修 士 論 文

# オーディオフィンガープリントシステムのハードウェアによる高速化に関する研究

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

高橋　宏幸

2009 年 3 月

修 士 論 文

# オーディオフィンガープリントシステムのハードウェアによる高速化に関する研究


指導教員　井口寧 准教授

審査委員主査　田中清史 准教授
審査委員　松澤照男 教授
審査委員　金子峰雄 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

610053 高橋　宏幸



提出年月: 2009 年 2 月

## 概 要

近年、インターネットをはじめとする高速ネットワークの普及に伴い、デジタルコンテンツ（オーディオファイル=楽曲）がオンラインでデジタルデータとして提供されることが増えてきている。しかし、便利になった反面、デジタルデータであるために、簡単にしかも一切劣化せず、オリジナルと一切同じ状態でネットワーク配信されてしまうということも起こっている。これは、権利関係がフリーな物でない限り問題である。著作権が存在するオーディオコンテンツを合法的なサービスとして提供する、オンライン課金システムがあったとする場合には、利用者は課金内容に応じたサービスの提供を受けられる事が必要となる。この時、圧縮等の加工を行ったファイルも、オリジナルのファイルもどちらも同じものであると識別出来るならば、より細やかなサービスの提供が行えると間がられる。また、誤識別率が低く、高速ネットワーク上でリアルタイムで識別処理を行う事が出来るシステムがあれば、非常に有用であると考える。処理時間の評価では、磯永の提案した、ハードウェアによるオーディオフィンガープリント ID による識別システムの結果である 315ms に対し、提案アルゴリズムをもちいてソフトウェア実装を行ったものでは 128ms の処理時間となり、約 2.4 倍の高速化が得られた。実際の実験では全て識別可能であり、そのままでは誤差率が導出出来ないため、識別結果は、実験から求めた標準偏差と平均をもちいた正規分布モデルであると仮定し、誤識別率を推定したところ、$1E^{-30}$ 以下の誤識別率であった。アルゴリズム的に、一番重い処理である特徴量抽出部の処理速度は、提案アルゴリズムでは 61ms であった。この速度を上回るハードウェア構成を推定したところ、1024 並列に出来たならば 32ms の処理速度になると想定でき、そのときの回路規模は対象 FPGA の Slice 数 92 %を使用したものと考えられる。

# 目 次

# 第1章　はじめに

## 1.1　研究背景

近年、インターネットをはじめとする高速ネットワークの普及に伴い、デジタルコンテンツ（オーディオファイル=楽曲）がオンラインでデジタルデータとして提供されることが増えてきている。しかし、便利になった反面、デジタルデータであるために、簡単にしかも一切劣化せず、オリジナルと一切同じ状態でネットワーク配信されてしまうということも起こっている。これは、権利関係がフリーな物でない限り問題である。著作権が存在するオーディオコンテンツを合法的なサービスとして提供する、オンライン課金システムがあったとする場合には、利用者は課金内容に応じたサービスの提供を受けられる事が必要となる。この時、圧縮等の加工を行ったファイルも、オリジナルのファイルもどちらも同じものであると識別出来るならば、より細やかなサービスの提供が行えると間がられる。また、誤識別率が低く、高速ネットワーク上でリアルタイムで識別処理を行う事が出来るシステムがあれば、非常に有用であると考える。

### 1.1.1　ファイルの識別法

オーディオファイルの識別技術としては、様々な種類の方法がある。大きく分けると電子透かし、DCD、電子指紋（フィンガープリント）などがあげられる。これらの代表的なもの [4] を、以下に列挙した。

#### 電子透かしを用いる技術

電子透かしは主に画像情報、音声情報等に使われており、人間の目や耳には区別がつかない程度に、画像/音声のデータ内に透かし鍵にもとづくランダムパターンを埋め込んでいる。識別処理時には、ランダムパターンから透かし鍵を用いて埋め込み情報を復号することができ、これによりファイルの識別が行える。デジタルデータは複製すると、オリジナルとコピーが区別出来なくなるが、あらかじめ透かしデータに、著作権者の情報を埋め込んでおけば、複製されたコピーにも、やはり透かしデータがついており、著作権の所在をはっきりさせることが出来る。電子透かしは一部のデータを切り取っても残るという特徴があり、オリジナルの一部を切り取り、別のコンテンツに流用するという事への抑止力ともなる。また、デジタルのままのコピーではなく、一度印刷などアナログ情報に変換

し、その後再度デジタルに変換する場合でも、電子透かしはアナログ時にもデジタル時にも残っている。

### DCD を用いる技術

あらかじめ IPR-DB（知的財産権データベース）に登録されたデータを検索する事で、デジタルコンテンツの識別が可能となっている。DCD（DistributedContentDescripptor) は流通コンテンツ記述子とよばれるものであり、これは、あらかじめデジタルコンテンツの最小限の属性情報をサブセットとしてコンテンツに付随させた物である。形態としては大きく分けて 2 種類あり、コンテンツと一体化（ただし、内部データに埋め込むわけでなく、ヘッダに追記する形）する種類と、コンテンツと完全に別ファイルとして持つ種類の方法がある。前者の一体型は、分離とそれによる改ざんがしにくいことが長所であり、短所は書き込み参照には専用の復号ソフトが必要となり、動画再生などでは再生に影響が出る事があげられる。後者のコンテンツと分離した場合は、多くの情報を持たせる事が可能な点や、構文解析などの手法で多くの数の DCD を検索できることが長所であり、短所は改ざん等が容易である点となる。

### 電子指紋を用いる技術

フィンガープリント（電子指紋）では、あらかじめファイルの特徴量をもととした、フィンガープリント ID を生成しておき、ID とオリジナルファイルとを 1 セットとし、データベース化しておく。識別時には対象ファイルからフィンガープリント ID を新規生成し、データベースに登録されている ID と、新規生成 ID とを比較すればよい。ファイルの特徴量が似ていれば、当然データベース登録 ID と新規生成 ID も似ており、ID 間の違いが 0 %に近くなる。逆に、特徴量が大きく異なっていれば、ID 間の違いは 50 %に近くなる。これより、ID 間の違いが 0 %に近ければ同一ファイル、50 %に近ければ相違ファイルとして識別できる。電子透かし、DCD は、ファイルが加工されたことしか判らないが、フィンガープリント ID による識別では、同一ファイルであるか否かを判別できる。また、ID 同士で比較を行うため、オリジナルのファイルそのものには加工を施さなくとも良いことが特徴である。また、電子透かしと同じく、アナログ化した後に再度デジタル化をされたとしても、特徴さえ似ていれば識別可能である。

## 1.2 研究目的

本研究では、以下の項目を達成することを目的とした。最終目標としては、磯永のハードウェアフィンガープリントシステムよりも、提案手法を用いたフィンガープリントシステムを高速にする事である。そのために、アルゴリズムの改良、ハードウェア化によるさらなる高速化、の 2 つの手法を用いる事とする。具体的な目標仕様としては、磯永のハー

ドウェアフィンガープリントシステムよりも高速となるように、1 ファイル識別が 315ms 以下で行える事、精度は同等であればよい事とした。本提案の第 2 段階で、ハードウェア化による高速化を考えているため、ハードウェ向けのアルゴリズムと考えられる Haitsma らのアルゴリズムを、大本のアルゴリズムとして採用することとした。

まず、どの部分を改良するかであるが、磯永のハードウェアフィンガープリントシステムの実験結果より、高速フーリエ変換処理部分が全処理時間の 90 %以上かかっていることが分かっており [3]、この部分を高速化することが非常に重要であると考える。元来フーリエ変換処理を必要としているのは、楽曲の波形データより、周波数スペクトルを取り出したいという必要性があるためである。周波数スペクトルとはどの周波数にどの程度のエネルギーが存在しているかを表したものであり、似ている演算に、ウェーブレット変換がある。ウェーブレット変換は、どの周波数の、どの時間の時にどの程度のエネルギーが存在しているのかを表す事ができる演算である。つまり、フーリエ変換が周波数とエネルギーのパラメータを持つならば、ウェーブレット変換は、時間と周波数とエネルギーのパラメータをもっていると言える。そのため、ウェーブレット演算はフーリエ変換の意味を包括し、置き換えも可能であろうと考えた。また、コンピュータを用いた計算等では連続値ではなく離散値を扱っているため、離散ウェーブレット変換（DWT）が用いられる。

つぎに、離散ウェーブレット演算を用い、ハードウェア化による高速化を考える。高速化手法としては並列化、パイプライン化がよく使われる手法である。ただし、どのような時にでも使えるわけではなく、演算が繰り返し処理であること。演算内容に依存性がないこと。が条件であげられる。また、ハードウェアはリソースが有限であり少ないため、以上の条件を多く満たし、回路量が少なくてすむウェーブレット変換の種類が求められる。

これら条件を満たすウェーブレットには、haar によるウェーブレット変換があげられる。haar のウェーブレットの構成は極めて単純であり、加算器 1 つと除算器一つ、減算器一つと除算器一つである。これらを用いればウェーブレット変換が行える。

## 1.3 本論文の構成

本論文は、全 5 章で構成されており、各章は以下のようになっている。1 章では、本研究の目的、研究背景と代表的な識別手法について述べる。2 章では、基本となる Haitsma らのアルゴリズムに用いられているフーリエ変換、提案手法に用いられているウェーブレット変換、オーディオファイルの構造、オーディオフィンガープリントに対する従来手法の一覧と Haitsma らのアルゴリズムの位置づけ、FPGA 等の一般的なハードウェアの基本構造について述べる。3 章では、Haitsma らのアルゴリズムと、提案したアルゴリズムについて述べる。4 章では、提案手法を用いたソフトウェア実装と、ハードウェア実装の推定に対する、実験および考察を述べる。5 章では、本研究のまとめを述べる

# 第2章　オーディオフィンガープリントIDによる識別システムについて

## 2.1　はじめに

オーディオフィンガープリントIDによる識別システムとは、先に述べた電子指紋を用いた識別技術のオーディオ向け識別システムである。基本的な特徴は同じであり、オーディオファイルからあらかじめIDを生成し、オーディオファイルとIDの対応をデータベース化しておき、識別時には対象ファイルから新規にIDを生成し、あらかじめデータベースに存在するID同士で比較を行い、識別する。フィンガープリントシステムはフィンガープリントID内に含まれている時間情報、周波数情報を利用している。情報抽出の手段として、短フーリエ変換を利用している。磯永は、高速かつ小回路量であること、圧縮や質の劣化に対して非常に頑健であること、特徴量生成アルゴリズムのステップは加算減算による簡素な構成であること、乗算および除算を多用しないこと、ハードウェア内で省スペースに格納するためフィンガープリントIDサイズが小さいこと、を考慮し、Haitsmaらによる、Philipsオーディオフィンガープリントシステムをハードウェア向けに改良し、システムを構成している。本研究では磯永のフィンガープリントシステムを参考にし、処理が重いと判明している高速フーリエ変換（FFT) 処理部分を主に高速化するために、オーディオファイルの加工としてよく用いられる、MP3、WMAなどの圧縮に対し、非常に強度があること、回路が加算器と減算器のみで作れ、回路規模が小さくなることが予想でき、演算中に依存関係も無いことから高速化も期待できるhaarのウェーブレットを用いることにより、Haitsmaらのアルゴリズムをハードウェア向けに改良すること、およびハードウェアにFPGAを用いこれを高速化する事を目的とし実装を行うこととした。

## 2.2　アルゴリズムについて

### 2.2.1　フーリエ変換とは

デジタルデータとは離散値であるので、離散フーリエ変換式の導出を行う。それにあたり、フーリエ級数展開、連続フーリエ変換、離散フーリエ変換の導出が必要となる。

### フーリエ級数展開

周期関数を $x(t)$ とする。ここで、$t$ は時間 $T_0$ は周期を表す。この関数がティリクレの条件を満たすとき、$T_0$ の整数倍の周期を持つ sin と cos の重ね合わせで表すことが出来る。つまり、以下の式が成立する

$$x(t) = \frac{a_0}{2} + \sum_{N=1}^{\infty} \{a_n \cos(n\omega_0 t) + b_n \sin(n\omega_0 t)\} \tag{2.1}$$

ここで、$\omega_0$ は、基本となる角速度であり、基本となる周期 $T_0$ に対応する基本周波数を $f_0$ とすると、$\omega_0 = 2\pi f_0 = 2\pi/T_0$ の関係が成立する。この $a_n$、$b_n$ という係数は、それぞれ $x(t)$ に含まれる $cos$ 成分と $sin$ 成分の大きさを表している。そして $xt$ が求まればこれらの値は決まる。逆に、$a_n$、$b_n$ を用いれば $x(t)$ を表現することが出来る。

### 連続フーリエ変換

複素係数 $C_n$ と周期 $T_0$ の積を、$x(jn\omega_0)$ とおくと、

$$C_n T_0 = X(jn\omega_0) \tag{2.2}$$

となり、$C_n$ を次元のない量とすると、$x(jn\omega_0)$ は時間の次元をもつ。この式を用い複素フーリエ級数展開の式を書くと、

$$x(jn\omega_0) = \int_{-\frac{T_0}{2}}^{\frac{T_0}{2}} x(t) e^{-jn\omega_0 t} dt \tag{2.3}$$

このとき、$T_0 \to \infty$ すなわち $\omega_0 \to 0$ の極限で、離散的な角周波数 $n\omega_0 (n = -\infty, ..., -1, 0, 1, 2, ..., \infty)$ は、連続的な角周波数 $\omega$ に置き換えられるものとする。このとき次の式が成立する

$$X(j\omega) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt \tag{2.4}$$

これが連続フーリエ変換の定義となる。次に、離散値を用いた離散フーリエ変換を導出する。

### 離散フーリエ変換

例えば整数を $N$ としてサンプリング周期 $T$、周期 $NT$ の関数を $x^*(t)$ とおくと、デルタ関数 $\sigma(t)$ を用い、次のように表すことが出来る

$$x^*(t) = \sum_{n=0}^{N-1} x(nT)\delta(t - nT) \tag{2.5}$$

ここで、1 周期分を考えたなら、$x^*(t)$ は離散信号 $x_0, x_1, x_2, x_3, ...., x_{N-1}$ の関数となる。次にこの館数 1 周期分について、これを複素フーリエ級数展開し、小さな数 $\epsilon$ とすると、次の式が求められる。

$$C_k = \frac{1}{NT}\int_{-\epsilon}^{NT-\epsilon} x^*(t)e^{-j2\pi k f_0 t}dt = \frac{1}{N}\sum_{n=0}^{N-1} x(nT)e^{-j\frac{2\pi kn}{N}} \tag{2.6}$$

次に、求めた複素フーリエ級数展開の係数 $C_k(k = 0, 1, 2, ..., N-1)$ について、$C_{k+N}$ を求めると、

$$C_{k+N} = \frac{1}{N}\sum_{n=0}^{N-1} x(nT)e^{-j\frac{2\pi kn}{N}}e^{-j2\pi n} = C_k \tag{2.7}$$

これより、複素フーリエ級数展開の係数 $C_k(k = 0, 1, 2, ..., N-1)$ にも、$N$ の周期性があることがわかる。いっぽう、離散周期信号は $C_k$ を用いて次のようにあらわせる。

$$x(nT) = \sum_{k=0}^{N-1} C_k e^{j\frac{2\pi kn}{N}} \tag{2.8}$$

これらより、離散フーリエ変換式が導出できる

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi kn}{N}} \tag{2.9}$$

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} e^{j\frac{2\pi kn}{N}} \tag{2.10}$$

**フーリエ変換まとめ**

フーリエ変換では、フーリエ級数展開で述べたように、$sin$ と $cos$ によって波形を表す事ができる事を利用していた。$sin$、$cos$ は無限に続く（周期性である）三角関数であるため、フーリエ変換を行った後のスペクトルは、時間情報がなくなり、ある周波数がどれだけのパワーをもっているかという情報のみを得ることとなる。

### 2.2.2 ウェーブレット変換とは

先に、信号に対してフーリエ変換を行った結果では、時間情報がなくなると述べた。時間情報も残し、周波数が時間と共にどのように変化するかを解析する手法にウェーブレット変換がある。フーリエ変換は $sin$ と $cos$ のみを基底とするが、ウェーブレット変換では、様々な種類の基底が存在する。このうち haar の基底を用いた haar のウェーブレット変換について述べる。

**haar のスケーリング関数**

スケーリング関数を次のように定義する。

$$\phi(x) = \begin{cases} 1 & 0 \leq x < 1 \\ 0 & \text{その他} \end{cases} \tag{2.11}$$

これを図 2.1 として示す。

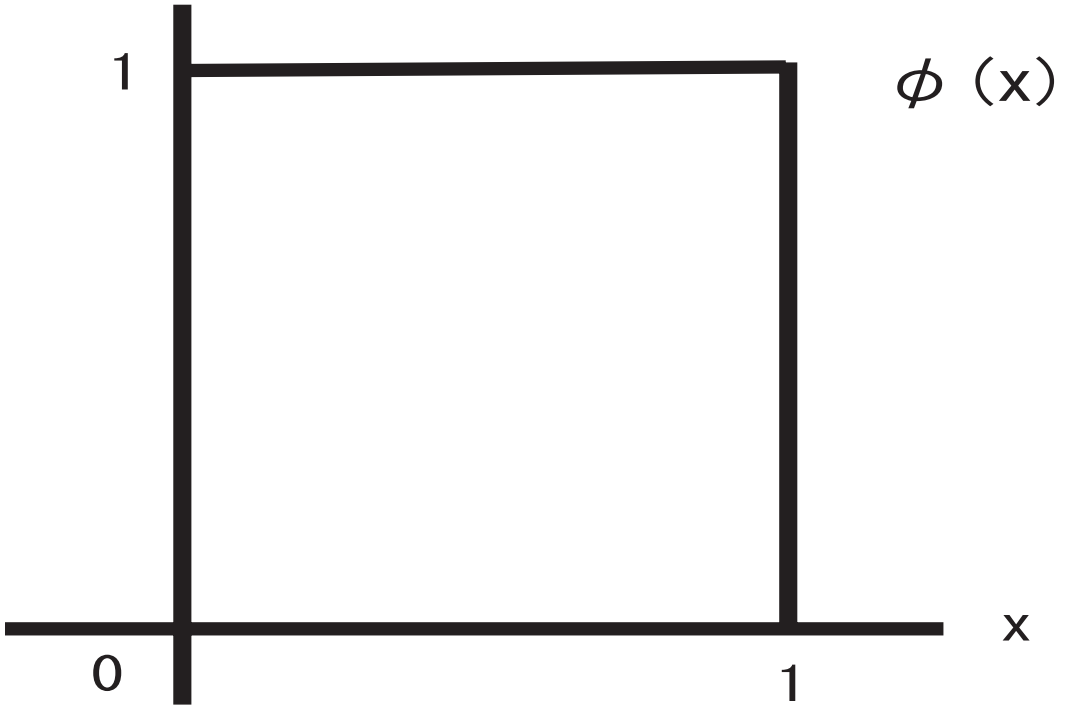



図 2.1: haar のスケーリング関数

一般に、スケール $2^j$ の信号は次のように書ける。

$$\begin{cases} f^j(x) = \sum_{k=0}^{N/2^j-1} c_k^{(j)} \phi(\frac{x}{2^j} - k) \\ c_k^{(j)} = \frac{c_{2k}^{(j-1)} + c_{2k+1}^{(j-1)}}{2} \end{cases} \tag{2.12}$$

**haar のマザーウェーブレット**

ウェーブレット母関数を次のように定義する。

$$\psi(x) = \begin{cases} 1 & 0 \leq x < 0/2 \\ -1 & 1/2 \leq x < 1 \\ 0 & \text{その他} \end{cases} \tag{2.13}$$

これを図 2.2 として示す。

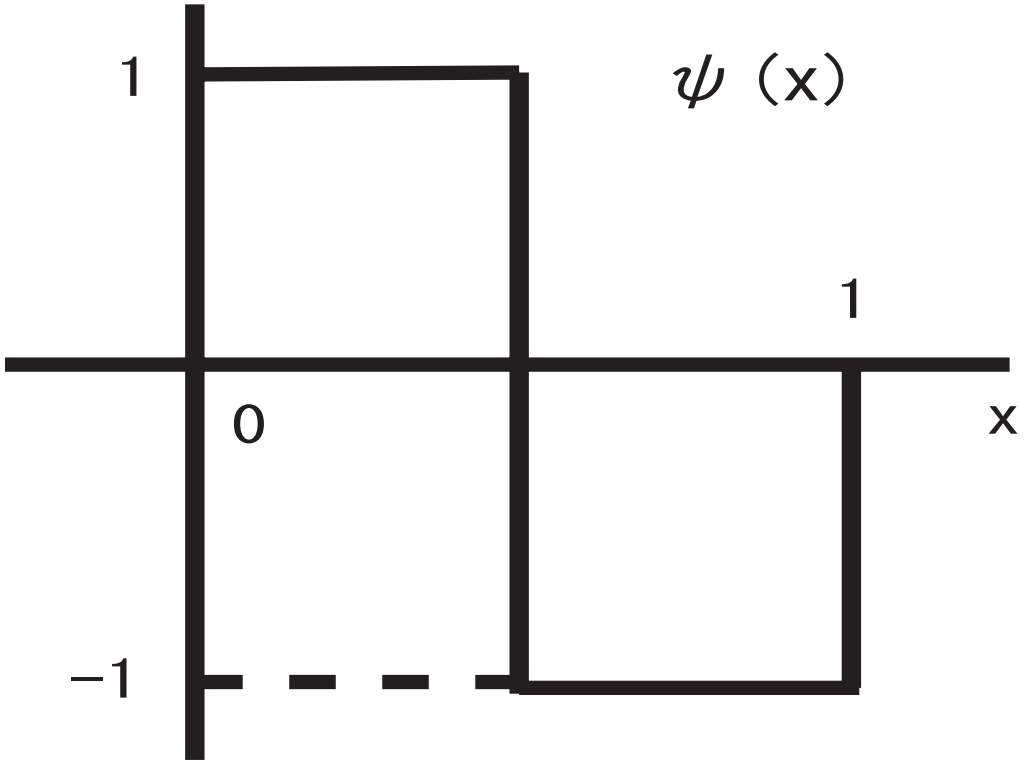



図 2.2: haar のウェーブレット母関数

これは、$[0, 1]$ を台とする関数であり、これを用いるとスケール $2^j$ の関数 $g^{(j)}(x)$ は次のように書ける。

$$g^{(j)}(x) = \sum_{k=0}^{N/2^j-1} \psi(\frac{x}{2^j} - k) \qquad j = 1, ..., n \tag{2.14}$$

ただし、次のようにおいた。

$$d_k^{(j)} = \frac{c_{2k}^{j-1} - c_{2k+1}^{(j-1)}}{2} \tag{2.15}$$

### ウェーブレット変換

$\psi_k^j(x)$ は、幅 $2^j$ の台上で+-1 であり、$\phi_0^{(n)}(x) = 1$ であるため、二乗積分は、

$$\psi(x) = \begin{cases} \|\psi_k^j\|^2 = \int_0^N \psi_k^{(j)}(x)^2 dx = \int_0^{2j} dx = 2^j \\ \|\phi_0^{(n)}\|^2 = \int_0^N \psi_0^{(n)}(x)^2 dx = \int_0^N dx = N \end{cases} \tag{2.16}$$

これから、ウェーブレット $\{\psi_k^j(x)\}$ および定数関数 $\phi_0^{(n)}(x)$ のノルムが、

$$\begin{cases} \|\psi_k^{(j)}\| = 2^{j-1} \\ \|\phi_0^{(n)}\| = \sqrt{N} \end{cases} \tag{2.17}$$

となる。また、空間 $V^{(0)}$ に属する任意の関数 $f(x)$ は、この直交基底に関して次のように展開できる。

$$f(x) = \sum_{j=1}^{n} \sum_{k=0}^{N/2^j-1} d_k^j \psi_k^j(x) + C_0^n \tag{2.18}$$

式 2.17 より、展開係数は次のように計算される。

$$\begin{cases} d_k^{(j)} = \frac{1}{2^j} \int_0^N f(x) \psi_k^{(j)}(x) dx \\ c_0^{(n)} = \frac{1}{N} \int_0^N f(x) dx \end{cases} \tag{2.19}$$

与えられた関数 $f(x)$ からこれらの展開係数を計算すること、およびその展開係数から式 2.18 によって、$f(x)$ の値を計算することを、あわせてウェーブレット変換とよぶ。

### 一般のウェーブレット（2 スケール関係）

ウェーブレット変換の基底に用いる事の出来る関数の定義を簡単に述べると、各レベルでスケールの小さい表現の全体は、スケールの大きい表現を含んでいるという関係が成り立っていなければならない。すなわち、スケーリング関数 $\phi(x)$ を 2 倍に広げると、元のスケーリング関数 $\phi(x)$ の平行移動の線形結合で表されなければならない。つまり、

$$\phi(\frac{x}{2}) = \sum_{k=-\infty}^{\infty} p_k \phi(x-k) \tag{2.20}$$

ただし、$\sum_{k=-\infty}^{\infty}$ は範囲を限定しないという意味であり、$k$ のある値以上とある値以下では、$p_k = 0$ であるとする。この式を、2 スケール関係式とよぶ。haar のウェーブレットでの、2 スケール関係を図 2.3 に示した。

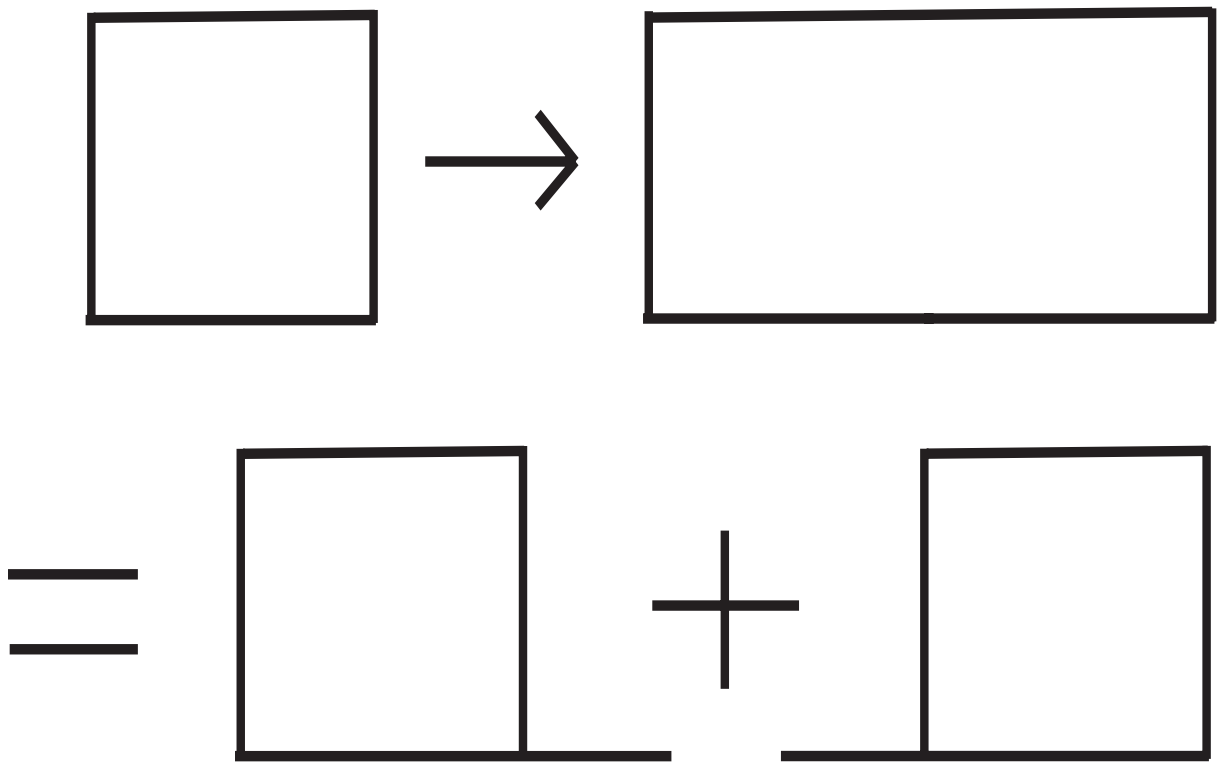


図 2.3: haar のスケーリング関数の 2 スケール関係例

ウェーブレット母関数 $\psi(x)$ は、信号のスケールの大きくする前と、その後の表現の差を表すものであり、スケーリング関数 $\phi(x)$ に応じて定まる。このことから、ウェーブレット母関数 $\psi(x)$ を 2 倍に広げると、スケーリング関数 $\phi(x)$ の平行移動の線形結合で、次のように表さなければならない。

$$\psi(\frac{x}{2}) = \sum_{k=-\infty}^{\infty} q_k \psi(x-k) \tag{2.21}$$

haar のウェーブレットでの階層的表現を、を図 2.4 に示した。

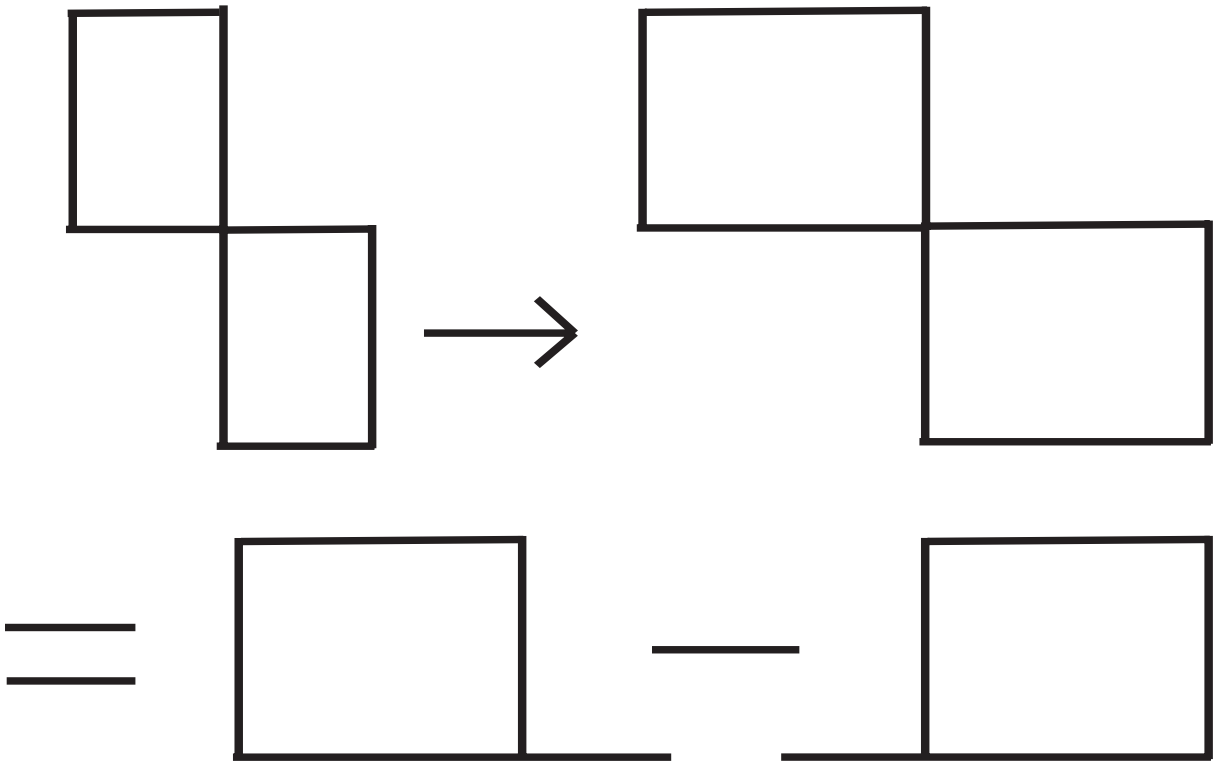


図 2.4: haar のウェーブレットでの階層的表現例

### 2.2.3 wav ファイルの構造（入力データ）

本研究では、もっともよく使われている音声形式である PCM を入力データとして用い、オーディオフィンガープリント ID の生成、ID による識別を行うこととする。PCM をデータとして持つファイルに wav がある。wav ファイルは。音声信号をデジタルデータにサンプリングしたものを格納する形式（チャンク）を規定している。符号化方式については規定が無く任意の形式が利用できる。PCM 以外にも、デコード方法を記述している

Codec を追加すれば、Mpeg-1 Audio Layer　(MP3) 等に代表される圧縮方式も利用できる。wav ファイルの構造を図 2.5 に示す。

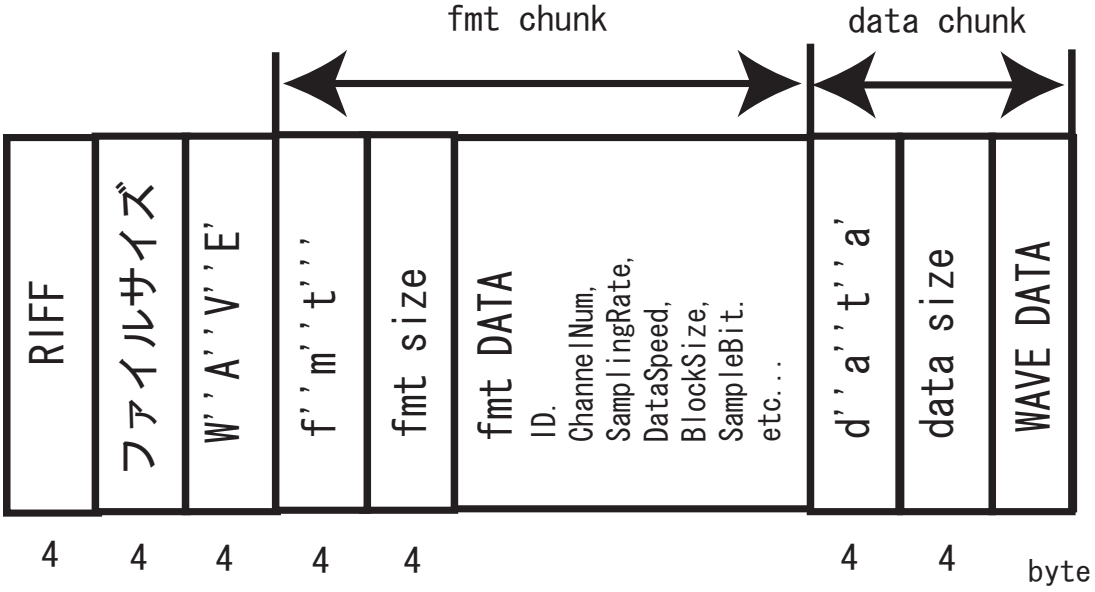



図 2.5: wav データの構造例

先頭から、RIFF、ファイルサイズ、W”A”V”E’、f”m”t”’、fmt DATA、d”a”t”a’、data size、WAVE DATA などが並んでおり、wav ファイルのフォーマット形式の判別に用いられる。全ての情報が無くともフォーマット形式の識別は可能である。実際に音声信号データが存在しているのは WAVE DATA 以降の部分である。今回は、入力データにはサンプリングレート 44.1kHz、チャンネル数 2、ビット数 16bit つまり、1 秒間に 44100 点のデータがありその値は signed int 16 の+-32768 となる。

### 2.2.4　オーディオフィンガープリントに関する研究

#### オーディオフィンガープリントシステムの従来研究

フィンガープリントシステムはフィンガープリント ID 内に含まれている時間情報、周波数情報を用い、特徴量を抽出している。手法としては、スペクトル扁平の特徴、スペクトルのピークの特徴、フーリエ係数の特徴、メル周波数ケプストラム係数の特徴、周波数時間のエネルギー差の特徴などが用いられている。具体的には、磯永ら [1] は、Haitsma ら [2] による、Philips オーディオフィンガープリントシステムのアルゴリズムを用いて、ハードウェアを用いたフィンガープリント ID 識別システムを構築している（以下、磯永らの、オーディオフィンガープリント ID による識別システムを、「磯永識別システム」とよぶ)。磯永らが、Haitsma らのアルゴリズムを選定した理由としては、ハードウェア実

装に向いており、高速かつ小回路量である。（MP3 などの）圧縮に対して非常に頑健である。乗算および除算が少なく、回路量がすくなくなる。生成 ID がコンパクトである。などをあげている。

## 2.3 ハードウェアについて

### 2.3.1 ロジックデバイスとプログラマブルロジックデバイス

論理仕様をプログラムする事により、様々な論理回路を構成することができるロジックデバイスを総称して、Programmable Logic Device(PLD) とよぶ。方式による歴史としては、1975 年にフューズ方式でプログラム可能な Field Programmable Logic Array(FPLA) が発表され、1978 年にバイポーラ素子を採用した Programmable Array Logic(PAL) が、さらに 1980 年代には CMOS 素子を採用し、電気的に消去/再書き込み可能な Generic Array Logic(GAL) が、これら単一の AND-OR アレイ構造（プロダクトターム）を持つものの総称を SimplePLD(SPLD) とよんでいる。その後、AND-OR アレイ構造を複数組み合わせた ComplexPD(CPLD) が、1985 年には SRAM ベースの Look-Up Table(LUT) に基づく基本論理ブロックをアレイ上に配置した Field Programmable Gate Array(FPGA) が発表された。これらは全て広義の PLD に含まれている。PLD の現状は、半導体製造技術の向上と共に、集積度、速度が急速に向上したことから主流は FPGA となっている。ロジックデバイスにはディスクリート素子も含め様々な物があるが、PLD/FPGA の位置づけを図 2.6 に示した。

次に、FPGA の構造について述べる。これを、図 2.7 に示した。

FPGA は、再構成可能な基本セルである、Configurable Logic Block(CLB)=論理ブロック、が、アレイ上に並べられており、これらをどのように構成するのかをプログラムによって記述することにより、目的の論理回路を生成できる。論理ブロックの内部は、FPGA であれば一般的に 4 入力の Look Up Table(LUT) と、FlipFlop(FF) で構成されている。この、論理ブロック内構造を図 2.8 に示す。

FPGA はクロックそのものは低速であるが、並列化、パイプライン化などの手法を用いることにより、一般的な CPU よりも高速に演算する事が可能である。

## 2.4 まとめ

本項では、アルゴリズム側の説明として、フーリエ変換、ウェーブレット変換、Wav データの構造、オーディオフィンガープリントに関する従来手法の説明を行った。これにより、フーリエ変換は $sin$、$cos$ を元にしているため、周期波形に用いるべきものであることがいえ、ウェーブレット変換にはそのような制限はなく、周期波形でないものに対しても適用可能なことが言える。

ハードウェア側の説明より、FPGA 等のプログラマブルなロジックデバイスを用い、計

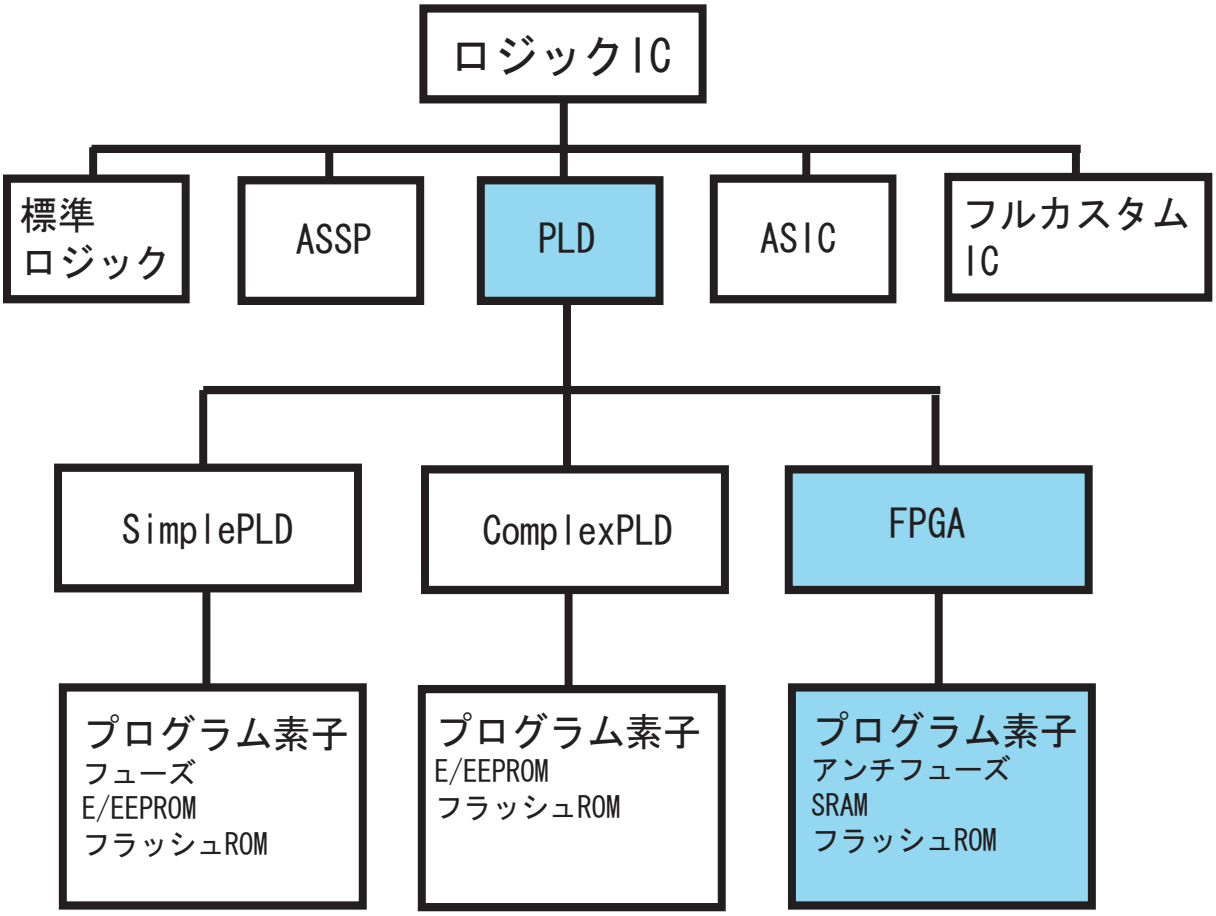



図 2.6: PLD/FPGA の位置づけ

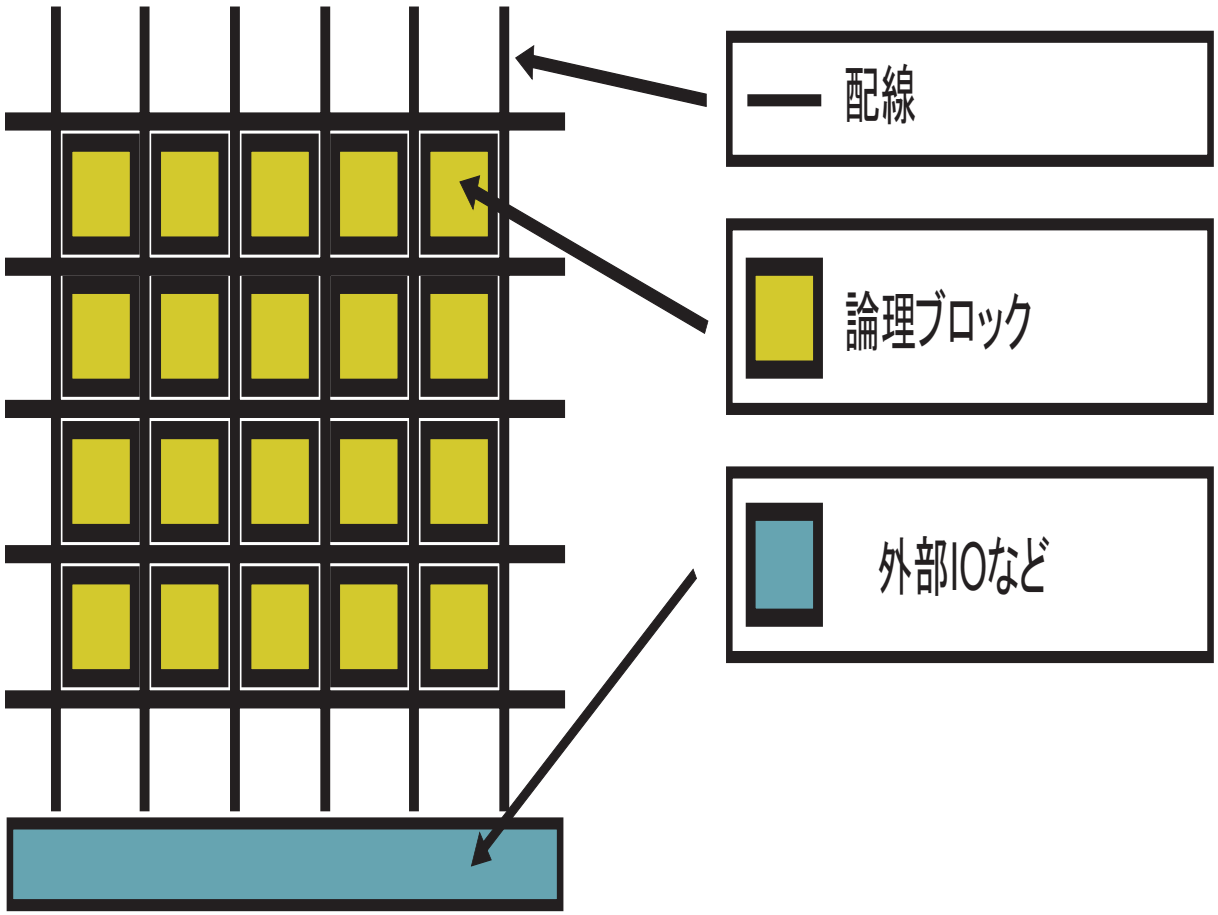
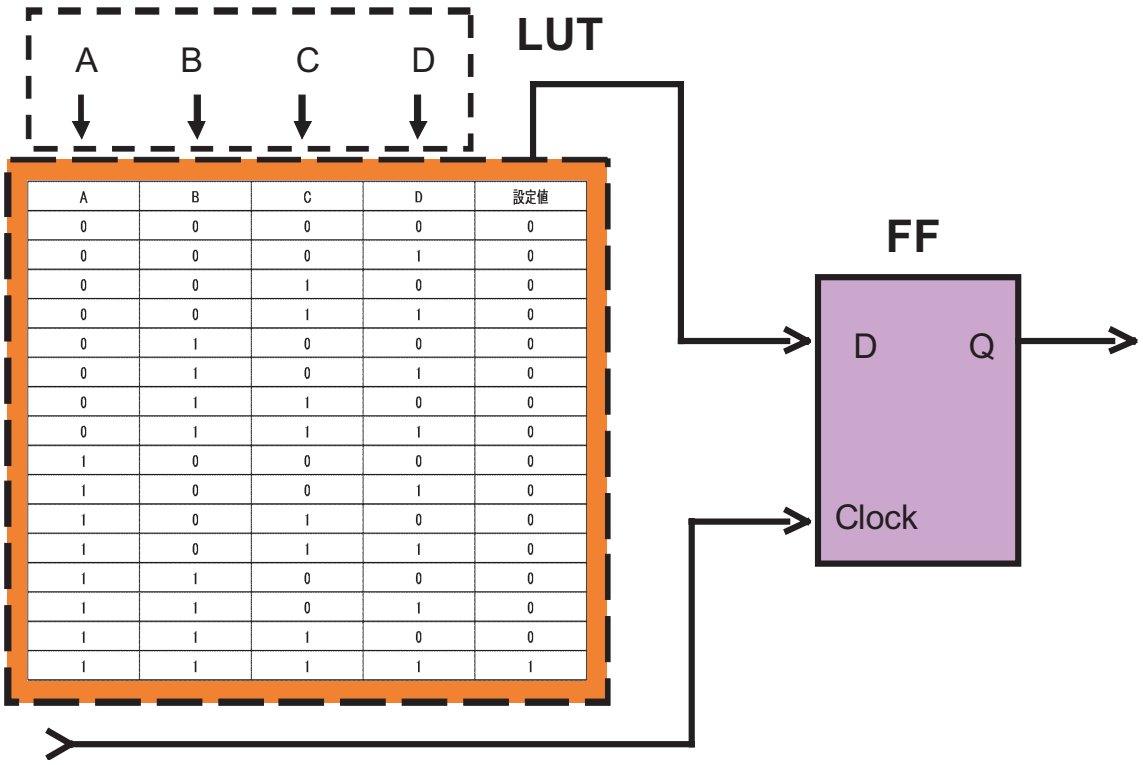



図 2.7: FPGA 内部の構造




| A | B | C | D | 設定値 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

図 2.8: 論理ブロック構造例

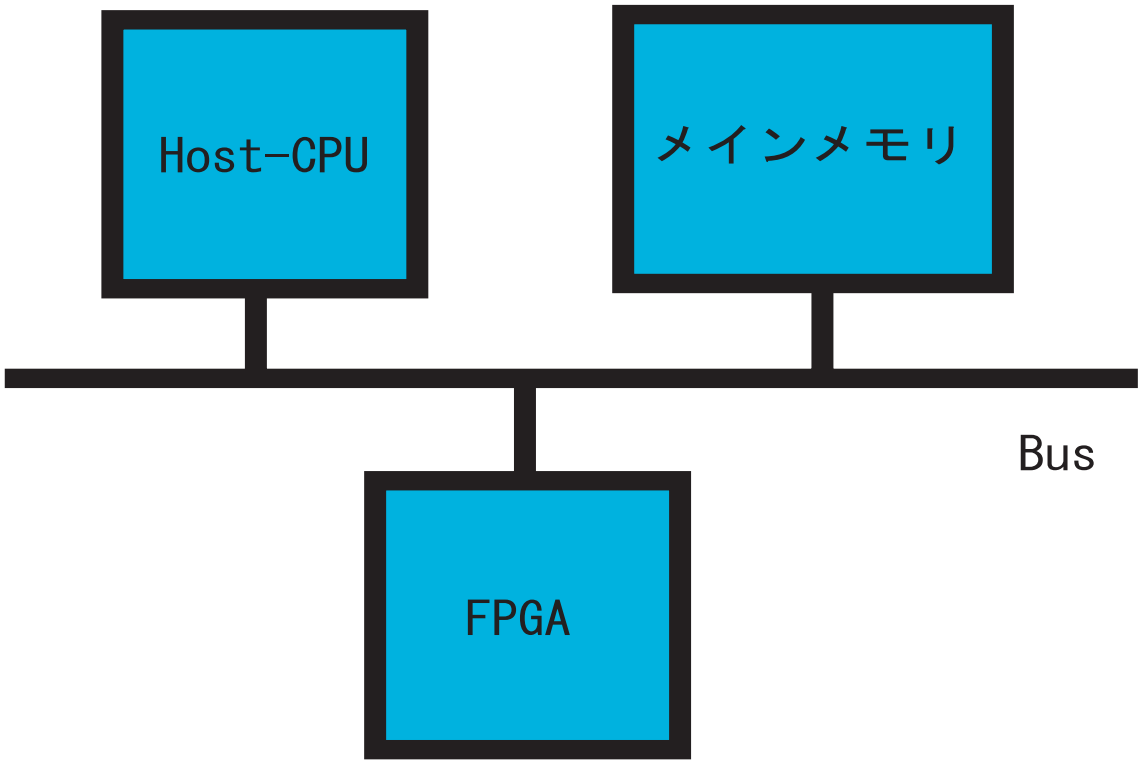



図 2.9: FPGA ボードがバスにつながっている場合の例

算が依存関係が少なく、繰り返し数が多いような場合は並列化がしやすく、ハードウェアを並列化できるような形に、演算内容を持って行ければソフトウェアの一般的な CPU を用いた処理よりも高速化出来ることが言える。

# 第3章　Haitsmaらのアルゴリズムと、提案アルゴリズム

## 3.1　はじめに

本研究では、磯永による、ハードウェアを用いたオーディオフィンガープリントシステムよりも、高速化することを目的としている。磯永のフィンガープリントシステムは、アルゴリズムにHaitsmaらの物をほぼそのまま用いており、アルゴリズムレベルではなく、ハードウェアレベルで並列化、パイプライン化などの手法を用い、ハードウェアに特化した高速化を行っていた。本提案では、アルゴリズムをハードウェア化に適したものにすることにより、アルゴリズムレベルにおいても高速化を行うアプローチをしている。本章では、磯永のフィンガープリントシステムで用いられているHaitsmaらのアルゴリズムを説明し、次に、提案アルゴリズムを説明する。

## 3.2　Haitsmaらのフィンガープリントアルゴリズム

図3.1に、Haitsmaらが提案したフィンガープリントシステムのID生成部を示す。

入力としては、オーディオファイルを想定し、最終出力には、256フレーム x 32ビット = 8192bitのIDを生成する事を目的としている。まず、オーディオファイルの370ms分の長さを1フレームとし、11.6msずつずらしながら257フレーム分足し合わせている。これは、図3.1の「1$frame$目、0$ms$〜370$ms$」「2$frame$目、0$ms$ + 11.6$ms$〜370$ms$ + 11.6$ms$」･･･に対応している。257フレーム分全ての長さは、370$ms$ +（11.6$ms$ × 256回）= 約3.3$s$となる。つぎに、窓関数を使い1フレームごとに重み付けする。これは次段で高速フーリエ変換を使う事になっているが、フーリエ変換は、周期性のある波形にたいし、用いる事が出来るという定義のため、本来は周期性のない入力波形であるが、窓関数を用いることにより開始時と終了時の値を等しくし、擬似的に窓の範囲内では周期性のある波形とする必要があるためである。これは「窓関数」の箇所に対応している。窓関数にはHanningWindowを用いており、特徴としては小さい波形のスペクトルを検出するのに向いている。

HanningWindowは、

$$h(n) = \begin{cases} 0.5 - 0.5\cos(\frac{2\pi n}{N-1})(0 \leq n < N-1) \\ 0 \text{ その他} \end{cases} \tag{3.1}$$

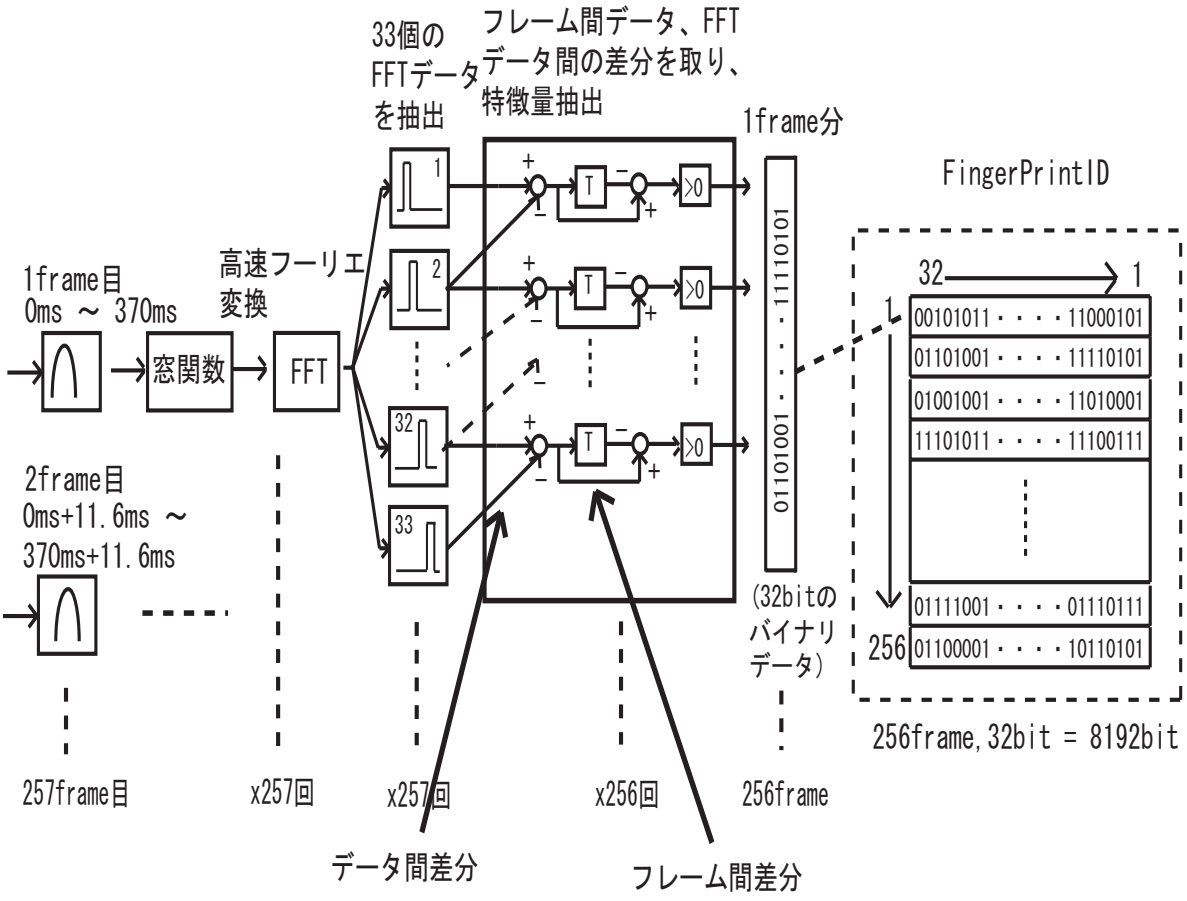



図 3.1: Haitsma らが提案したフィンガープリント ID 生成アルゴリズム、概念図

で与えられ、概形図は図 3.2 となる。

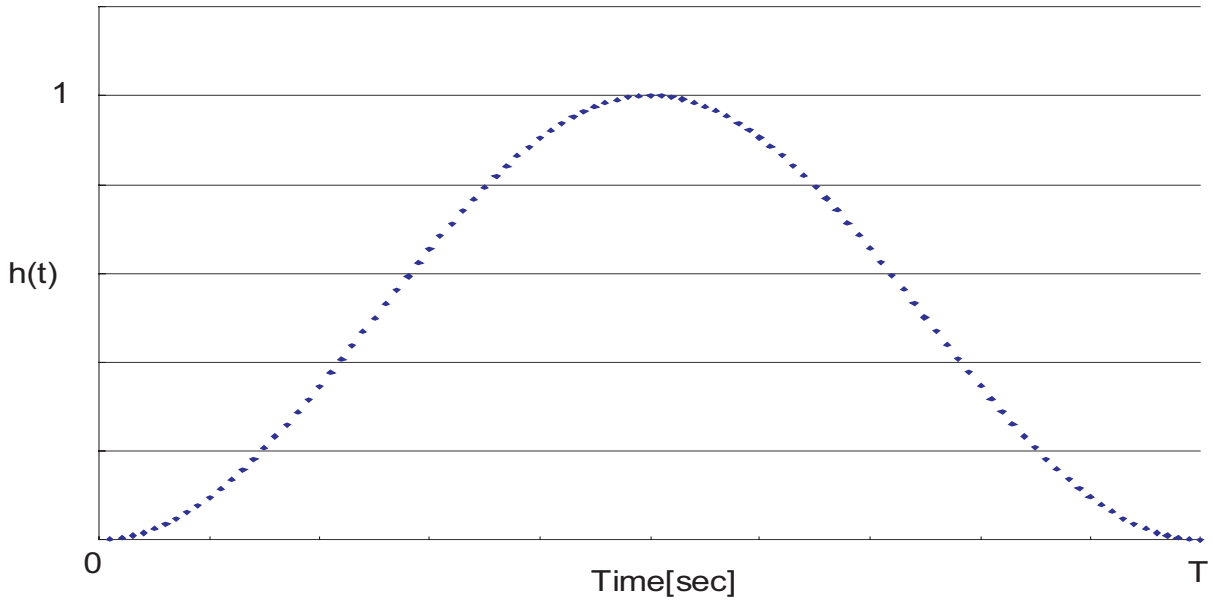



図 3.2: HanningWindow

ここまでで、370ms/1 フレームに HanningWindow をかけた波形が求められた。次にこの波形を使い高速フーリエ変換により 33 個のデータを抽出する。高速フーリエ変換を用いる事により、1 フレーム分の入力データから 33 個の各周波数でのエネルギーが抽出できる。これは「高速フーリエ変換、33 個の FFT データを抽出」に対応している。次の段の処理は、「フレーム間データ、FFT データの差分を取り特徴量抽出」である。ここで述べているように、フレーム間データの差分をとる必要があるため、高速フーリエ変換処理までの演算は、257 フレーム分全て終わらせておく。特徴量抽出段の演算を行うためには、257 フレーム分 × 33 個の高速フーリエ変換結果が必要であり、フレーム間の高速フーリエ変換結果、高速フーリエ変換結果間の差分を取る事によって求められる。特徴量抽出段の結果は、次の定義の形式になる。

フレーム n の、周波数帯 m のエネルギーを、$Ener[n][m]$ と表す。これらのエネルギーの差分、$EnerDiff[n][m]$ は時間軸情報と周波数軸情報から計算され、

$$EnerDiff[n][m] = (Ener[n][m] - Ener[n][m+1]) - (Ener[n-1][m] - Ener[n-1][m+1]) \tag{3.2}$$

となる。このシステムにおいては、$n = 0$〜256、$m = 0$〜32 に対応する。最終的に、8192bit のフィンガープリント ID を生成したいため、$EnerDiff[n][m]$ の情報を、ビットで保持する事としてする。そのために 1 フレームあたり 32bit のサブフィンガープリント ID（1 フレーム分=32bit の ID）$F[n][m]$ は、

$$F[n][m] = \begin{cases} EnerDiff[n][m] > 0 \text{ のとき } 1 \\ EnerDiff[n][m] <= 0 \text{ のとき } 0 \end{cases} \tag{3.3}$$

を用いて算出する。これを 256 フレーム分繰り返すことにより、256x32 = 8192bit のフィンガープリント ID が生成できる。これは、「FingerPrintID」に対応しており、縦軸はフレーム数（時間軸)、横軸は、高速フーリエ変換結果から抽出した周波数情報に対応しており、1 か 0 の bit の値は、エネルギーに対応している。つまり、擬似的にある時にある周波数がどれだけのエネルギーをもっているかを表しているとも言える。そのため、わずか 8192bit と小さな容量ながら、元ファイルの特徴を反映しており、識別対象楽曲が様々な加工等により変化しても、識別が可能となる。また、何をもって同一楽曲か、相違楽曲かであるかは、あらかじめデータベースに登録しておいた ID と、新規に生成したフィンガープリント ID の差を、BitErrorRate(BER) とよび、このパラメータ数値により、BER が閾値以下であれば同一楽曲、BER が閾値を超えていれば相違楽曲となる。このシステムにおいては閾値は 0.35 に設定されている。

## 3.3 提案したフィンガープリントアルゴリズム

### 3.3.1 はじめに

本研究では、Haitsma らのアルゴリズムの高速フーリエ変換処理部分を離散ウェーブレット変換処理へと置き換え、次段の特徴量抽出部分も離散ウェーブレット変換処理に応じた形に改良を行った。なぜ置き換えを行うという発想に至ったかというと、フーリエ変換は「周波数に対するエネルギー量を抽出」する事が出来、ウェーブレット変換は「ある時間の周波数に対するエネルギー量を抽出」する事が出来るためウェーブレット変換は、フーリエ変換の情報に加え時間情報を持つため置き換えを行ってもかまわないであろうと考えたためである。ただし、元々のアルゴリズムは、フーリエ変換によるデータに対して意味を持つように次段が作られているため、単純に高速フーリエ変換と離散ウェーブレット変換を置き換えるだけでは無く、離散ウェーブレット変換によるデータに対し意味を持つような形に変更する必要があった。離散ウェーブレット変換へ置き換え、特徴量抽出部を改良し、ハードウェアで高速に動作するようなアルゴリズムを提案した。提案アルゴリズムによる ID 生成フローを、図 3.3 に示す。

これは、Haitsma らのアルゴリズムを元としているため、多くの部分は同じである。入力としては、オーディオファイルを想定し、最終出力には、256 フレーム×32 ビット = 8192$bit$ の ID を生成する事を目的としている。Haitsma らのアルゴリズムと、提案アルゴリズムの主な相違点に関しては表 3.1 に示した。

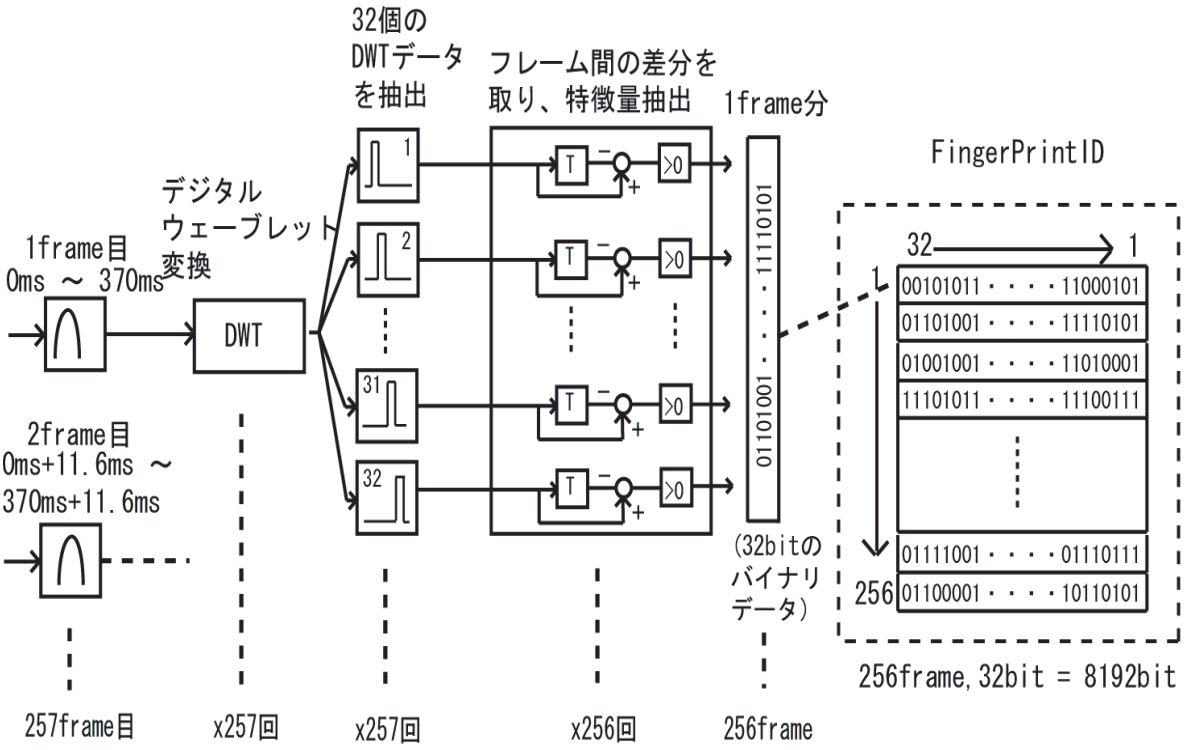



図 3.3: 提案アルゴリズムによるフィンガープリント ID 生成、概念図

| | Haitsma らのアルゴリズム | 提案アルゴリズム |
|---|---|---|
| 入力データ | オーディオデータ | オーディオデータ |
| フーリエ/ウェーブレット変換 | 窓関数 | なし |
| | フーリエ変換 | haar-ウェーブレット変換 |
| | 33bit、257frame | 32bit、257frame |
| 特徴量抽出 | 33bit 間分 & 257frame 間差分 | 257frame 間差分のみ |
| 出力データ | 256frame x 32bit=8192bit | 256frame x 32bit=8192bit |

表 3.1: アルゴリズム間の違い

### 3.3.2　提案アルゴリズムの説明

#### 入力データ部

まず、入力データがはじめにくる。これは図 3.3 中の「1 フレーム目 0ms〜370ms」･･･に対応しており、ここは Haitsma らのアルゴリズムとは変わらない。1 フレームあたり、オーディオファイルの 370ms 分の長さ（44.1kHz サンプリング周波数ならば、16384 点に相当）である。2 フレーム目以降は 11.6ms ずつずらしている。257 フレーム分全ての長さは、$370ms +$（$11.6ms \times 256$ 回）= 約 $3.3s$ となる。

#### 省略箇所（窓関数省略）

次の段は変更点であり、窓関数を省略した。Haitsma のアルゴリズムにおいては、フーリエ変換を用いるため、その前段階として窓関数が必要であった。これは、フーリエ変換であれば、変換対象の波形は周期性のあるものでなければならないため窓関数を用い、初点と終点の値を等しくする事で、擬似的に周期性のある波形であるとするためであった。これに対し、ウェーブレット変換は、変換対象の波形には周期性が必要ないため、窓関数を使う必要がなかった。

#### DWT 変換部

その次の段も変更点である。「DWT 変換」「32 個の DWT データを抽出」に対応しており、Haitsma では高速フーリエ変換処理を行っていた部分を、離散ウェーブレット変換に置き換えた。一般に、ウェーブレット変換は、Lo 側：(LPF = Lo-Pass-Filter = 低域側周波数のみ通過させ、高域側周波数を遮断するフィルタ）と、Hi 側：(HPF = Hi-Pass-Filter = 高域側周波数のみ通過させ、低域側周波数を遮断するフィルタ）の 2 つに分解できる。また、Lo に対し、さらにウェーブレット演算を行うと、Lo-Lo、Lo-Hi、に分けることが出来、Hi に対しウェーブレット演算を行うと、Hi-Lo、Hi-Hi に分けることが出来る。つまり、繰り返しウェーブレット演算を行う事により、より細やかな周波数成分ごとに分解することが可能となる。本提案では、基本的に Lo 側の結果のみを繰り返し用い、データ数が 32 個となったときの結果のみ、Hi 側を用いた。元オーディオデータには、44.1kHz サンプリング周波数のものを想定したため、サンプリング定理により、元波形には $0Hz$〜$20.05kHz$ までの情報が含まれている。$0Hz$〜$20.05kHz$ のデータに対し、ウェーブレット変換を行うと、Lo 側である $0Hz - 10.025kHz$ と、Hi 側である、$10.025kHz - 20.05kHz$ と、半分ずつの周波数成分をもつデータに分解出来る。1 回目（Lo）¿ 2 回目（Lo）¿ 3 回目（Lo）¿ 4 回目（Lo）¿ 5 回目（Lo）¿ 6 回目（Lo）¿ 7 回目（Lo）¿ 8 回目（Lo）¿ 9 回目（Hi）と取ると、9 回目（Hi) の結果は 32 点$-40 - 80Hz$ であった。

**特徴量抽出部**

次の段で特徴量を抽出する。これは、「フレーム間の差分を取り特徴量抽出」に対応している。ここも Haitsma らのアルゴリズムと異なる点であり、提案手法ではデータ間で差分を取る計算を省略し、フレーム間差分のみを用いている。理由として、Hi 側データを導出する過程において、ウェーブレット変換ではデータ間差分をとる演算を用いており、特徴量抽出段で再度差分をとる計算を行う必要はないと考えたためである。また、フレーム n の、周波数帯 m のエネルギーを、$Ener[n][m]$ と表すと、これらのエネルギーの差分、$EnerDiff[n][m]$ は時間軸情報と周波数軸情報から計算され、

$$EnerDiff[n][m] = Ener[n][m] - Ener[n-1][m] \tag{3.4}$$

となる。このシステムにおいては、n=0〜256、m=0〜32 に対応する。最終的に、8192bit のコンパクトなフィンガープリント ID を生成したいため、$EnerDiff[n][m]$ の情報を意味を保ったまま、データサイズを小さくするためにビットで保持する事とする。そのために 1 フレームあたり 32bit のサブフィンガープリント $F[n][m]$ を、

$$F[n][m] = \begin{cases} EnerDiff[n][m] > 0 \text{ のとき } 1 \\ EnerDiff[n][m] <= 0 \text{ のとき } 0 \end{cases} \tag{3.5}$$

を用いて算出する。これを 256 フレーム分繰り返すことにより、$256 \times 32 = 8192bit$ のフィンガープリント ID が生成できる。ウェーブレット変換は、フーリエ変換による結果である、「ある周波数にどれだけのエネルギーがあるか」という事よりも時間軸情報が増え、「ある時間の時には、ある周波数にどれだけのエネルギーがあるか」という事を知ることが出来る。

## 3.4 まとめ

本章では Haitsma らのアルゴリズムと、本提案アルゴリズムについて説明した。本提案アルゴリズムも、Haitsma のアルゴリズムと同様な特徴を持ち、元となる楽曲が様々な加工により変化したとしても、特徴が似ていれば識別が可能である。なお、楽曲の長さに関わらず、3.3s 分のデータさえあれば識別である。以上の動作をまとめ、図 3.4 に示した。

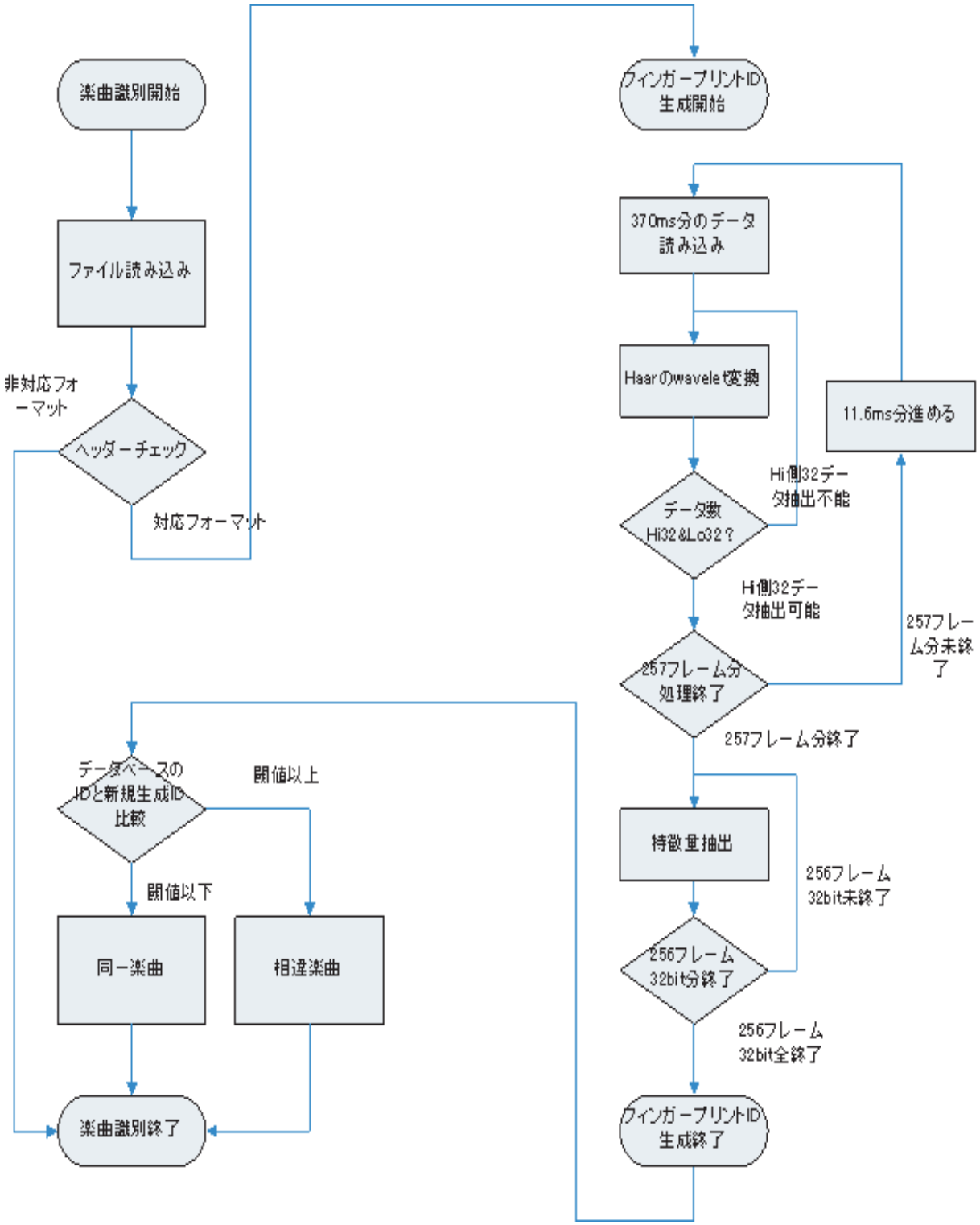



図 3.4: 提案アルゴリズムフローチャート

# 第4章　実験および結果

## 4.1　はじめに

本章では、前章で述べた提案手法の実験および評価について述べる。まず、提案アルゴリズムのソフトウェアによる実装と評価、次にハードウェアによる実装と評価について述べる。

## 4.2　提案アルゴリズムのソフトウェア実装

### 4.2.1　はじめに

本提案では、オーディオフィンガープリントシステムのアルゴリズムに、提案したアルゴリズムを用いた。そのため、Haitsma らのアルゴリズムを用いた場合と比較すると、識別性能、処理速度、等は当然異なった結果になることが予想される。そのため、識別は可能であるのか、そのときの誤識別率はどの程度であるか、処理速度はどの程度であるか、等を確かめるため、提案アルゴリズムを、ソフトウェアに実装し、実験により検証を行った。

### 4.2.2　ソフトウェア開発、実行環境

まず、表 4.2.2 にソフトウェアの実験環境を示す。

ソフトウェア開発には、OS として WindowsXP-Pro を用いた。これは、音声加工用ソフトウェアが Windows には豊富にあるため選択した。また、開発言語、開発環境として VisualStudio2005（C++）を用いた。C++言語を用いたのは、C 言語系は一般に高速に処理が行えるためである。C++言語を使用したが、異なる OS（例えば Unix、Linux 等）に移植する場合も考え ANSI-C に準拠しプログラムを行った。（ただし、Windows と Linux では text の文字コード処理が異なっているため、プログラム内で text を処理している部分は、そのままでは移植できない）

プログラムは Win32 コンソールアプリケーション (DOS プロンプト上での実行形式）にて生成し、最適化オプションは/O2 を指定している。実験手順としては、1：ID 生成のみのプログラムを起動し、全ての対象ファイルからそれぞれ ID を生成。2：ID 生成と比較を行うプログラムを起動し、まず識別対象から新規 ID を生成、その新規生成 ID と手順

1 にてあらかじめ生成しておいた ID とを比較。3：ID の相違率（= 新規生成 ID と、あらかじめ生成してあった ID との相違ビット/8192 ビット）が、識別閾値よりも低ければ同一楽曲、識別閾値よりも高ければ相違楽曲としている。

本フィンガープリントシステムのアルゴリズムには、提案アルゴリズムを用いており、様々な楽曲を識別する実験を行った。識別対象楽曲のジャンルは、演歌、ジャズ、クラッシック、ロック、JPOP、アニメ等のいろいろなジャンルである。また、識別実験と共に、同時に処理時間も測定し、Text ファイルに測定データの出力を行い、データ処理を行いやすいようにしてある。

### 4.2.3 PCM のそれぞれ異なる楽曲 100 曲の実験

ソフトウェアでの実験においては、提案アルゴリズムによって、ファイルが識別可能であるのか、可能であるならば処理速度はどの程度なのかを調べることとした。まず、最も基本となる、44.1kHz-16bit-PCM データの、それぞれ全て異なる楽曲 100 曲を識別可能であるかを実験した。実験方法は、表 4.2 のような組み合わせ表を作り、

楽曲 1 と楽曲 1 との比較、楽曲 1 と楽曲 2 との比較、楽曲 1 と楽曲 3 との比較 …· と全ての組み合わせを試した。つまり $100 \times 100$ 曲分 $= 10000$ 曲分 の組み合わせ数となる。出力結果は相違率であり、同一楽曲であれば全く同じフィンガープリント ID が生成されるはずなので相違率は 0 %、全く異なる楽曲であれば全ての情報がランダムに異なる（負の相関、正の相関が共にランダムとなるため 50 %となる）フィンガープリント ID が生成されるはずなので相違率は 50 %となるのが理想である。相違率の理想値を、以下に示す。

$$\text{相違率} = \begin{cases} \text{完全に同一である楽曲} = 0\,\% \\ \text{完全に異なる楽曲} = 50\,\% \end{cases} \tag{4.1}$$

この比較実験を、提案手法（特徴量抽出部にフレーム間差分のみを用いた場合）を用いたものと、特徴量抽出部に離散ウェーブレット結果間差分+フレーム間差分を用いたも

表 4.1: 開発、実行環境

| | 開発環境 | 実行環境 |
|---|---|---|
| ホスト CPU | Sempron3000+ 1.8GHz | Pentium4 2.8GHz |
| ホストメモリ | 2GB | 1.5GB |
| OS | WindowsXP-Pro | WindowsXP-Pro |
| 開発環境 | VisualStudio2005 | |
| 開発言語 | C++ | |
| 総組み合わせ生成スクリプト | ActivePerl5.10 | |
| 連続実行 | | bat ファイル |

の、特徴量抽出部に離散ウェーブレット結果間差分のみを用いたものの、3つの手法を用いて行った。これら特徴量抽出部の違いを図 4.1 に示した。

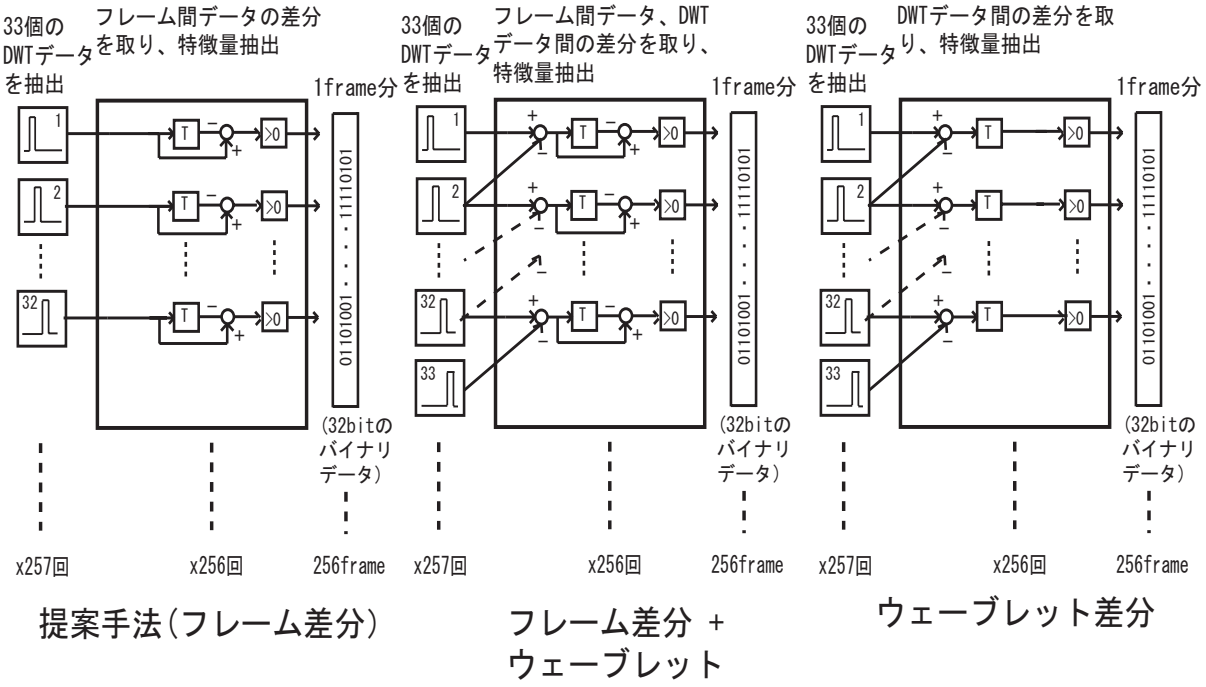



図 4.1: 3 手法の特徴量抽出部の違い

提案手法の結果はグラフ 4.2 の様になった。

楽曲の識別に関しては、同一楽曲同士の識別は全く同じもの同士を比較しているため、相違率 0 %であり、相違楽曲側の識別は、相違率平均値 49.152 %、標準偏差 3.406 となった。

続いて、特徴量抽出部に離散ウェーブレット結果間差分+フレーム間差分を用いたものは、グラフ 4.3 の様になった。

楽曲の識別に関して、同一楽曲の相違率は 0 %、相違楽曲の相違率は 49.064 %、標準偏差 3.972 となった。

| | 楽曲 1 | 楽曲 2 | 楽曲 3 | ••• | 楽曲 100 |
|---|---|---|---|---|---|
| 楽曲 1 | 0 | 44.4 | 49.4 | | 50 |
| 楽曲 2 | 44.4 | 0 | 47.4 | | 48.3 |
| 楽曲 3 | 49.4 | 47.4 | 0 | | 51 |
| ••• | | | | | |
| 楽曲 100 | 50 | 48.3 | 51 | | 0 |

表 4.2: 異なる PCM100 曲 x100 曲の ID 比較

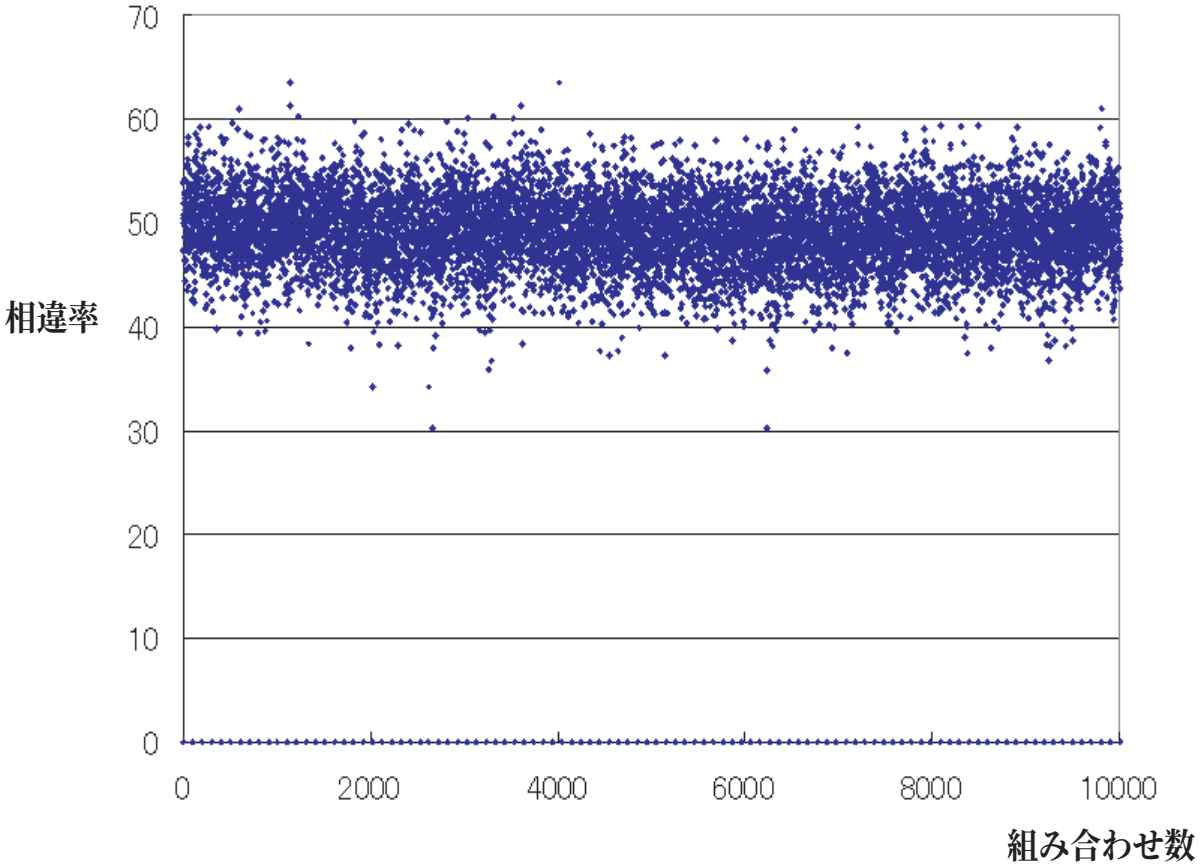



図 4.2: 提案手法（フレーム間差分を用いた特徴量抽出）

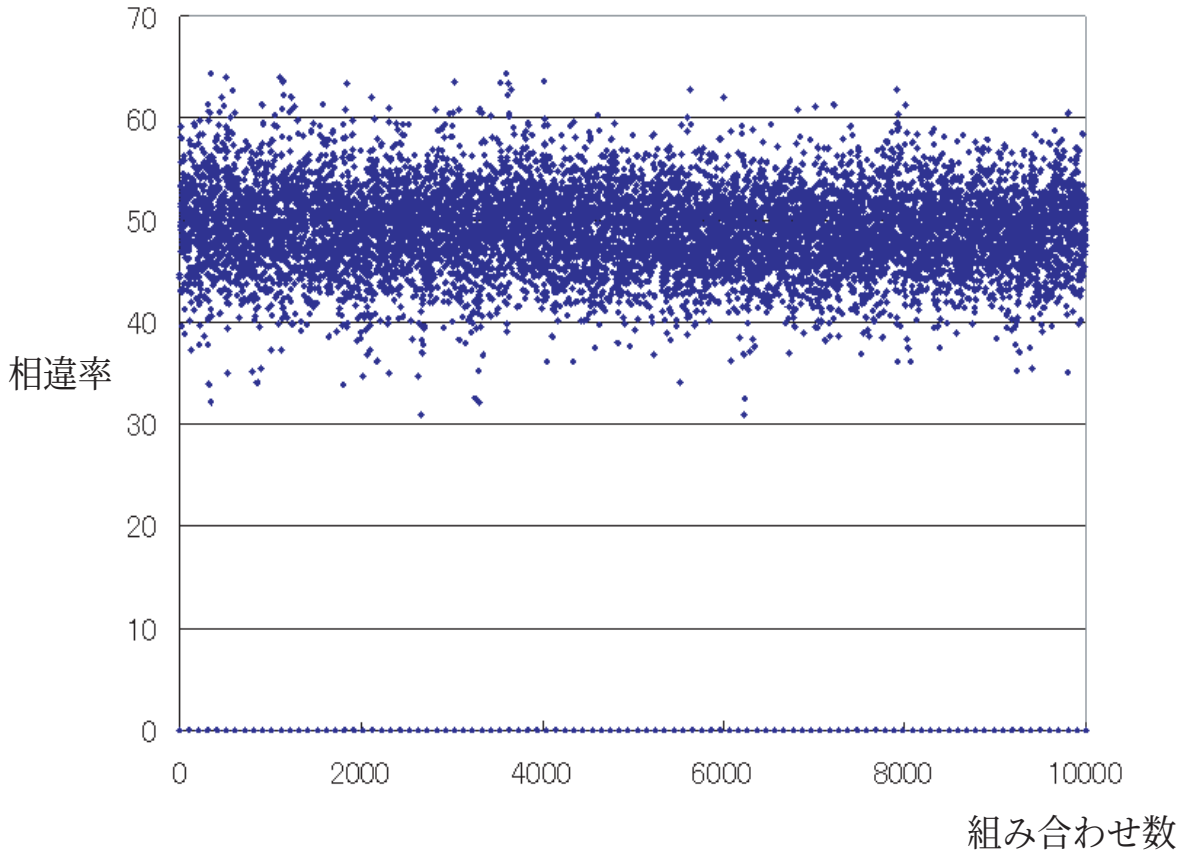



図 4.3: 離散ウェーブレット結果間差分+フレーム間差分を用いた特徴量抽出

続いて、特徴量抽出部に離散ウェーブレット結果間差分を用いたものは、グラフ 4.4 の様になった。

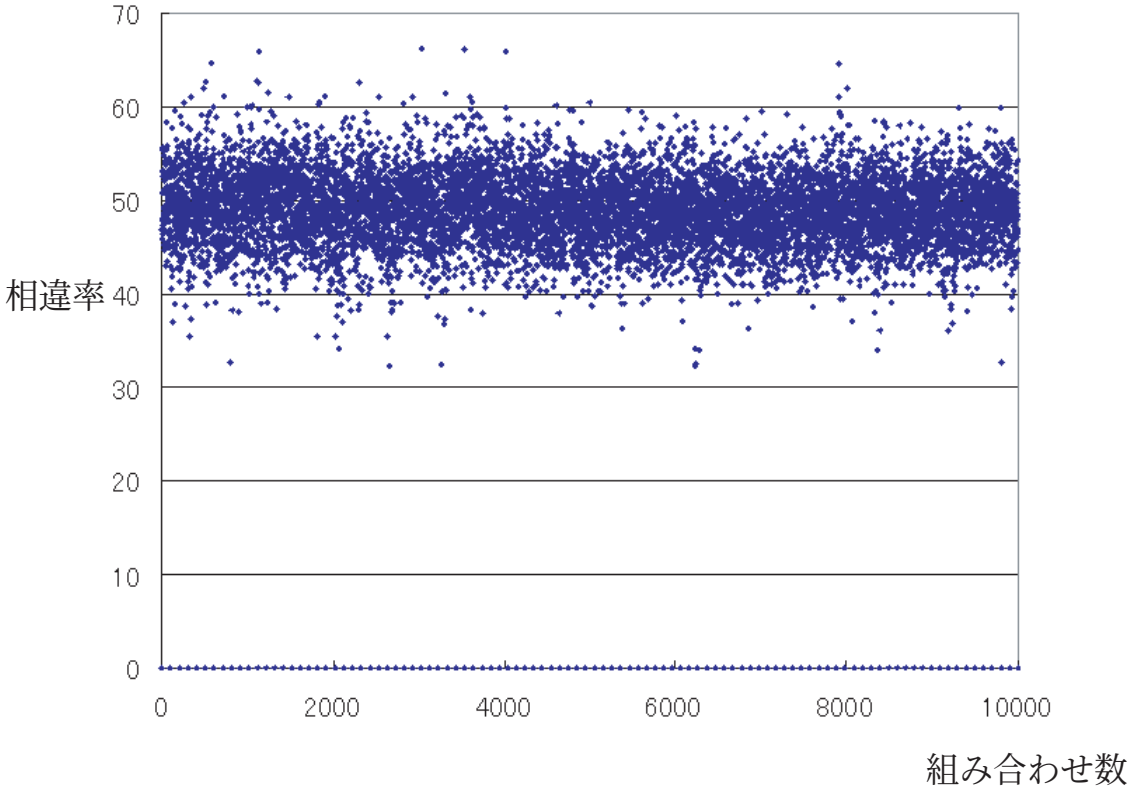



図 4.4: 離散ウェーブレット結果間差分を用いた特徴量抽出

楽曲の識別に関して、同一楽曲の識別は 0 %、相違楽曲の識別は 48.973 %、標準偏差 3.716 となった。これら 3 種の結果をまとめると、表 4.3 のようになる。

これらから分かるように、提案アルゴリズムによる結果は、相違率の平均値が一番（理想値である）50 %に近く、標準偏差も一番小さい。また、処理時間は総処理時間が 250ms 程であり、そのうち 200ms=80 %程の割合が、フィンガープリント ID を生成する時に必要な時間となっていた。よって提案の通りに、特徴量抽出処理にはフレーム間差分のみ用いる事が一番良いと言える。

### 4.2.4 PCM と MP3 それぞれ異なる 20 曲の実験

提案アルゴリズを用いて、PCM と MP3 の 20 曲ずつの識別実験を行った。CBR128kbps-44.1kHz の MP3 に LAME[14] を用いてエンコードし、（実装ソフトウェアの入力が、44.1kHz

の PCM のみが入力可能であるという仕様にしていたため）再度 LAME を用いて PCM-44.1kHz-16bit にデコードした。元の PCM が、楽曲 1、楽曲 2、楽曲 3、… 楽曲 20 とあった場合に、MP3 化した楽曲は、楽曲 1’、楽曲 2’、楽曲 3’、… 楽曲 20 ’と表す。（楽曲 1’ は、楽曲 1 を MP3 化したものである）

実験方法は、表 4.4 のような組み合わせ表を作り、楽曲 1 と楽曲 1’ との比較、楽曲 1 と楽曲 2 との比較、楽曲 1 と楽曲 2’ との比較 …· と全ての組み合わせを試した。つまり 20x20 曲分=400 曲分の組み合わせ数となる。この結果も、元の PCM 楽曲同士と、それを MP3 化したもの（例えば、楽曲 1 と楽曲 1ID、および、楽曲 1 と楽曲 1’ の組み合わせ等）であれば極めて似ているフィンガープリント ID が生成されるはずなので、それらの違いは 0 %に近くなり、全く異なる楽曲であれば全く同じフィンガープリント ID が生成されるはずなので違いは 50 %となるのが理想である。

提案手法の結果はグラフ 4.5 の様になった。

楽曲の識別に関しては、元 PCM と MP3 化した楽曲はほとんど同じ特徴を持つ楽曲であり、このように似ている楽曲間（楽曲 1 と楽曲 1’ や、楽曲 2 と楽曲 2’ 等）の相違率は 1.765 %、標準偏差 2.256 となり、全く異なる特徴を持つ相違楽曲側（楽曲 1 と楽曲 2 や、楽曲 1 と楽曲 2’ 等）の識別は、平均値 49.751 %、標準偏差 2.908 となった。

### 4.2.5 PCM と WMA それぞれ異なる 20 曲の実験

提案アルゴリズを用いて、PCM 形式と WMA 形式の 20 曲ずつの識別実験を行った。CBR128kbps-44.1kHz の WMA 形式に、WindowsMediaAudio[15] を用いてエンコードし、再度 WMA を用いて PCM-44.1kHz-16bit にデコードした。元の PCM が、楽曲 1、楽曲 2、楽曲 3、… 楽曲 20 とあった場合に、WMA 化した楽曲は、楽曲 1”、楽曲 2”、楽曲 3”、… 楽曲 20” と表す。（楽曲 1” は、楽曲 1 を WMA 化したものである）

実験方法は、表 4.5 のような組み合わせ表を作り、楽曲 1 と楽曲 1” との比較、楽曲 1 と楽曲 2 との比較、楽曲 1 と楽曲 2” との比較 …· と全ての組み合わせを試した。つまり 20x20 曲分=400 曲分の組み合わせ数となる。この結果も、極めて似ているフィンガープリント ID の相違率は 0 %に近くなり、全く異なる楽曲であれば相違率は 50 %に近くなるのが理想である。

PCM と WMA の識別結果はグラフ 4.6 の様になった。

| | 提案手法（フレーム差分） | フレーム差分+ウェーブレット差分 | ウェーブレット差分 |
|---|---|---|---|
| 平均値 | 49.152 % | 49.064 % | 48.973 % |
| 標準偏差 | 3.406 | 3.972 | 3.716 |
| 処理時間 | 200ms/250ms | 200ms/250ms | 200ms/250ms |

表 4.3: 手法ごとの平均値、標準偏差

| | 楽曲 1 | 楽曲 1’ | ••• | 楽曲 20 | 楽曲 20’ |
|---|---|---|---|---|---|
| 楽曲 1 | 0 | 0.5 | | 49.5 | 51.3 |
| 楽曲 1’ | 0.5 | 0 | | 50.1 | 48.8 |
| ••• | | | | | |
| 楽曲 20 | 49.5 | 50.1 | | 0 | 0.3 |
| 楽曲 20’ | 51.3 | 48.8 | | 0.3 | 0 |

表 4.4: PCM20 曲と MP3 の 20 曲の ID 比較例

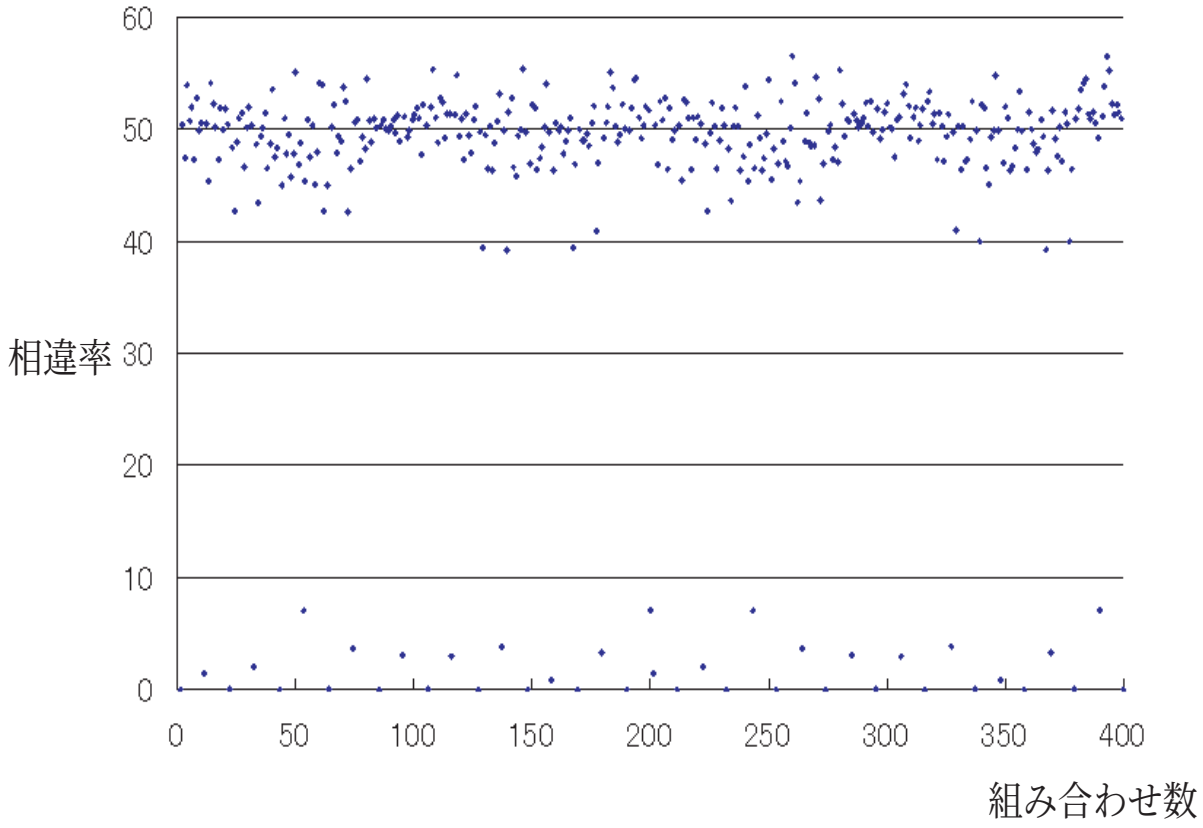



図 4.5: PCM と MP3 のフィンガープリント ID 相違率

| | 楽曲 1 | 楽曲 1” | ••• | 楽曲 20 | 楽曲 20” |
|---|---|---|---|---|---|
| 楽曲 1 | 0 | 0.5 | | 49.5 | 51.3 |
| 楽曲 1” | 0.5 | 0 | | 50.1 | 48.8 |
| ••• | | | | | |
| 楽曲 20 | 49.5 | 50.1 | | 0 | 0.3 |
| 楽曲 20” | 51.3 | 48.8 | | 0.3 | 0 |

表 4.5: PCM20 曲と WMA の 20 曲の ID 比較例

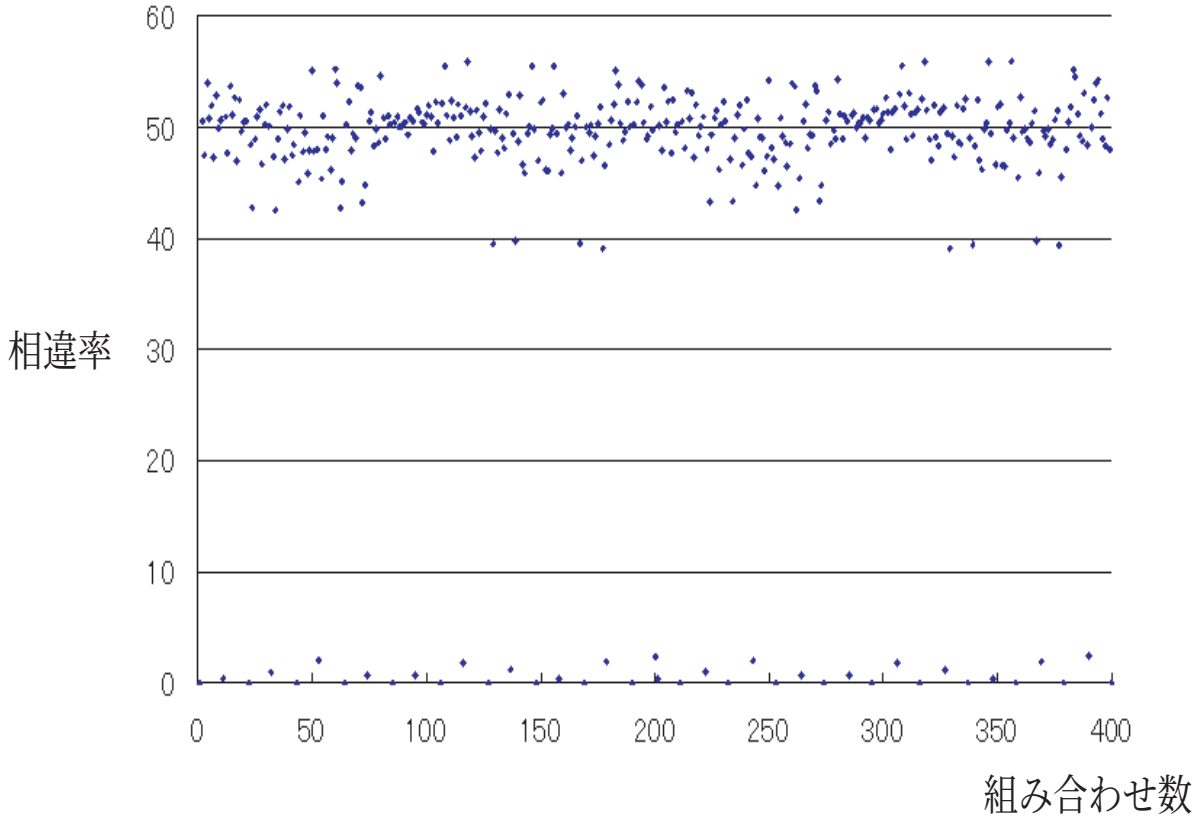



図 4.6: PCM と WMA の相違率

楽曲の識別に関しては、元PCMとWMA化した楽曲はほとんど同じ特徴を持つ楽曲であり、このように、似ている楽曲間の相違率は0.625％、標準偏差0.803となり、全く異なる特徴を持つ相違楽曲間の識別は、平均値49.723％、標準偏差2.942となった。

### 4.2.6 PCMとリサンプリングファイルによるそれぞれ異なる20曲の実験

提案アルゴリズを用いて、PCMとリサンプリング後元に戻した楽曲を20曲ずつ（合計40曲）の識別実験を行った。SoundPlayerLilith(Lilith)[13]を用いて、22.05kHzにダウンサンプリングを行い、再度Lilithを用いて44.1kHzにアップサンプリングを行った。元のPCMが、楽曲1、楽曲2、楽曲3、･･･楽曲20とあった場合に、WMA化した楽曲は、楽曲1”’、楽曲2”’、楽曲3”’、･･･楽曲20”’と表す。（楽曲1”’は、楽曲1をリサンプリングしたものである）

実験方法は、表4.6のような組み合わせ表を作り、楽曲1と楽曲1”’との比較、楽曲1と楽曲2との比較、楽曲1と楽曲2”’との比較 ････ と全ての組み合わせを試した。つまり20x20曲分=400曲分の組み合わせ数となる。PCMとリサンプリングの識別結果はグラフ4.7の様になった。

楽曲の識別に関しては、元PCMとリサンプリングを行った楽曲はほとんど同じ特徴を持つ楽曲であり、同一と思われる楽曲間の相違率は0.294％、標準偏差0.446となり、全く異なる特徴を持つ相違楽曲間の識別は、平均値49.709％、標準偏差2.916となった。

### 4.2.7 PCMとテンポチェンジ（+4％）20曲の実験

提案アルゴリズを用いて、PCM形式とテンポを高速化（+4％した）楽曲、20曲ずつの識別実験を行った。テンポチェンジにはLilithのDSP機能を用いた。通常、圧縮加工は良く行われると想定されるが、テンポの高速化はあまり行われないと考える。ただし、提案アルゴリズムが全ての加工に対して頑健であるわけではないことを示すために、あらかじめ、提案アルゴリズムは時間変化に極めて弱いことが予想できていたため、本実験を行った。元のPCMが、楽曲1、楽曲2、楽曲3、･･･楽曲20とあった場合に、テンポチェ

| | 楽曲1 | 楽曲1”’ | ･･･ | 楽曲20 | 楽曲20”’ |
|---|---|---|---|---|---|
| 楽曲1 | 0 | 0.5 | | 49.5 | 51.3 |
| 楽曲1”’ | 0.5 | 0 | | 50.1 | 48.8 |
| ･･･ | | | | | |
| 楽曲20 | 49.5 | 50.1 | | 0 | 0.3 |
| 楽曲20”’ | 51.3 | 48.8 | | 0.3 | 0 |

表4.6: PCM20曲とリサンプリングの20曲のID比較例

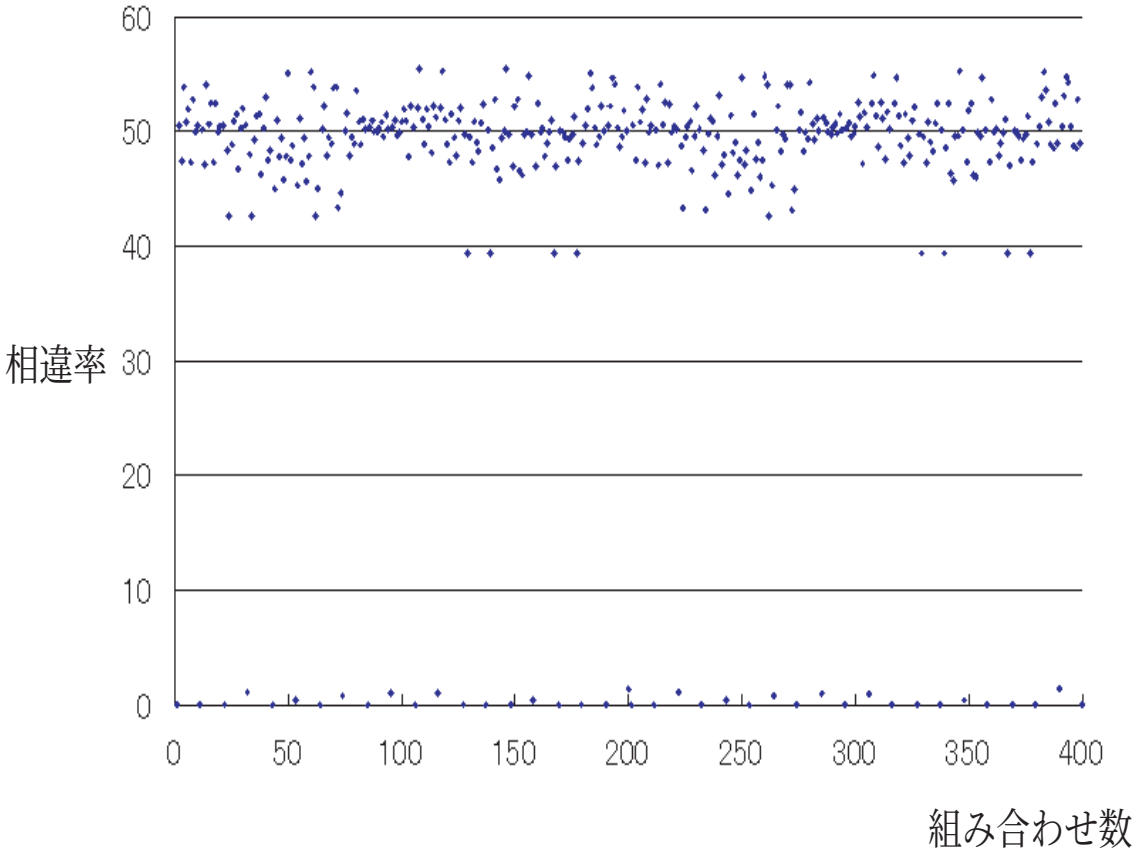



図 4.7: PCM とリサンプリングファイルの相違率

ンジした楽曲は、楽曲 1””、楽曲 2””、楽曲 3””、··· 楽曲 20”” と表す。（楽曲 1”” は、楽曲 1 をテンポチェンジ（+4 %）したものである）

実験方法は、表 4.7 のような組み合わせ表を作り、楽曲 1 と楽曲 1”” との比較、楽曲 1 と楽曲 2 との比較、楽曲 1 と楽曲 2”” との比較 ···· と全ての組み合わせを試した。つまり 20x20 曲分=400 曲分の組み合わせ数となる。PCM とテンポチェンジ（+4 %）の識別結果はグラフ 4.8 の様になった。

この結果は事実上識別不能であり、同一楽曲であると思われる楽曲の相違率は、平均値 17.468 %、標準偏差 17.800 となり、相違楽曲と思われる楽曲は、平均値 50.207 %、標準偏差 3.001 となった。

### 4.2.8 PCM、MP3、WMA、リサンプリングファイルのそれぞれ異なる 10 曲の実験

提案アルゴリズを用いて、PCM、MP3、WMA、リサンプリングの楽曲を 10 曲ずつ（合計 40 曲）で識別実験を行った。それぞれの単体データをとっているのにも関わらず、再度全てを含めて実験しているのは、PCM からの加工を 1 次加工とし、加工後さらに加工（2 次加工）するような事は、一般的によく行われており、このような場合を想定し、きちんと識別が行えるか知りたかったためである。なお、圧縮加工は良く行われるが、テンポを変える事はあまり行われないと考える。そのためテンポチェンジの項目は含まなかった。ファイルの変換方法は前節で述べた通りである。実験方法は、元の PCM が、楽曲 1、楽曲 2、楽曲 3、··· 楽曲 10 とあった場合に、MP3 化した楽曲は、楽曲 1’、楽曲 2’ ··· 、WMA 化した楽曲は楽曲 1”、楽曲 2” ··· 、リサンプリングを行った楽曲は、楽曲 1”’、楽曲 2”’ ··· と表す。

PCM、MP3、WMA、リサンプリングファイル、全てをまとめた結果はグラフ 4.9 の様になった。

結果として、同一と思われる楽曲間の相違率は、平均値 1.860 %、標準偏差 2.207 となり、全く異なる特徴を持つ相違楽曲間の識別は、平均値 49.754 %、標準偏差 2.913 となった。また、同一楽曲、相違楽曲の平均値、標準偏差をもちい、それぞれ正規分布であると

| | 楽曲 1 | 楽曲 1”” | ··· | 楽曲 20 | 楽曲 20”” |
|---|---|---|---|---|---|
| 楽曲 1 | 0 | 0.5 | | 49.5 | 51.3 |
| 楽曲 1”” | 0.5 | 0 | | 50.1 | 48.8 |
| ··· | | | | | |
| 楽曲 20 | 49.5 | 50.1 | | 0 | 0.3 |
| 楽曲 20”” | 51.3 | 48.8 | | 0.3 | 0 |

表 4.7: PCM20 曲とテンポチェンジ（+4 %）の 20 曲の ID 比較例

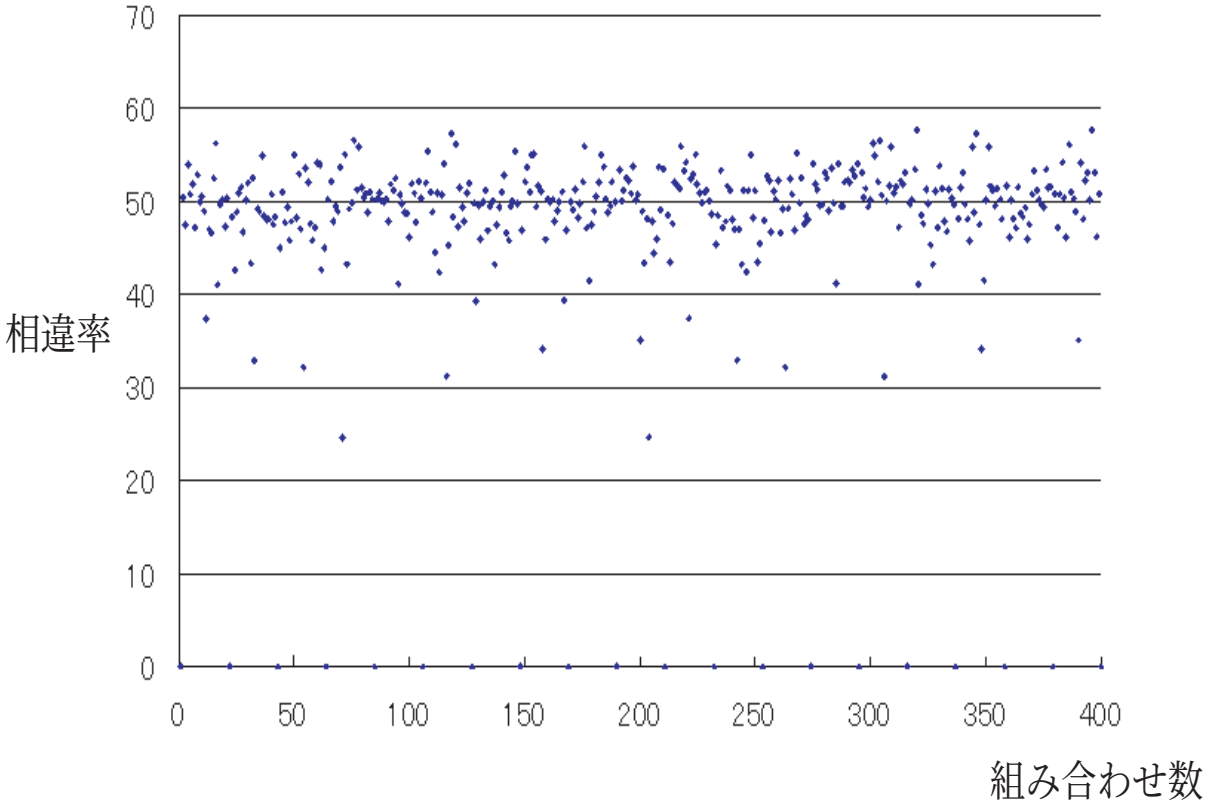



図 4.8: PCM とテンポチェンジ（+4 %）の相違率

| | 楽曲 1 | 楽曲 1’ | 楽曲 1” | 楽曲 1”’ | ••• | 楽曲 10”’ | 楽曲 10”” |
|---|---|---|---|---|---|---|---|
| 楽曲 1 | 0 | 0.5 | | | | 49.5 | 51.3 |
| 楽曲 1’ | 0.5 | 0 | | | | 50.1 | 48.8 |
| 楽曲 1” | 0.5 | 0 | | | | 50.1 | 48.8 |
| 楽曲 1”’ | 0.5 | 0 | | | | 50.1 | 48.8 |
| ••• | | | | | | | |
| 楽曲 10”’ | 49.5 | 50.1 | | | | 0 | 0.3 |
| 楽曲 10”” | 51.3 | 48.8 | | | | 0.3 | 0 |

表 4.8: PCM、MP3、WMA、リサンプリング 10 曲ずつの ID 比較例

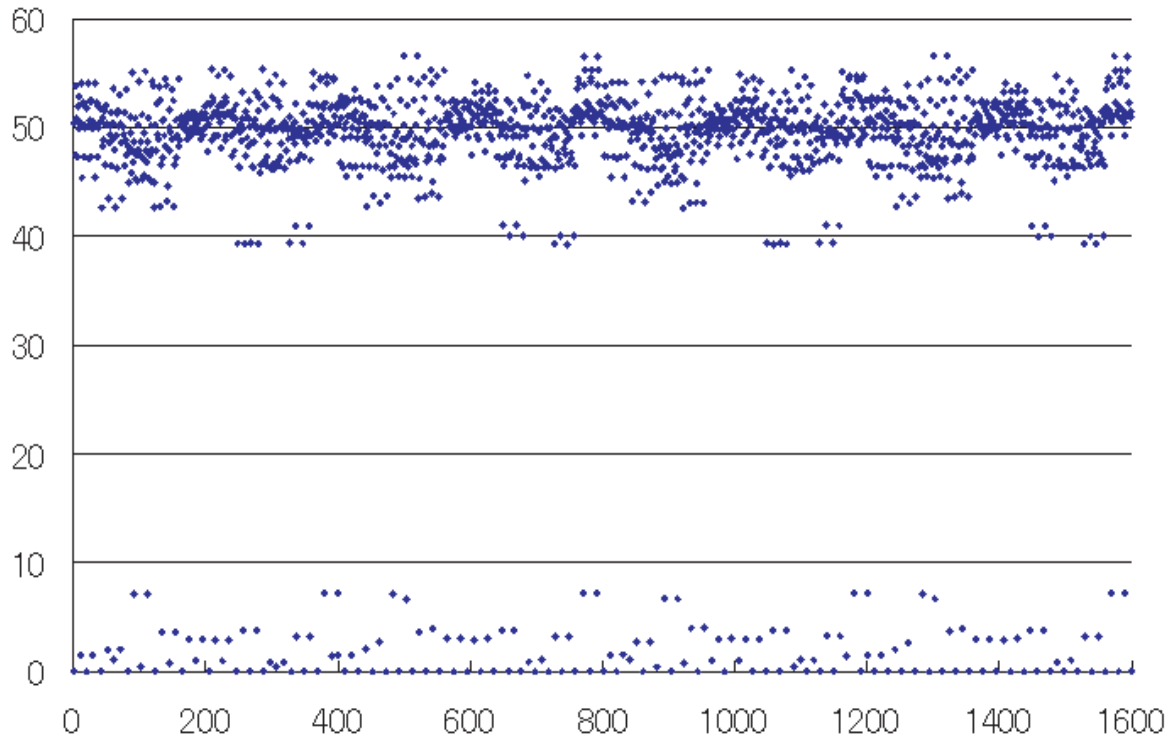



図 4.9: PCM、MP3、WMA、リサンプリングファイルの相違率

仮定した場合、$f(x) = \frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(x-u)^2}{2\sigma^2}$ に当てはめて推定線を引いた。これを、グラフ 4.10 に示す。

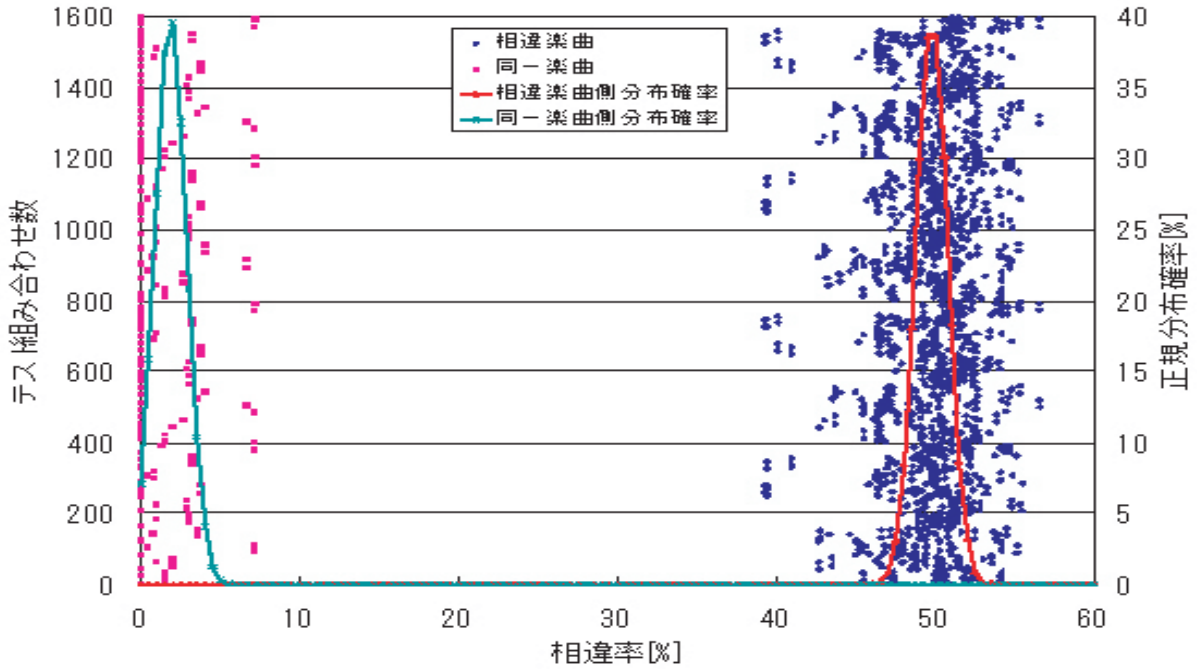



図 4.10: 識別閾値の推定

推定線の交点は最も誤識別率の低くなる箇所であり、これを識別閾値とし、23 %とした。このときの推定誤差率は $1E^{-30}$ 以下となった。

### 4.2.9 実装上の工夫

提案アルゴリズムをソフトウェアに実装したときに、いくつか工夫を施した。まず 1 つ目としては、きちんとした比較を行うために、入力データにあらかじめオフセットをかけ、確実にデータが存在するところを使用した事、2 つ目はウェーブレット変換をさらに高速化するために haar ウェーブレットの式を変形した事である。

#### 入力データの使用領域

まずは、1 つ目のオフセットについて述べる。入力データとなる楽曲は、先頭 1 秒前後が無音状態（0 データ）なものがまれに存在した。無音状態ではデータが無いため、特徴も存在しない。そのため無音部を用いてはならず、データが存在する箇所を用いなければ

本来の識別が行えない事となる。データが確実に存在するように、2秒進んだところから入力データをとることとした。本来のアルゴリズムでは先頭データより約3.3秒までの領域で必要データを読み終わる事となっているが、この場合では2秒分オフセットし、2秒から約5.3秒までの領域で必要データを読む事としている。

#### ウェーブレットの高速化および整数化

次に、2つ目のhaarウェーブレットの式変形について述べる。提案アルゴリズムのウェーブレット変換部にはhaarを用いており、これは、元データの隣り合う数値を足し合わせ、2で割るものをLo（つまり、情報量を減らし近似化している)、元データの隣り合う数値を引き、2で割るものをHi（つまり元データとLo側結果との差分を表す）としていた。提案アルゴリズムの最終結果には、ウェーブレット変換部の結果をそのまま用いているのではなく、特徴量抽出部でフレーム間差分を取り、値が0以下か0を超えるかで最終的なフィンガープリントIDを生成している。そのため、ウェーブレット変換結果の絶対値が合っている必要はなく、HiとLoの関係が成り立っていれば（Loデータと Hiデータから、元のデータに復元できるならば）式変更を行ってもかまわないと考えた。この考えに従い、単純に隣り合う数値を足し合わせたものをLo、隣り合う数値の差分をとったものをHiとした。除算を無くし、加算減算のみにしたことから、副次的に整数化も行われた。この結果をグラフ4.11に示す。

グラフ結果より、フレーム間差分を用いた特徴量抽出-グラフ4.2と、まったく同じ結果になっていることが分かる。高速化に関して、整数化および除算の排除により、Pnetium4-2.8GHzにおいて、離散ウェーブレット処理時間を141msから61msへと短縮できた。

## 4.3 提案アルゴリズムのハードウェア実装

### 4.3.1 はじめに

提案アルゴリズムの中で、最も演算負荷が高い部分は、離散ウェーブレット変換部分であった。そのため、提案アルゴリズムを用いたフィンガープリントシステムを高速化する場合、ハードウェアにより高速化を行えば、最も効果がある箇所であると考えた。ここでは、離散ウェーブレット変換部の、並列化箇所の決定、最大並列化数の推定、最大並列化数での処理時間の推定を行った。

### 4.3.2 ハードウェア実験環境

まず、図4.12に、ハードウェアの実験環境、図4.13に、RCHTXボードの概略図を示す。

ホストPCの環境は、CPU-Opteron2216 2.4GHz-DualCPU、メモリ-4GB、OS-CentOS4.6 x64（Linux-kernel2.6系-64bit）が使用されている。ハードウェアにはRCHTXボードが使

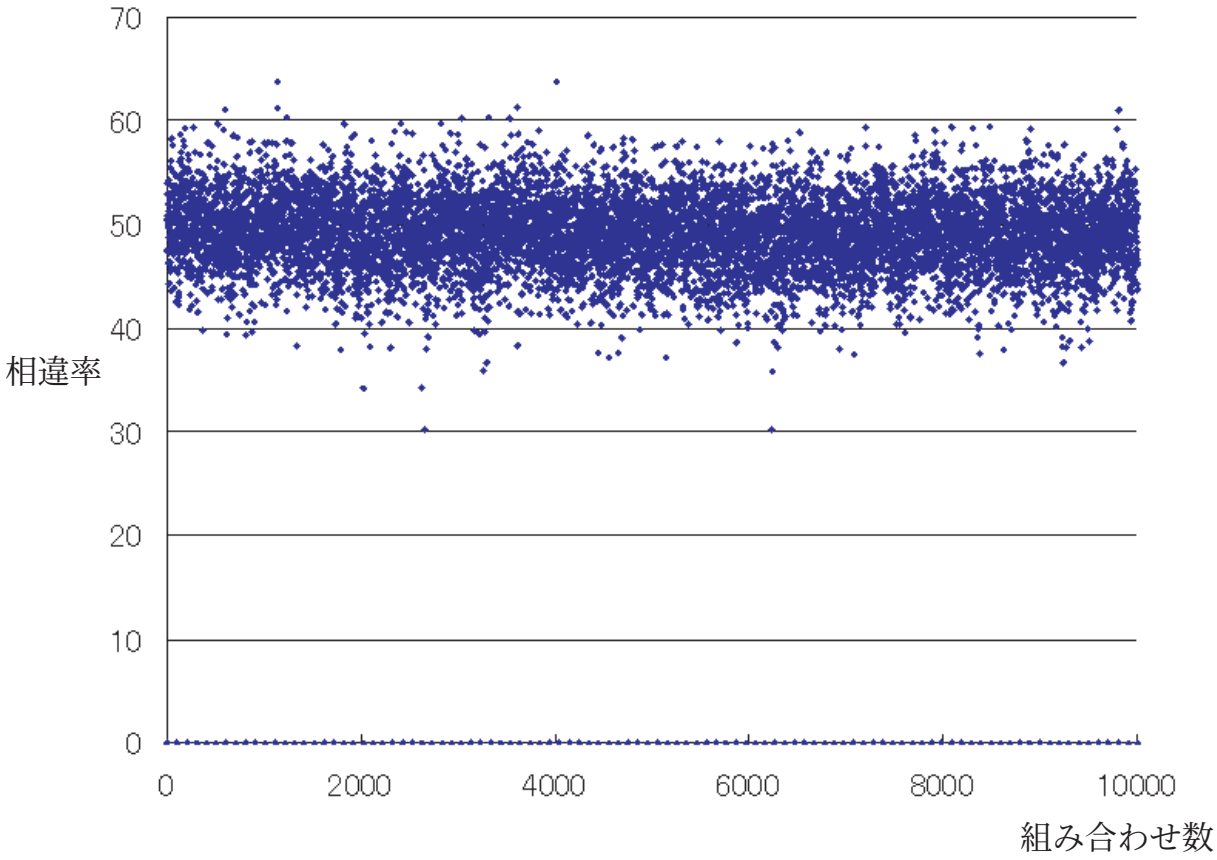



図 4.11: 整数化したアルゴリズムでの、100 曲の異なる PCM ファイルの相違率

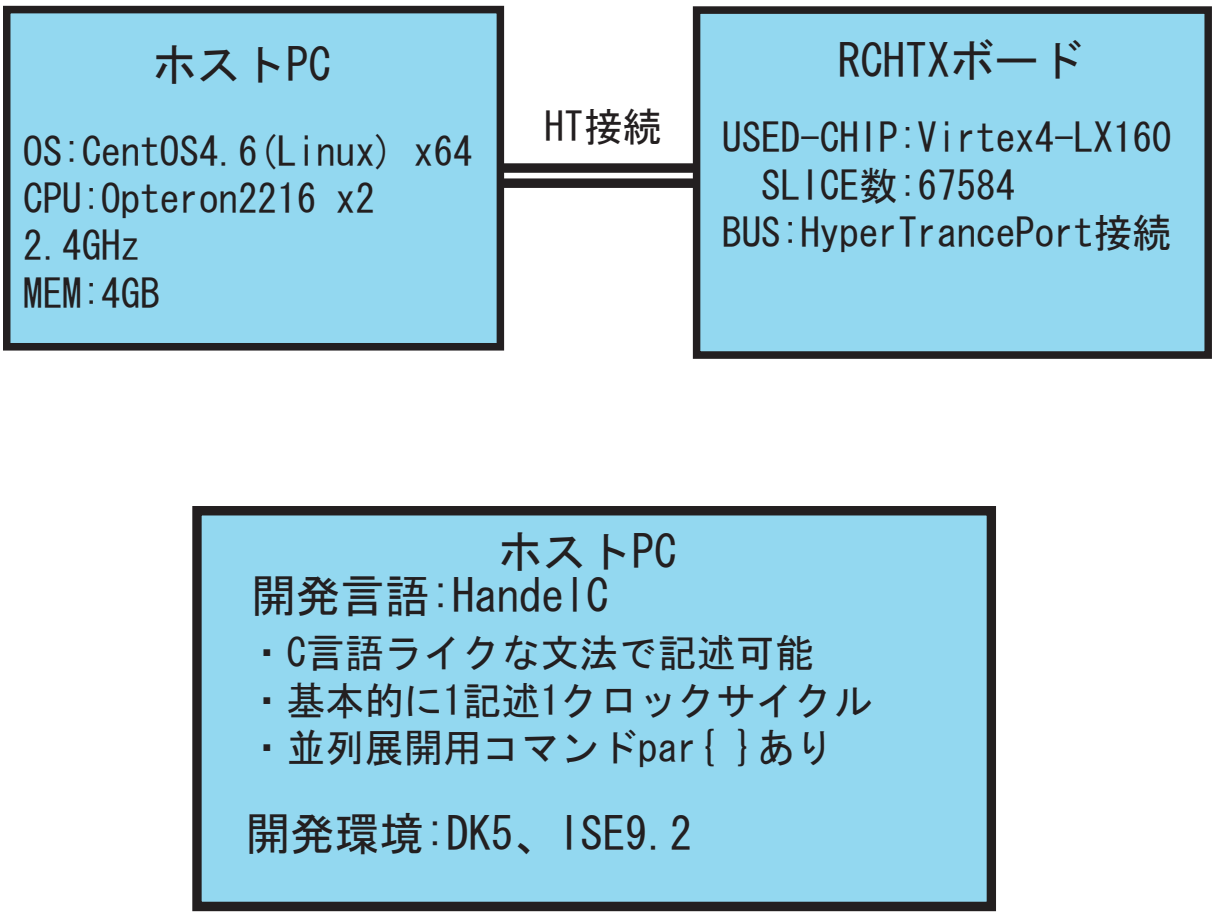



図 4.12: ハードウェアの実験環境

## RCHTX Functionality

The RCHTX HPC board provides the following functionality:

- User FPGA:
  - Xilinx Virtex-4 XC4VLX160 or XC4VSX55 FPGA
  - 4 banks 2M * 16 QDR SRAM
  - VGA/DVI video port
  - Gigabit Ethernet port (PHY only, requires soft MAC)
  - 8 user-controllable status LEDs
  - LVDS expansion I/O port
- Kernel FPGA
  - Xilinx Virtex-4 XC4VLX40
  - 2 banks 2M * 16 QDR SRAM
  - Gigabit Ethernet port (PHY only, requires soft MAC)
  - Flash memory for Kernel and User FPGA programming
  - 4 status LEDs
- 2 Programmable clocks
- HTX bus 16-bit data path running in either the 200MHz or 400MHz HyperTransport mode

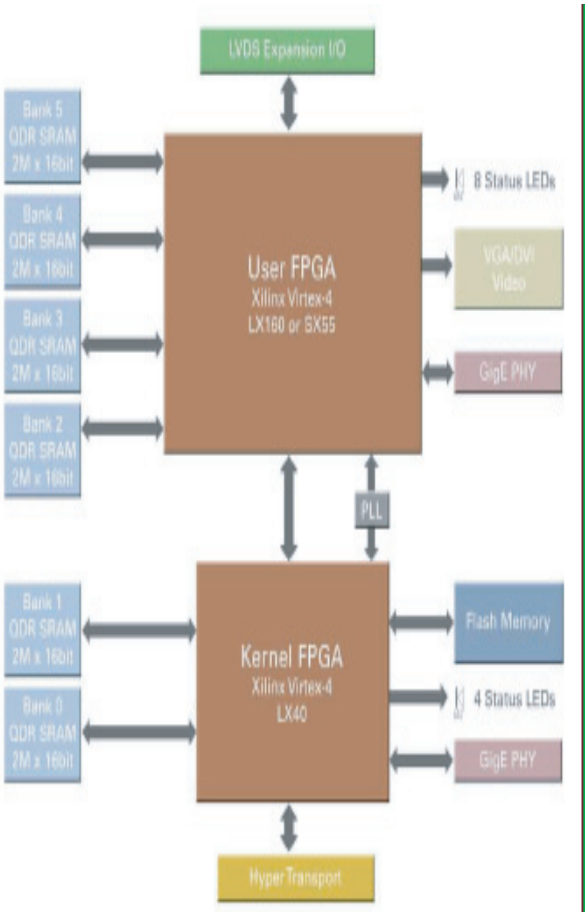



図 4.13: RCHTX ボード概略図

用されており、これには、ボードコントロール用 FPGA チップに Virtex4-LX40、ユーザ使用 FPGA チップとして Virtex4-LX160 の合計 2 つのチップが搭載されている。Virtex4-LX160 の全 Slice 数は 67584 であり、これを全て用いて分散 RAM を構成した場合、1056kb 分の RAM をを作ることが可能である。また、Virtex4 の最大周波数は 500MHz であるが、RCHTX ボードと DK を用いた場合は 300MHz が最大となる。これらの内容を、表 4.9 にまとめた。

RCHTX ボードの大きな特徴としては、高速バスである HT バスでホスト PC と接続されている事があげられる。ホスト PC と RCHTX ボードは、16bit-200MHz、400MHz で接続でき、本システムでは 200MHz で動作している。一般に、FPGA ボードをアクセラレーターとして用いる場合、ホスト PC と FPGA ボードとの通信バスがボトルネックとなるが、本システムでは 16bit-200MHz=400MB/s の帯域があり、PCI-32bit-33MHz=133MB/s よりも 3 倍ほど高速なため、バスがボトルネックにはなりにくい。

### 4.3.3 ハードウェアによる離散ウェーブレット演算の並列化数、処理速度の推定

入力データを $input[\ ]$ とし、データ番号を $i$ とすると、Lo 側は $Lo = (input[2 \times i] + input[2 \times i + 1])$ として表され、意味としては、「元の波形データの近似」となる。Hi 側は $Hi = (input[2 \times i] - input[2 \times i + 1])$ として表され、意味としては「元の波形データと Lo 側データとの差分」となり、この部分の演算フローは、図 4.14 となる。

#### シーケンシャル時

フィンガープリントシステムの中で、演算負荷が重い部分は離散ウェーブレット変換部であると分かっているため、まずはこの部分を FPGA に実装する場合を考える。表 4.10 に、haar のウェーブレット演算のシーケンシャル動作での 1 フレームあたりの演算量を推定した。

また、離散ウェーブレット演算時のクリティカルパスと Slice 数を、VHDL と ISE によりシミュレーションにより求めた。演算内容は、$Lo = (input[2 \times i] + input[2 \times i + 1])$ と、$Hi = (input[2 \times i] - input[2 \times i + 1])$ の 2 つであり、結果を表 4.11 に示した。

これらの、演算回数と、クリティカルパスより、演算時間 = 演算回数 × クリティカルパス によって演算時間が推定できる。実際に数値を入れて計算すると、1 フレームあたり

| | 最大 clock | 構成可能最大分散 RAM | Slice 数 |
|---|---|---|---|
| Vertex4-LX160 | 500MHz | 1056kb | 67584 |

表 4.9: Virtex4-LX160 の最大 Slice 数と最大分散 RAM 量

の演算時間 $= 101ms$ となる。本システムでは 257 フレーム分処理を行わなければならないため、257 フレーム $\times 101ms = 26s$ となり、シーケンシャル動作ではソフトウェア処理の 61ms と比べるととても遅いことが分かる。

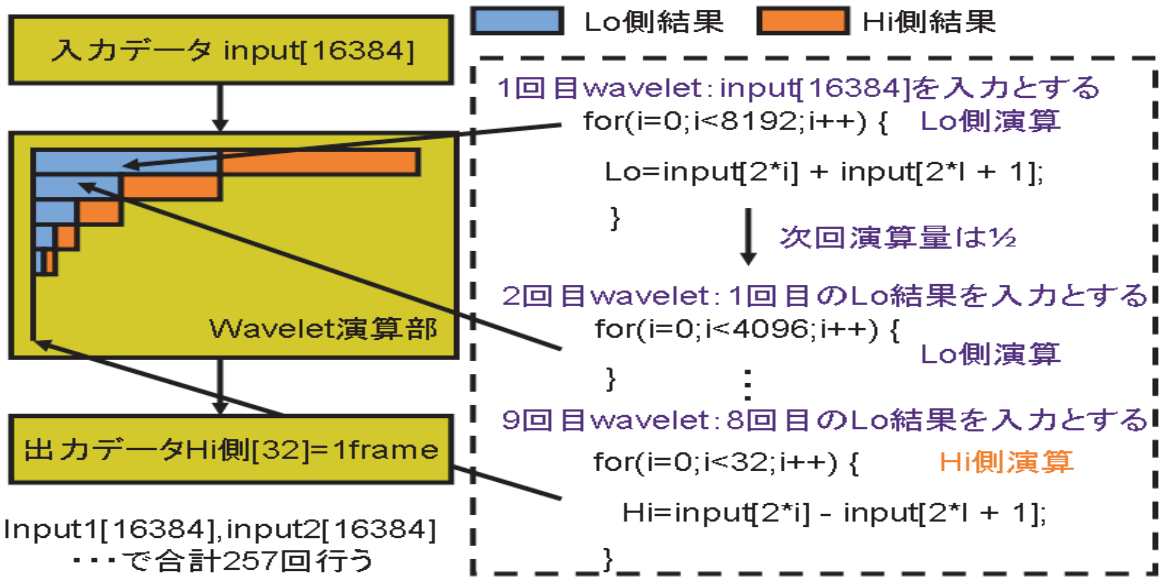



図 4.14: haar-wavelet 演算フロー

### 並列化可能最大数

ハードウェア処理を速くする手法にはいくつかあるが、今回は並列化により処理速度を向上させる手法を考え、最大並列数を推定した。

並列化するということは、同一回路を複数生成するということである。FPGA においては回路規模は Slice 数により制限を受ける。まず、離散ウェーブレット演算を行うためには、何が必要であるかを考えると、入力である入力バッファ、出力となる出力バッファ、残りは実際の演算となり、3 つの項目が必要であることがわかる。これらの回路規模をそれぞれ推定すると、入力バッファ $= int24bit[16384] = 384kb$、出力バッファ $= int24bit[257][32] = 193kb$ であり、Virtex4-LX160 の全 Slice 数 67584 を全て分散 RAM にした場合、1056kb が構成出来ることを用い、入力バッファ、出力バッファの Slice 数を推定できる。また、1024 並列を行うとした場合の演算器数 = 加算器 1024 個 + 減算器 32 個 = 1056 個 となり、24bit 加算器減算器は、1 演算器につき 24Slice 必要であるとシーケンシャル時のシミュレーション結果より分かっているため、こちらも Slice 数が推定できる。これらをまとめ、Slice 数を表 4.12 に示した。

| 離散ウェーブレット回数 | 演算回数 | 演算回数合計 |
|---|---|---|
| 1 回目 (Lo) | 8192 | 8192 |
| 2 回目 (Lo) | 4096 | 12288 |
| 3 回目 (Lo) | 2048 | 14336 |
| 4 回目 (Lo) | 1024 | 15360 |
| 5 回目 (Lo) | 512 | 15872 |
| 6 回目 (Lo) | 256 | 16128 |
| 7 回目 (Lo) | 128 | 16256 |
| 8 回目 (Lo) | 64 | 16320 |
| 9 回目 (Hi) | 32 | 16352 |

表 4.10: 1frame あたりの演算数

| | クリティカルパス | Slice 数 |
|---|---|---|
| 24bit 加算器 | 6.157ns | 24 |
| 24bit 減算器 | 6.157ns | 24 |

表 4.11: 加算器減算器のシミュレーションによるクリティカルパスおよび Slice 数

| 用途 | 回路状態 | 必要量 | 使用 Slice 数 | 残り Slice 数 |
|---|---|---|---|---|
| 入力バッファ | 分散 RAM | 384kb | 24576 | 43008/67584 |
| 出力バッファ | 分散 RAM | 193kb | 12352 | 30656/67584 |
| 演算 | 加算器演算器 | 1056 個 | 25334 | 5312/67584 |

表 4.12: 1024 並列時の必要 Slice 数

1024 並列を行った場合には、残り Slice 数が 5312 となっており、これは全 Slice 数の 92 %を使用したということとなる。512 並列の時は同様に全 Slice 数の 65 %を使用となる。2048 並列では Slice 数が足りず、対象 FPGA チップ (Virtex4-LX160) では 1024 並列が最大であることが分かる。

#### 1024 並列時の演算時間

Virtex4-LX160 では、Slice 数の制限から、離散ウェーブレット演算部は最大で 1024 並列までしか構成出来ないことが分かり、この時の離散ウェーブレット演算の必要処理時間を推定し、表 4.13 に示した。

演算時間 = 演算回数 × クリティカルパス であるため、1 フレームあたりの演算時間 $= 20 \times 6.157ns = 123ns$ となり、257 フレームの演算時間 $= 257 \times 123ns = 32ms$ となる。よって、離散ウェーブレット処理部が 1024 並列であった場合、32ms かかることが分かる。同様に、512 並列の時には 54ms となる。

## 4.4 まとめ

### 4.4.1 ソフトウェア実装のまとめ

フィンガープリントシステムの特徴量抽出に用いた、haar のウェーブレットを整数化および除算の排除により、Pentium4-2.8GHz での処理時間を 141ms から 61ms へと短縮できた。これにより、本提案アルゴリズムによるフィンガープリント ID 生成は、全ての処理にかかる時間が 128ms となり、ソフトウェア実装の段階でありながら、磯永のハードウェアによるフィンガープリント ID 生成の 315ms よりも、約 2.46 倍高速となった。なお、この時の誤識別率は推定 $1E^{-30}$ 以下となった。

### 4.4.2 ハードウェア実装推定のまとめ

提案アルゴリズムでの離散ウェーブレット部分の CPU 処理時間は 61ms であったので、ハードウェア化する場合、それよりも高速に処理できることが求められていた。Virtex4-LX160 を用いてシーケンシャルに実装した場合、26s かかることが予想され、これを高速化するために並列化手法を用いた場合、Slice 数の制限から最大 1024 並列までとなる。512 並列時の処理速度は 54ms、使用 Slice 数の割合は 65 %と推定でき、1024 並列時の処理速度は 32ms、使用 Slice 数の割合は 92 %になると推定される。これはソフトウェアの 1.9 倍高速となり、ハードウェアによる高速化を行えると推定できる。

| 離散ウェーブレット回数 | 演算回数 | 演算回数合計 |
|---|---|---|
| 1 回目 (Lo) | $8192/1024 = 8$ | 8 |
| 2 回目 (Lo) | $4096/1024 = 4$ | 12 |
| 3 回目 (Lo) | $2048/1024 = 2$ | 14 |
| 4 回目 (Lo) | $1024/1024 = 1$ | 15 |
| 5 回目 (Lo) | $512/512 = 1$ | 16 |
| 6 回目 (Lo) | $256/256 = 1$ | 17 |
| 7 回目 (Lo) | $128/128 = 1$ | 18 |
| 8 回目 (Lo) | $64/64 = 1$ | 19 |
| 9 回目 (Hi) | $32/32 = 1$ | 20 |

表 4.13: 1024 並列時の 1frame あたりの演算数

# 第5章　まとめと今後の課題

本研究の最終目的は、磯永のハードウェア実装結果である、入力から識別終了までの処理時間315msよりも高速に識別処理を行う事であった。本論文中で提案したアルゴリズムを用い、ソフトウェア実装を行い、実装ソフトウェアを用いた実験によりPentium4-2.8GHzでのソフトウェアでの処理時間が128msとなる事が確かめられた。この時点で、磯永のハードウェアフィンガープリントシステムの2.46倍の処理速度となっており、ソフトウェア実装の段階で高速化目的は達せられた。

また、識別精度においても、PCM、MP3、WMA、リサンプリング等の一般的によく使われるであろうと想定される加工に対しては極めて識別精度が良く、実験結果から算出した標準偏差、平均値を元に、識別結果が正規分布に基づくモデルであると仮定し、誤識別率を推定した。推定結果は$1E^{-30}$以下となり、ほぼ誤識別がおこらないであろう精度となった。

提案アルゴリズムを用いた識別法では、入力ファイルサイズが288kByte(3.3秒分）以上あれば、識別を行える。磯永の識別法では入力ファイルサイズが8.38MByte以上なければならなかった。オーディオファイルのイントロや、ちょっとしたフレーズでも3.3秒は超えると思われるので、そのような短い時間の楽曲に対しても識別が行えるというのも本提案アルゴリズムの利点である。

提案アルゴリズムを用いたソフトウェア実装で、すでに磯永のハードウェアフィンガープリントシステムよりも十分高速化されているが、FPGAを並列化して用いさらなる高速化が行えないかと推定を行った。ソフトウェアの総処理時間128msの内、haarのウェーブレット演算の処理時間は61msかかっており、処理時間の割合が大きいhaarのウェーブレット部分を高速化する場合について推定した。対象FPGAチップ(Virtex4-LX160)のSlice数の制限より、演算部は最大で1024並列までとなり、512並列では推定使用Slice数割合は65 %、推定処理速度54ms、1024並列での推定使用Slice数の割合は92 %、推定処理速度は32msとなった。

今回、FPGAを用いて高速化が行えないかと推定を行ったが、Slice上に分散RAMとして入出力バッファを構成すると仮定したため、分散RAMによるSliceの使用量が大きく、入出力バッファを構成すると、残りのSlice数の割合は45 %しか無かった。今回は、分散RAMを用い演算器を並列化する事による高速化の推定を行ったが、FPGAチップ外のRCHTXボード上のRAMを用いた場合や、並列化手法だけでなくパイプライン手法等について検討する事などが今後の課題となる。

# 謝辞

本研究を行うに当たり、多くの御助言、御指導を賜りました北陸先端科学技術大学院大学の情報科学センター 井口 寧 准教授に深く感謝すると共に、ここに御礼申し上げます。貴重な御助言、御意見をいただいた本学の松澤照男教授、金子峰雄教授、田中清史准教授に深く御礼申し上げます。また、貴重な御助言、御意見をいただいた佐藤 幸紀助教と研究のサポートをしていただいた情報科学センターの方々に深く御礼申し上げます。

井口研究室で、貴重な御意見、討論をしていただいた、Yu Chen 氏、Sun Wei 氏、近藤 裕貴氏、松山 周平氏、熊坂 史彬氏、長瀬 文彦氏、Li Qi 氏、Li Bo 氏、荒木 光一氏、伊藤 徹也氏、中尾 哲也氏に深く感謝いたします。

ハードウェア合同ゼミで貴重な御意見、討論をしていただいた皆様、本研究を応援してくださった、日比野研究室の矢澤 慶樹氏、田中研究室の請園 智玲氏、松本研究室のメンバーに深く感謝いたします。

本研究を進めるにあたり、すばらしい環境を提供していただいた北陸先端科学技術大学院大学、および Celoxica 社、Agility 社に感謝いたします。

最後に、ここまで見守ってくださった、父、母、姉に深く感謝いたします。

# 参考文献


[1] Hisashi Isonaga, Yasushi Inoguchi, "A Highly Robust Audio Fingerprinting System", Proc.ISMIR 2002 3rd International Conference on Music Informmation Retrival.

[2] Jaap Haitsma,Ton Klker, "A Highly Robust Audio Fingerprinting System", Proc ISMIR 2002 3rd International Conference on Music Information Retrieval.

[3] 磯永久史, "FPGA を利用したコンテンツフィンガープリンティングの高速化に関する研究", 北陸先端科学技術大学院大学情報科学研究科 2005 年度 修士学位論文.

[4] 安田浩, 安原隆一, コンテンツ流通, アスキー,2003.

[5] 金谷健一, これなら分かる応用数学教室, 共立出版,2003.

[6] 末吉敏則, 天野英晴, リコンフィギャラブルシステム, オーム社,2006.

[7] Hisashi Isonaga,Yasushi Inoguchi, "Implimentation of high speed audiofinger-print system using FPGAs", VLD2005-88, CPSY2005-44, RECONF2005-77,pp1-6.

[8] Agility.Corp, "http://www.agilityds.com/"

[9] Celoxica.Corp, "http://www.celoxica.com/"

[10] Xilinx.Corp, "http://www.xilinx.com/"

[11] http://laputa.cs.shinshu-u.ac.jp/ yizawa/InfSys1/basic/index.htm

[12] http://www.kk.iij4u.or.jp/ kondo/wave/

[13] http://www.project9k.jp/

[14] http://lame.sourceforge.net/

[15] http://www.microsoft.com/japan/windows/windowsmedia/9series/codecs.aspx