウェアの処理時間	処理速度
500	159	341	476Mbps

次に，ソフトウェアとハードウェアでの検出時間の比較を図 4.2 に示す．ソフトウェアは，提案手法を利用した検出時間を示している．グラフより，1 つの透かしを検出するときはソフトウェアもハードウェアもあまり時間の差はないことが推測できる．しかし，ソフトウェアでは検出する透かしの数に比例して検出時間が増加するが，ハードウェアでは透かしの数に依存せず一定の検出時間である．よって，複数の透かし検出にハードウェアを利用することは非常に有用である．50 個の透かしを検出するときは，ハードウェアはソフトウェアの約 23 倍の検出速度である．

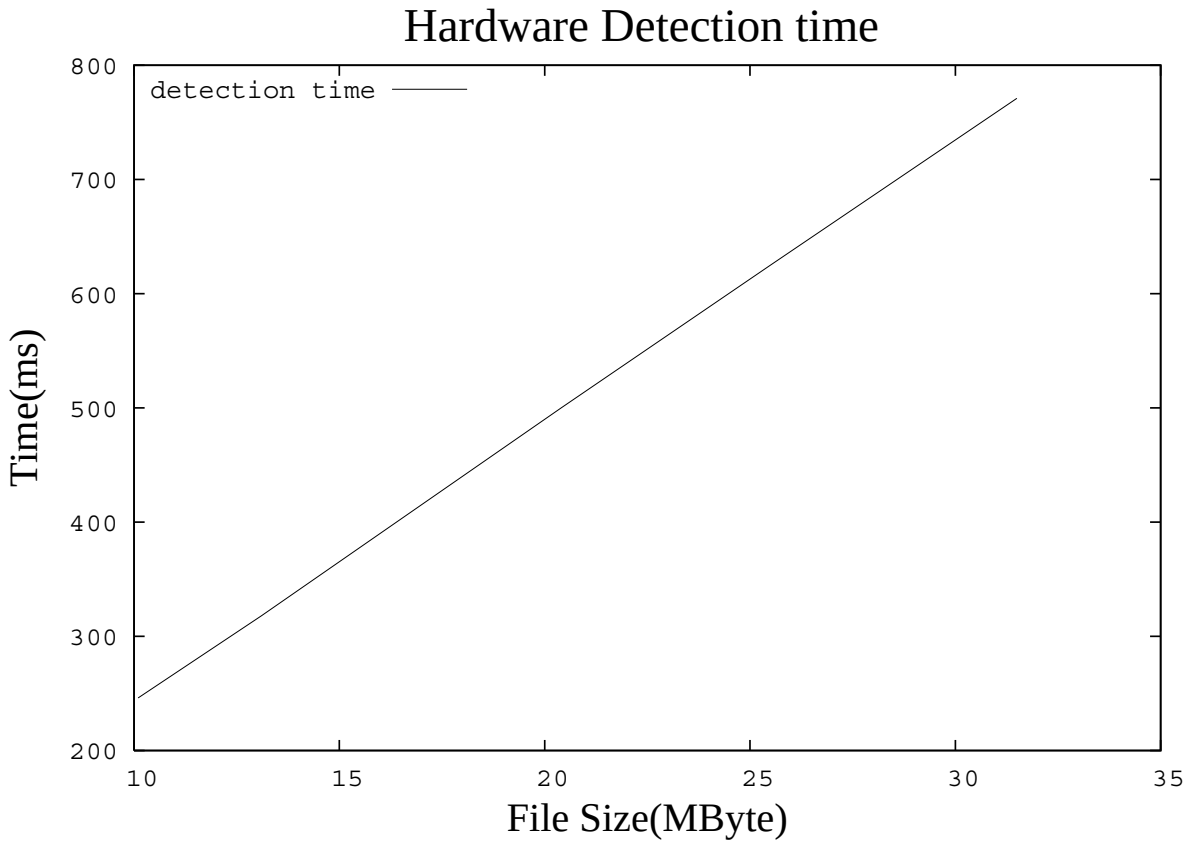

図 4.3: 検出時間の推移

4.4 ファイルサイズの増加による検出時間の推移

前節では，特定のファイルを利用して検出を行なった結果を示した．本節では，様々なサイズのファイルを用いてハードウェアにより検出を行なったときの検出時間の推移を示す．計測範囲と動作速度は，前節と同様とする．利用したファイルは，左チャンネルのデータが 10.1MByte,13.1MByte,20.4MByte,25.5MByte,31.5MByte の 5 種類のファイルを利用した．結果を図 4.3 に示す．

結果より，ファイルサイズの増加に比例し検出時間も増加する．よって，ハードウェアにより透かしを検出した場合，325Mbps の速度で検出が可能である．

4.5 検出回路量とクリティカルパス

本節では，25 個の透かし検出回路と 50 個の透かし検出回路のクリティカルパスと回路量を表 4.4 に示す．

それぞれの回路で，クリティカルパスが 25ns 以下であることから，40MHz で動作可能で

表 4.4: 回路量とクリティカルパス

	4 input LUTs	Number/total(%)	critical path
25 個の透かし検出回路	29,485(93,184)	31%	24.696ns
50 個の透かし検出回路	57,135(93,184)	61%	24.740ns

あることがわかる．そこで，40MHz で動作させたときの処理時間を計算すると，20.4MByte のファイルでは約 265ms となるため，ハードウェア単独では 615Mbps の処理速度を達成できる．

今回は，2 つの回路でクリティカルパスに違いが生じているが，これは回路構築の影響を受けていると考えられる．25 個の透かし検出回路では SRAM を 1 バンク利用し，50 個の透かし検出回路では SRAM を 2 バンク利用してる．この違いがクリティカルパスの違いにつながる可能性がある．また，本研究では回路記述に Handel-C を利用したため，コンパイラの影響を受けてクリティカルパスに違いが生じた可能性がある．

今回利用した RC2000 ボードの場合，50 個の透かし検出で FPGA 全体のおよそ 61%の回路を消費するため，およそ 82 個の透かし検出回路が構築できる．今回の検出回路は，まだ改良の余地があり回路量の削減などでさらに多くの透かしが検出できる．

4.6 まとめ

本研究では，オーディオ電子透かし検出アルゴリズムをハードウェア化を行うことで，高速な複数の透かし検出を行なった．ハードウェアによる高速検出のため，浮動小数点演算や乗除算の削減と，演算回路を並列化する手法を提案した．

本章では，ソフトウェアとハードウェアでの透かし検出時間を測定を行い比較を行った結果，ハードウェアがソフトウェアの約 23 倍の検出速度を得ることができ，325Mbps 程度の速度を達成した．また，クリティカルパスの評価より 40MHz での動作が可能であることを示すことができ，ハードウェア単独では 615Mbps で処理できることを示した．さらに，今回構築した 50 透かし検出回路では，全体の 61%の回路量であるため，82 個程度の透かしが検出できる回路が構築可能であることが示せた．今回構築した回路は，まだ改良の余地を十分含んでいるため，改良することでより多くの透かしを高速に検出することが可能である．

第5章　不正配布監視システムへの応用

5.1　はじめに

近年，ネットワークの高速化に伴い，電子化された音楽を配信するサイトが多く存在するようになってきた．また，Apple コンピュータの iPod などのデジタルオーディオプレーヤの普及に伴い，多くの音楽が電子化されるようになった．電子化された音楽は，非常にコピーが容易で品質の低下もないため，簡単にネットワークを経由して転送することが可能である．そのため，ファイル交換ソフトなどを利用することで簡単にオーディオファイルを入手することが可能である．しかし，多くのオーディオファイルは著作権が存在するため，これらのファイルを不特定多数にダウンロードを許可することは違法となる．多くのファイル交換ソフト利用者は著作権問題に関心が薄く，自分が著作権を侵害していることをあまり意識していない．そこで，ネットワークを経由して転送されるファイルの中で，不正に配布されているものは転送許可しないことができれば著作権保護が可能である．

本研究では，問題となる不正配布を防止する目的で，今回構築した電子透かしハードウェアを用いた，オーディオファイルの不正配布を監視する不正配布監視システムを提案する．本章では，提案システムの概要を示しその実用性について議論する．

5.2　概要

本研究で提案する不正配布監視システムは，オーディオファイルの不正配布を対象に監視する．本研究では，オーディオファイル中に透かしが埋め込まれているファイルは転送許可がないものと考える．そこで，ネットワーク中を流れるオーディオファイルから電子透かしの検出を行い，透かしを検出したファイルの転送はシステムで転送を中断する．この透かしの検出は今回構築した透かし検出ハードウェアを用いる．そのため，対象となるオーディオファイルは wav ファイルとする．提案システムでは，ネットワークの中継地点となるルータなどに導入することで効率的に監視を行う．

システムのフローチャートを図 5.1 に示す．ネットワークを流れるデータはパケット単位で転送されるため，まずパケットを解析しオーディオファイルの転送であるかを判別する．このパケット解析については，多くのソフトウェアで実装されているためソフトウェアによる解析を行う．パケット解析によりオーディオファイルであると判断された場合は，そのファイルのパケットをバッファに蓄える．バッファに蓄えられたデータから左チャンネルのデータを抽出し，透かし検出ハードウェアで透かしの検出を行なう．データの保存

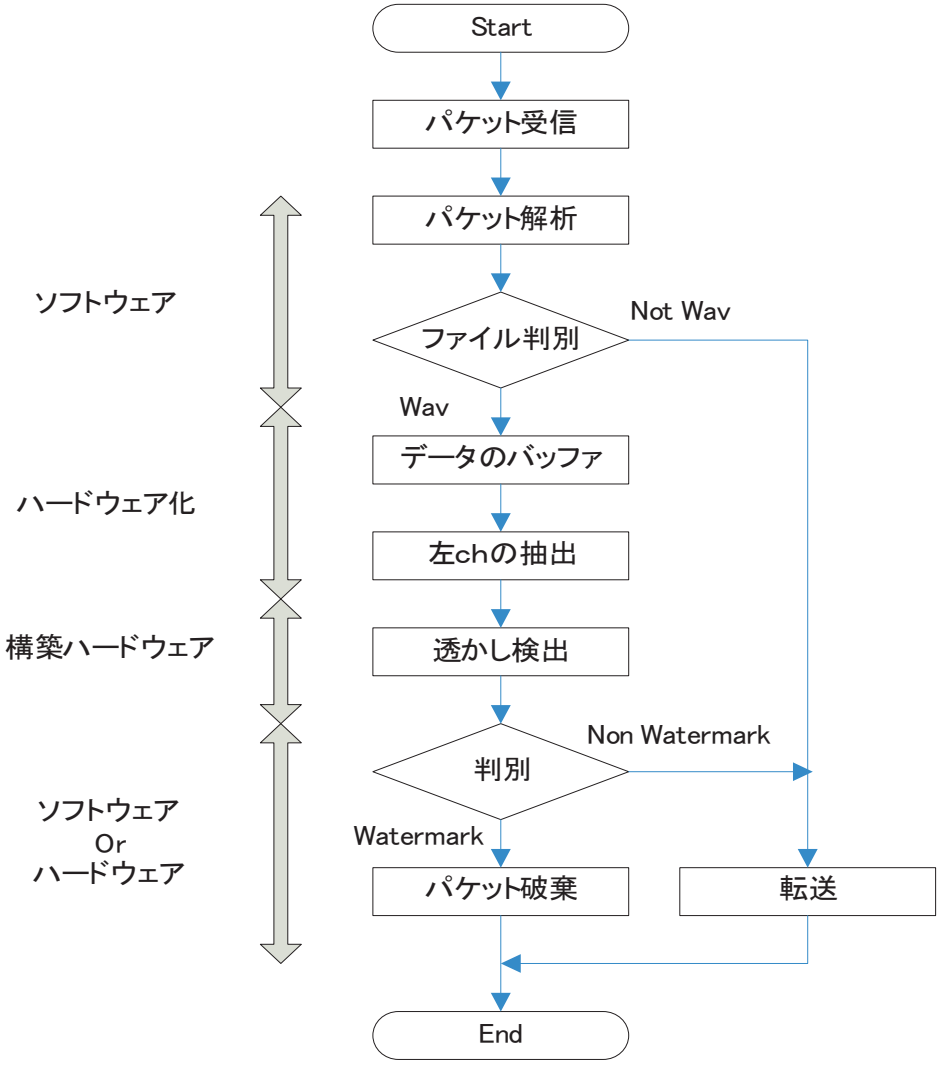

図 5.1: 不正配布監視システムのフローチャート

と左チャンネルのデータ抽出は，ハードウェアにより行うこととする．抽出したデータと，今回構築したハードウェアで透かしの検出を行なう．その結果，透かしが検出されたファイルのパケットはシステムにより破棄される．ファイルから透かしが検出されない場合や wav ファイルでない場合は，データの転送を許可する．このような流れで不正配布を監視する．

提案システムでは，検出ハードウェアは FPGA 上に構築されているため，回路構成の変更が容易であるという利点を持つ．そのため，検出したい透かしの変更や，アルゴリズムの変更も容易にできる．提案システムは，運用形態にあわせた回路を構築することで，柔軟な監視が可能となる．

5.3 実用性

提案システムを用いることで，不正なオーディオファイルの転送を防ぐことが可能であると考える．しかし，提案システムを導入することでさまざまな問題が生じると考えられる．そこで，考えられる問題点を示し，提案システムの実用性について議論する．

5.3.1 検出速度

提案システムでは，ネットワークの中継地点となるルータなどにシステムを設置する．ここで問題となることは，提案システムを導入することでネットワークのボトルネックになる可能性である．そこで，提案システムを構築した場合のシステムのスループットを見積もり，ネットワークのボトルネックになるかを議論する．

まず，図 5.1 のパケット解析は，現在ルータなどで行うことができる．wav ファイルであるかを判別するだけなので，この部分はボトルネックになることはない．wav ファイルをバッファに蓄え左チャンネルを抽出する部分は，前章で評価していないがハードウェア化することでボトルネックにならないと考えられる．もっとも，ボトルネックとなると考えられるのは透かしの検出である．本研究で構築したハードウェアの検出時間は 500ms であった．これは，オーディオファイルの左チャンネルのデータがハードウェアに転送が開始され，結果をホストコンピュータで表示する場合までの時間である．ここで，データは PCI を利用して転送されている．PCI バスは 32bit/33MHz で転送されるため約 1 Gbps の転送速度となる．今回の計測では，20.4MByte のファイルを利用したためデータ転送に要した時間は 159ms となり，ハードウェアで要した時間は 341ms である．よって，ハードウェア単独の処理速度は 476Mbps である．今回はハードウェアを 30MHz で動作させたときのスループットを示したが，実際は 40MHz で動作することができるため 615Mbps のスループットを実現できる．そこで，wav ファイルのバッファと左チャンネルの抽出を透かし検出ハードウェア上で実装することで，データ転送を高速化しボトルネックとならない高速なスループットが実現できると考える．

今回のシステムでは，wav ファイルのみが透かし検出の対象であるため，ほとんどのファイルはパケット解析のプロセスで転送が許可される．つまり，wav ファイルの転送は 500Mbps 程度となるが，wav ファイル以外の転送はシステムの影響をほとんど受けないため十分実用的なスループットを提供できると考える．

5.3.2 検出可能な透かし数

電子透かしは著作権者固有であるため，著作者の数だけ透かしが存在する．そのため，すべての透かしに対して検出できることが理想的であるが，これは非常に困難である．提案システムを利用した場合，透かしの検出は 1 度で 50 個の透かししか検出できない．ハードウェアの改良と大規模な FPGA を利用することでさらに多くの透かし検出に対応する

ことが可能であるが，すべての透かしを検出することは事実上不可能である．そのため，提案システムを分散的に配置し各システムで違う透かしを監視することである程度不正配布を防ぐことはできるが，十分な結果を得ることは難しいと考えられる．つまり，本研究で構築したハードウェアでは数百程度の透かしについては十分に対応することが可能であるが，不正配布を防ぐには検出する透かしの数が小さいということが問題である．この問題に対処する方法として，利用方法を考察することで提案システムが有効に利用できると考える．そこで，現時点で考えられる利用方法を 2 つ提案する．

提案 1　提案システムでは，すべての透かしを検出することは不可能であるが数百程度なら可能であるため，オーディオファイルに埋め込む透かしの数を限定することで利用可能となる．例えば，制作会社単位や著作権保護を目的とした会社単位で透かしを割り当てることで，透かしの数を小さくする．オーディオファイルは，それぞれの会社の透かしを埋め込むことで保護される．会社の数は，著作者数より小さいため数百程度に減らすことが可能となり，提案システムを有効に利用できると考える．

提案 2　すべてのオーディオコンテンツに共通する透かしをいくつか設定しておき，その透かしをシステムで検出する．この透かし検出は，学校や会社などの LAN からインターネットへ出て行くルータで検出を行なう．これにより，その透かしが埋め込まれたファイルの流出を防ぐことができる．たとえ，透かしが埋め込まれたコンテンツがインターネットに流出しても，提案システムを設置している LAN から流出したものでないことを証明することに使うことが可能である．

これらの提案のように，提案システムでは不正配布を完全に防ぐためには，透かしの検出可能な数が十分ではないが，利用方法を工夫することで十分利用価値があると考える．

5.3.3　暗号化された転送

提案システムは，一般的なファイル転送を想定しているため，暗号化されたファイルの転送を防ぐことは不可能である．例えば，Winny などを利用したファイル転送は暗号化された転送となるため，提案システムを利用して不正配布を防ぐことはできない．そこで，Winny などの P2P ファイル交換ソフトを利用した転送を監視するサービス [16] と提案システムを併用することで，ある程度の不正な配信を防ぐことが可能である．

5.4　まとめ

本章では，本研究で構築したハードウェアを用いた不正配布監視システムを提案した．提案システムは，ルータなどのネットワークの中継地点に設置し，ネットワークを流れるオーディオファイル監視して不正配布を防ぐものである．提案システムのスループットは

500Mbps 程度と想定できるため，ほぼネットワークのボトルネックにならないと考えられる．また，検出可能な透かしの数が小さいため完全に不正配布を防ぐことは不可能であるが，利用方法を工夫することで利用価値があると考える．P2P ファイル交換ソフトによる転送では，暗号化されたデータが転送されるため，提案システムでは検出をすることが出来ない．しかし，P2P を監視するサービスと提案システムを併用することで，ある程度の不正配布を防ぐことができる．提案システムは，運用形態を工夫することで有用なシステムになると考える．

第6章　結論

6.1　まとめ

本論文では，オーディオ電子透かしの検出をハードウェアで行なうことで，ソフトウェアでは難しい高速な複数の透かし検出を実現した．さまざまなオーディオ電子透かしアルゴリズムの中から，ハードウェア化に適したアルゴリズムを考察し，複数の透かしを高速に検出する手法を提案した．実現したハードウェアでは，ソフトウェアの 23 倍の検出速度を達成することができた．

第 2 章では，一般的なオーディオ電子透かしアルゴリズムの概要を説明し，ハードウェア化に適していると考えられるアルゴリズムを考察した．本研究では，時間領域を利用し，検出時に透かしを利用するアルゴリズムがハードウェア化に適していると考えた．

第 3 章では，オーディオ電子透かし検出ハードウェアの構築を行った．高速に複数の透かしを検出するため，乗除算の削減と並列化の手法を提案した．さらに，除算を必要としない新たな検出法を提案し，25 個の透かしと，50 個の透かしを検出する回路を構築した．

第 4 章では，構築したハードウェアの性能評価を行った．オーディオ電子透かし検出アルゴリズムのハードウェアとソフトウェアでの検出時間の比較を行ない性能を評価した．本研究で構築したハードウェアでは，ソフトウェアの 23 倍の検出速度で検出が可能であることが示せた．また，構築したハードウェアのクリティカルパスより 40MHz で動作可能で，回路量より最大 82 個の透かし検出回路が構築可能である．

第 5 章では，構築したハードウェアを用いた，オーディオファイルの不正配布監視システムを提案した．ハードウェアを用いることで，ネットワークのボトルネックとならない監視が可能であることを示し，運用形態の工夫により有用なシステムであることを示した．

6.2　今後の課題

本研究で構築したハードウェアは，まだ改良の余地を十分含んでいて，より高速に多くの透かし検出が可能なハードウェアの構築は可能である．改良可能部分として，データのビット幅の検討が考えられる．今回は，データのビット幅を 48 ビットで統一しているため，実際には 48 ビットも必要ない部分で回路量の増加と演算速度の低下が起きてると考えられる．そのため，最適な回路を構築することが求められている．

また，本研究ではハードウェア化に適したアルゴリズムを採用することで，高速に複数の透かし検出を可能にすることができた．しかし，電子透かしの特性上，透かしの聞こえ

にくさや透かしの強度については考慮することは重要である．そこで，さまざまな電子透かしアルゴリズムについて検討を行い，そのアルゴリズムの特性を考慮したハードウェア化することで，さまざまな電子透かしに対応することがよいと考える．これにより，本研究で提案したシステムがより実用的なものとなっていくであろう．

謝辞

本研究を行うにあたり，多くの御助言，御指導を賜りました情報科学センター 井口 寧 助教授に深く感謝するとともに，ここに御礼申し上げます.

適切な御意見，御助言を頂きました本学の松澤照男教授，田中清史助教授に深く御礼申し上げます.

貴重な御意見，討論を頂いた井口研究室学友の磯垣順氏，広瀬勇人氏，渡辺浩二氏，我妻直樹氏，Yin Wuotang 氏，鈴木祐一氏，松本健太郎氏，宮子陽一氏に深く感謝致します．井口研究室の先輩である Zhang Yuanyuan 氏には様々なアドバイスを頂き，後輩である礒永久史氏，近藤裕貴氏，清水昭尋氏，高畠義和氏，松山周平氏には日々の愚痴を聞いて頂きました．また，日常の相談にも答えて下さった敷田研究室の皆さんと，日々のお世話をして頂いた山上由美氏にも感謝いたします.

最後に，日頃より私を支えて下さった両親に深く感謝致します.

参考文献

[1] 電子透かしとコンテンツ保護, オーム社 (2001).

[2] 電子透かしの基礎–マルチメディアのニュープロテクト技術–, 森北出版株式会社 (1998).

[3] JASRAC, : プレスリリース,2003.1.22: http://www.jasrac.or.jp/release/index.html.

[4] Boney, L., Tewfik, A. H. and Hamdy, K. N.: Digital Watermarks for Audio Signals, in *International Conference on Multimedia Computing and Systems*, pp. 473–480 (1996).

[5] Nedeljko Cvejic, T. S.: Improving Audio Watermarking Scheme Using Psychoacoustic Watermark Filtering: http://www.mediateam.oulu.fi/publications/pdf/93.pdf.

[6] Hong Oh Kim, N.-Y. L., Bae Keun Lee: Wavelet-based Audio Watermarking Techniques:Robustness And Fast: http://amath.kaist.ac.kr/research/paper/01-11.pdf.

[7] Bassia, P. and Pitas, I.: Robust audio watermarking in the time domain, in *9th European Signal Processing Conference (EUSIPCO'98)*, pp. 25–28, Island of Rhodes, Greece (1998).

[8] Bassia, P. and Pitas, I.: Robust audio watermarking in the time domain, *IEEE Transactions on Multimedia*, Vol. vol 3, No. no. 2.

[9] Gruhl, D., Lu, A. and Bender, W.: Echo Hiding, in *Information Hiding*, pp. 293–315 (1996).

[10] 三上直樹：アルゴリズム教科書, CQ 出版社 (1996).

[11] http://www.celoxica.com.

[12] http://www.xilinx.com.

[13] Celoxica Ltd.: *DK Design Suite User Guide For DK Version 3.0* (2004).

[14] Celoxica Ltd.: Handel-C 言語仕様マニュアル Version 2.1.

[15] Celoxica Ltd.: *Platform Developer's Kit ADM-XRC2Platform Support Library,SDK and Examples Manual.*

[16] 株式会社ネットアーク：P2P Finder：http://www.netarc.jp.