修 士 論 文

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
情報科学研究科情報システム学専攻

礒永久史

2006年3月

修 士 論 文

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

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　日比野靖 教授
審査委員　松澤照男 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

410010 礒永久史

提出年月: 2006 年 2 月

概 要

近年の楽曲数の増加に伴い、電子透かしやフィンガープリントなど、いくつかの識別方式が提案されている。しかしながら、近年のファイル交換ソフトウェアを使用した不正なネットワーク配信は著作権侵害や CD 売り上げ数減少の原因となり、深刻な問題となっている。DRM(デジタル著作権管理) システムにてインターネット上に流通している膨大な数のコンテンツを識別し、著作権保護の手段とするためには、そのコンテンツの識別処理能力を向上させることが望ましい。オーディオ識別技術であるオーディオフィンガープリントとは、オーディオ内容の知覚や感性の部分をコンパクトに表現したものであり、信号加工処理のため著しく質が劣化したオーディオにおいても、識別するために最も近似しているフィンガープリントを使用できる。本研究では並列処理や演算器のループ展開の手法を用いて、高速なオーディオフィンガープリントをハードウェア上に構築した。処理時間の評価では、ソフトウェアによる処理と比較して、約 10.6 倍の性能が得られた。

目 次

図 目 次

表 目 次

第1章　序論

1.1　研究の背景と目的

本項ではおよび研究の背景と ID を用いたコンテンツ流通情報管理の概要について述べる。

1.1.1　背景

ブロードバンド回線の普及につれて、インターネット上でのコンテンツ流通が盛んに行われている。インターネットの普及率の増加と利用の高度化・多様化に伴い、コンテンツ産業へのビジネス全体に対する脅威が非常に問題になりつつある。脅威とは、つまり、「デジタルコンテンツの不正蓄積」や「デジタルコンテンツの不正流通」である。これらは CD 売り上げ枚数、創作意欲の低下などコンテンツ産業に対して、悪影響を及ぼしつつあり、日本における CD セールスも 2000 年をピークに減少の一途を辿っている。したがって、健全なコンテンツ流通環境にするために、コンテンツの海賊行為 (P2P を用いたファイル交換等) などの行為に対する対策を講じなければならない。

そこで、そのような海賊行為に対処するために、コンテンツ流通機能として DRM システムを導入することが盛んになってきている。DRM システムは、既存のストリーミングサービスなどのコンテンツ配信システムと連携を図ることで実現される。特にコンテンツを事前に DRM システムへ対応させる必要があり、そこには電子透かしやフィンガープリントなどのコンテンツ識別技術が適用される。DRM システムが配備されるブロードバンド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，複数の楽曲識別を高速に試みなければならない。

現在，オーディオフィンガープリント技術は処理速度に着目した研究が行われていないため，ソフトウェアでの超高速なフィンガープリンティングは非常に困難であり，検出コストの大きさから、データベースの中から楽曲を検索するにはさらに多くの時間が必要となる。クラスタなどの並列計算機を利用することで高速なフィンガープリンティングは可能となるが，膨大な経済的コストが大きくなり実用的とは言えず、オーバースペックである。

そこで，簡易的かつ高速にオーディオフィンガープリントするには，LSI を利用した超細粒度並列処理による高速化が非常に有効であると考える。

ハードウェアでオーディオフィンガープリントためのカスタム回路を構築することで、高速に信号を解析し、フィンガープリンティングが可能となる。ソフトウェアでボトルネックとなるディジタル信号処理のステップを DSP に特化したコアを使用することにより、格段に処理速度が上がる。また，開発時間の短縮や低コスト化のために，FPGA 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 の特徴として、簡単にリコンフィギャラブル可能であるという点である。RAM データベースへのフィンガープリントのアップデートや、フィンガープリント回路の並列数の変更などのために再構成するなど利用状況にあった自由な回路構築が可能である。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、経済コスト、スペースコストを大幅に削減することができ非常に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楽曲データの信号そのものに ID を高速に割り振るフィンガープリント回路を FPGA 上に構築する手法を提案する。FPGA はさまざまな特性や制約を持つため、これらを考慮し、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，オーディオフィンガープリントを用いたネットワーク上での楽曲識別手法を提案する。

1.1.2 流通情報管理の概要

コンテンツ流通情報管理とは、ネットワーク上に流れている特定のデータの捕捉を行い、管理することである。流通情報の管理には、基本となる４つのステップで構成される。

ネットワーク上を流通しているコンテンツの状況を把握するためには、対象となるコンテンツを収集する必要がある。コンテンツを収集する際には、コンテンツ自身だけではなく、存在時間、存在場所また、どのように存在していたかという情報もあわせて収集しなければならない。（収集）

次に、収集した物が対象となるコンテンツかどうかを識別・判定する必要がある。ここでは収集したコンテンツの中から特定のコンテンツを見つけ出すという作業を行い、識別子となるコンテンツ ID を特定する。（識別技術）

さらに、コンテンツ ID で識別されたコンテンツがどんなコンテンツであるか、内容を分析するためにコンテンツの内容を示すメタデータを取得しなけらばならない。テキスト情報とメタデータを用いて、どういうコンテンツがどのように使用されているのかを分析する。（分析）

得られた結果をデータベース等に保存して必要に応じて検索できるようにしておき、次の収集や分析の際に使用する。（検索）

上記で述べた、従来の４ステップにおける流通情報管理にて近年の膨大化したデジタルコンテンツ数を管理・対応するためには、このうちの「識別」と「検索」に関する処理コストの削減が急務と考え、それは将来的に DRM システムの性能向上へと繋がると考えた。したがって、本研究では「識別」の処理をハードウェアにて高速化し、コンテンツ流

通情報管理における計算負荷を削減することを目標とする。また、今回はコンテンツのターゲットをデジタルミュージックとした。　米 Apple Computer が運営する i-tune などの音楽配信サイトの利用が高まり、その結果、mp3 等のデジタルミュージックのタイトル数の増加の一途をたどっている。今後、デジタルミュージックにおいては著作権保護を前提とした流通管理の需要が高いと考えられる。

1.1.3　コンテンツの識別

流通情報の管理を行うためには、収集したコンテンツがどのようなものか確認する必要があるが、コンテンツに結合もしくは付加されているコンテンツ ID を知ることができれば、例えば管理データベースに問い合わせることで詳細を調べることも可能である。コンテンツを識別する方法として、信号データの解析には、データの特徴量を用いてコンテンツの ID を識別する方法がある。

データの特徴量を ID とする技術の代表的なものが「フィンガープリント」である。これはあらかじめコンテンツに埋め込まず、コンテンツの信号データの特徴量を利用する手法であり、電子指紋技術とも呼ばれるこの技術は、データ構造上の特徴ではなく信号データそのものの特徴を利用しているため、電子透かしとは異なり、あらかじめ ID を埋め込んでおくのではなく、信号データの特徴量を検索の鍵に用いて、あらかじめハードディスクやメモリなどの大規模データベース等に保存されている ID を検索することによって、コンテンツ ID を特定することが可能である。

1.1.4　フィンガープリント技術の特徴と問題点

近年、音楽、映像などのデジタルコンテンツの膨大化により、ネットワーク上に流通している識別データを捕捉し、コンテンツの識別、管理を行うために、前項で述べたように、３つの流通情報管理の識別方式が提案されてきた。流通情報管理の識別方式の一つであるフィンガープリントは、データ信号そのものの特徴を利用した、強固性が非常に強い技術である。データの特徴量を識別子としているため、データ部分への事前の著作情報埋め込みの必要が無く、ノイズに敏感な音楽ファイルなどに適している。オーディオフィンガープリントの処理の流れを図 1.1 に示す。フィンガープリントはシステムに入力されたオーディオ信号の周波数帯域などの特徴量を解析し、算出された識別子を、データベースに問い合わせることで楽曲の検出処理を行う。しかし、フィンガープリントは計算負荷が高い離散フーリエ変換などデジタル信号処理を用いて特徴解析をしなければならないので、検出時間が長く、他の識別方式と比較して読み取りコストが高くなる問題がある。

ソフトウェア処理ではオーディオの特徴をつかむまでの時間が長くなり、処理速度も低いため、広帯域ネットワーク上で流通している膨大な数の音楽ファイルの管理・補足・識別は不可能である。

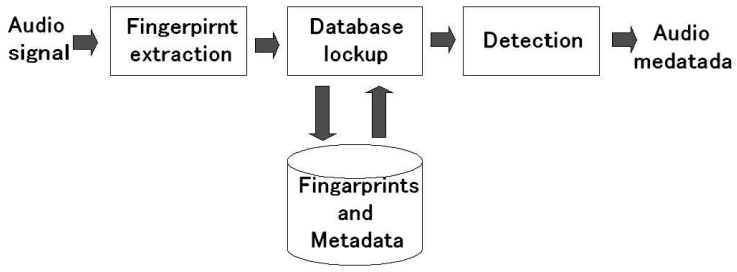

図 1.1: フィンガープリントを利用した音楽の識別

そこで、本研究ではデジタル信号処理を得意とするハードウェアを用いて、オーディオフィンガープリントシステムの構築を行った。しかし、ソフトウェアで実装されたアルゴリズムをそのままハードウェア化しただけでは、性能の向上は不可能である。例えば今回ハードウェア化するアルゴリズムはパイプラインレジスタ等の高速化手法を用いない場合、I/O 時間を含めるとソフトウェアによる処理時間と同等程度となってしまい、意義を失う。したがって、本研究ではパイプライン処理や、モジュールレベルの並列展開などの手法を適用することで、楽曲の検索までの処理時間の削減を目指した。

1.2 本文の構成

本論文は，以下の構成からなる。２章ではオーディオフィンガープリントの概要とハードウェア実装向けアルゴリズムの選定・実装の戦略を示し、選定アルゴリズムの詳細を示す。また、３章ではハードウェア構築の概要、およびフィンガープリント生成回路と楽曲識別回路の詳細を説明する。４章では構築したフィンガープリントシステムの回路量とクリティカルパスの測定、ハードウェア処理とソフトウェア処理の処理時間比較、および考察を行う。５章ではオーディオフィンガープリントを用いた音楽ファイルの不正流通監視システムを想定した実用評価を行う。最後に 6 章でまとめと本研究の課題について述べる。

第2章　オーディオ用フィンガープリントについて

2.1　概論

これまでのオーディオフィンガープリントの従来研究に関しては、強度の向上、サイズの効率化が主であった。Cano らは、全てのフィンガープリントをオーディオに内在する時間-周波数表現から導く方式のプロトタイプとなるオーディオフィンガープリントの概要 [1] を提案した。フィンガープリント算出までの主な違いとは、それらがフィンガープリントを構成するために使用する特徴によって決定される。つまり特徴とは、スペクトル扁平の特徴 [2]、スペクトルのピークの特徴 [3]、フーリエ係数の特徴 [4]、メル周波数ケプストラム係数の特徴 [5]、そして周波数帯間のエネルギー差の特徴 [6] などである。また、Prarthana Shrestha らの研究 [9] では P2P ネットワークによる権限のない音楽ファイルの不正交換を防ぐために、オーディオフィンガープリントに基づく分散システムを提案した。それはネットワークにおける現在の音楽内容を認識することを可能にした。この提案されたシステムは強健であり、動的かつヘテロジニアスな p2p ネットワーク環境に最適である。これは既存するフィンガープリントの検索に対するレイテンシの削減がターゲットである。一方、これまではオーディオデータの特徴抽出部分の高速化に関する研究は確認されていない。

本研究では高速かつ少回路量であり、信頼性の高いハードウェアフィンガープリントシステムの構築を実現するために、MP3 などの圧縮に対して、非常に強健であり、生成アルゴリズムのステップが加算と減算などの簡素な構成であること、回路量が膨らむ乗算および除算が比較的少ないこと、また生成されたフィンガープリントサイズがコンパクトで、FPGA 内で省スペース (FPGA の 4input-LUT 消費など) で格納・容易に検出ができること、などを考慮した結果、Haitsma と Kalker らによる、Philips オーディオフィンガープリントシステム [6] をハードウェア用に改善し、実装を行うことにした。

2.2 オーディオ用フィンガープリントアルゴリズムのハードウェア化への考察

2.2.1 ハードウェア実装の考慮点

本研究では、オーディオ用フィンガープリント処理の計算負荷が高い箇所をハードウェア化することで高速にデジタルコンテンツにコンテンツ ID、つまりフィンガープリントを付加し、同一 LSI 内での楽曲識別を可能とするなど、高性能なオーディオフィンガープリントシステムの構築を目指す。ここでは、オーディオフィンガープリントアルゴリズムをハードウェア化するために考慮する点を明らかにし、ハードウェアに特化したアルゴリズムを検討する。

また、本研究では短い設計時間、開発の低コスト化のため、回路構築は FPGA 上に行う。FPGA を用いることで容易に論理回路の構築が可能となるばかりではなく、論理回路の再構成が行えるため、デバックも短時間で実施可能である。ただし、デメリットとして、商用 FPGA では依然、業務用 ASIC などと比較して、回路規模、メモリ容量の制限が厳しく、限られたリソースの中で性能を向上させる必要がある。浮動小数点による演算や乗除算を含む演算回路は、回路量が大規模となってしまい、FPGA 内では、これらの演算の多用は不可能である。したがって、低リソースにて高性能なオーディオフィンガープリントシステムを実現するため、機能を満たす上で最適な回路を構築することも、研究の重要なポイントとなる。なお、本研究では LSI の消費電力についての性能は議論しない。

ハードウェアでは、演算回路のパイプラインレジスタを用いた並列同時処理や最適化で高速な演算を可能とするため、効率的かつ並列処理を行うことができる演算回路構成とすることが見込めるオーディオ用フィンガープリントアルゴリズムを用いることが、より高性能なハードウェアのシステムを実現できる鍵となる。一般的に浮動小数点演算や乗除算の回路は所要クロック数が多く、ファンアウトの増加に伴いクリティカルパスが増えてしまうなど、演算速度、つまりスループットの低下の原因となる。

以上の議論を考慮した結果、以下のようなアルゴリズムがハードウェアに特化すると考える。

1. パイプラインレジスタによる並列処理や効率的な演算が行えるアルゴリズム
2. 浮動小数点演算や乗除算が少なく、おおむね加算減算によって構成されるアルゴリズム
3. 簡単な定義式で強度、信頼性を実現できるアルゴリズム
4. フィンガープリントサイズがコンパクトなアルゴリズム
5. フィンガープリントの出力までのステップが少ないアルゴリズム

6. **音声特徴解析** (DSP) **ステップにおいて、回路構築が容易、かつ高性能が演算が見込めるアルゴリズム**

2.2.2 適用アルゴリズムの検討、および選定の根拠について

本項では前項の箇条書きで述べた点を考慮しながら適用アルゴリズムへの道筋を開く。

本研究では、オーディオ用フィンガープリントアルゴリズムをハードウェアに実装することで、楽曲に対してフィンガープリントの短時間で付加し、膨大な楽曲の中から検索・識別を同時に行う高性能なシステムを構築する。そこで、ハードウェア化に適したアルゴリズムの検討を行い、そのアルゴリズムを論理回路にて構築する。

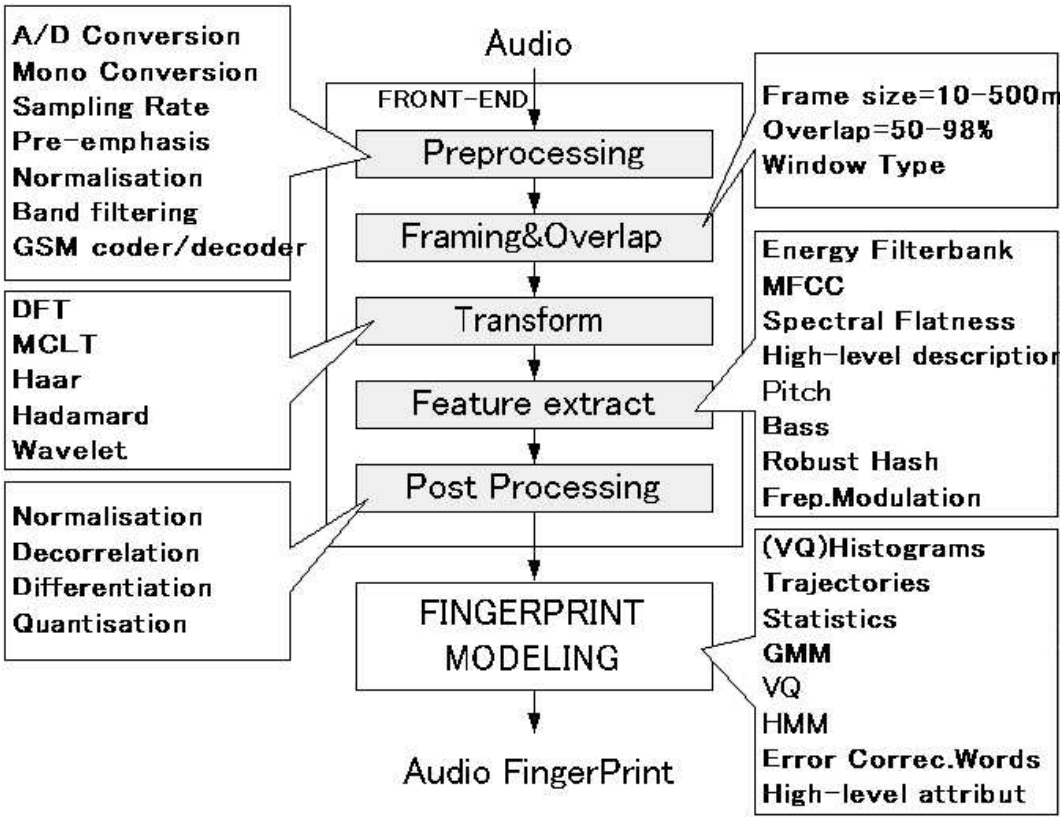

図 2.1: **オーディオフィンガープリント生成** Framework

図 2.1 **は従来手法のオーディオフィンガープリントのフレームワークをまとめたものである。**

オーディオフィンガープリントは信号そのものの特徴を捉えることを基本としているので、信号解析を行うための「Transform**」ステップは必要不可欠なものとなってくる。しかし、この「**Transform**」は依然として計算負荷が高く、フィンガープリントの計算コストを大きくする要因となる一つである。この「**Transform**」の候補の中で、特に離散フー**

リエ変換は、乗除算を多用しているため、回路量も多少ながら消費するが、他のデジタル信号処理にも広く用いられ、ハードウェアにおいてはアクセラレータなどの用途で使用するために、研究者や開発者らによって一般的な変換モジュールとして高速化、省スペース化、省電力化が達成されている。本研究では、比較的容易に構築できる「フーリエ変換」を用いたアルゴリズムがハードウェアによるフィンガープリントシステムにとって、高速性、拡張性、利便性の面から最良であると考えた。

楽曲の特徴を算出する「Feature extract」は信頼性・ロバスト性を決定づけるステップであり、フィンガープリントの核となる部分である。Haitsuma らの研究 [6] ではエネルギバンドの時間的ディレイの差分（時間軸と周波数軸の両方）のサイン値「Hash String」を得るために３３に区分けされたエネルギー値を利用する特徴抽出法を提案した。これは従来手法と比較して、楽曲の知覚的な数値を ID とするのでロバスト性が非常に高く、また、その特徴抽出式は加算と減算だけで定義されているため、計算量も多くはない。

つづいて、適切なアルゴリズムを得るため、オーディオフィンガープリントシステムのパラメータの違いについて焦点を当てる。主なパラメータとして、ロバスト性、信頼性、フィンガープリントサイズ、粒度、検索速度・拡張性が挙げられる。

- ロバスト性
  たとえ、厳しい信号劣化の後でも、オーディオを識別できるかどうかを測るパラメータである。高いロバスト性を達成するためにはフィンガープリントを信号劣化に対して不変である「知覚的特性」に基づかせるべきである。望ましくは、厳しい劣化の楽曲も、かなり類似したフィンガープリントにつながる。ロバスト性は「偽陰性」の確率によって表現され、その誤り率とは知覚的には類似しているように見える楽曲があまりにも誤差が大きく、整合できなかった場合に発生する。

- 信頼性
  間違った識別が何度起こるかを数値化したパラメータである。つまり、この確率は通常、「偽陽性」を参照した場合に発生する。信頼性とロバスト性は、トレード・オフの関係にあり、システムとしての最適値を状況に応じて決定する必要がある。

- フィンガープリントサイズ
  高速にフィンガープリントを検索するために、通常、フィンガープリント特徴を行うデバイスと高速アクセスが可能な大容量のディスクなどに保存される。それゆえ、フィンガープリントサイズ（bit per sec もしくは bit per song で表現）はフィンガープリントデータサーバーのために必要とする膨大なメモリリソース量を決定付けるパラメータとなる。

- 粒度
  楽曲を識別するために、どれだけの秒数が必要であるかを示すパラメータである。粒度はアプリケーション次第であり、あるアプリケーションでは識別のために、楽曲の全体を利用する必要があり、他においては、むしろ、楽曲の短い抜粋のみでの

識別をするアプリケーションもある。当然、粒度は小さければ小さいほど、処理時間を削減できるが、ロバスト性、信頼性を考慮した場合、それは一概に良いとは言えない。

- 検索速度/拡張性
  現在、問い合わせる楽曲が、フィンガープリントを格納するデータベース中から検索するのに、どれだけの時間が必要かを指し示すパラメータである。検出を完了するまでの時間は「フィンガープリント特徴抽出時間」と「データベース検索時間」の和となる。商用のフィンガープリントシステムでは、この検索スピードと拡張性は重要なパラメータとなる。検索スピードは (限られた計算機資源のみの利用で) １０万曲以上を含むデータベース中から、ミリセカンドオーダを達成することが望ましい。

これらの基本的となる５つのパラメータは、お互いに多大な影響を与え合う。例えば、少粒度を求めるならば、同じだけの信頼性を得るためには、大きなフィンガープリントが必要である。これは偽陽性の割合が、フィンガープリントサイズと反比例関係にあることに起因する。他の例では、ある設計においてフィンガープリントのロバスト性次第で、検索速度は増加、もしくは減少する。これはフィンガープリント検索自体が近似検索であることに起因する。つまり、近似のロバスト性がより小さくなる特性である場合、類似したフィンガープリントも見つけられなければならない。それゆえに、検索速度は増加する。

前項でも述べたように、Haitsuma らの研究 [6] は楽曲の知覚的な数値を ID とするのでロバスト性が非常に高く、かつ計算量も少ない。また、フィンガープリントサイズは１曲当たり 1KB とコンパクトなサイズで表現可能であり、それは大容量の楽曲データベースの構築が安価な設備で構築できることを示す。その上、粒度、信頼性においても非常に優れている。

さらに、ハードウェア化の観点から検討した場合に、[6] のアルゴリズムはステージ毎に区切られていることから、パイプラインレジスタによる並列動作、もしくは演算器モジュールの並列展開が可能であり、大幅な性能向上が可能であると考えた。したがって、本研究では「Haitsma と Kalkerr らのオーディオフィンガープリント特徴生成アルゴリズム [6]」を採用することにした。

2.3 Haitsma と Kalkerr らのフィンガープリント特徴抽出アルゴリズム

図 2.3 に Haitsuma らが提唱したフィンガープリントのシステムの生成ステージの概観を示す。最初に音声信号は、31/32 のオーバラップされた要素である、0.37 秒のフレームに分割され、入力される音声信号は窓関数「Hanning Window」により重み付けられる。この窓関数は中央値が 1 の Raised Cosine の波形になっている。主成分の周波数分解能は

やや劣るが、サイドローブが比較的小さいため、小さい電力のスペクトルを検出するのに向いている。Hanning Wondow は

$$h(n) = \begin{cases} 0.5 - 0.5\cos(\frac{2\pi n}{N-1}) & (0 \leq n < N-1) \\ 0 \quad (\text{その他}) \end{cases} \tag{2.1}$$

で定義される。

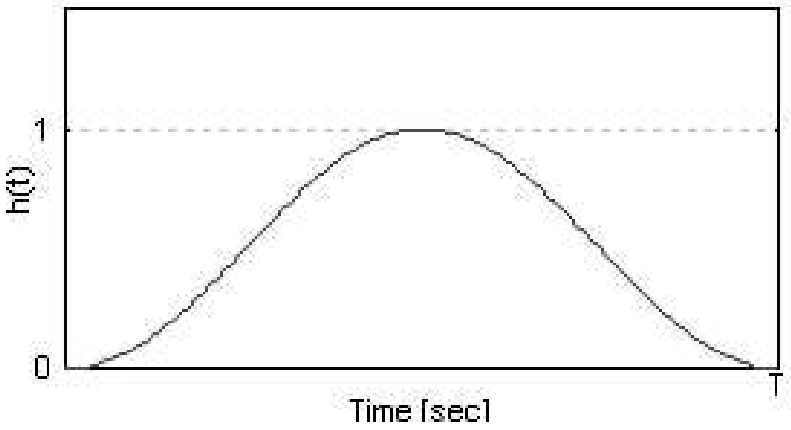

図 2.2: ハニングウィンドウ

分割されたフレームのコンパクトな表現は、サブフィンガープリントと呼ばれる。それは 11.6ms(370ms) 毎の間隔で、32 ビットのサブ-フィンガープリントを生成する。

オーバラップ（周波数の重なり）が大きいため、サブフィンガープリントの系列は大きな類似性を持ち、時間内で少しづつ異なっていく。 楽曲はサブフィンガープリントの 256 個の連続した系列で構成し、データベースに格納される。

それぞれのフレームに 32 ビットのサブフィンガープリントを生成するために、フーリエ変換から出力されたパワースペクトル密度関数 (power spectral density function) にて 33 個の非重である周波数帯のエネルギーが算出される。パワースペクトル密度とは信号のパワーを一定の周波数帯域毎に分割し、各帯域毎のパワーを周波数の関数として表したものである。単位は振幅の２乗となる。パワースペクトル密度関数は、簡単なピリオドグラム概算を使用することで見積もられる。

音声データ等の循環的な波形の変動は、周期の異なるコサイン関数の結合を用いて記述できる。また各々のコサイン関数は、その周期と位相の係数を持っており、位相の係数は初期条件を表わすため、変動の持つ循環的な特徴は周期の係数に集約されている。このように、過程 {xt} を、時間領域 (time domain) ではなく、含まれる循環の周期の長さで区分した周波数領域 (frequency domain) において扱うのが、スペクトル解析と呼ばれる分野である。

簡単な循環系列

$$x_t = R\cos(\theta t + \xi) \tag{2.2}$$

を考えた場合。f=θ/(2π) はコサイン波の単位時間あたりの周波数、ξ は初期位相、R は振幅を示している。また、1/f=(2π)/θ は波長または周期である。

より複合的な循環系列は、単純な循環構造である式 (2.2) を組み合わせることで、

$$x_t = R_1 \cos(\theta_1 t + \xi_1) + R_2 \cos(\theta_2 t + \xi_2) + \bullet\bullet\bullet + R_M \cos(\theta_M t + \xi_M) \tag{2.3}$$

のように構成することが可能である。

一方、三角関数の関係から

$$\cos(\theta t + \xi) = \cos\theta t \cos\xi - \sin\theta t \sin\xi \tag{2.4}$$

が成り立つ。これを用いて (2.2) を書き直すと、

$$x_t = \sum_{j=1}^{M} R_j \cos(\theta_j t + \xi_j) = \sum_{j=1}^{M} \{a_j \cos(\theta_j t) + b_j \sin(\theta_j t)\} \tag{2.5}$$

$$(a_j = R_j \cos\xi_j, b_j = -R_j \sin\xi_j)$$

と表すことができる。したがって、過程 {xt} から、各種の $\theta_j (0 \leq \theta_j \leq \pi)$ についての a_j, b_j の値を求めることができれば、循環成分毎の影響力 R_j を推定することができる。また、この考え方を θ に関する連続的な分布概念に拡張したものがパワースペクトル密度関数 (power spectral density function) であり、(2.4) はその特殊ケースと考えることも可能である。式 (2.4) を、標本サイズ 2M のデータについて $\theta_j = (j/M)\pi(j = 1, .., M)$ と設定する分析方法は、フーリエ (Fourier) 解析あるいは調和解析と呼ばれる。以下では、平均がゼロでないデータも扱えるように、$\theta_j = 0$ の項を j=0 のケースとして付加すると、

$$X_t = \sum_{j=0}^{M} R_j \cos(\pi j t/M + \xi_j)(t = 1, ...2M, \xi_0 = \xi_M = 0) \tag{2.6}$$

$$= \sum_{j=0}^{M} \{a_j \cos(\pi j t/M) + b_j \sin(\pi j t/M)\}(t = 1, ..., 2M, b_0 = b_M = 0) \tag{2.7}$$

となり、これがピリオドグラム (periodogram) 概算となる。

これらのピリオドグラム概算から得られる周波数帯域の範囲は人の聴覚の特性と同じ 300Hz から 2000Hz と対数関数により区切られる。

Haitsma と Kalker は式 (2.6) で算出されるエネルギー差のサインが多くの種類の処理に非常に強健な特性であることを示した [6]。

文献 [6] の記法を使用して、ここではフレーム n の周波数帯 m のエネルギーを E(n、m) で記す。これらのエネルギー差 ED(n,m) は時間と周波数で計算され、

$$ED(n, m) = (E(n, m) - E(n, m + 1)) - (E(n - 1, m) - E(n - 1, m + 1)) \tag{2.8}$$

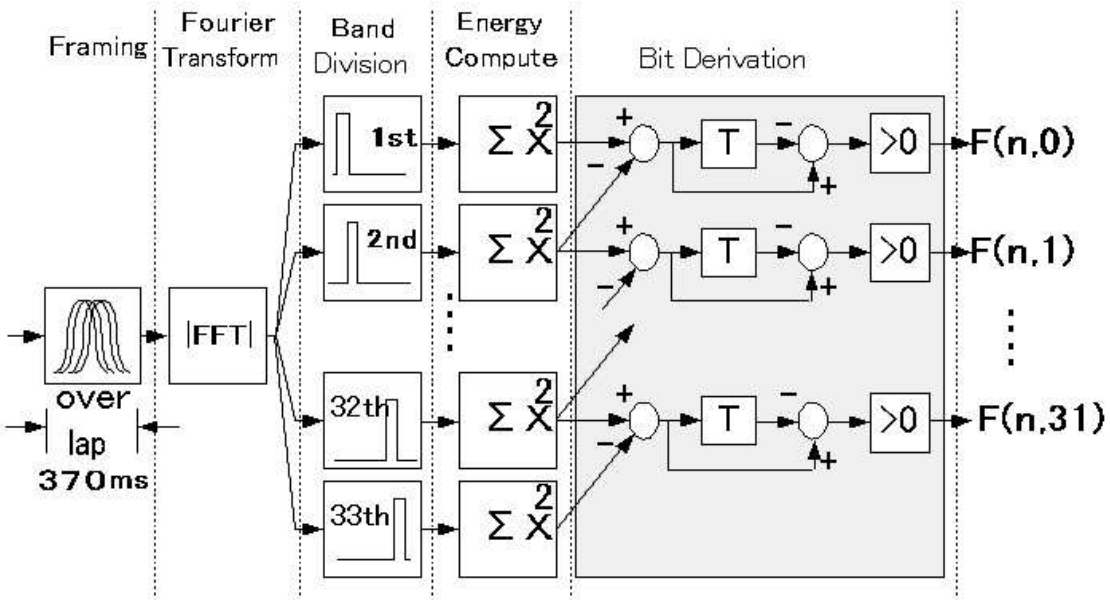

図 2.3: **左：**Haitsuma **らのオーディオフィンガープリント特徴生成 右:**FP **例 白→** F(n,m)=1 **黒→** F(n,m)=0

(**図** 2.3 **の網掛け部分**)

サブフィンガープリントのビット F(n,m) **は**

$$F_{(n,m)} = \begin{cases} 1 & ED_{(n,m)} > 0 \\ 0 & ED_{(n,m)} \leq 0 \end{cases} \tag{2.9}$$

から算出され、式 (2) **の** F(n、m) **は** n **フレーム目の第** m **ビットを意味する。図** 2.3 **の右側は実際のフィンガープリントの例を示す。白い部分は、エネルギー差が** 1 **であることを示して、黒い部分は、**0 **であることを示す。 フィンガープリントブロックの周波数軸は、**Haitsuma **らの定義、すなわち人の聴覚の特性である** 300Hz **から** 2000Hz **の中から抽出した** 33 **個の基本周波数成分の間の違いに対応した** 32 **ビットから成る周波数帯の傾向であり、ブロックの横軸は時間軸** (1 **フレームを** 370ms、256 **フレーム分のフィンガープリントを生成する**) **に対応する。**256 **個のサブフィンガープリント** (**合計サイズ** 1KB) **を発生させて、たとえ楽曲のフレームがさまざまな信号加工処理のために品質劣化しても、このシステムは、大容量データベースでわずか** 3.3 **秒の音楽のフレームを識別することを可能とした。「生成されたフィンガープリント」と「データベースのフィンガープリント」の間の** Bit Error Rate(BER) **は識別のための類似性測定のパラメータとして使用される。**

BER **が閾値の下にある場合は、照合値がデータベース項目で見つけられたということである。なお、このシステムでは、閾値は** 0.35 **に設定される。**

2.4　wav ファイル構造

本研究では、Windows 標準の音声ファイルの形式である wav ファイルの特徴量を解析し、オーディオフィンガープリントを付加することとする。wav ファイルは音声信号をデジタルデータに変換したものを記録するための保存形式などを規定している。符号化方式については規定しておらず、任意のものを利用することができる。デフォルトでは PCM(無圧縮) 方式や ADPCM 方式などの圧縮方式に対応している。Microsoft 社や他社が提供する「CODEC」と呼ばれるソフトウェアを追加することにより、MPEG-1 Audio Layer III 方式 (MP3) や、Indeo Audio 形式などの様々な圧縮方式を利用することができるようになるなど、デジタルミュージックの最も基本となるファイル形式である。

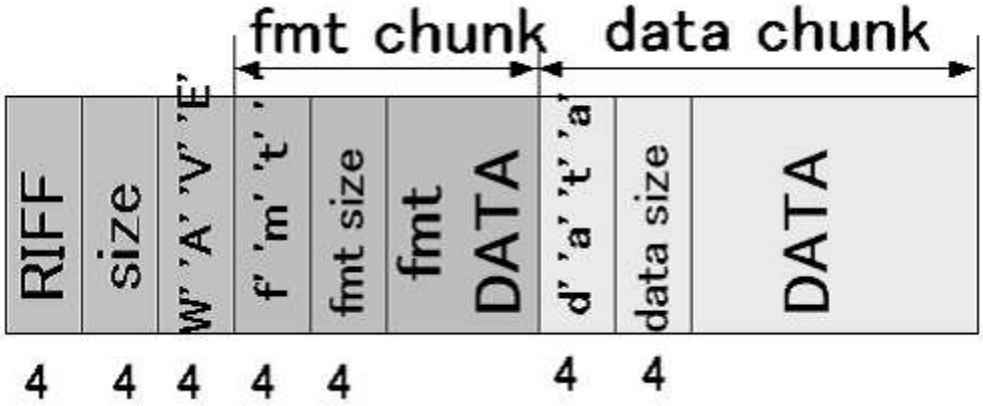

図 2.4: wav ファイルの構造例

wav ファイル構造を図 2.4 に示す。図中の「fmt chank」の DATA 部に wav のファイル情報 (サンプリングレート、チャンネル数など) が格納され、実際の音データは「data chank」の DATA 部に格納される。本研究で利用する wav フォーマットはサンプリングレート 44.1KHz、チャンネル数は 2(ステレオ)、ビット数は 16bit とする。つまりサンプリングレートが 44.1kHz なので、１秒間に 44100 個のサンプルがあることを示し、その中の 1 個のサンプルは 16bit ということである。

2.5　ソフトウェア実装による先行実験

本項では本研究で採用する [6] のアルゴリズムの性能を知るために、ソフトウェアにて実装することで予備実験を行った。

2.5.1　フローチャート

図 2.5 はソフトウェア処理の全体の流れを示す。まず、ファイルオープン関数で wav ファイルを読み込み、ファイルフォーマットをチェックする。もし、本研究で使用するフォーマット以外の形式であれば処理を終了する。使用するフォーマットはデータの以下のとおりである。

図 2.5: ソフトウェア処理全体の流れ

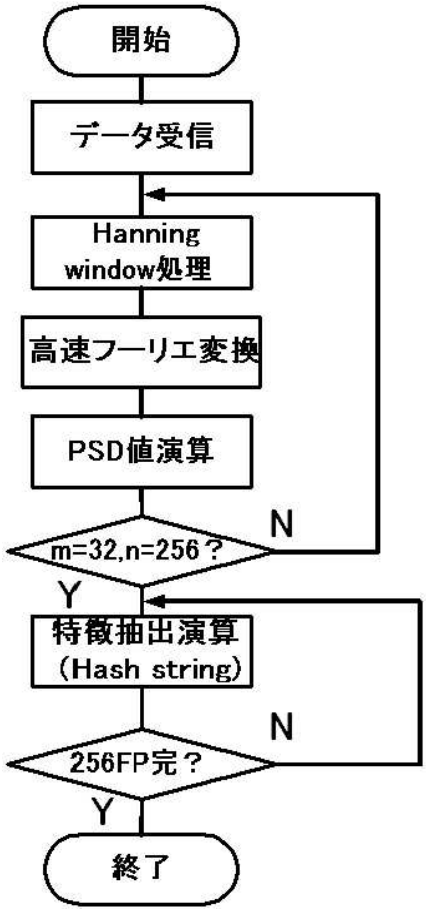

図 2.6: フィンガープリント生成の流れ

- 4Byte 'R' 'I' 'F' 'F' (データ先頭)
- 4Byte これ以降の Byte 数 (= ファイルサイズ - 8)

- 8Byte 'W' 'A' 'V' 'E' 'f' 'm' 't' ' '(←スペース)
- 4Byte WAVEfmt 欄の Byte 数，つねに 16 (10 00 00 00)
- 2Byte データ形式 (PCM: 01 00)
- 2Byte channel 数 (モノ: 01 00 ステレオ: 02 00)
- 4Byte sampling rate (44100Hz なら 44 AC 00 00)
- 4Byte Byte/sec (44100Hz ステレオ 16Bit なら 10 B1 02 00)
- 2Byte Byte ／ sample × channel 数 (ステレオ 16Bit なら 04 00)
- 2Byte Bit ／ sample (16Bit なら 10 00)
- 4Byte 「欄」の名前 ('d' 'a' 't' 'a'，'f' 'a' 'c' 't' など)
- 4Byte この「欄」の Byte 数 n
- nByte 「欄」の中身

ただし、欄は 'd' 'a' 't' 'a' だけを必須とした。欄 'd' 'a' 't' 'a' の内容には次の形式でデータが入っている。

- ステレオなら L R L R ... の順
- 8Bit データは 0〜255 (無信号は 128)
- 16Bit データは -32768〜+32767 (無信号は 0)

読み込んだ wav データがステレオである場合、左チャンネルのデータのみをメモリに保存し、フィンガープリント関数にデータを渡す。図 2.6 にフィンガープリント生成処理の流れを示す。フィンガープリントを生成するために、まず、音声データファイルを入力としてフーリエ変換を用いて基本周波数をフレーム（最大フレーム数２５６）毎に抽出することが最初のステップとなる。離散フーリエ変換（高速フーリエ変換も同様）では、暗黙のうちにデータの周期性が仮定されているため、 右端と左端のデータ値が大きく異なると、その部分で急峻に変化しているような影響が現れ、 結果として高調波成分の歪が発生してしまう。 その対策として、切り出した 16bit の音声データの両端の影響をどのように押さえるため、データに窓関数を掛け合わせる。フレーム毎に３３個の基本周波数に分割され、基本周波数毎にパワースペクトル密度値を算出する。求められた値を Haitsuma らが定義した、式 (2.8) から特徴量を算出する。

表 2.1: 予備評価の実装環境

OS	Fedora Core3
CPU	Opteron プロセッサ 2GHz × 2
Memory	2048MB
利用ファイル	wav 44.1KHz、16bit、左 ch

2.5.2 実装と測定

表 3.1 に示す環境にて Haitsuma らのフィンガープリントアルゴリズム [6] のソフトウェア実装を行った。なお、開発言語は C 言語とした。

プログラムの実行時間を time コマンドにて測定を行った。表 3.1 に測定結果を示す。当初、フーリエ変換は離散フーリエ変換 (計算オーダ $O(n^2)$) にて演算していたが、演算時間が非常に長いため、高速フーリエ変換 (計算オーダ $O(nlogn)$) へ置き換えを行った。wav データ 256 フレーム分 (約 16384 サンプル/1 フレーム-370ms つまり 16bit × 16384 サンプル× 256 フレーム=8.38MByte) をフィンガープリント付加するユーザー CPU 時間（user）は 3.353[sec] となり、その結果から、処理ビットレートは 19.95Mbps となった。周波数解析に高速フーリエ変換を用い、高性能なデュアルプロセッサにて処理したにもかかわらず、高速な処理は不可能であった。またメモリ／ CPU 間のデータ入出力など時間も含めると、実時間 (real) は約 7 秒となってしまう。フィンガープリントの処理時間は、結局、信号解析を行うための変換部や特徴抽出部の演算時間に大きく依存していると思われ、その過程の演算時間の削減が、本研究の重要な課題となることがここで判明した。

表 2.2: アルゴリズム [6] のソフトウェア処理時間

	real	user	sys	速度 (bps)
FFT 使用	7.115s	3.353s	0.078s	19.95M

2.6 まとめ

本項ではオーディオ用フィンガープリントの概要を説明した。オーディオ用フィンガープリントは、データそのものの特徴量を ID とすることから、信号解析処理・演算に非常に時間を費やすため、ほとんど実用化されてこなかった技術である。本研究では計算負荷の高いステップを DSP を得意とするハードウェアを利用することで、フィンガープリンティングの高速化を実現する。ハードウェア化するフィンガープリントアルゴリズムは、一般的なハードウェアの性能向上手法である「パイプラインレジスタ」を適用できることを第一に検討した。また、少回路量での構成を目指すため、可能な限り、浮動小数点演

算、乗除算を使用しないアルゴリズムを選定した。第３番目にフィンガープリントサイズが、コンパクトなアルゴリズムが重要な鍵となる。ハードウェア上に実装している、限りあるメモリ資源に、なるべく多くの楽曲データベースを格納することが、フィンガープリントシステムとして望ましいからである。

これらの意義深い検討の結果、本研究では数ある文献の中から、フィンガープリントの特徴抽出アルゴリズムは Jaap Haitsma , Ton Klker らの論文題名"A Highly Robust Audio Fingerprinting System"出展"ISMIR 2002 3rd International Conference on Music Information Retrieval"のアルゴリズムを選定した。Haitsuma らのアルゴリズムの特徴抽出式はパワースペクトル密度の時間的位相差から成り立っている「Hash　String 法」を採用しているため、これらは音楽の感性的な領域を、そのまま ID 化する技術であるので、非常に強度性（ロバスト性）が高いと言う事ができる。つまり、圧縮やその他の処理が原因により多少、質が悪化したデジタルミュージックにおいても、識別を可能とする特長を持つ。また、計算の流れ、ステップ形態がパイプラインを実装が容易であり、ハードウェアに特化している。さらに、フィンガープリントサイズが 1KB/１曲と非常にコンパクトで、少ないメモリ資源で大容量の楽曲データベースが構築が可能である点が、このアルゴリズムを採用した最大の特典である。楽曲の検索アルゴリズムについては、「判定確率を意識した BER に基づく楽曲識別法」を提案し実装を行う。アルゴリズムの詳細については３章に示す。

また、本項では予備実験と題して、採用したアルゴリズム [6] のソフトウェア処理での性能を検証した。測定結果から、ソフトウェアによる手法では、局所的なコンテンツ識別には対応できる速度であるが、100Mbps クラス以上の広帯域なネットワーク上などの環境において、次から次へ流れてくる膨大な数の音楽コンテンツの捕捉・識別は不可能であると考える。このことから従来、この技術がオーディオファイルの管理技術として、あまり有効利用されてこなかった理由が少なくとも掴めたと思われる。

第3章　ハードウェア構築について

3.1　本研究のハードウェア実装の特徴

一般的にソフトウェアで実装されたアルゴリズムをそのまま vhdl 記述等でハードウェア化しても、高速化できるとは限らない。例えば今回ハードウェア化するアルゴリズムはパイプラインレジスタ等の高速化手法を用いない場合、オーディオデータ 1 フレーム当たりの処理時間は N(データ数) $= 256, \tau_{non}$(パイプラインを用いない時の１ $task$ の処理時間 $) = 17.55\mu s$

$$T_{nonpipe} = N \times \tau_{non} = 256 \times 17.55(\mu s) = 4492.8(ms)(1\ \text{フレーム}) \tag{3.1}$$

全てのフィンガープリントを完了する 256 フレームでは 1150ms 以上の演算時間が必要となる。これでは I/O 時間を含めるとソフトウェア処理時間と同程度となってしまい、意義が薄れ、オーディオフィンガープリントを DRM システムに対応させる能力に至らないと判断する。

LSI を使用して性能向上させる手段として、超細粒度並列処理が挙げられる。これは計算負荷の高い専用回路を並列展開し、負荷を分散させることで、計算時間の削減を達成する。今回、使用する開発環境はホスト PC と FPGA を PCI バスで通信させる形態を取っている。したがって、FPGA の入力はシリアルであるため、論理回路内にて入力データの分散処理を行うことは極めて難しく、それに伴うオーバヘッドも大きくなってしまい、効率の良い実装が不可能であると判断した。

もう一つの案として、ひとつのフィンガープリント回路の中で並列処理を行うことである。つまり、各組み合わせ回路間にパイプラインレジスタを挿入することで各演算の並列処理を行う。この手法により、高スループットにてフィンガープリンティングが可能になると思われる。

採用するアルゴリズムの 33 個の周波数帯を演算する独特のステップに着眼した比較器のループ展開を行う。これにより比較処理全体で必要となるクロックサイクル数は 1/32 に削減されることで、さらなる処理時間の短縮化を実現する。

楽曲識別部は BER に基づいた楽曲識別法を提案し、その提案アルゴリズムを実装する。BER に基づいた楽曲識別法は BER が低いとその楽曲であるという、相補誤差関数を用いた確率定義式から立案した。この BER は問い合わせフィンガープリントとデータベースフィンガープリントのハミング距離から算出されるもので、同時に楽曲の識別を行うため、並列に BER 演算を行う。一度に複数の楽曲識別が可能なため、並列数を多くするこ

とで、検索時間の短縮が見込まれる。既存のパタンマッチングと融合した高速探索アルゴリズムと比較しても同等以上の性能が得られる。

3.2 ハードウェア構築の概要

本項では、ハードウェアにおける、オーディオフィンガープリントシステムの高速設計手法について説明する。本研究で提案したハードウェアにより、高速化されるシステムのTOP構成は既存のフィンガープリントシステムと同様、フィンガープリント生成部と楽曲検索・識別部のステップ構成にVirtual Turbo専用API制御LSIと同期を取り、リード、ライト制御を行うためのインターフェイス部を加えた３ステップ構成とした(図3.1参照)。ここでのフィンガープリント生成部とは、図2.3のHaitsumaらのフィンガープリントアルゴリズムを、ハードウェア向けに改良したものである。ハードウェア部はホストPC上

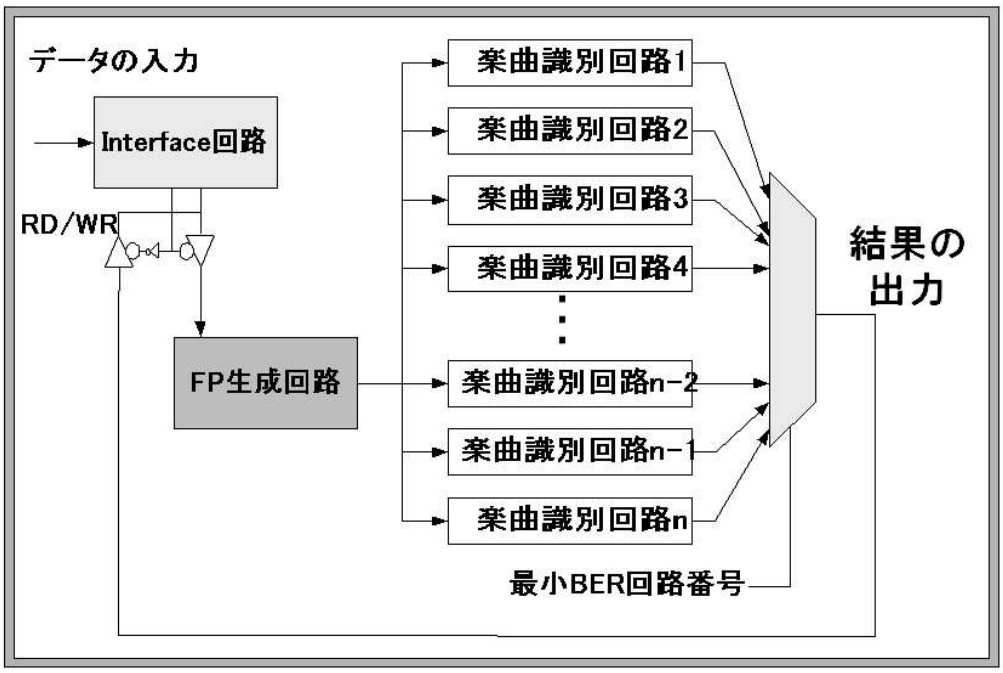

図 3.1: フィンガープリント生成回路の構成

のC++プログラムと連動を行う。ホストPC上でFPGAボードの専用の書き込み用API関数を実行することで、送信されてきた音楽ファイル(wave、16bit、44.1KHz)を入力とし、2章で既に説明したHaitsumaらの生成アルゴリズムに準じてフィンガープリントを生成する。フィンガープリント生成部は各演算部間へのパイプラインレジスタ挿入や比較器の並列展開などの手法により、性能の向上を目指した。楽曲識別回路にはFPGAに標準搭載されているBRAMもしくはCLBによって構成されるROMを配置し、そのメモリ領域には、あらかじめ入力が予想される楽曲のフィンガープリントを格納しておく。楽曲

識別回路を並列に配置し、同時に複数の識別処理を行うことにより、楽曲識別の所要時間の削減を行う。

3.3 実装環境

表 3.1: ハードウェア構築の実装環境

FPGA ボード	日立　Logic Bench
HW/SW インターフェイス	日立 Virtual Turbo PCI
搭載 FPGA	Xilinx XC2V6000FG1156-4×4
ホスト PC CPU	Athron 3.6GHz
ホスト PC OS	Windows2000 Pro
ホスト PC メモリ	１ GB
連動アプリケーション	Visual Studio.net
開発環境	Xilinx ISE6.3i ,ISE8.1
シミュレーションツール	Mentor Modelsim for Xilinx
論理合成ツール	Synplicity Synplify Pro V7.3.4
ロジックベンチ論理分割ツール	Logic bench Compiler,ACE Compiler
開発言語	VHDL 、C++

オーディオフィンガープリントシステムの実装環境を表 1 に示す。日立製の FPGA ボード「Logic bench」は、Xilinx の FPGA「XC2V6000」を４個搭載しており、ゲート数は 2400 万システムゲートと大規模論理検証が可能である。

また「XC2V6000」は、BRAM を約 9Mbit 相当を搭載している。ホスト PC と FPGA 間のブリッジは日立製「Virtual Turbo　PCI インターフェイスボード」を使用した。この PCI ボードは SW・HW 連動シミュレーションのための専用 API が用意されており、多様なシステム開発に適応できる。

また、基幹クロックは 33MHz ではあるが、OSC の予備ソケットが搭載されているので、さらに高速なクロック周波数で動作させることも可能でであり、拡張性に優れている。

3.4 実装手順

本項では構築するフィンガープリントシステムの実装手順を簡単に説明する。図 3.3 に実装手順を示す。まず FPGA ボード「Logic Bench」に実装する論理回路を設計する。vhdl で構築するフィンガープリント回路を Xilinx ISE6.3i 付属の合成ツール XST(Xilinx Synthesis Technology) にて論理合成を行い、Mentor の Modelsim を使用することでタイ

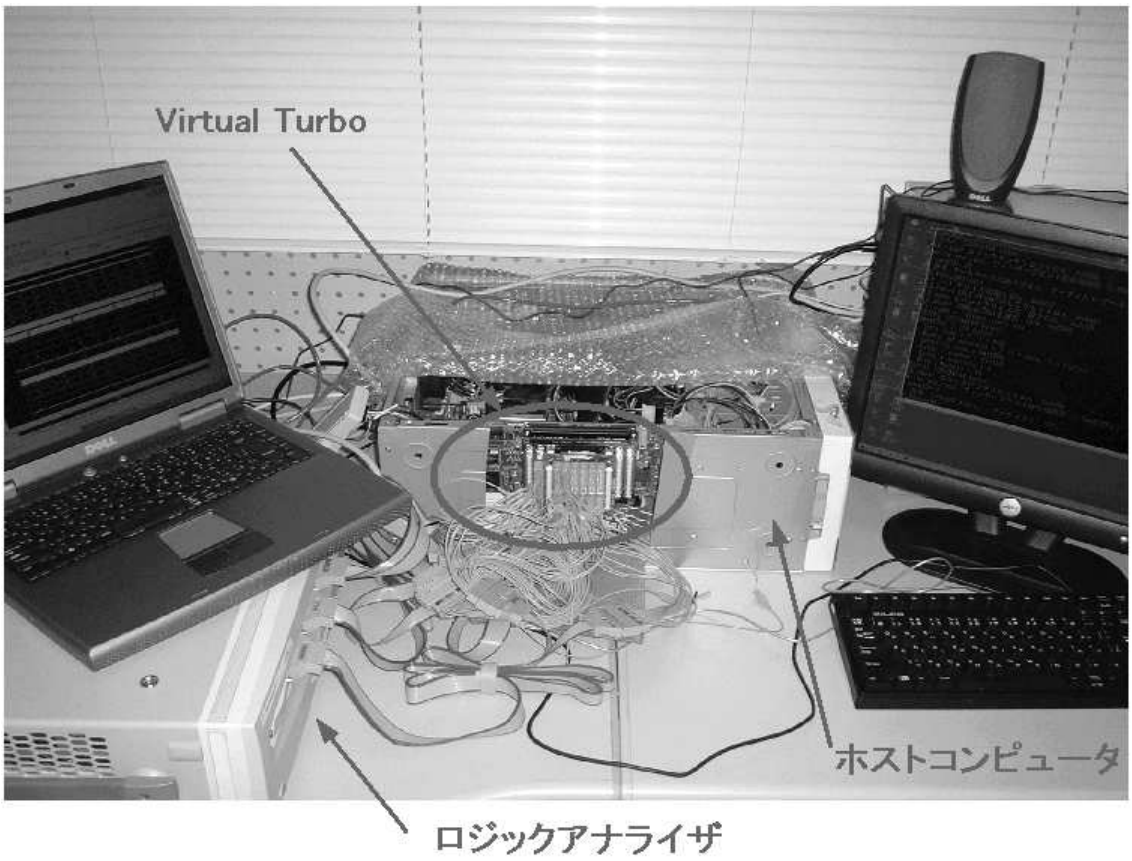

図 3.2: **実装環境　風景**

ミング確認などの簡単なデバックを実施した（フィンガープリント回路に対応したテストベンチファイルを作成）。ある程度の動作確認が取れた段階で、今度は Synplify Pro **にて** edif(Electronic Design Interface Format) **形式に論理合成をする。** Logic bench **上の４つの** FPGA **に効率的に論理分割を行うためのツール「**Logic bench Compiler**」(以下** LBC) **の** EDIF **入力は最上位層とその直下の層である機能ブロックだけの規定となっているため、ここで論理合成する** vhdl **ファイルは「**top.vhd**（最上位）」、「**interface.vhd**（インターフェイス部）」、「**afp.vhd**（フィンガープリント生成部）」、「**afp_database.vhd(**楽曲識別部**)」の４つのファイルとなる。** LCB **は論理分割、ピン配置制約を実施し、それが反映された各** FPGA **に実装するための** EDIF **ファイルおよび** UCF(User Constraints File) **を出力する。** UCF **とはユーザが定義したピン配置・配線や制約が反映されたファイルである。** EDIF **と** UCF **はマッピングされるべき** FPGA **全ての数** (switch **用** FPGA **も含む) だけ生成される。生成された** EDIF **と** UCF **から** Xilinx ISE **の** Generate programming file **モードにて、** FPGA **に書き込むための** bit **ファイルを生成する。** bit **ファイル生成パラメータは以下とした。**

- -g StartUp **Ｃｌｋ**:**ＪＴＡＧＣｌｋ**

- -g DONE_cycle:4
- -g GTS_cycle:DONE
- -g GWE_cycle:DONE
- -g M0Pin:PULLNONE
- -g M1Pin:PULLNONE
- -g M2Pin:PULLNONE
- -g DonePin:PULLUP
- -g UnusedPin:PULLUP

Logic bench の制約により未使用ピンはコンフィギュレーション後に、PULLUP されるように設定した。Virtual Turbo には JTAG 方式で FPGA が接続されているため、Parallel Cable を使用して実機への bit ファイルのダウンロード (コンフィグレーション) を行った。IMPACT のバウンダリスキャンモード (図 3.4) を使用し、9 個の FPGA にコンフィグレーションを行う。(マッピングしない FPGA でもダミービットファイルを書き込む。)

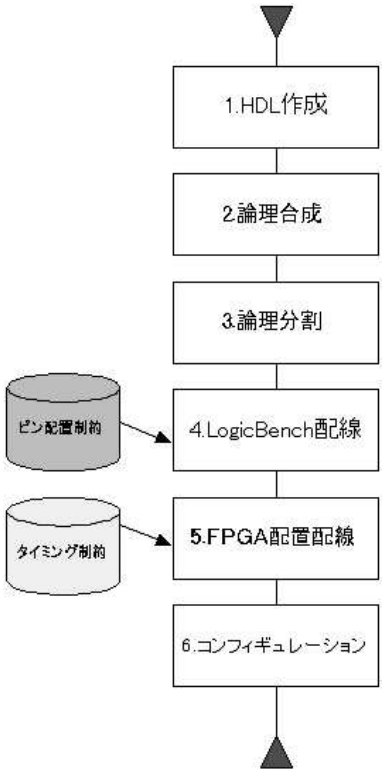

図 3.3: 実装手順

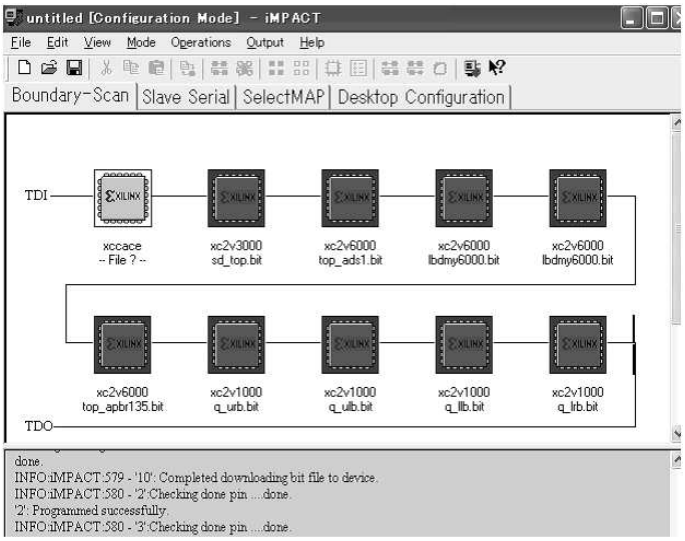

図 3.4: IMPACT を使用した場合のコンフィギュレーション風景

3.5 ホストPC-ハードウェアの通信、処理分担について

本研究では Virtual Turbo で C プログラムと LogicBench を連動させる。そこで、Virtual Turbo のイベント型アクセスという機能を利用した。イベント型アクセスの概念図は以下のようになる。(図 3.5 参照) 動作させたいフィンガープリント回路を Logic Bench にマッピングさせ、ユーザアプリケーション（Visual Studio.net）から API を呼び出すことにより、PC 上のソフトウェアと Logic Bench 上の回路と書き込み、読み出しなどの通信が可能となる。また、フィンガープリントシステムのソフトウェアとハードウェアの処理分担を図 3.6 に示す。C プログラムの計算負荷が高い部分をハードウェアで高速化する（アクセラレータ）概念となる。Virtual Turbo には書き込みや読み込みのための専用 API が用意されており、その関数をホスト PC で実行する C プログラムに記述することでデータの受け渡しを行う。

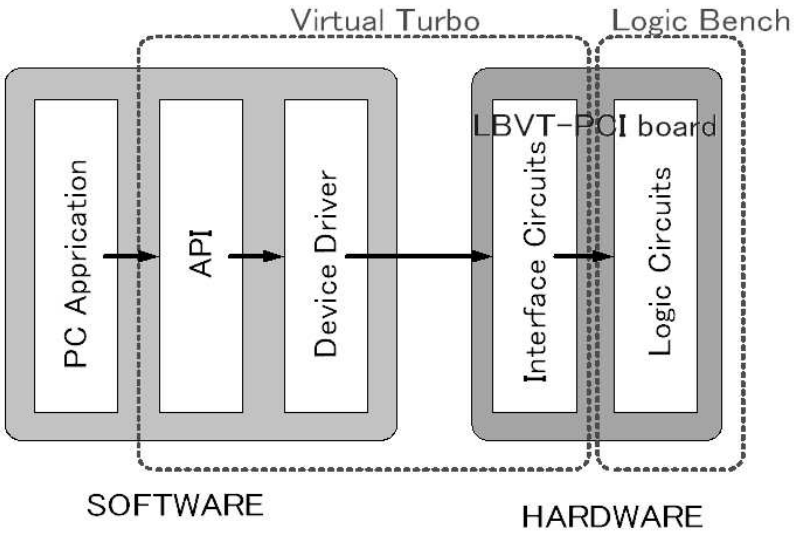

図 3.5: **ホスト** PC **と論理回路のインターフェイス概要**

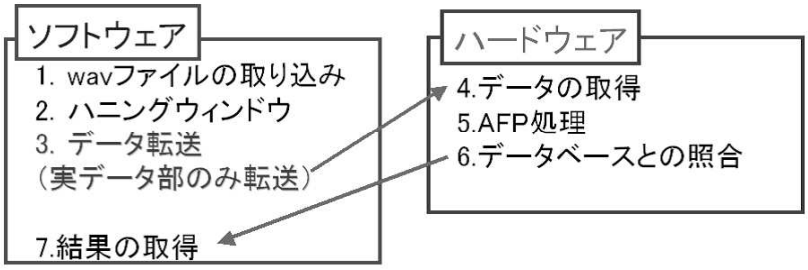

図 3.6: **ハードウェアとソフトウェアの処理分担**

3.6 インターフェイス部

入力されるオーディオデータは、最初にインターフェイス部を介することとなる。インターフェイス部の働きは、

- API **制御用** LSI **との同期を確立する**
- **リード/ライトフィンガープリント部に対して制御を行う（トライステート制御）**

の２点である。図 3.7 **に** Logic bench **と** API **制御用** LSI(LIBF-FPGA) **間のインターフェイスを示す。前項でも説明したように、構築した回路は、今後の拡張性を考慮した結果、３つの** FPGA **に論理分割する。今回は** interface **部は** FPGA/UL **のみに実装する。**LIBF-FPGA **と通信するデータは** 32bit **バスの「**DATA[31-0]**」であり、双方向バスである。**LIBF-FPGA **からのバス制御情報「**MODE[1-0]**」、アウトプットイネーブル** (Low-act) **「**OEB**」、ライトイネーブル（**Low-act**）「**WEB**」らは** LIBF-FPGA **から** Logic Bench **への片方向である。**

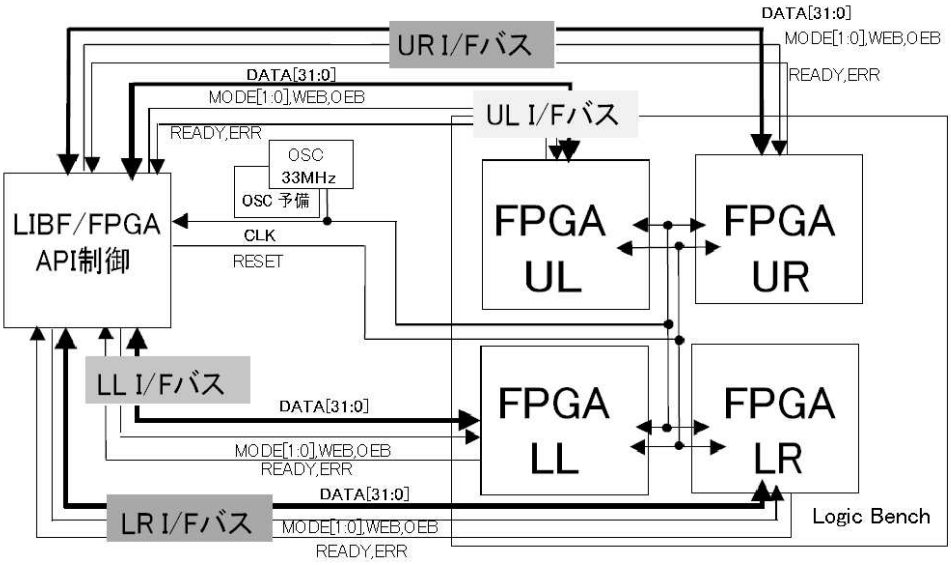

図 3.7: API **制御** LSI **と** FPGA **のインターフェイス**

「READY**」はリード可能となった時に発行される** LogicBench **から** LBIF-FPGA **へのレディ信号である。また、**4 **つの** FPGA **および** LIBF-FPGA **は、共通の** 33MHz **のクロックで動作するため、**LIBF-FPGA **から出力もしくは** Logic Bench **上の** FPGA **から出力されるデータは、出力先の** LSI **に入力するまでにどうしても遅延が生じてしまう。例えば、書き込み制御の場合、図** 3.8 **に記載されているセットアップ/ホールド時間をその遅延により守ることができなければ、正常動作ができない。したがって、論理合成ツールにて** 33MHz **においてタイミング制約を行った。**

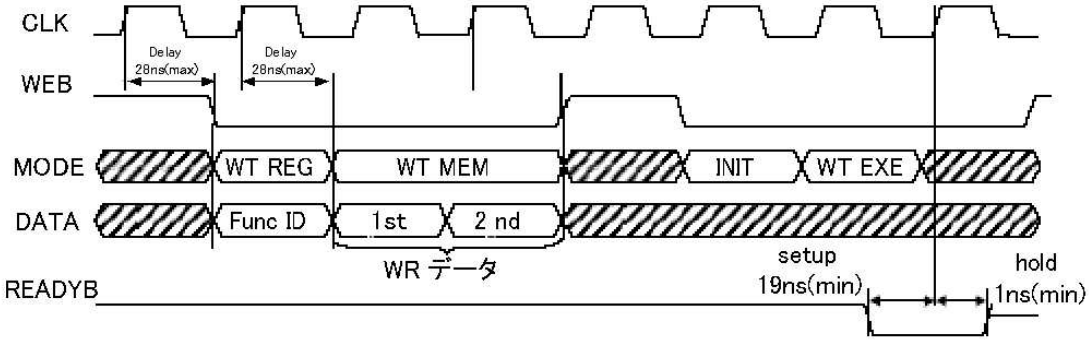

図 3.8: LogicBench **搭載** FPGA **の書き込みタイミング**

本研究では、高速性を重要課題としているため、図3.8 **に示されるタイミングチャート中の** MODE **で「**WT MEM**」時はメモリにデータをライトするのではなく、そのまま、データをフィンガープリント部に出力する仕様とした。そうすることで、メモリからのリード**

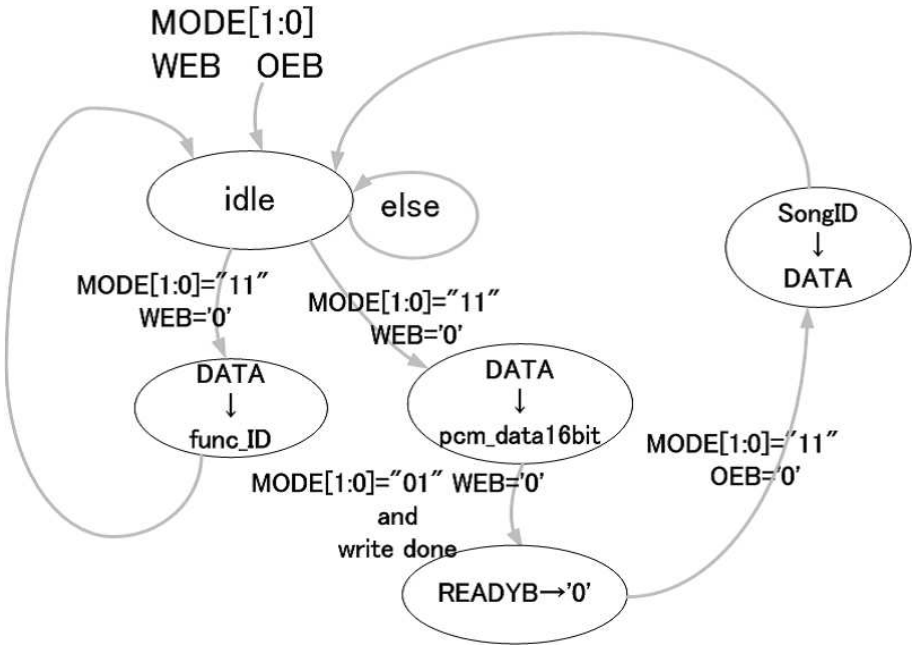

図 3.9: interface **部の状態遷移**

時間を削除することが可能である。したがって、MODE **が** WT MEM **時の時は双方向のデータを制御するためのライト用トライステートのイネーブルの論理を** ACT**、リード用のトライステートのイネーブル論理を** NON-ACT **にすることで、データをフィンガープリント部へ出力する使用とする。また、データのリードに関しては** OEB ＝'0' **かつ** READYB **の論理が負になった時点で、トライステートのイネーブルの論理を前者と逆にする。図中の「**FuncID**」はホスト** PC **で動作する** C **プログラム中に組まれる制御** API **がデータ送出回路を指定する場合の機能であり、今回は使用しない。**

3.7　フィンガープリント生成部の高速化

3.7.1　概要・構成

本項では図 2.3 の処理に相当するフィンガープリント生成部の構成について説明する。フィンガープリント生成部の最上位層を図??に示す。生成部は計算負荷の高い演算を含むので、設計手法次第で、その性能は大きく異なると思われる。設計方針として、パイプラインレジスタを用いた並列処理、ループ展開によるクロック数の削減、を適用する。高速フーリエ変換器、加算器、減算器、符号なし乗算器、コンパレータ等の演算器で構成する。データのビット幅は wav の PCM 方式の量子化 16bit に合わせ、全て固定小数点演算にて処理を行う。また、周波数成分の番号「xk_index」および FFT 完了フラグ「fft_done」は Reg 部にて初めて必要となるため数段のフリップフロップを介することで、データと周波数成分の番号を同期させる。ソフトウェアにてフレーミングされたオーディオデータは 64 サンプル毎に FFT により周波数分割 (Band Division) され、逐次的に出力された各周波数成分の実数部 (Re) と虚数部 (Im) からパワースペクトルの振幅の総和、つまりフレーム毎のエネルギーを図中の Reg 内にて計算する (Energy Compute)。次に見積もられた n フレーム目の成分 m であるエネルギーと成分 m+1 であるエネルギーとの差分を計算し、Divider 内のレジスタに格納する。その後、格納されている n フレーム目のエネルギー差分と n+1 フレーム目のエネルギー差分を周波数成分毎に比較することでサブフィンガープリントを抽出する。(Bit Derivation)

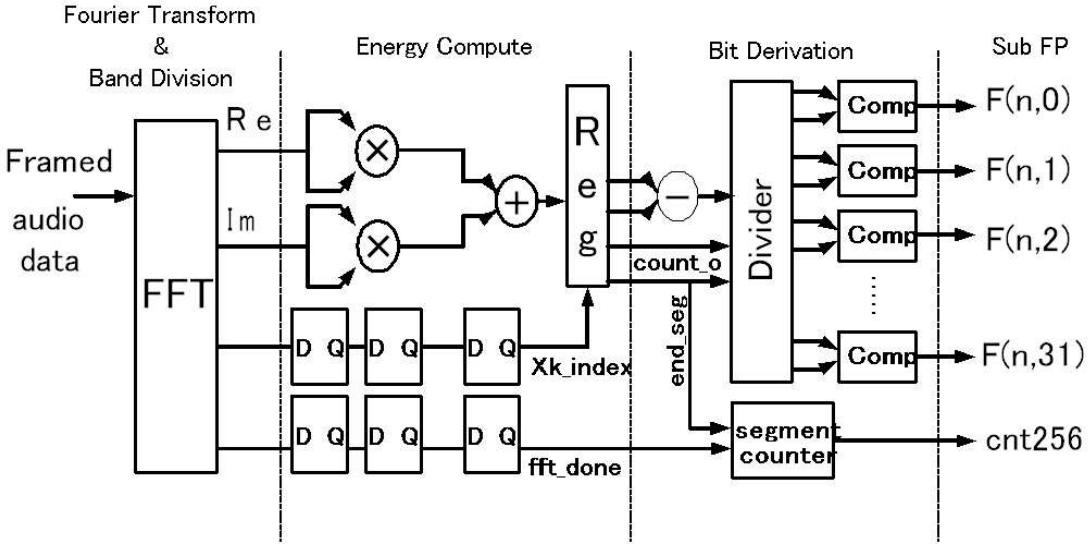

図 3.10: フィンガープリント生成回路の構成

以下に、このフィンガープリント生成回路を構成する６つの演算ステージを説明する。

3.7.2 省リソース高速フーリエ変換

FFT は離散フーリエ変換 (DFT: Discrete Fourier Transform) における計算のむだを省くため，三角関数の周期性と線形性を実にうまく利用した計算手法である。FFT であっても DFT であっても計算結果には差がない。DFT は，すべてのスペクトル F(k)[k = 0,1,2,N - 1] を求めるためには N^2 回の積和演算が必要となる。例えば，N = 1024 の場合は，約 10^6 回もの膨大な積和演算が必要となる。高速フーリエ変換アルゴリズムを用いた場合，その計算回数は $N \log_2 N$ 回となり，N = 1024 とするとその計算回数は約 104 回となり，大幅に計算量を削減することが可能である。ただし，FFT は普通，データ点数に制約があり．2 のべき乗 (2,4,8,16,32, 64, 128...) にデータ数をとるのが最も一般的である．このような制約がありながらも，実用上は計算速度の点でのメリットが大きいので，必然的に FFT が用いられている．離散フーリエ変換は入力デジタル信号 (N=64 サンプル)$\{f_n\}_{n=0}^{n=63}$ とし、出力の実数部、虚数部各々の周波数成分 $\{F_k\}_{k=0}^{k=63}$ とすると、

$$F_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n \ (e^{-j(2\pi/N)})^{nk} \quad (k = 0, 1 \bullet\bullet 63) \tag{3.2}$$

ただし、$e^{-j(2\pi/N)} = cos(2\pi/N) - jsin(2\pi/N)$

にて定義される。この高速フーリエ変換部は、設計の効率化のため、Xilinx 社が ISE 設計ツールで無償で提供する IP コアを活用した。この「Radix-2、Minimum Resources FFT」の IP コアは離散フーリエ変換の演算に基数 2 の分解（時間引き）を利用したものである。パイプラインレジスタで構成されているため、省リソースの回路量にて高速な演算を実現できるものである。

また、以下の表に、本研究で選択したパラメータにおける使用リソース量、変換時間、最大クロック周波数を示す。パラメータは Point Size=64、input data/phase factor bit width=16、Block RAM=3、embedded hardware multiplier=3 と設定した。

表 3.2: 高速フーリエ変換のリソース

Point Size	Input Data Width	PhaseFactorWidth	slices	BlockRAM	Mult18x18
64	16	16	709	3	3

表 3.3: 高速フーリエ変換の性能

—	Data Load + Trans. Timex	
max clock frequency	Clock Cycle	Times
169(MHz)	329	1.95(us)

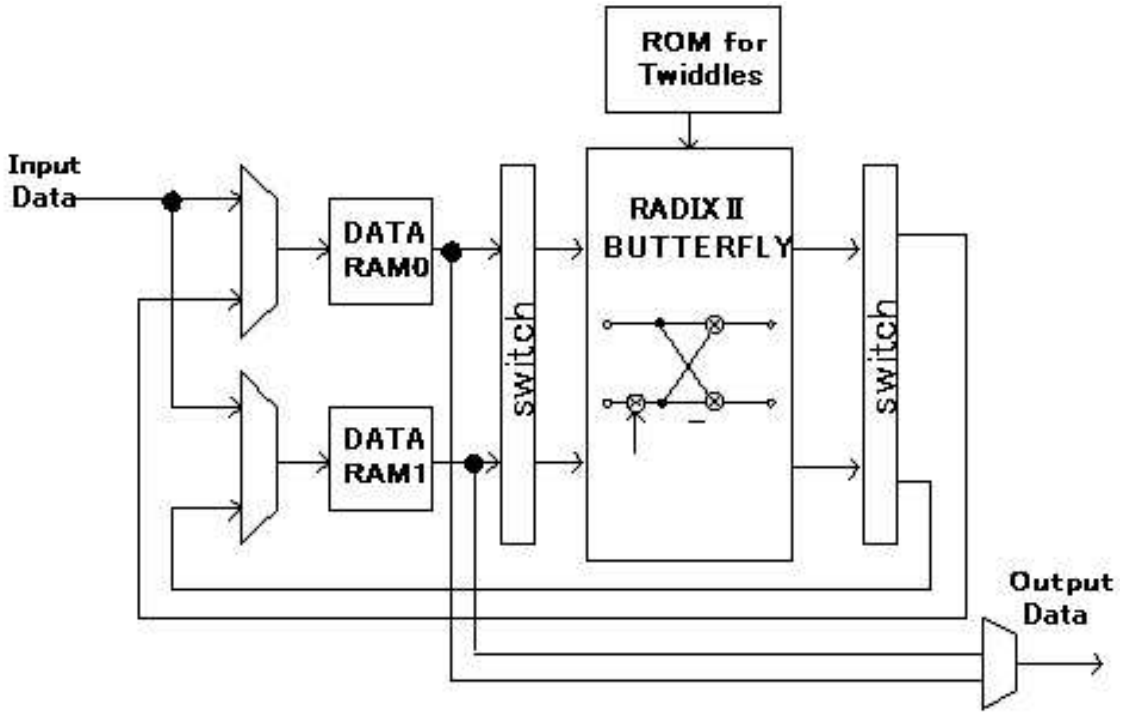

図 3.11: Radix- 、**最小回路量** FFT **の構成**

3.7.3 パワースペクトルの算出

フーリエ（振幅）スペクトルは概念的にはＸ軸にｃｏｓの係数値を実数部（リアルパート Re）に、Y 軸にｓｉｎの係数値を虚数部（イマジナリパート Im）としてベクトルで現すことが出来る。ここで求めるべきパワースペクトル P はフーリエスペクトルの大きさとして求めるので、$P = \sqrt{\mathrm{Re}^2 + \mathrm{Im}^2}$ となり、位相は $\tan\theta = \frac{Im}{Re}$ で表される。したがって、実数部と虚数部の２乗を算出するため、各々にエンデベッド 18x18 乗算器を割り当てることとする。

エンデベッド 18x18 乗算器を使用したパワースペクトル算出

本研究で使用する xc2v6000ff1152-4 デバイスには、あらかじめ 18 ビット X 18 ビットの 2 の補数エンベデッド乗算器が１４４個も組み込まれている。このエンベデッド乗算器を使用すると、18 ビット X 18 ビット (符号付き) 乗算の積がすばやく効率的に得られ、しかも、乗算器ブロックはブロック SelectRAM メモリと配線リソースを共有しているため、多くのアプリケーションで効率も上がる。カスケード接続した乗算器も、ローカルの Virtex-II スライスにある追加ロジック リソースでインプリメントできるといった利点がある。そして、一番の利点は CLB を消費することなく乗算器が配置でき、フィンガープリントシステムの構築を少回路量で実現するには好都合である。本研究では、CORE

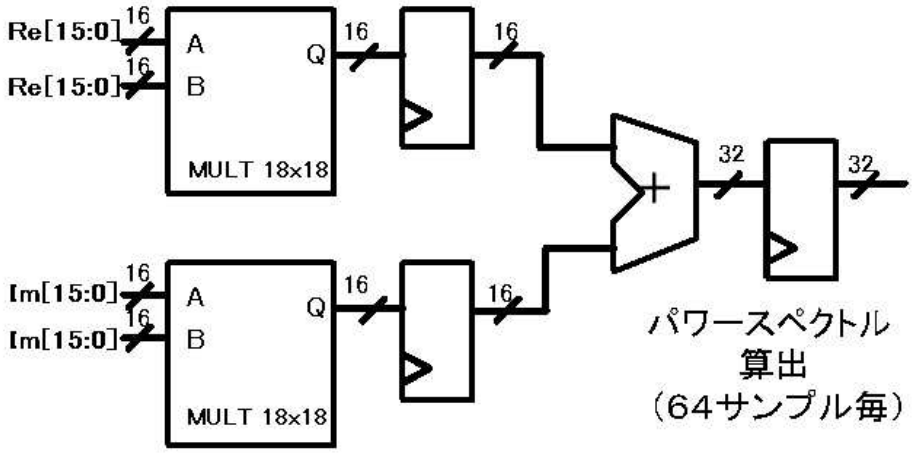

図 3.12: **パワースペクトル演算部**

Generator **を使用し、**Virtex-II **デバイス用乗算器コアの** 18 **ビット** X 18 **ビット** 2 **の補数エンベデッド乗算器** (**バージョン** 2.0 **以降**) **を使用して、乗算器を生成した。図** 3.12 **にエンデベッド乗算器を使用したスペクトル算出部の構成を示す。これは単純に** 64 **サンプル毎の周波数成分の実数部と虚数部の２乗をそれぞれ計算を行い、各々の２乗の和を計算する。**

パワースペクトル総和演算

レジスタ内にて、１フレーム分の各周波数成分のパワースペクトルの総和を算出する。高速フーリエ変換は 64 **サンプル入力毎の周波数成分を出力するため、**1 **フレーム** (16384 **サンプル**) **当たり、**256 **回繰り返し加算を行うことで、各周波数成分の総エネルギーを算出し、その隣合う差を出力する。図** 3.13 **に回路構成を示す。**

64 **サンプルの高速フーリエ変換が終了時に１クロックサイクル分だけ**”H” **となる** fft_done **をトリガにカウントアップする** 256 **周期カウンタ値**”255” **（１フレーム分に相当）に達するまで、高速フーリエ変換から出力される、データに同期した３３個の周波数帯番号「**xk_index[5:0]**」にを基に各周波数帯毎のパワースペクトルの総和を計算する。図中央の「**divider**」は周波数帯「０」のデータを** REG[0]**、周波数帯「３２」のデータを** REG[32] **というように、周波数帯番号に対応したレジスタに入力データを分配する役目をする。加算終了** (256counter=”255” **時**) **と同時に図中の** 64 **周期カウンタはカウントアップを開始し、そのカウンタ値に同期させ、**REG0 **から** REG32 **までの３３個の減算器の入力をシフトさせていく。**

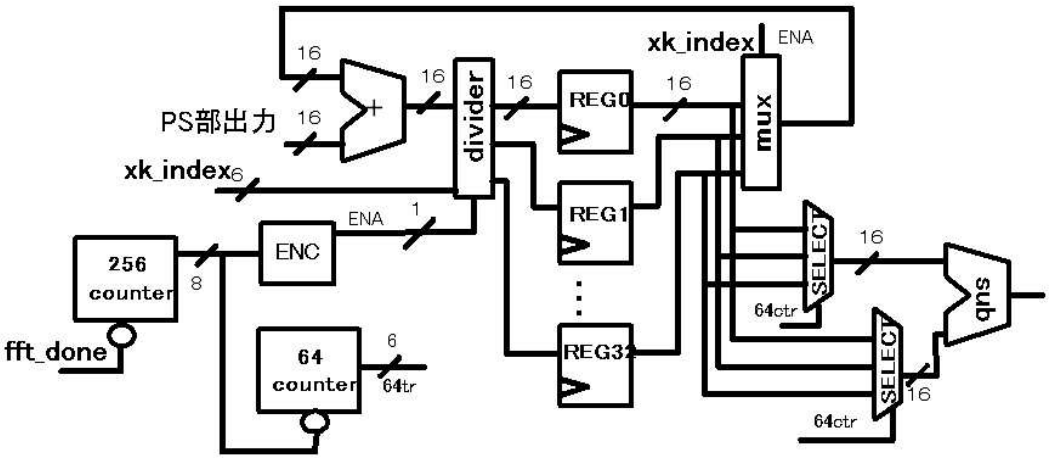

図 3.13: **パワースペクトル総和部**

3.7.4 比較器のループ展開によるクロック数の削減

まず、ここでは 1 **フレーム毎の各周波数成分のパワースペクトル総和の差を、**32 **個に並列展開したコンパレータに分配する。ソフトウェアでこの処理を行う場合、１フレーム当たりコンパレータを** 32 **回ループさせて、ようやく** 32bit **のサブフィンガープリントを得ることができるが、回路量の少ないコンパレータを** 32 **個ループ展開させると、**1 **フレーム当たり１回の比較で良いということになる。表** 3.4 **にループ展開を採用した場合と、そうでない場合のフィンガープリント完了するクロック数の比較を示す。**

表 3.4: **ループ展開によるクロック数削減**

	オーディオ１フレーム	オーディオ２５６フレーム	性能比
ループ展開無し時	3200cycle	819200cycle	1.0
ループ展開採用時	100cycle	25600cycle	32.0

今回使用するコンパレータが必要とするクロックサイクル数を x **サイクルとすると、ループ展開を行わない場合は１フレームの比較に** $32 \times x$ **クロックサイクル必要となってくる。フィンガープリント完了となる** 256 **フレームで考えると、合計で** $8192 \times x$ **クロックサイクル必要で** 33MHz(**周期** 30ns) **での動作周波数で動作すると考えると、**$8192 \times x(cyc) \times 30(ns)$ **の時間を比較処理だけに費やすことになる。一方、ループ展開を採用した場合は、**$256 \times x(cyc) \times 30(ns)$ **となる。したがって、**Bit Derivation **部に関して、ループ展開をすることで** 1/32 **の処理時間でサブフィンガープリントが算出できる。**

式 (2.8)**、**(2.9) **によるとフィンガープリントは、前後のフレームのエネルギー（パワースペクトル総和）**E(n,m) **の「差」を** 0 **と大小比較することによって決定されるが、式** (2.8) **および** (2.9) **の定義式を以下のように辺を移項することで**

$$F_{(n,m)} = \begin{cases} 1 & (E_{(n,m)} - E_{(n,m+1)}) > (E_{(n-1,m)} - E_{(n-1,m+1)}) \\ 0 & (E_{(n,m)} - E_{(n,m+1)}) \leq (E_{(n-1,m)} - E_{(n-1,m+1)}) \end{cases} \tag{3.3}$$

減算器を削減することができ、結果的に処理時間と回路量の削減となる。

図??にループ展開した比較器とその周辺制御図を示す。図中の「Divider」と呼ばれる比較器への分配器には前に到達したフレームのエネルギー差 (式 (3.2) の右辺) と後から到達したフレームのエネルギー差 (式 (3.2) の左辺) の二つ値を記憶しておくレジスタが必要である。そこで、３２個に並列配置された前到達用レジスタと後到達用レジスタを設けることで、比較器への同時出力を可能とする。

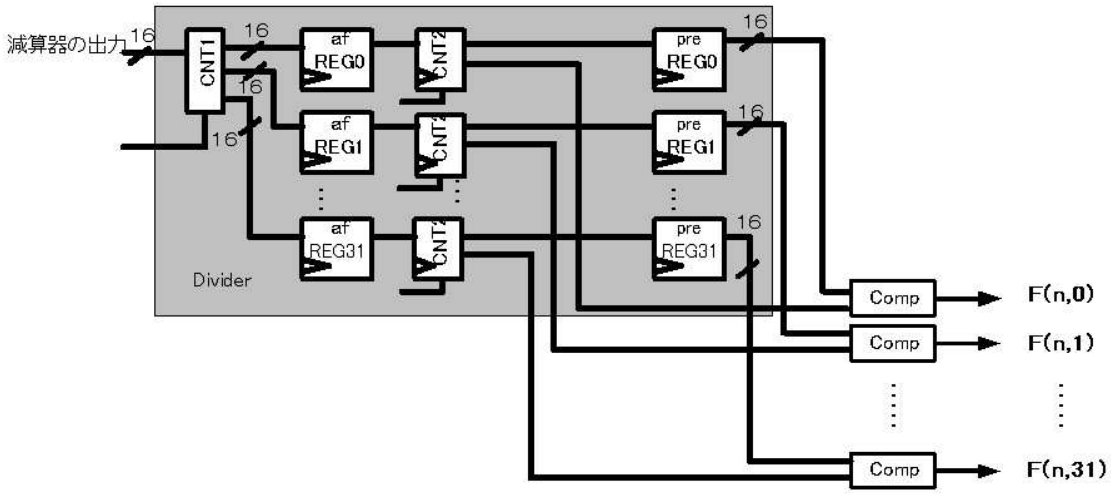

図 3.14: 比較器のループ展開

図??中左の制御回路「CNT1」はパワースペクトル総和演算部の 64ctr を制御要素とする。この 64ctr の情報 (ctr64=”0〜31”) を基にエネルギー差 ED(n,m) を 32 個のレジスタ af_REG[0]〜af_REG[31] へ出力する。ctr64=”62”時に３２個の af_REG と pre_REG は ED 値を出力を行い、並列に配置されたコンパレータはフレーム毎にサブフィンガープリントを３２ bit 同時に算出する。

また、フレームの終わりを示すフラグ（前項の 256counter=”255”時）のアクト状態で af_REG に格納されていた３２個の ED 値は一斉に pre_REG へシフトされる

3.7.5 パイプライン化

スループットを向上させるために前項で用いたようなループ展開を多用し、処理クロック数を減らそうとすると、かえって動作クロック周波数が落ちてしまう可能性がある。そこで回路規模がある程度大きくなるのを容認した上でクロック周波数低下を防ぎ、かつスループットを上げるのが処理のパイプラインレジスタ化である。ただし、組み合わせ回路

間に単純にパイプラインレジスタを挿入しただけでは、クリティカルパスとなるステージの伝播時間に τ_{pipe}（パイプライン導入時の１ステージ分の処理時間）が制限されてしまう。

各ステージ内に無駄な時間を生じさせないような、パイプライン処理を効率的に行うためには、各ステージ間の機能粒度をおおむね均衡にする必要がある。したがって、各ステージの機能粒度を均衡にするため、所要クロック数を考慮したパイプラインステージの配置を行うことで、クリティカルパスの遅延を抑え、スループットの向上を実現した。図 3.15 に本研究でのパイプランレジスタを用いた処理の流れを示す。機能粒度は約 64 クロックサイクルを 1 ステージと考えた。

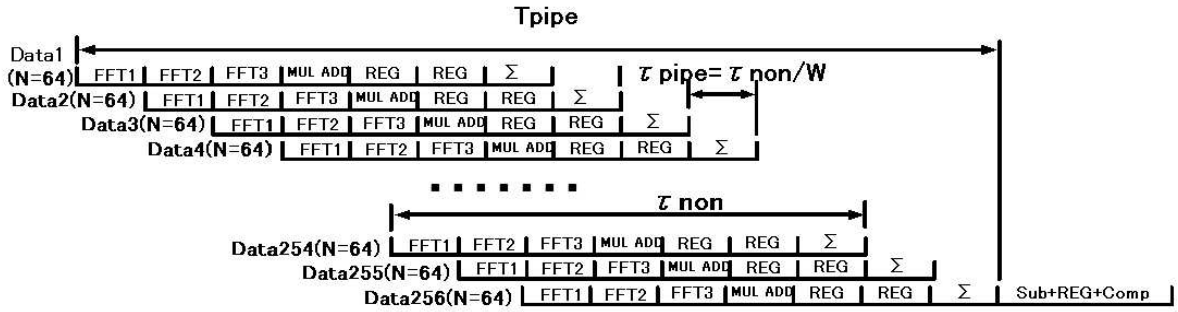

図 3.15: パイプラインレジスタを使用した場合のデータの流れ (1 フレーム分)

非パイプライン処理において１つの入力データの処理時間を τ_{non} とした場合、フィンガープリント 1 フレームを処理するデータの総要素数（64 サンプル単位）を 256 とすると非パイプライン処理における総データ処理時間は

$$T_{nonpipe} = N\tau_{non} \tag{3.4}$$

となる。

そこで、機能粒度を考慮したパイプライン処理を行う場合のステージ数を L とし、各ステージが均等に

$$\tau_{pipe} = \tau_{non}/L \tag{3.5}$$

の時間に分割できる場合、最初のオーディオデータがパイプライン処理され Σ に至るまで $L \times \tau_{pipe}$ の時間がかかるが、残りの 255 個のデータは τ_{pipe} 毎に Σ まで到達できる。したがって、パイプライン処理時における全データの処理時間 T_{pipe} は

$$T_{pipe} = (L + N - 1)\tau_{pipe} = (L + N - 1)\tau_{non}/L \tag{3.6}$$

となり、Radix2 の FFT を実装した場合、L=10、N=256 で、L に対して N が十分大きいので、

$$T_{pipe} \approx 256\tau_{pipe} = 256\tau_{non}/10 \tag{3.7}$$

となることから、非パイプライン処理時と比較してパイプライン処理を採用した場合は、ステージ数、つまり 10 倍の性能向上を得ることが出来る。

3.8 楽曲検索・識別部の提案

3.8.1 Bit Error Rate(BER)

フィンガープリントシステムにおいて、任意の２つの全く異なる楽曲では、全く異なるフィンガープリントとなるが、同一の曲でも片方が MP3 への圧縮などの加工により、質が劣化したものである場合、それらのオーディオフィンガープリントは原曲のものと完全一致するとは限らず、多少は誤差が出てくる。そのような加工されたオーディオファイルでも識別できることがシステムとしてロバスト性が向上するという点望ましいと考えられ、ある程度の誤差も許容しなければならない。そこで、識別を決定するパラメータとし

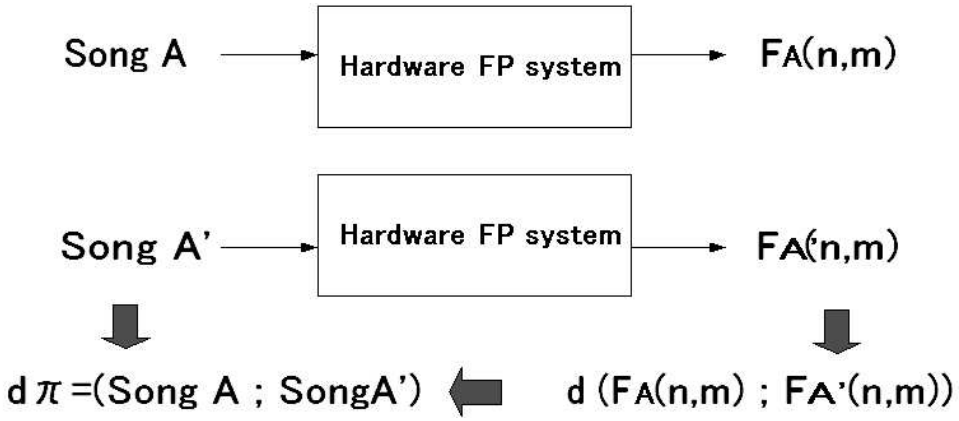

図 3.16: 知覚的な音質の差とフィンガープリントの差の関連性

て、BER(Bit Error Rate) を本研究では採用する。ここでは、原曲 A から生成されたフィンガープリントを $F_A(n,m)$、加工処理された曲 A' から生成されたものを $F'_A(n,m)$ とする。BER は比較する二つのフィンガープリント $F_A(n,m)$ と $F'_A(n,m)$ のハミング距離

$$F_{diff}(n,m) = F_A(n,m) \oplus F_{A'}(n,m) \tag{3.8}$$

から算出される。したがって、ハミング距離に基づいた BER は

$$BER = \frac{1}{32N} \sum_{n=0}^{N-1} \sum_{m=0}^{31} F_{diff}(n,m) \quad (N = 1, 2 \bullet\bullet 256) \tag{3.9}$$

となる。この二つのフィンガープリントのハミング距離は、曲 A と曲 A' の知覚・感性的な差 $d_\pi(SongA, SongA')$ と関係するとされている。つまり、BER の差は原曲と加工された曲との感性の違いを数値化したものであり、２者間で BER が低い場合、類似性により同一楽曲であるという確率が非常に高くなる。

3.8.2 識別の確率 (偽陽性と偽陰性)

2 個のオーディオデータ 256 フレーム分におけるフィンガープリントブロック間のハミング距離（すなわちビットエラー）の総和を算出し、閾値 T を下回る場合、2 個の音声信号は非常に似ているものだと判断される。この閾値 T は偽陽性率 P_f を決定する。つまり、音声信号の信頼性が低いと判断されるものは、T は小さい値であり、確率 P_f も小さくなる。

また、他方で閾値 T が小さな値の場合は、偽陰確率 P_n に対して悪く作用する。P_n は 2 つの信号が‘ 等しい’ が、特定されないという確率である。ビットエラーが起こる確率 p、総ビット数 n（ $8192 = 32 \times 256$) とするとエラー平均値

$$\mu = np \tag{3.10}$$

と標準偏差

$$\sigma = \sqrt{(np(1-p))} \tag{3.11}$$

での二項分布 (n,p) によって正規分布 (誤差関数) へ近似することができる。フィンガープリントブロック F1 が与えられる場合、閾値 $T = BER$ n エラー以下である任意に選択されたフィンガープリントブロック $F_{A'}$ が F_A であるという確率 $P_f(BER)$ は

$$P_f(BER) = \frac{1}{\sqrt{2\pi}} \int_{1-2BER}^{\infty} e^{-x^2/2} dx \tag{3.12}$$

によって与えられる。

図 3.17 に相補誤差関数から算出される識別確率と BER の関係を示す。BER が 1 に近い場合は問い合わせフィンガープリントブロックが今ハミング距離を計算しているフィンガープリントブロックに対して「近似している」という確率は低くなる。逆に BER が少なくなるにつれて、指数的にその確率が増加する。したがって、本研究では BER を利用した楽曲判定をハードウェア上で行う。

3.8.3 エンデベッド BRAM(Select-RAM) を有効利用したフィンガープリント格納

FPGA 中にメモリを作成しようとする場合、普通に配列で作ると、ロジック回路に使う CLB(Configurable Logic Block) という領域を使用する。このメモリは FPGA 分散 RAM であり、FPGA 分散メモリの容量などは特に規定は無い (最大 CLB 数) が、貴重な CLB を膨大に消費するため、フィンガープリントによる大量の楽曲データベースを構築しようとすると FPGA の最大回路量を上回ってしまう。

そこで、ザイリンクス社FPGA(アルテラなど他社FPGA も同様) の今回使用する Virtex-II の XC2V6000 には、分散 SelectRAM メモリに加え、18Kb のブロック SelectRAM メモリが 8Mbit 標準搭載されている。ブロック SelectRAM メモリは真のデュアル ポート

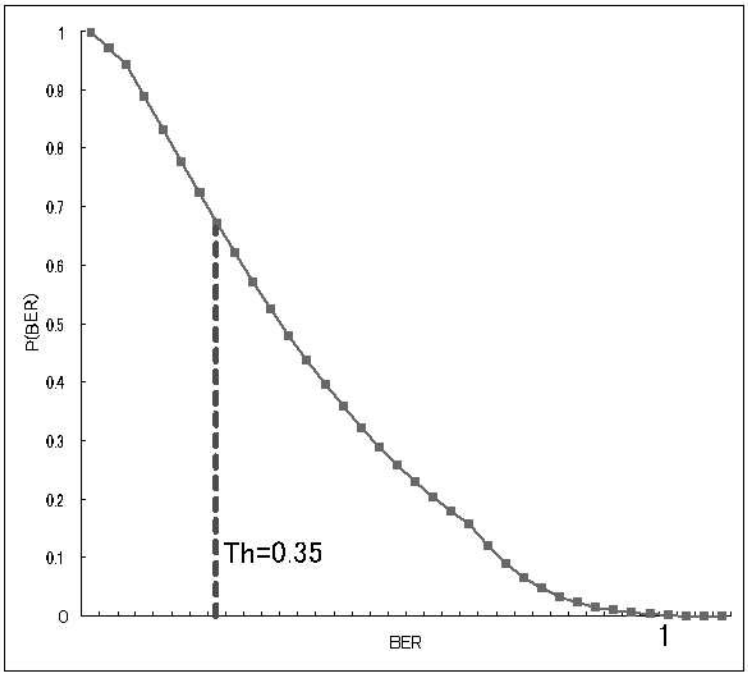

図 3.17: **識別の確率と** BER **の関係**

表 3.5: **フィンガープリント格納** RAM **のポートの深さと幅の関係**

幅	深さ	アドレスバス	データバス
32	512	ADDR[8:0]	DATA[31:0]

RAM **であり、デバイス上で高速で離散型の大容量ブロック メモリとして使用可能である。メモリは縦に並べられており、ブロック** SelectRAM **メモリの総容量は** Virtex-II **デバイスのサイズによって異なります。**18Kb **のブロックはカスケード接続可能で、ビット数とワード数の多いメモリをインプリメントでき、特殊な配線リソースによりタイミング遅延を最小限に抑えることができる。本研究で活用するフィンガープリント格納用および** Look Up Table **用の** BRAM **のポートの深さと幅の関係を表** 3.5 **に示す。**

一楽曲あたりのフィンガープリントは 32bit × 256 **フレームの構成となっているが、**SelectRAM **の規定により、ポートのデータ幅とアドレスバス幅の比率が決まっており、**32bit **のデータ幅に対しては深さは** 512 **となる。したがって、フィンガープリントを格納する領域は** RAM **のアドレス** [0x000〜0x0FF] **とし、アドレス** [0x100〜0x1FF] **の範囲は遊休領域となってしまうため、**1 **楽曲のフィンガープリントにあたり、やむを得ず、**2KB **消費することになる。各** BRAM **は** 32 **幅×** 512 **深さのデュアルポートで構成され、リード/ライトは** virtual turbo **で使用するグローバルクロック** 33MHz **に同期する。(図** 3.18 **参照)**

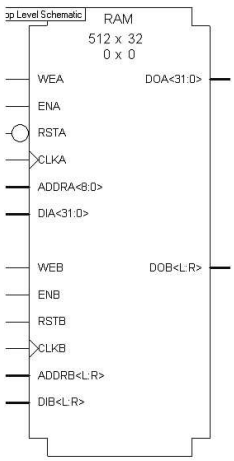

図 3.18: **フィンガープリント格納** RAM

3.8.4 Bit Error Rate(BER) に基づいた楽曲識別法の提案

本項では、前項で述べた BER **を利用した楽曲識別を** FPGA **内で行う手法とその構成について説明する。本研究で提案する並列性を生かした楽曲識別アルゴリズムを図** 3.19 **に示す。**

この楽曲識別手法の特徴として、図 3.1 **に示すように楽曲識別回路を並列配置することで、「問い合わせフィンガープリント (システムに入力された音楽データから生成されたフィンガープリント)」と「データベースフィンガープリント (データベースにあらかじめ格納されているフィンガープリント)」を比較対象とし、ハミング距離、**BER **演算を並列で行う。本提案手法は、**BER **演算を用いているため、単純に他の探索アルゴリズムと計算量の比較は意味が無く、高速性を特長とするのではなく、つまり、一定のロバスト性を保有した高速な楽曲検索および識別を可能とする。**

並列性を生かした探索手法はモジュールレベル（大規模な機能回路）の並列展開しなければならないが、それは、ファンアウトを拡大させクリティカルパスが大きくなる原因となる。したがって、論理合成時に出力されるレポートファイルの結果を参考にしながら、識別回路の並列数の調整を行う。

3.8.5 楽曲検索・識別部の回路構成

図 3.19 **の検索アルゴリズムを実行する回路構成を図??に示す。**Look Up Table **の入力の接続先はフィンガープリント生成部の出力であり、楽曲の検索終了にて格納したフィンガープリントはクリアする。まず、**Look Up Table **に格納された「問い合わせフィンガープリント」を並列に展開された** XOR **へ入力することで、「**Select-RAM **に格納されている複数のフィンガープリント」に対して同時にハミング距離の算出を行い、その値から** BER

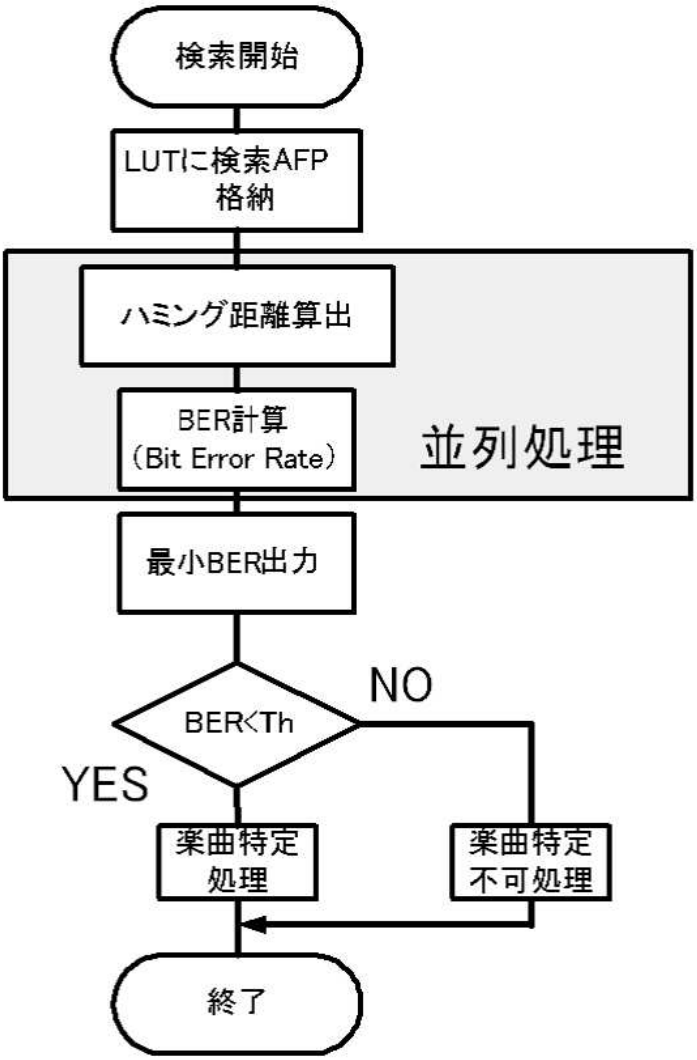

図 3.19: **楽曲識別アルゴリズム**

を計算する。つまり、複数の BER **値を並列処理により高速に計算する。次に、複数の比較器にて構成される最小** BER **選定回路を用いて、** BER **計算部から出力された複数の** BER **候補の中から最小の** BER **を選定する。選定された最小** BER **はさらに比較器に入力され、** BER **が閾値より低い場合は、楽曲番号をレジストしている** FF **に対してイネーブルを与えることで、楽曲特定したというメタデータ (楽曲番号）をホスト** PC **へ返す。**

3.8.6 LUT への生成フィンガープリントの格納

フィンガープリント生成部にて生成されたフィンガープリントを LUT(Look Up Table) **に格納する。** LUT **は** RAM(**データバス幅** 32 **×アドレスバス幅** 8) **で構成し、** N **フレーム目**

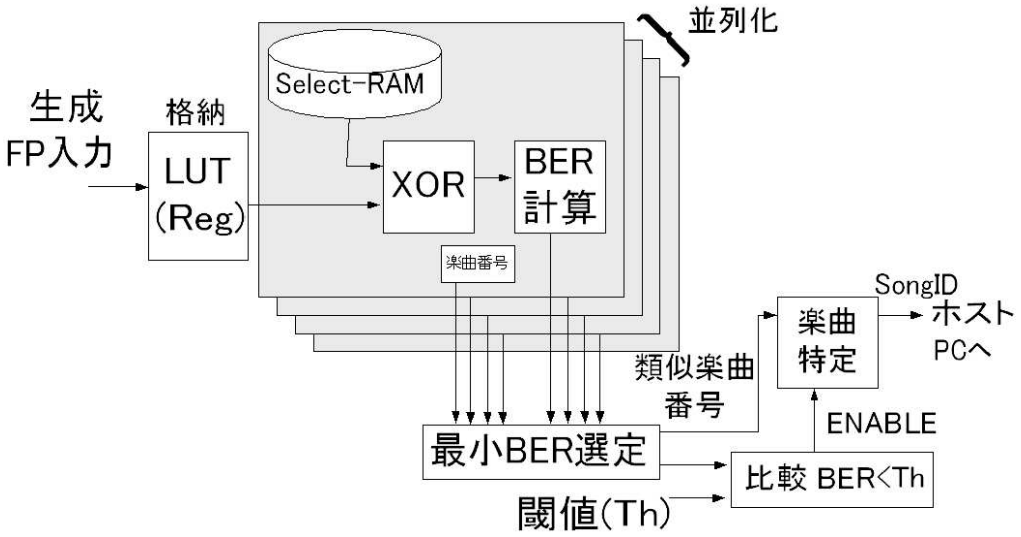

図 3.20: 楽曲識別・検出回路のブロック図

のフィンガープリントの格納アドレスは N-1 という規則性を持たせる。例えば 6 フレーム目の 32bit サブフィンガープリントはアドレス:0x05 に格納する。RAM は FPGA に標準搭載されているものを使用する。本研究で実装を行う Xilinx 社の FPGA XC2V6000FG1156 は約 9Mbit の BRAM を含んでいるので、1 つの FPGA には最大 100 個 (8192bit=1KB/1 フィンガープリント) のフィンガープリントデータベースが蓄積ができる。

図 3.21 に LUT のタイミングチャートを示す。LUT はフィンガープリント生成回路部から出力されてくるサブフィンガープリントをフレーム番号と同期しながらメモリ領域に格納する。(書き込みイネーブル「WR_ENE」はアクト状態) 256 フレーム全てメモリへの書き込みが終了と同時に「OUTPUT_ENE」がアクト状態になり、２５６周期の「SEARCH_CNT」がスタートする。また、問い合わせフィンガープリント「SEARCH_FIN」はサーチカウンタと同期して出力される。並列に配置展開されている複数の楽曲識別部に含まれる BRAM のアドレスピン (信号名：SEARCH_CNT) およびデータ入力ピン (信号名：SEARCH_FIN) へ入力される。そうすることで、複数の楽曲フィンガープリントとのハニング距離の算出が可能となる。

一般的に、電子回路の世界では１つの出力ピンがどれだけの入力先をドライブできるかという「ゲート出力のドライブ能力」を知る必要がある。(つまりファンアウト数の最大値) 標準 TTL などのインターフェイスでのファンアウト数は約１０個が限界であるが、FPGA 内である論理回路でのファンアウト数は１０００個を超えるケースは多々あり、論理合成ツールにてファンアウトの低減がなされるので、楽曲識別部へのファンアウト数をマニュアルで検討することは必要ないと判断した。

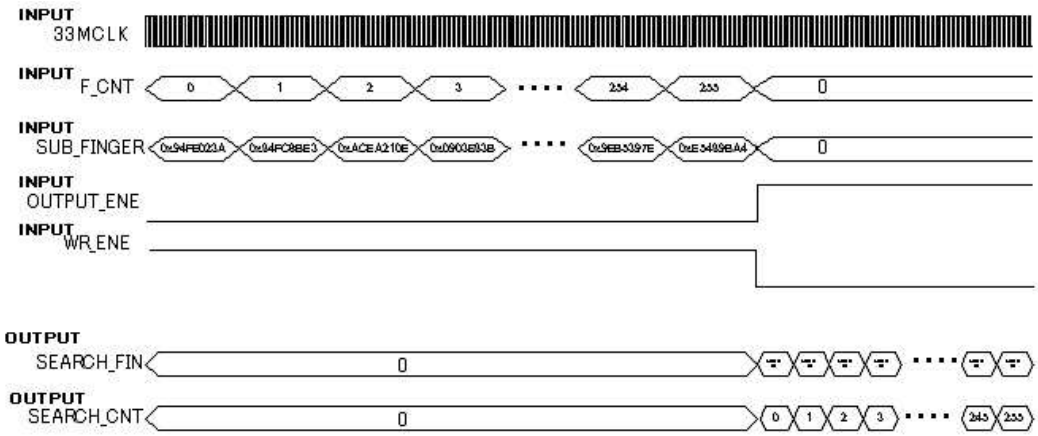

図 3.21: LUT **のタイミングチャート**

3.8.7 BER の並列計算

並列処理にて同時に、複数の「データベースフィンガープリント」と「問い合わせフィンガープリント」間の BER**(式** (3.3)**) を算出する。２つのフィンガープリント間のハミング距離の算出は、アドレス:**0x00 **から開始し、**0xFF **で終了する。つまり** 1 **フレーム目から始まり２５６フレーム目まで昇順にハミング距離を算出する。フレーム番号毎にハミング距離の数を算出し、ハミング距離の総和に加算する。**

また、式 (3.3) **は除算を使用している。除算を使用することにより、小数点以下の値が発生することで、浮動小数点演算を行う必要が出てくる。これ以降の** BER **演算のステップを全て浮動小数点演算で実行することは、回路量を膨大に消費することが予想される。（ましてや、並列展開するならばなおさらである。）そこで、式** (3.3) **の定義式を以下のように変形することで、回路量の削減を行う。**

$$32 \times N \times BER = \sum_{n=0}^{N-1} \sum_{m=0}^{31} F_{diff}(n,m) \quad (N = 1, 2 \bullet\bullet 256) \tag{3.13}$$

つまり、この BER **計算ブロックでは、ハニング距離の総和のみを計算することとなる。また、変形する前の閾値を「**th**」と定義すると、閾値「**th**」も** 32N **倍する必要があり、**BER **変形後の** BER **に対応した閾値「**th'**」は**

$$Th' = 32 \times N \times Th \tag{3.14}$$

3.8.8 クリティカルパスを抑える最小 BER の選定回路

複数の楽曲識別回路から出力された BER **の比較を行い、最小の** BER **値と、そのフィンガープリントデータベース番号を出力する。**

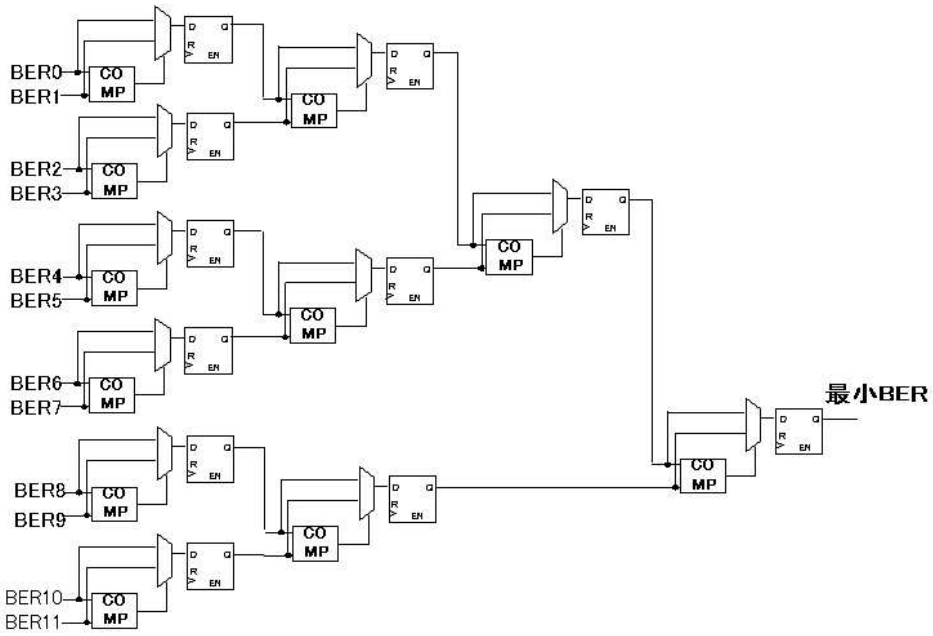

図 3.22: クリティカルパスを抑えた最小 BER 選定回路

ここではクリティカルパスの遅延を抑えるため、シーケンシャルに比較を実行していくのではなく、均衡の取れた２分木構造で比較を行う。例として図 3.22 に 12 個の BER から最小 BER を選定する回路を示す。各ノードは比較器とフリップフロップ、セレクタにて構成され、比較器の２入力には BER を入力し、低い方の BER 値と楽曲番号を選択し、次ステージへ選択されるトーナメント方式である。各ノードは全て同等の機能粒度であるため、各ノードは同時処理で比較を行い、処理を進行する。

3.8.9 最小 BER と閾値の比較

手動にて設定した閾値と、出力された最小の BER の比較を行う。採用した Haitsuma らのフィンガープリントアルゴリズム [6] では、BER が閾値が 0.35 より下にある場合は、問い合わせ楽曲がデータベース中にある可能性が高いと実験にて証明した。同じ特徴生成アルゴリズムをハードウェア化した本研究では閾値を th=0.35 で設定し、BER が閾値より下にある場合（図 3.17 の点線より右）は楽曲識別フラグ、つまりデータベース番号を出力する。最小 BER が閾値より上にある場合は、特定不可フラグ ALL"1"を出力する。

ただし、前項での回路量削減のための式変形から設定する閾値 th' は

$$Th' = 32 \times N \times th = 32 \times 256 \times 0.35 \simeq 2867 \tag{3.15}$$

となる。

3.9 まとめ

本研究で構築したハードウェアによるフィンガープリントシステムは、インターフェイス部、フィンガープリント部、楽曲識別部の３つのブロックから構成され、それぞれのブロックに適宜、性能向上を目指すための設計を施した。フィンガープリント部に関しては機能粒度（所要クロック数）を考慮したパイプラインレジスタ処理によるスループットの向上、また、比較器のループアンローリングによるクロック数の削減の２つの手法に基づいた設計を行い、一番計算負荷が高いと考えられる「Transrate」のステップを最小の回路量での高速フーリエ変換 IP コアの配置した。また、楽曲識別部に関しては問い合わせフィンガープリントとデータベース中のフィンガープリント間の BER 計算を並列に処理することで、同時に複数のデータベースの中から楽曲の識別・検出を可能とした。算出した BER の最小選定回路には２分木構造とすることで、クリティカルパスの遅延の増加を抑える工夫もした。また、識別判定にスレッショルドを用いることで、たとえ問い合わせる楽曲が劣化している場合でも、識別・検出可能となり、これでロバスト性の高いシステムとなるはずである。次項より、本項で構築したハードウェアの評価を行い、性能が要求を満たしているかどうか確かめる。

第4章　評価

4.1　概要

前項にて構築したハードウェアフィンガープリントシステムの性能を確かめるために、以下の評価を実施する。

- フィンガープリント処理時間測定
- 楽曲識別・検索時間測定
- 回路量・クリティカルパス測定
- ロバスト性、信頼性評価

フィンガープリント処理時間はソフトウェアによる処理時間とハードウェアによる処理時間を比較し、ハードウェアで処理する優位性を示した。回路量・クリティカルパスの測定については、楽曲識別部の並列数の増加に伴う最大周波数の推移についても測定し、楽曲識別部の最大並列数について議論を行った。ロバスト性評価は音質を変化させた楽曲に対して、システムはどれだけの耐性があるかを測定し、数値に示す。信頼性評価については音楽ファイル誤検出の回数を測定し、誤検出率を算出する。また、得られた結果から構成の再検討を行い、改善手法によりさらなる性能向上を目指した。

4.2　処理時間の測定およびソフトウェア処理との比較

表 4.1: 処理時間の比較

	FP 生成時間	楽曲識別時間	性能比
ソフトウェア	3353ms(19.95Mbps)	ー	1.0
ハードウェア	650.24ms(102.78Mbps)	0.543ms	5.152

まず、ハードウェアでのフィンガープリント処理時間の測定を行った。測定系を図 4.1 に示す。ホスト PC の C 言語にて、wave のヘッダファイルを取り除く処理を行い、wave の左チャンネルのデータ部を書き込み API を呼び出すことにより、PCI バス経由で FPGA

ボードに転送する。FPGA 上の PCI-FPGA ブリッジは VirTual Turbo の規定タイミングにリタイミングを行い、FPGA にデータを書き込む。このフィンガープリントシステムを

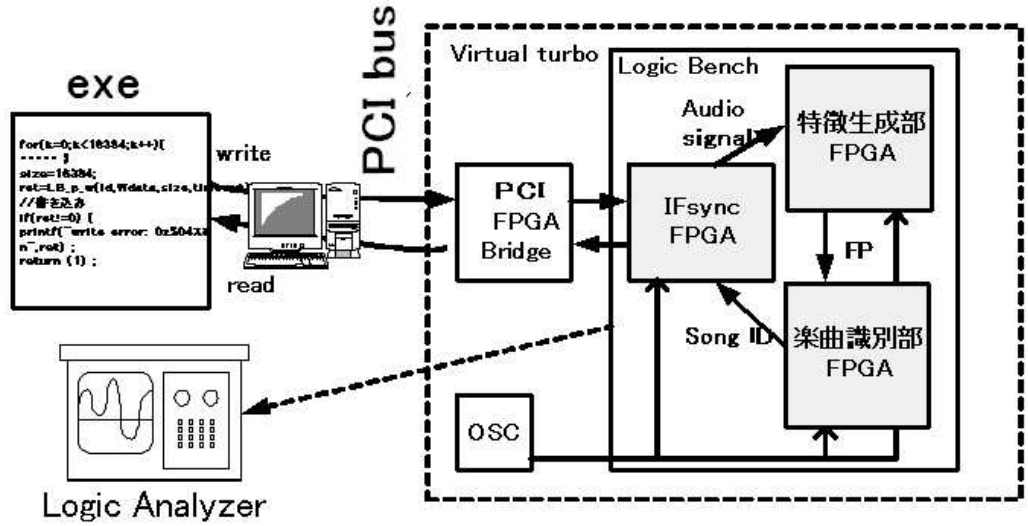

図 4.1: 測定系

今後、大規模なものへ拡張するため、システムゲートの余裕を考え PCI-FPGA との同期を行う IF sync 部、特徴生成部、楽曲識別部をそれぞれ３つの FPGA に論理分割を行った。

事前準備として、それぞれの楽曲識別部には異なる 30 曲分のフィンガープリントを蓄えておき、ホスト PC から音楽データ１曲を入力し、楽曲識別を行う。時間測定は書き込み API 実行から読み取り API 終了までの clock 関数を用いた時間測定、およびロジックアナライザのモニタにて、フィンガープリント所要実時間測定を実施した。システムクロック 33MHz で動作させた場合、wav ファイル 256 フレーム (約 8MB) のフィンガープリントにかかる時間は約 650ms（ビットレートに換算すると 102Mbps）、楽曲識別は約 0.54ms という結果が得られた。ソフトウェアのフィンガープリント生成速度と比較すると約 5 倍の性能を達成した。

ロジックアナライザでの解析によると、所要時間の大部分は高速フーリエ変換の演算時間である。入力信号 64 サンプルにつき、高速フーリエ変換 (radix-2 の場合) の演算所要時間が約 9.7 μ S かかり、音楽データ 256 フレームのフィンガープリント生成には入力信号 4177152 サンプルの演算をしなければならないので FFT 合計時間は、少なくとも約 600ms は必要である。

4.3 回路量とクリティカルパス

本項で構築した 10 並列、30 並列楽曲識別回路の消費回路量、およびクリティカルパスを表 4.2 に示す。30 並列の回路においても 4-input-LUT は 5888 と、少ない回路量で構築することができた。これは実装環境である XC2V6000 全体の回路量では 10 %しか消費していないことになる。したがって本稿で実装した環境においては楽曲識別回路の並列度数は回路量ではなくて、FPGA に含まれる RAM 容量の上限にて制限される (最大 100 並列)。

表 4.2: 回路量とクリティカルパス

	4 input LUTs	critical path	max freq
FP 生成部	1997(67584)	11.500ns	86.957MHz
楽曲識別部 10 並列	2901(67584)	25.970ns	38.505MHz
楽曲識別部 30 並列	3899(67584)	24.983ns	40.027MHz
楽曲識別部 40 並列	4331(67584)	26.583ns	37.617MHz
楽曲識別部 50 並列	4875(67584)	26.758ns	37.371MHz
楽曲識別部 60 並列	5330(67584)	26.871ns	37.214MHz
全体 (FP 生成部+楽曲識別部 10 並列+IF 部)	5107(67584)	25.027ns	39.956MHz
全体 (FP 生成部+楽曲識別部 30 並列+IF 部)	5501(67584)	26.138ns	38.258MHz
全体 (FP 生成部+楽曲識別部 40 並列+IF 部)	6701(67584)	26.564ns	37.645MHz

また、10 並列時のクリティカルパスは 25.027ns となり、フィンガープリント生成部単体と比較して２倍以上になっている。これは楽曲検索部内にて、ビットエラー総数を求める部分のファンアウトの増大がクリティカルパスを著しく増加させている原因と推測される。

しかしながら、10 並列から 60 並列同時検索・検出システムへ拡張した結果、図 4.3 からクリティカルパスは 1.1ns 程度の増加、周波数は 1.1M 程度の減少に留まっているため、楽曲検索回路の並列数はクリティカルパスの増加には大きく影響しないことが分かる。つまり、33M の動作周波数で動作させた場合は 100 程度の並列数では動作周波数は大きく減少せず、並列数を著しく増加させても現状の OSC（水晶発信器）のクロック周期での動作が問題なく行うことが可能である。また、30 並列時においては、クリティカルパスの遅延が 10 並列時よりも少なくなっているが、これは 30 並列時に限りザイリンクス XST による論理合成が最適に行われた結果であると想定される。つまり FPGA による論理回路構築の最適性は合成ツールとの相性に依存するところが非常に大きいということになる。

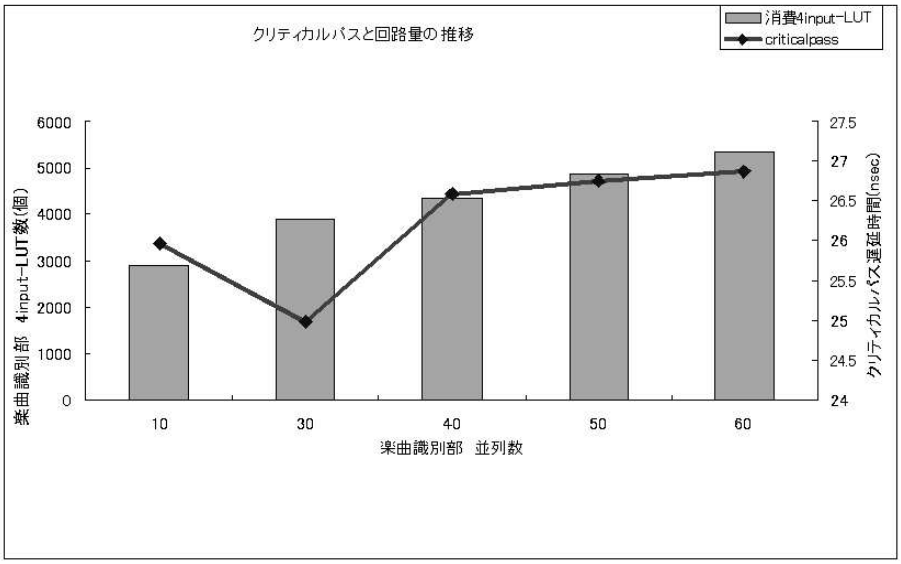

図 4.2: クリティカルパスと回路量の推移

4.4 ロバスト性　信頼性評価

4.4.1 ロバスト性の評価

ロバスト性は音質が変化したオーディオを識別可能かどうかを示すパラメータである。本研究ではロバスト性の実値は偽陰性（見落とし) の確率で表し、認識率と等しいと考える。

ロバスト性 (認識率) の定義式を以下に示す。

$$\textbf{ロバスト性}\, rob(\%) = (1 - \frac{\textbf{検出不可回数}}{\textbf{総試験回数}}) \times 100 \tag{4.1}$$

評価方法はオーディオファイルの音質の複数のパラメータを徐々に変化させていき、認識率を測定する方針とする。

オーディオファイルの音質の変化は wav **編集ソフト** SoundEngine Free[12] **を使用した。音質のパラメータは、「質感」、「高域強調」、「丸み」、「**CV **制限」（変化量の制限)、「ボリューム」（表** 4.3 **には記さない）の５つとし、このパラメータを組み合わせた１０通りのモード (表** 4.3 **参照) で生成されたオーディオファイルをフィンガープリントシステムに入力する。質感 「-」では、高域を強く「+」では低域を強くする。また、高域強調では「**0**」以上で高域を強め、** CV **制限では** wav **データの音圧 (**-32768 **〜** +32767**) の変化量を強制的に制限する。また、ある** J-pop **の** wav **音楽データを** mode **毎に加工した後の波形の様子を図** 4.4 **に示す。認識率の測定の結果、音質に丸みを持たせる加工処理をしたオーディオファイルの認識率が** 100% **では無いことがわかる。音質に丸みを持たせる加工処理**

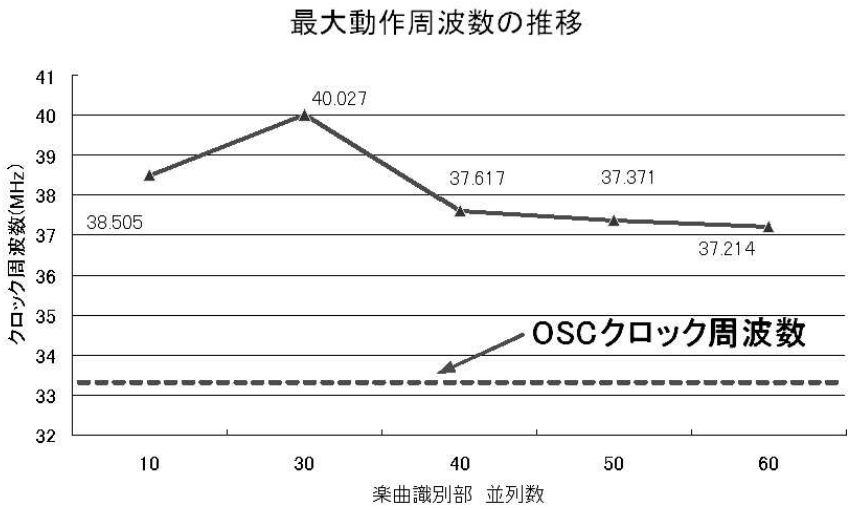

図 4.3: 最大動作周波数の推移

には耐性は低いが、おおよその常識的な音質加工処理に対しては本研究で提案するハードウェアフィンガープリントシステムのロバスト性を示すことができたと考える。

4.4.2 誤検出確率の評価

フィンガープリントシステムとしてファイルの誤検出はシステムとしての性能低下を招くため、検出したファイルが本当に正しいかどうかの指標は信頼性としてパラメータ化される。信頼性は間違った識別が何度起こるかを確率にしたパラメータであり、つまり「偽陽性」（ファイル誤検出）が発生した場合に信頼性が低下する。本研究で取り扱う誤検出率を以下の式にて定義する。

$$\textbf{誤検出率}\ f(\%) = (1 - \frac{\textbf{誤検出回数}}{\textbf{試験総回数}}) \times 100 \tag{4.2}$$

誤検出測定の評価として、フィンガープリントのデータベースに順番に複数の楽曲 (A,B,C,D.wav **ファイル**) **を１００回にづつ入力させ、誤検出をする確率を算出する。**

測定の結果（図 4.4 **参照)、全てのオーディオファイルは総測定回数４００回のうち一度も誤検出されることはなかった。つまり、本研究のフィンガープリントシステムは低誤検出率から考慮した結果、関して非常に高い信頼性が達成できたと思われる。**

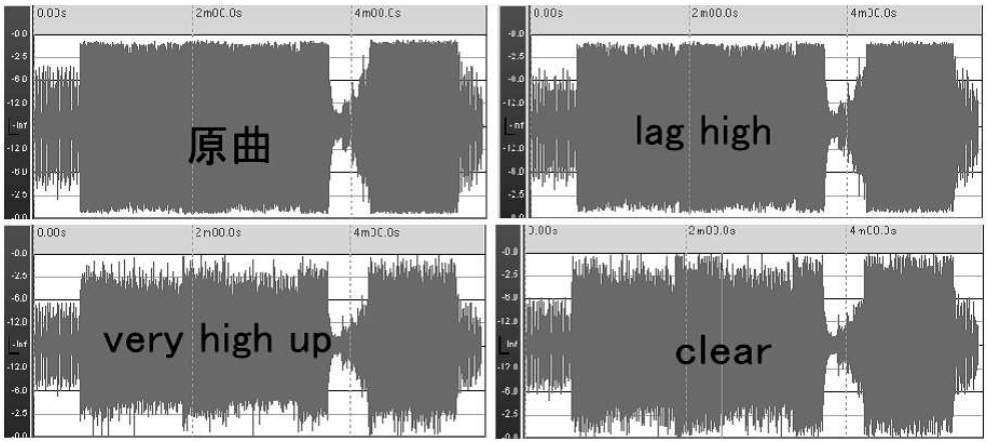

図 4.4: **音質加工後の波形**

4.5　更なる性能向上に向けての修正と評価

4.5.1　【改善策１】フィンガープリント部の並列展開

構成

フィンガープリント生成回路をモジュールレベルで並列化し、演算の処理を分散させることで、さらなる高速化を目指した。図 4.5 **に、構築した回路構成を示す。まず、フィ**

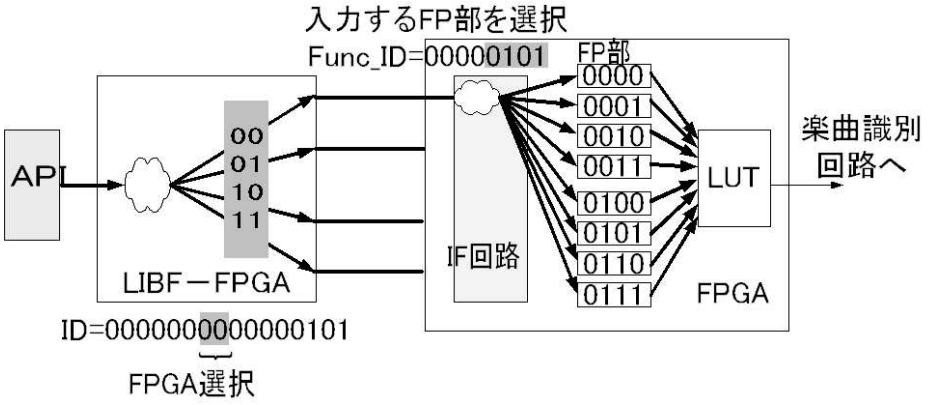

図 4.5: FP **回路の並列展開**

ンガープリント回路を８個に並列展開し、それぞれのフィンガープリント回路に 0 **から** 7 **（**2 **進）の** ID **を割り振る。ホスト** PC **上の** C **プログラム上にて、書き込み** API **の「**id**」という引数を、**0 **から** 7 **までインクリメントさせることで、データの書き込み先を変化させる。各フィンガープリント回路にはオーディオデータ３２フレームづつ処理させる。この** id **は** Func_ID **として** FPGA **内のロジック回路に通知される。３章で説明したように、イ**

表 4.3: 音質変化モードと認識率

モード名	質感 (%)	高域強調 (dB)	丸み (%)	CV 制限 (音圧)	認識率 (%)
clear	-50.0	0	0	no limit	100
high limit	0	0	0	4909	100
high limit2	0	0	0	7999	100
high up	0	3.0	0	no limit	100
very high up	0	6.0	0	no limit	100
lag high	0	0	50.0	no limit	0
lag mid	0	0	20.0	no limit	80
lag low	0	0	20.0	no limit	90
quality change	-30.0	3.0	15.0	9993	100
quality change2	50.0	6.0	30.0	no limit	100
warm	50.0	0	0	no limit	100

表 4.4: 誤検出評価

	A.wav	B.wav	C.wav	D.wav	平均
誤検出回数 (回)	0/100	0/100	0/100	0/100	/0100
誤検出確率 (%)	0	0	0	0	0

ンターフェイス部にて書き込みデータの先頭に位置する id 値を読み取ることで、書き込み先のバス線をセレクタから切り替える構成とした。

考察・問題点

以下、表 4.5 にフィンガープリント部を 8 並列で配置した場合の回路量と最大動作周波数を示す。4-input LUTs は 27449 となり、フィンガープリント回路の単並列時と比べて約

表 4.5: FP 部 8 並列実装時の回路量とクリティカルパス

	4 input LUTs	Slices:	critical path	max freq
FP 部 8 並列+楽曲識別部 30 並列+IF 部)	27449(67584)	23225(33792)	29.578ns	33.808MHz

5.5 倍ほど増加した。表 4.2 によると、フィンガープリント回路の 4-input LUTs は約 2000 であり、フィンガープリント回路 8 並列で約 16000、その他の回路量消費の増加の原因として、並列数増加に伴う周辺回路、問い合わせフィンガープリントを格納するルックアッ

プテーブルの追加が挙げられる。しかし、クリティカルパスの遅延が大幅に悪化した。これは、書き込み先 ID を認識し、出力先の切り替えを行うインターフェイス回路内の最終段のフリップフロップの出力からフィンガープリント回路の最初のフリップフロップの入力までの配線のファンアウトが増加したことが原因である可能性が最も高い。かろうじて、Virtual Turbo のクロック周波数での動作は可能だが、これ以上の並列数の増加は、さらなるクリティカルパスの増加の原因となり、駆動周波数 33MHz での正常動作は不可能になると思われる。結果としてこの並列展開でのフィンガープリント処理速度の短縮は不可能であった。それは、シリアルで入力されるオーディオデータの分散処理は Virtual Turbo の書き込み API の制限上不可能であることが原因である。並列に配置された高速フーリエ変換のイネーブルを同時にアクト状態にすることができず、並列処理が行うことができない。つまり、そのイネーブルは前処理が終った段階でしか、順次アクト状態にならないため、結局フィンガープリント回路の１実装時と処理時間は変わらない。

4.5.2 【改善策 2】radix4 Burst FFT コアの使用

ザイリンクス社の Core Generator ver8.1 は Radix-4 バタフライ演算を採用した FFT コアを提供している。回路構成を図 4.6 に示す。

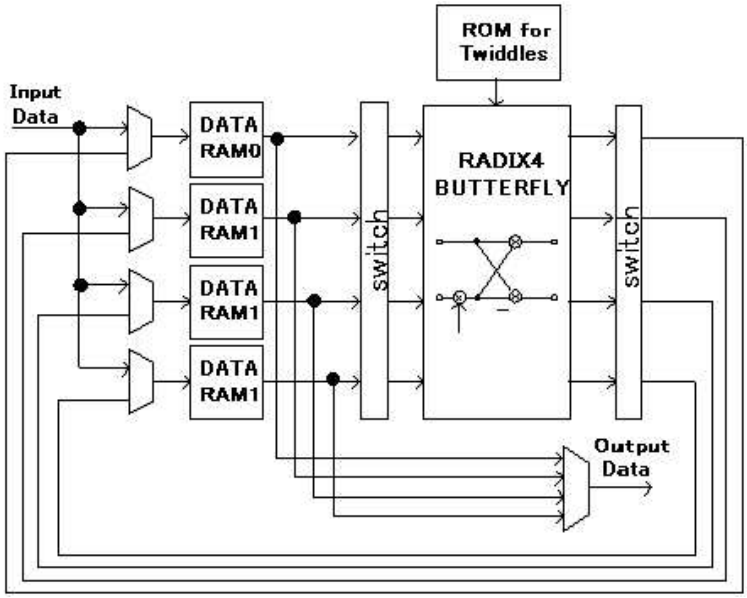

図 4.6: 基数４分解バタフライ演算を用いた高速フーリエ変換回路図

Radix-4 とは Radix-2 バタフライを拡張したもので、Radix-2 が 2 入力であるのに対して Radix-4 バタフライは 4 入力で計算を行う。Radix-2 における計算量は，$N \log_2 N$(シグナルフロー図の段数に相当) なのに対して Radix-4 での計算量は，$N \log_4 N$ で処理することができる。このことから，本研究では１回当たりの入力サンプル数は 64 としているの

で、計算量は全体で約 50%の削減となる。この回路は入力を 64 サンプルとした場合、変換時間は約 1.04sec であり、これをそのまま、radix2 のフーリエ変換に置き換えた場合、radix-2 では約 650ms 必要であったのに対して、350ms から 400ms 程度でフィンガープリント処理が可能ということになる。radix-4 版 FFT に置き換えた場合の、回路量を表 4.6 に示す。これは ISE8.1 の論理合成ツール XST で合成した。(理由:radix-4 の FFT を生成する Coregenerator8.1 に対応しているのは ISE8.1 のみであるため。レポートファイルを確認するだけのために使用)

表 4.6: 4radixFFT 実装時の回路量とクリティカルパス

	4 input LUTs	critical path	max freq
全体 (FP 生成部+楽曲識別部 30 並列+IF 部)	7615(67584)	23.607ns	42.360MHz

結果を見ると radix-2FFT の実装時 (表 4.2) と比較して、動作周波数が向上し、4-input LUT は約 40%ほど増加している。動作周波数の向上は ISE8.1 の単に合成技術がさらに効率化されていることが理由であると考え、合成結果は radix2 とは単純比較は不可能である。また、radix-4FFT 実装時のフィンガープリント処理時間を radix-2 時と同様、図 4.1 の測定系にて、ロジックアナライザの波形観測にて算出した。

表 4.7: 処理時間の比較 2

	FP 生成時間	楽曲識別時間	性能比
ソフトウェア	3353ms(19.95Mbps)	ー	1.0
ハードウェア radix-2 FFT を使用	650.24ms(102.78Mbps)	0.543ms	5.152
ハードウェア radix-4 FFT を使用	314.62ms(213.30Mbps)	0.543ms	10.66

上記のフィンガープリント処理時間の 314ms という結果は radi-4FFT の計算量が radix-2 の計算量の約半分であるということを考えた場合、理論どおりの結果と言っても良い。以上の結果より、最終的にハードウェア処理で得られた性能はソフトウェア処理の約 10.66 倍となった。

4.6 考察

4.6.1 高速化手法の検証

本稿では、本研究の特徴である機能粒度を考慮したパイプラインレジスタ並列処理の検証を行う。機能粒度を均衡な箇所にパイプラインレジスタを配置し、非パイプライン時と比較してステージ分 (10) の 1 の処理速度となる設計方針で構築を行った。図 4.7 に

Radix=2 の高速フーリエ変換実装時の仕様どおりのオーディオ 1 フレーム分のデータ処理の様子を示す。仕様で動作する場合、1 フレームのパイプライン時の処理時間は

$$T_{pipe1} = (L + N - 1)\tau_{pipe1} = (10 + 256 - 1)30(ns) \times 64(cyc) = 0.508(ms) \quad (4.3)$$

となる。したがって、オーディオフィンガープリント完となる 256 フレーム分だと、処理時間は 130.25(ms) となる。その後の処理過程である、減算や比較を加えても、140(ms) 程度での処理が可能である。

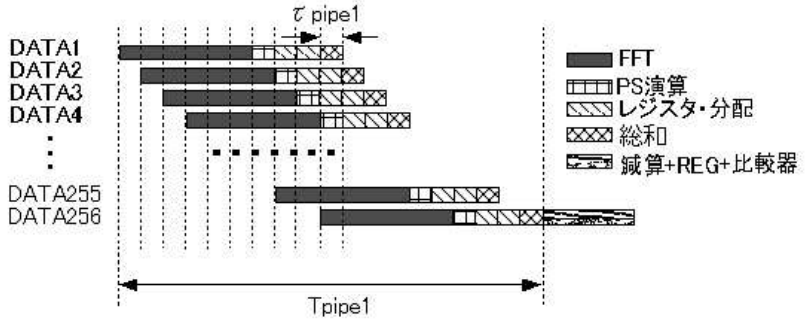

図 4.7: 仕様の動作

しかしながら、測定結果を見ると、650.24(ms) となっていることから、フーリエ変換器の動作を HDL シミュレータ「Modelsim」を使用して検証した。出力波形測定の結果、フーリエ変換器に入力から出力まで 329(cyc) 必要で、リアルタイムに結果が出力されていないことがわかった。(出力待ち状態が発生している)

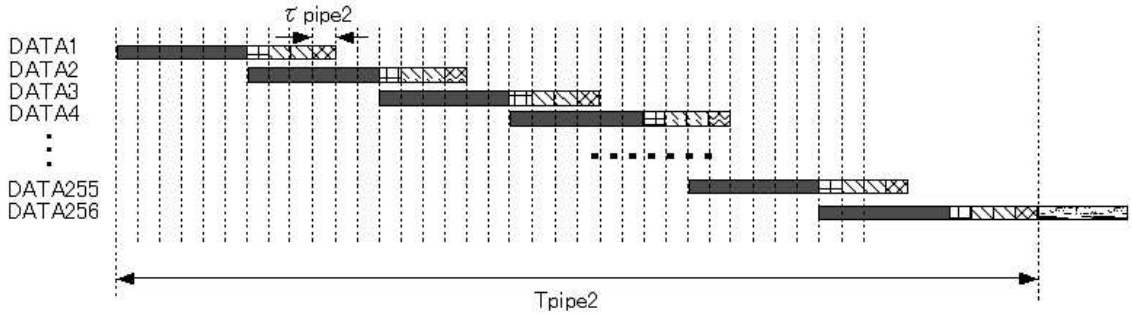

図 4.8: 実際の動作

したがって、１フレーム時のデータ処理の流れは図 4.8 の通りとなる。つまり、最初の 5 段のパイプラインの効果は無く、後４段のパイプラインの効果が表れている。１フレームの処理時間は

$$T_{pipe2} = (30(ns) \times 329(cyc) \times 256) + (30(ns) \times 64(cyc) \times 4(\text{演算器})) = 2.53(ms) \quad (4.4)$$

256 フレーム分だと 647.68ms という結果となり、測定結果とほぼ等しいことがわかる。

4.6.2 ハードウェア化による性能への影響

4.6.3 客観的性能比較

4.7 まとめ

本項での評価結果を考察すると、まずフィンガープリント処理時間については、radix-2FFT を実装した時は 650ms(ソフトウェアによる処理速度の約 5.1 倍)、radix-4FFT を実装した時は 314ms(ソフトウェアによる処理速度の約 10.6 倍) で達成できた。これはフィンガープリント生成部における高速化に関する２つの方針、機能粒度を考慮したパイプラインレジスタ配置、および所要クロックの削減のためのコンパレータ（比較器）のループ展開の効果であると考える。また、フィンガープリント処理過程において一番計算負荷が高い、変換部に効率的な FFT を配置ができたことも２番目の勝因ともいえるであろう。楽曲識別部のクリティカルパスの遅延の大きさは、全体の最大動作周波数を大きく下げる原因となってしまったが、楽曲識別部の並列展開によるクリティカルパスの増大、最大動作周波数の減少はそれほど、重大な問題ではなく、100 並列以上の展開も可能であることを示した。また、ロバスト性については、音質の変化による耐性を評価した。ほぼ常識的な音質の加工に対しては耐性があり、ロバストなハードウェアフィンガープリントシステムであることを証明した。また、サンプリングレートを 44.1KHz から変更した wav ファイルの認識は不可能であった。それは音声データの参照するサンプル数 44.1KHz のみに対応させて設計しているためである (44.1KHz=16384 サンプル/フレーム)。Haitsuma らの理論上での非常にロバストなフィンガープリントアルゴリズムをハードウェア化し、そのロバスト性、信頼性を実験によって検証したことは、ソフトウェアによる処理が一般的であるオーディオフィンガープリント分野にとって、革新的ではないかと考える。

第5章　フィンガープリントを用いたコンテンツ不正流通対策の実験

5.1　概要

ブロードバンド回線の普及につれて、インターネット上でコンテンツ流通が盛んに行われている。インターネットの普及率の増加と利用の高度化・多様化に伴い、デジタルコンテンツ産業への悪影響が非常に問題になりつつある。悪影響とはおおむね「不正な蓄積」や「インターネット上での不正流通」の二つといえる。音楽ファイルなどの不正流通は、

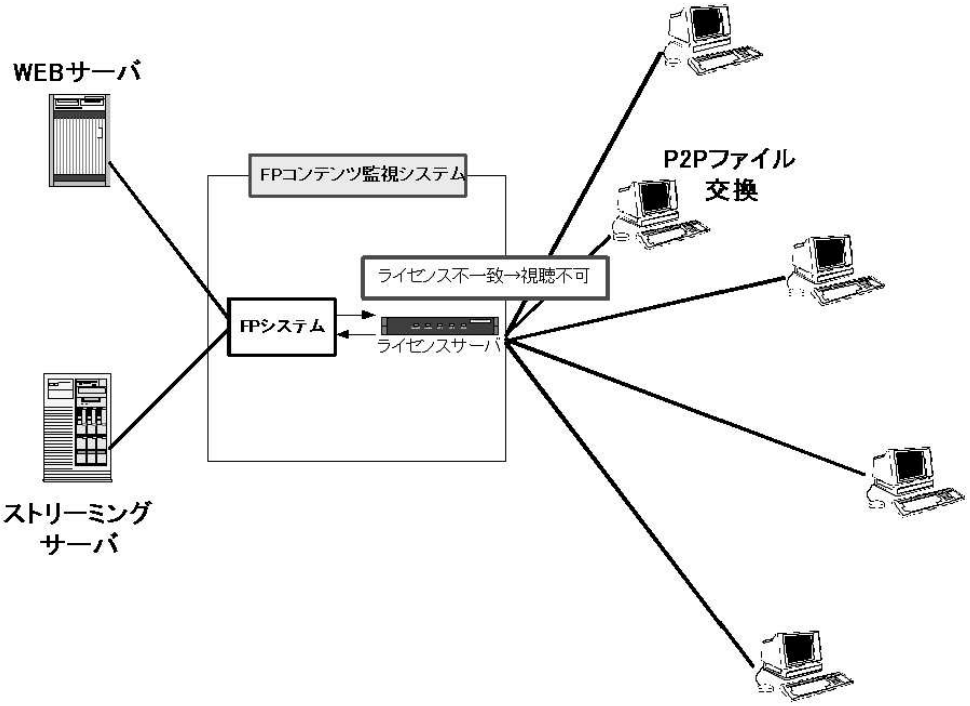

図 5.1: オーディオ用フィンガープリントを用いた流通監視（DRM システム）

悪意のある一般ユーザによって、ウェブサーバやストリーミングサーバなどから違法なソフトウェアを使用することでコンテンツの不正な蓄積がなされ、コンテンツを p2p 上などで共有する。著作権団体がネットモラルを訴え続けても、その効果は殆ど無い。近年では

Winny などの P2P ファイル交換ネットワーク上で、mp3 音楽ファイルなどの不正蓄積されたコンテンツの２次不正流通が行われており、本来購入すべき楽曲が無料で入手できてしまうことから、CD 売り上げの低下に直結している。

著作権侵害行為に対処するために、近年ではコンテンツ流通機能として DRM（著作権管理）システムが導入されている。本研究ではその DRM の識別技術としてオーディオ用フィンガープリントを用い、音楽コンテンツを ID 化し、利用者個別にライセンスを付与することで、権限管理を行うシステムを提案する（図 5.1 参照）。このシステムにより悪意のある利用者が、たとえコンテンツを取得したとしても、ライセンス不一致のため視聴不可となる。このシステムは、既存のコンテンツ配信システムと連携を図り実現される。主に一度通過した音楽ファイルを事前にフィンガープリントを付加させ、その情報をライセンス サーバにあらかじめ登録しておく。利用者の音楽購買履歴により、ライセンスサーバ上には権限管理された個別のライセンス情報が蓄積される。これにより、コンテンツ配信の仕組みを変更することなしに、音楽の著作権保護を向上させることができると考える。

5.2 実用評価

将来的にオーディオフィンガープリントを用いた DRM システムを実用性のあるものへ発展させるため、図 5.2 に示すようなネットワーク系を構築し、ネットワーク上を流れているパケットに対して、フィンガープリントを付加し楽曲を識別する簡易的な実用評価を行った。

また、図 5.3 に実用評価系の写真を示す。写真中の　は FPGA 搭載 PC、　は音楽を ino-www からダウンロードするための PC、　はキャプチャPC である。

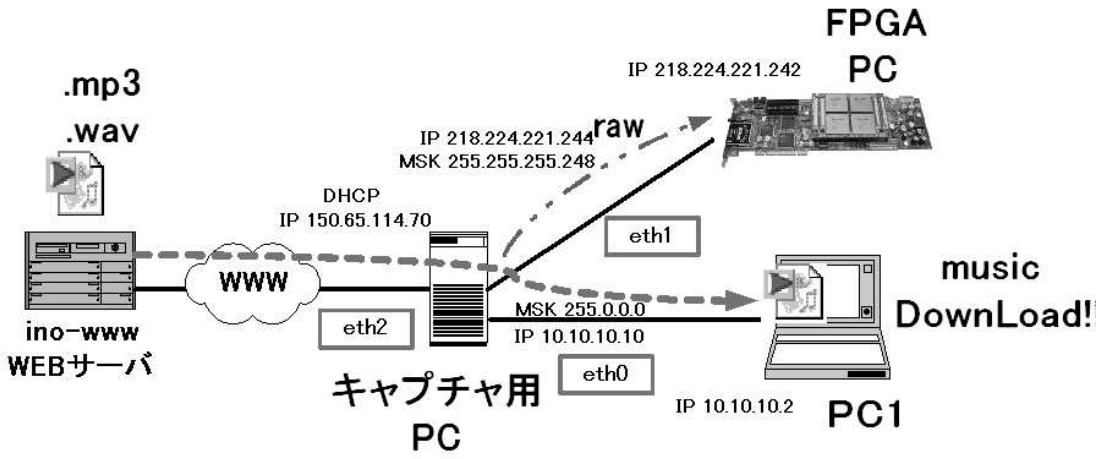

図 5.2: 実用評価系

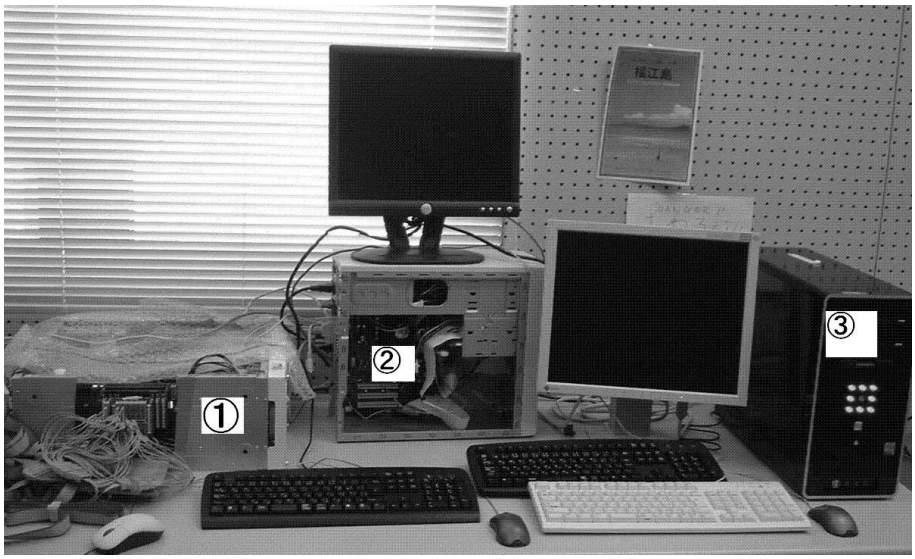

図 5.3: **実用評価系の風景**

この評価系は３つのサブネットから構成され、パケットキャプチャ用 PC **は３つのポート** (eth0〜2) **を持ち、**TCP/IP **で通信する。**eth0 **は楽曲ダウンロード用** PC(PC1) **間のサブネットに接続し、**eth1 **は** FPGA **ボードを搭載する** PC(FPGA PC) **間のサブネット、**eth2 **は外部インターネット接続用ポートである。つまり、キャプチャ用** PC **はこの３つのサブネットのブリッジ機能と楽曲ファイルのキャプチャを行い、**mp3 **および** wav **ファイルを** raw **データへ変換するデフレーマーの２つの役割を担う。図** 5.4 **に実用評価の流れを示す。キャプチャ用の** PC **は** eth2 **から** eth0 **へのダウンロードを監視すると同時に、**mp3 **または** raw **ファイルを検出すると、**eth1 **に情報を送信する。**

外部のインターネットに接続可能な、任意の場所に配置されている PC1 **にて、**ino-www **の** WEB **サーバにアップされている** mp3 **もしくは** wav **ファイルをダウンロードする。楽曲ファイルのダウンロードの直前には、あらかじめ** FPGA PC **はソケットを作成、バインドを行い、リクエストの受信待機（**listen **関数）記述したプログラムを実行させることで、キャプチャ用** PC **からのデータの受信に備える。キャプチャ用** PC **は、キャプチャした音楽ファイルが** mp3 **フォーマットの場合、**wav **ファイルにフォーマット変換を行った後に、**wav **ファイルのフレームをはずすことで、**16bit **の**.raw **ファイルにする。キャプチャ用** PC **は、この抽出した**.raw **ファイルを** eth1 **サブネットの固定** IP**「**218.224.221.244**」へデータを送信することで、**FPGA PC **は接続を確立しデータの受信を行う。受信したデータは** Virtual Turbo **専用の書き込み** API **関数に渡され、**FPGA **での処理が開始される。以上の流れに従い実験を行った結果、ローカルで行った評価と同様に複数の楽曲データベースの中から、楽曲が検出できた。**

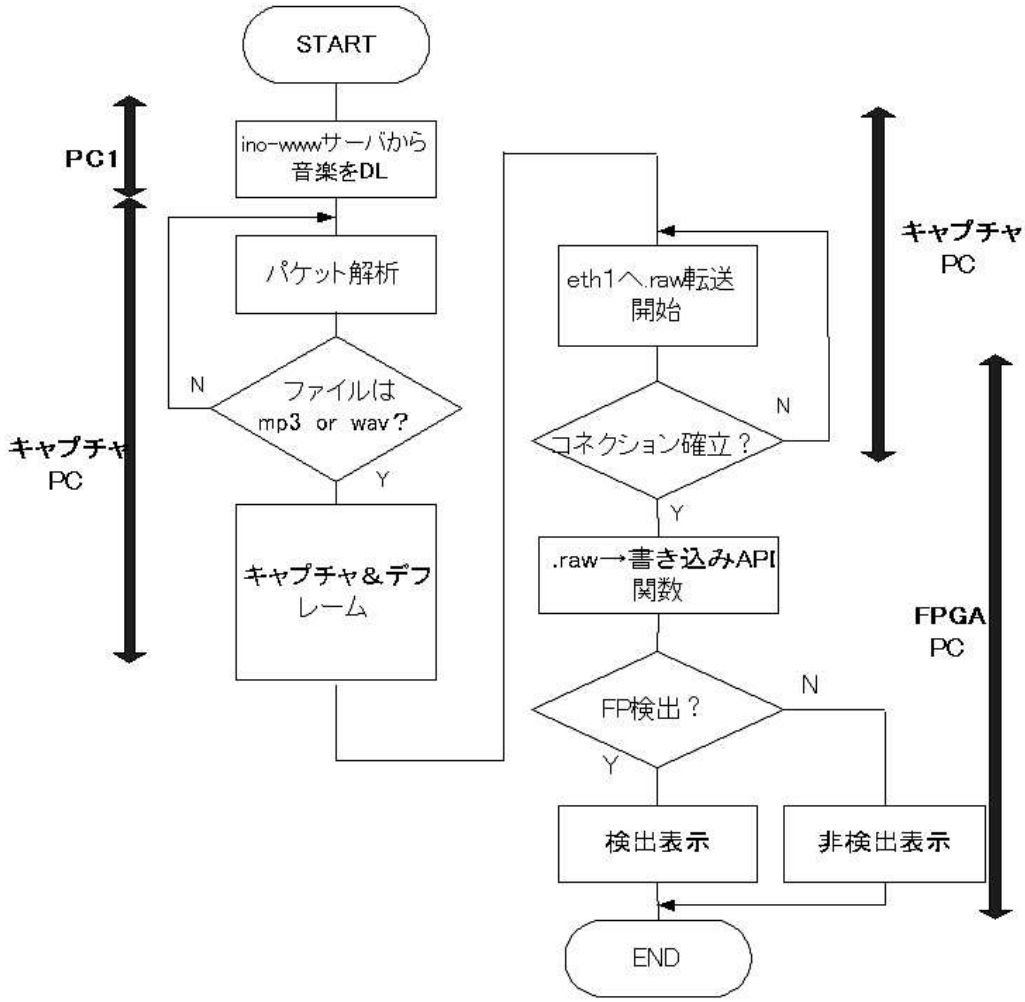

図 5.4: **実用評価の流れ**

5.3 考察

オーディオフィンガープリントの DRM **システムへの適用においては、誤検出により不正者の誤認や機器の誤動作などの重大な問題を引き起こす可能性があり、これを許容範囲内に収めることが必要である。また、検出情報に基づいて不正流通の監視をする際に関しても、誤検出を防止し、検出情報の信頼性を高めていく必要がある。強度性と信頼性はトレードオフの関係にあり、偽陰性と偽陽性の妥協点を見出さなければならない。音楽ファイル誤検出により、無実のユーザが受ける不利益は最も避けるべき事態であるため、現状では前者の精度を下げることが望ましいと考える。したがって、本研究ではスレッショルドと表現した閾値のパラメータを厳しくし、ほぼ原曲と等しい質、つまりクオリティの高いファイルのみを検出し、ある程度、質の劣化した不本意ではあるが音楽は見送る手段をとるしかないと思われる。**

また、音楽ファイルではないファイルの誤検出の可能性も 0 ではない。本研究で採用したフィンガープリントアルゴリズムに限ってはパワースペクトルの時間的ディレイの差から算出されたフィンガープリントのため、インターネットに流れている html,text,gif などの他種類のファイルのバイナリ列がデータベースに格納されているバイナリ列とマッチングするとは考えられない。

他種類ファイル誤検出を防ぐ確実な手段は、mp3 もくしくは wav 等のファイルフォーマットを判定した直後から、このフィンガープリント検出処理を行うことである。ファイルフォーマットの判定処理はソフトウェア処理でもボトルネックとならず、検出速度を低下させること無く、信頼性を向上させることができる。

第6章　結論

6.1　まとめと今後の課題

コンテンツへの海賊行為に対処するために、コンテンツ流通機能としてDRMシステムを導入することが盛んになってきている。DRMシステムはコンテンツを事前にDRMシステムへ対応させる必要があり、そこには電子透かしやフィンガープリントなどのコンテンツ識別技術が適用される。DRMシステムが配備されるブロードバンド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，複数の楽曲識別を高速に試みなければならない。しかしながら、オーディオフィンガープリントは信号そのものの特徴量を解析するため、計算負荷が高く、ソフトウェアで処理するとボトルネックとなり、DRMシステムに組み込むにはスペック不足である。したがって、本研究ではハードウェアを用いることで、処理速度を上げ、将来的にはDRMシステムへのオーディオフィンガープリントの導入を可能とすることが狙いであった。

そこで、ハードウェアにて高速化するフィンガープリントアルゴリズムは以下の点に留意して、選定を行った。

- パイプラインレジスタによる並列処理や効率的な演算が行えるアルゴリズム
- 浮動小数点演算や乗除算が少なく、おおむね加算減算によって構成されるアルゴリズム
- 簡単な定義式で強度、信頼性を実現できるアルゴリズム
- フィンガープリントサイズがコンパクトなアルゴリズム
- フィンガープリントの出力までのステップが少ないアルゴリズム
- 音声特徴解析(DSP)ステップにおいて、回路構築が容易、かつ高性能が演算が見込めるアルゴリズム

Haitsumaらのアルゴリズムの特徴抽出式はパワースペクトル密度の時間的位相差から成り立っている「Hash　String法」を採用しているため、これらは音楽の感性的な領域を、そのままID化する技術であるので、非常にロバスト性が高い。しかも、特徴抽出式が加算・減算のみの簡素な構成のため、あまり、回路量を消費することなく特徴抽出ができるメリットがある。また、サイズが1KB/曲と非常にコンパクトでありFPGAの少ないメモ

リ資源に対しても、多数のフィンガープリントを格納ができる。また、フィンガープリントの探索についてはBERに基づいた楽曲識別法を提案し、その提案アルゴリズムを実装した。BERに基づいた楽曲識別法はBERが低いとその楽曲であるという、相補誤差関数を用いた確率定義式から立案した。このBERは問い合わせフィンガープリントとデータベースフィンガープリントのハミング距離から算出されるもので、同時に楽曲の識別を行うため、並列にBER演算を行う。一度に複数の楽曲識別が可能なため、並列数を多くすることで、検索時間の短縮が見込まれる。

また、フィンガープリント部は「パイプラインを用いた並列処理」と「比較器のループ展開」の２つの高速化手法により、高性化を目指した。特に、パイプライン処理においては、機能粒度を考慮したレジスタ挿入を行う特徴強いアプローチで設計を行った。これにより、アルゴリズムをそのままハードウェア化するよりも、２倍以上の性能を得ることが可能となった。

構築したハードウェアは、オーディオデータから高速にフィンガープリント生成を行い、複数の楽曲を同時検索することを確認した。30楽曲の同時識別を試みた場合では213Mbpsの速度で楽曲識別が可能となりソフトウェアによる処理と比較して約10.66倍(Radix-4FFT実装時)の性能向上が得られた。これは広帯域なネットワーク上での音楽ファイルの捕捉や将来的にDRMシステムに組み込むことが可能となるなど、コンテンツ流通・管理の処理能力向上という点で有益な結果が得られたと言って良い。

今後の課題として、フィンガープリント処理速度のさらなる向上を挙げる。これは評価結果にも記載したようにフーリエ変換の演算待ちにより、パイプラインレジスタが半分しか機能しなかったため、理想値では140msで処理ができるところを、今回は650msかかってしまった。それでもソフトウェアの処理時間と比較して5.1倍の性能が得られたが、パイプライン処理を確実とするFFTに置き換えることで、約２３倍の性能が達成できるものと推測する。

また、今回は楽曲識別部のクリティカルパスの大きさが、最大動作周波数の低下を招いた。BERのループ加算部が原因と想定され、構築記述の変更により低減できれば、楽曲識別部の並列数増加に伴う、最大動作周波数の低下は防げると思われる。

このハードウェアオーディオフィンガープリントを楽曲識別システムとしての有効利用を考えた場合、世の中に存在する膨大な音楽のタイトル数を考えると、楽曲のデータベース数に関して言えば容量不足である（現状、少数のタイトルしか識別可能としないため)。HMVなどの大手レコード会社がインターネットで配信する音楽のタイトル数は、60万〜70万タイトルと増加する一途であり、本稿で提案したシステムをさらに実用的とするためには、FPGAと高速に通信可能な大規模な記憶媒体が必要である。もしくは、今回で構築したFPGAにおけるフィンガープリンティング部をサーバーなどで構成される大規模なオーディオフィンガープリントサービスの一部にアクセラレータとして組み込むような実用方法も考えられる。

参考文献

[1] P. Cano, E. Batlle, H. Mayer, and H. Neuschmied,“ Robust sound modeling for song detection in broadcast audio, ”in AES 112th International Convention, May 2002.

[2] E. Allamanche, J. Herre, O. Hellmuth, B. Frbach, and M. Cremer,“ Audioid: Towards content-based identification of audio material, ”in 100th AES Convention, May 2001.

[3] A. Wang,“ An industrial strength audio search algorithm, ”in 4th Int. Symposium on Music Information Retrieval (ISMIR), Oct.2003

[4] Y. Cheng,“ Music database retrieval based on spectral similarity, ”in 2nd Int. Symposium on Music Information Retrieval (IS- MIR), Oct. 2001.

[5] P. Cano, E. Batlle, T. Kalker, and J. Haitsma,“ A review of algorithms for audio fingerprinting, ”in International Work- shop on Multimedia Signal Processing,, Dec. 2002.

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

[7] Pedro Cano , Eloi Batlle ,“ A Review of Algorithms for Audio Fingerprinting ” Proc.IEEE International Multimedia Signal processing 2002

[8] Kazuhiro SAKAKIBARA,Yasushi INOGUCHI, ”Detection of the audio watermark on FPGA”, CPSY2004-80, VLD2004-114, pp. 25-30

[9] Prarthana Shrestha, Ton Kalker ”AUDIO FINGERPRINTING IN PEER-TO-PEER NETWORKS”, 2004 Universitat Pompeu Fabra.

[10] **小野 束，電子透かしとコンテンツ保護，オーム社，**2001.

[11] **安田 浩，安原 隆一**, **コンテンツ流通，アスキー，**2003.

[12] http://www.cycleof5th.com/index.htm

本研究に関する発表論文

- FPGA を用いた音声フィンガープリントの高速化手法, 礒永 久史, 井口 寧, 2005 年度 電気関連学会 北陸支部連合大会,E-40, 2005/9/24, 石川工業高等専門学校
- FPGA を利用した高速オーディオフィンガープリントシステムの構築, 礒永 久史，井口 寧, 2005 年度システム LSI 設計技術研究発表会, 信学技報, 2006-SLDM-123,P1-P6, 2006/1/17, 慶応大学 理工学研究科

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，日比野靖教授に深く御礼申し上げます．また、実用評価のキャプチャPC用のソフトウェアを作成していただいた、株式会社ハートソリューションズ様にも厚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近藤裕貴氏，清水昭尋氏，高畠義和氏，松山周平氏に深く感謝致します．井口研究室の先輩である Doan Son 氏,Zhang Yuanyuan 氏,Vinh trong Le 氏,Hieu Xuan Phan 氏,Sun Wei 氏, 広瀬 勇人氏には様々なアドバイスを頂き，後輩である長瀬 文彦氏, 熊坂 史彬氏,Li Qi 氏,Li Bo 氏, には日々の愚痴を聞いて頂きました．また、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，長い間、私を支えて下さった長崎の家族に深く感謝致します．