社団法人　情報処理学会　研究報告
IPSJ SIG Technical Reports

2005－SLDM－118（18）
2005／1／26

FPGAを用いたオーディオ電子透かしの検出

榊原　憲宏†　　井口　　寧††

† 北陸先端科学技術大学院大学 情報科学研究科　〒923-1292　石川県能美郡辰口町旭台 1–1
†† 北陸先端科学技術大学院大学 情報科学センター, 科学技術振興機構 さきがけ研究 21「機能と構成」
〒923-1292　石川県能美郡辰口町旭台 1–1
E-mail: †{s-kazu,inoguchi}@jaist.ac.jp

あらまし　近年，デジタルコンテンツの不正なネットワーク配信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コンテンツに著作者の情報を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によって行っており，検出速度が低いためネットワークで不正に配信されるコンテンツそのものの流通を防ぐことはできなかった．そこで，本研究では，FPGA 上に電子透かしの検出回路を並列に実装することにより透かしの検出速度を大幅に向上させ，ネットワーク中で交換されるコンテンツから電子透かしをリアルタイムで検出することで，不正配信を防ぐ手法を提案する．提案したハードウェアでは，ソフトウェアで実現が難しい複数の透かしの高速検出を，検出回路の並列化により実現した．処理時間の評価より，提案ハードウェアは，ソフトウェアによる電子透かし検出の 148 倍の検出速度を達成した．

キーワード　FPGA，オーディオ電子透かし，並列化，ネットワーク，C レベルデザイン

Detection of the audio watermark on FPGA

Kazuhiro SAKAKIBARA† and Yasushi INOGUCHI††

† Japan Advanced Institute of Science and Technology　Asahidai1–1, Tatsunokuchi, Ishikawa 923-1292 Japan
†† Japan Advanced Institute of Science and Technology, "Infomation and Systems", PRESTO Japan Science and Technology Agency　Asahidai1–1, Tatsunokuchi, Ishikawa 923-1292 Japan
E-mail: †{s-kazu,inoguchi}@jaist.ac.jp

Abstract　Recently,illegal spread of digital contents via network becomes a serious problem. In order to solve this problem, digital watermarking has been proposed which shows copyright information. Digital watermarking is the technique to embed copyright information that people can not recognize into digital contents. However, digital watermark detection by software is too slow and it can not catch illegal audio files exchanged over network. In this paper, we propose a high speed digital watermark detection scheme by implementing parallel detector on a FPGA chip. Since it is difficult to detect many watermarks at high speed by software, we use hardware to detect many watermarks at high speed by implementing the detection circuit of digital watermarking in parallel. Estimation results show that the proposed hardware scheme is 148 times as fast as software.

Key words　FPGA,audio watermarking,parallel,network,C level design

1. まえがき

近年，デジタルコンテンツのネットワーク配信が盛んに行われているが，デジタルコンテンツの場合，簡単にコピーができるため，不正に利用されてしまうことが問題となっている．そのため，コンテンツの著作権の保護と主張，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著作者の情報をコンテンツに埋め込む技術である．この技術を用いることで，コンテンツの著作権の保護と主張が可能となる．ネットワークにおける著作物の違法配布に対し，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，ロボット検索プログラムを用いて，ネットワーク上にある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，電子透かしの有用性を実証し違法サイトの情報収集が可能であることを示した．

しかしながら，この方法では，コンテンツの不正配信の抑制は可能であるが，未然に防ぐことはできない．そこで，コンテンツの不正配信を未然に防ぐために，ネットワーク中で交換されるコンテンツから電子透かしをリアルタイムで検出し，透かしの入った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．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を必要とする．また，ネットワーク上では複数の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，複数の透かしの検出に対応しなければならない．現在，電子透かしの検出はソフトウェアにより行われているため，リアルタイムに透かしを検出することは非常に困難である．さらに，複数の透かしを検出するには多くの時間が必要となってしまう．

そこで，本稿ではオーディオファイルを対象とした電子透かし技術を利用し，ネットワーク中で交換されるオーディオファイルからリアルタイムに電子透かしを検出するハードウェアを構築する．ソフトウェアで実現が困難な，複数の透かしの検出は，検出回路を並列に構築することで実現する．また，リアルタイムな検出のため，検出回路の簡略化によりリアルタイム検出を達成する．本稿では，提案する検出回路を FPGA 上に実装し，回路の性能を評価した．

本稿の構成は，２章でハードウェア実装向けのオーディオ電子透かしアルゴリズムを明らかにする．３章では，ハードウェア回路の簡略化法と並列化法を示し，４章で，検出ハードウェアの評価を行う．５章でまとめと今後の予定を述べる．

2. オーディオ電子透かし

2.1 オーディオ電子透かしの概要

現在，オーディオ電子透かしとして多くの手法が提案されている．多くの場合，人間の耳に聞こえない音を著作者固有の透かしデータとして埋め込んでいる．その手法として，周波数領域へ透かしを埋め込む方法[4]と，時間領域へ透かしを埋め込む方法[5][6]が一般的である．さらに，透かしを検出する場合でも元のデータが必要な場合や透かしデータが必要な場合がある．このように，さまざまな方法の電子透かしが提案されている．

2.2 利用アルゴリズム

本稿では，ソフトウェアでは困難である高速な複数の透かしの検出を，ハードウェアを用いて実現させる．しかし，ハードウェアを利用した場合，回路量の制約や利用可能なメモリサイズなどさまざまな制約がある．そこで，さまざまなオーディオ電子透かしアルゴリズムの中より，ハードウェアを用いることで高速に複数の透かしを検出することができるアルゴリズムを検討する．

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する．そのため，浮動小数点演算や乗算除算を多用に利用することとなり比較的回路量が多くなる．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となるが，これは周波数領域を利用した場合でも同じである．しかも，フーリエ変換をしないという点から，回路量は周波数領域を利用した場合より比較的小さくなると考えられる．ハードウェアでは回路量の制約で，より小さい回路が望ましいため，時間領域を利用するほうがよいと考える．

検出時に必要なデータとして多くのアルゴリズムの場合，透かしデータか元のデータを必要とする．本稿では複数の透かしを検出することが目的のひとつである．そのため，検出時に元のデータを必要とする場合，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．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ので，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を用いる場合がよいと考える．

以上の観点から，本稿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電子透かしアルゴリズムをハードウェア化する．本稿では，[6]で提案されているアルゴリズムのコア部分のみを改良しハードウェア実装する．これは，本稿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ネットワークのボトルネックにならない高速な検出を目的としているためである．

2.3 埋め込みアルゴリズム

オーディオデータを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(1)の計算し透かしデータを埋め込む．元のデータを $x(i)$，透かしデータを $w(i)$，埋め込まれたデータを $y(i)$ とする．$w(i)$ の値は，著作者固有の乱数配列とし，要素の値は-1か1である．また-1と1の要素数は等しくないとする．本稿では，$\alpha = 1/128$ とし，$N = 44100$ とする．

$$y(i) = x(i) + \alpha|x(i)|w(i) \tag{1}$$

2.4 検出アルゴリズム

透かしが埋め込まれたデータ $y(i)$ を N サンプルずつセグメントにわけ，セグメントごとに検出を行なう．まず，S を次のように定義する．

$$S = \sum_{i=1}^{N} y(i)w(i) \tag{2}$$

式(1)，(2)より，式(3)を得る．

$$S = \sum_{i=1}^{N} [x(i)w(i) + (\alpha|x(i)|w(i))w(i)] \tag{3}$$

ここで，$\Delta w = \sum_{i=1}^{N} w(i)$ とすると $\Delta w \neq 0$ となるため，次のような式が得られる．

$$S = \sum_{i=1}^{N-\Delta w} x(i)w(i) + \sum_{i=1}^{\Delta w} x(i)w(i) + \sum_{i=1}^{N} \alpha|x(i)|w(i) \quad (4)$$

式 (4) の第 1 項は，$w(i)$ の-1 と 1 の数が等しいためおよそ 0 となる．もし，透かしが入ってない場合は，S の値は式 (4) の第 2 項の値とおよそ等しくなる．透かしが入っている場合，S の値は式 (4) の第 2 項と第 3 項の合計値とおよそ等しくなる．検出時にはもとのデータである $x(i)$ の代わりに $y(i)$ を利用する．$y(i)$ を利用した場合でもあまり大きな影響は生じない．そのため，式 (4) 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 (5) で与える．

$$r \triangleq \frac{S - \frac{\Delta w}{N}|S|}{\sum_{i=1}^{N}(\alpha|y(i)|w(i))w(i)} \quad (5)$$

もし，透かしがある場合 r の値はほぼ 1 になり，透かしがない場合 r の値はほぼ 0 になる．検出時には多少の誤差があるため，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 以上で検出できたことにする．

3. ハードウェア化手法

3.1 ハードウェア化の概要

本稿では，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の検出を目指す．そこで，次の 2 点に着目してハードウェア化を行う．

- 乗算除算の削減による検出の効率化．
- 複数の透かしの並列検出．

ハードウェアの実装は，FPGA を用いて行う．本稿では，オーディオデータの左チャンネルに透かしデータを埋め込むこととする．そのためホストでは，ファイルのチェックと左チャンネルのデータの取り出しを行い，そのデータをハードウェアに転送する．ハードウェアは受け取ったデータより検出を行う．検出した透かしの番号をホストに転送し，ホストで結果を表示する．

3.2 実装環境

本稿での実装環境は表 1 のとおりである．

表 1 実装環境

FPGA ボード	celoxcia 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
開発環境	DK-version3
開発言語	Handel-C

RC2000 ボード [7] には，Xilinx 製 xc2v8000-4 [8] と，SRAM が 4MB×6 バンク搭載されている．開発環境として DK-version3 [9] を利用し，開発言語は Handel-C [10] を利用した．これは，RC2000 に対応した API [11] があるということと，開発の短縮のためである．

3.3 乗除算回路の削減

乗算除算は，加減算に比べ演算に多くの回路を使い，複数のサイクルを消費する．そこで，乗算除算をなるべく省くことで効率的な演算を行う．

まず，式 (2) より $y(i)w(i)$ の乗算を削減する．$w(i)$ は－1 か 1 であるため，$w(i)$ が 1 である場合 $y(i)$ を足し，そうでない場合 $y(i)$ を引くことで S を求めることができる．また，式 (5) より $\Delta w/N$ の値は，透かしごとに決まる定数である．そのため，その結果を計算して与えておくことで除算を削減することができる．さらに，α の値を $1/2^7$ としておくことで掛け算をシフト演算で処理することが可能となる．

3.4 検出回路の並列化

図 1 で，検出回路の並列化について示す．本稿では，1 つの透かしを検出する回路を並列に構築し，複数の透かしの同時検出を行う．これにより，ソフトウェアでは達成が困難な複数の透かしの高速検出に対応する．また，1 つの透かしの検出回路内では，式 (5) の分子である $S - \frac{\Delta w}{N}|S|$ の計算と，分母の $\sum_{i=1}^{N}(\alpha|y(i)|w(i))w(i))$ の計算が並列に実行可能である．これにより，さらに高速に検出が可能となる．

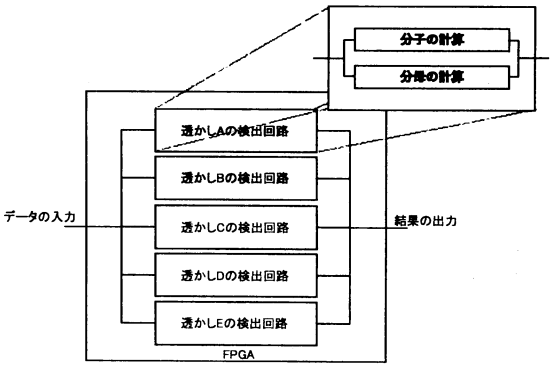

図 1 検出回路の並列化

3.5 メモリアクセスによる並列化の制限

複数の透かしを並列に検出するためには，それぞれの透かしデータが必要となる．しかし，透かしデータは数十キロビットあり，多くの透かしデータを FPGA 上に置くことは困難である．そこで，RC2000 ボード上の SRAM にデータを保存しておき，そのデータを利用する．

SRAM では 1 つのアドレスに 32 ビットのデータが保存でき，1 度のロードで 1 アドレス分しかデータを得られない．しかし，SRAM のブロックは 6 ブロック存在し，各ブロックへの同時アクセスは可能なため，1 度のロードで最大 6 アドレス分のデータを転送することができる．また，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，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，複数の透かしの同時検出は最大で 32×6 個の透かしまでとなる．

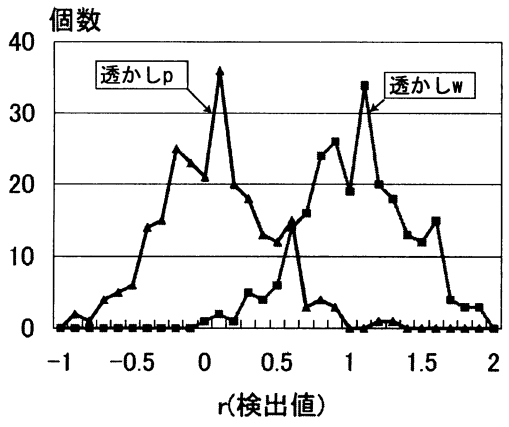

図 2　透かし w の検出時の r の度数分布

3.6　透かしの有無の判別

ソフトウェアの場合，式 (5) の r の平均値が 0.5 以上でその透かしを検出したことになる．しかし，平均値を求めると小数点演算及び除算を多く利用するためハードウェアでは好ましくない．そこで，本稿では別の方法で透かしの有無を判別する．

本稿のアルゴリズムで得られる r の値の分布は，図 2 のようになっている．図 2 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を利用し，透かし w の検出を行った例である．ちなみに，透かし w≒透かし p である．

図 2 より透かし w が入っているファイルでは，約 80 パーセント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 パーセント以下である．よって，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有無を判別できると考える．さらに，セグメント全体が無音である場合，式 (5) の分母は 0 となり r の値は無限になる．そこで，$0.5 \leqq r \leqq 2$ 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透かし有無を判別する．r の値を得るためには除算が必要なため，次のように式を変形し除算を削除する．

まず，計算の便宜上，式 (5) を式 (6) とする．S1 は S，S2 は $\frac{\Delta w}{N}|S|$，分母を αF とし，F は $\sum_{i=1}^{N}|y(i)|$ とする．

$$r = \frac{S1 - S2}{\alpha F} \tag{6}$$

r は 0.5 以上 2 以下の値を数えるため，次の条件を満たすセグメントを数える．

$$\frac{1}{2} \leqq \frac{S1 - S2}{\alpha F} \leqq 2 \tag{7}$$

さらに式 (8) に変形することで，除算を削減することができる．

$$\frac{1}{2}\alpha F \leqq S1 - S2 \leqq 2\alpha F \tag{8}$$

今回は，オーディオファイルの長さを 2 分以上 6 分未満を対象とした．2 分のオーディオファイルは，120 セグメントに分割され r の値が計算される．そこで，全体の 80 パーセント以下に閾値がなければならない．また，6 分のオーディオファイルの場合 360 セグメントに分割され r の値が計算される．もし，透かしが存在しない場合を考え，閾値が全体の 20 パーセント以上でなければならない．よって，閾値は 360 セグメントの 20 パーセント以上，120 セグメントの 80 パーセント以下とする．そこで，今回は閾値を 80 セグメントとした．

3.7　ハードウェア構築

本稿で構築した検出回路を図 3 に示す．まず，式 (8) の S1 と F の計算を並列に行う．S2 は S1 の結果より計算する．S2 は $\frac{\Delta w}{N}|S1|$ である．$\frac{\Delta w}{N}$ は小数であるため整数に変換する．そのため，全体を 2^{16} 倍する．しかし，F は $\alpha = 2^{-7}$ の掛け合わせがあることを考慮すると，式 (9) となる条件を数える．

$$2^9 * F \leqq S1 * 2^{16} - \frac{\Delta w * 2^{16}}{N}|S1| \leqq 2^{11} * F \tag{9}$$

$\frac{\Delta w \cdot 2^{16}}{N}$ は，透かしごとに決まる定数であるため，あらかじめ計算しておき，検出する透かしの数だけハードウェア上に与えておく．また，全体を 2^{16} 倍するため，データ幅は 48 ビットとしておく．

本稿では，図 3 の回路を並列することで，複数の透かしの同時検出をする．実際に，25 個の透かしを同時に検出する回路と，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 の 1 ブロックに 25 個の透かしデータを保存し，1 つの透かし検出回路を 25 並列にすることで実現した．また，50 個の透かし検出回路は，SRAM を 2 ブロックと，25 個の透かし検出回路を 2 つ利用して構成した．並列化された回路はそれぞれ異なる透かしの有無を判別し，透かしを検出した回路の透かし番号を出力値とした．

4.　評　　価

4.1　ソフトウェアによる検出速度

採用したアルゴリズムをソフトウェアにより実装を行い，予備実験として検出速度を測定した．

計測環境として，Pentium4 2.8GHz，メモリ 512MB，OS は WindowsXP である．利用するオーディオファイルは，wave ファイル，PCM，44.1Hz，ステレオとする．透かしの埋め込みは左チャンネルだけに埋め込む．計測方法として，ファイルチェックなどの時間は除き，メモリ上にオーディオファイルの左チャンネルのデータのみが存在する状態から計測を始め，検出の計算が終わり結果を表示するところまでを測定した．結果を表 2 に示す．

表 2　ソフトウェアでの検出時間

	file size	1 透かしの検出時間	1 透かしの検出速度
fileA	20.4MB	1633ms	99.93Mbps
fileB	12.9MB	1116ms	92.47Mbps

結果より，検出速度は 90Mbps 以上であるが，ファイルチェックなどの時間を含めると 50Mbps ぐらいまで下がってしまう．また，複数の透かしを検出する場合，透かしの数に比例して検出時間が増える．

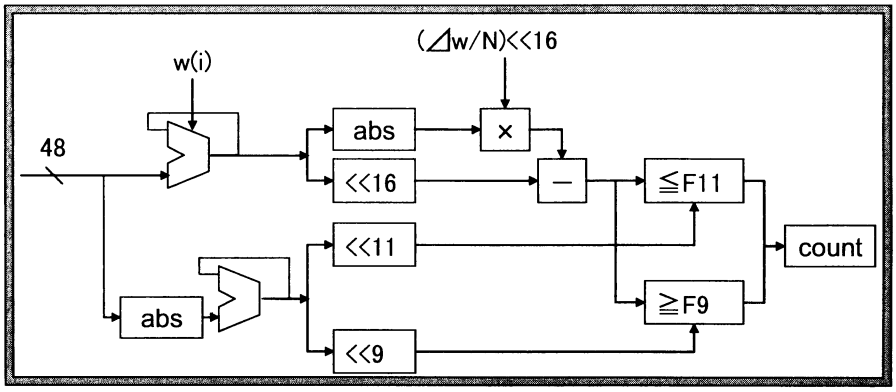

図 3　1 つの透かしの検出回路

表 3　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	27,715(93,184)	29%	24.727ns
50 個の透かし検出回路	53,660(93,184)	57%	24.867ns

表 4　ハードウェア及びソフトウェアでの検出時間
(ハードウェアの動作速度=30MHz)

	ソフトウェア	ハードウェア
25 透かしの検出時間　(検出速度)	38181ms(4.2Mbps)	500ms(326.4Mbps)
50 透かしの検出時間　(検出速度)	74351ms(2.2Mbps)	500ms(326.4Mbps)

4.2　検出回路量とクリティカルパス

本稿で構築した 50 個の透かしの検出回路と，25 個の透かし検出回路の回路量及びクリティカルパスを表 3 に示す．

今回利用した RC2000 ボードの場合，50 透かしの検出で FPGA 全体のおよそ 57%の回路を消費するため，およそ 90 個の透かし検出ができる．しかし，SRAM の制限として，最大 $32 \times 6 = 192$ 個の透かしまで検出が可能である．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と考える．

また，検出回路の並列度を上げても，クリティカルパスはほとんど増加しなかった．そのため，今回は 30MHz でハードウェアを動作させた．

4.3　検出時間の比較

50 個の透かしを検出する回路と 25 個の透かしを検出する回路を RC2000 ボード上で構築し，それぞれの検出時間を計測した．計測方法として，ソフトウェアから左チャンネルのデータを転送開始から，ハードウェアの結果を表示するまでの時間を測定した．また，ソフトウェアにより 50 個と 25 個の透かしを検出する時間をそれぞれ計測し，ハードウェアとの比較を行った．利用したオーディオファイルは，左チャンネルが約 20.4MB のデータを利用した．結果は表 4 に示す．

25 個の透かし検出を試みた場合，ハードウェアはソフトウェアの約 76 倍の検出速度で，50 個の透かし検出を試みた場合は約 148 倍の検出速度を達成した．今回は，動作速度を 30MHz としたため，さらに高速に検出が可能である．また，表 2 で示したように，クリティカルパスは検出回路の並列度にほとんど依存しないため，さらに多くの透かし検出回路を並列化することはより効果的である．結果的に，ネットワーク中で交換されるオーディオファイルからリアルタイムで検出することができる速度を達成したと考えられる．

5.　まとめと今後の予定

本稿では，オーディオ電子透かしの検出ハードウェアを次の点を考慮して構築した．

- 時間領域の利用したアルゴリズム
- 乗算除算回路の削減
- 並列化

この検出ハードウェアでは，複数の透かしを同時検出と高速な検出を可能とし，50 個の透かし検出を試みた場合，ソフトウェア検出と比べ 148 倍の速度を達成した．結果として，ネットワーク中で交換されるオーディオファイルからリアルタイムに電子透かしを検出する速度を達成したと考える．

今後，本稿で示した電子透かしの検出ハードウェアを用いて，不正配布を未然に防ぐシステムを提案し構築する予定である．また，検出回路規模をより小さくし，より多くの透かし検出に対応する予定である．

文　献

[1] 小野 束，電子透かしとコンテンツ保護，オーム社，2001.
[2] 松井 甲子雄，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/03/01_3.html
[4] P. Bassia and I. Pitas, Robust Audio Watermarking in the

Time Domain, (EUSIPCO'98), Rhodes, Greece, vol. I, pp. 25-28, 8-11 September 1998

[5] Laurence Boney and Ahmed H. Tewfik and Khaled N. Hamdy, Digital Watermarks for Audio Signals ,International Conference on Multimedia Computing and Systems,pp473-480,1996

[6] P. Bassia and I. Pitas, Robust audio watermarking in the time domain, IEEE Transactions on Multimedia, vol 3 no. 2, June 2001, pp. 232 -241

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

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

[9] Celoxica Ltd., DK Design Suite User Guide For DK Version 3.0,2004

[10] Celoxica Ltd., Handel-C 言語仕様マニュアル Version 2.1,2002.

[11] Celoxica Ltd., Platform Developer's Kit ADM-XRC Platform Support Library,SDK and Examples Manual ,2004.