修 士 論 文

ハイブリッド処理を用いた フィンガープリント情報の 高速検索に関する研究

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

熊坂 史彬

2008 年 3 月

修 士 論 文

ハイブリッド処理を用いた フィンガープリント情報の 高速検索に関する研究

指導教官　井口寧 准教授

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

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

0510033 熊坂 史彬

提出年月: 2008 年 2 月

概 要

本稿では，コンテンツ識別技術としてある各種方式を紹介し，その中でもフィンガープリント情報を用いる方式に着目しコンテンツ管理を行う方法を示す．また，コンテンツ管理のシステムに求められる制約を現実的なコンテンツの流通の分析によって明らかにする．具体的にはフィンガープリント情報を用いたコンテンツ管理方式の場合，対象とした入力されるフィンガープリント情報を既知のフィンガープリント情報データベースと比較する．これによって，フィンガープリント情報をキーにして，コンテンツを識別が可能になる．

問題点としては，現実面からのシステム運用を考えた場合，どうやって大量のコンテンツをごく限られた短時間で扱うか，と言うことになる．

最も基本的な方針として考えられるソフトウェアでのアプローチを行い，結果を示す．ここではこの結果によって，ソフトウェアのみでのアプローチに限界があることが理解できた．(ワンタイム検索の要求で 300 秒程度．100 万曲データベースを備えている場合)

現実的なコンテンツの流通具合を分析した結果，(特に音楽ファイルでは）単位時間当たりのデータ要求が「流行」という要素にだいぶ左右されることが分かった．「流行」は常に変動する要素として与えられるので一意な解を設けておく事が良い結果にはつながらない．「流行」を先読みできる事が最も良いアプローチだと思われるが，この方法で成立するシステムが存在しないため我々は可能な限りの予測と言う方法を用いる事にした．

話題の楽曲ファイルのリリースによって「流行」が始まる傾向があることをコンテンツ分析から得られたためである．また，時間経過に伴って「流行」の変動やリリース数の売れ行き減少などもみえてくるので，その予想もある程度のスパンで入れ替えが必要になる．(分析によって得られる最頻の参照なデータを何とかすると高速化できそうだ，と言う着目点の説明と紹介．)

本稿では，これを比較するデータベースを既知として扱うので，最も最近リリースされた話題の楽曲を「流行」の楽曲と予想して一部分を再構成可能なハードウェア上に搭載することにした．これにより，単位時間当たりのシステムに対する入力のうち多数の割合を占める流行の楽曲の検出を可能にした．また，再構成可能デバイスを用いる事で，流行の変動時に対する変更も可能にした．ハードウェア上での処理は高速であるが，容量に制限がある．

これによってソフトウェアのみではなしえなかった検索時間の短縮を，ハイブリッドシステムによって実現することができた．提案システム全体として，ハードウェアをソフトウェアを両方用いるハイブリッド構成にした．これによって，大容量のデータをソフトウェアによって扱う事が可能となる．その中での高速応答を求められる部分をハードウェア処理する事により，大容量のデータを高速に検索・検出することが可能になった．

目 次

図 目 次

表 目 次

第1章　序論

1.1　研究の背景と目的

近年の高速なネットワークの普及によってマルチメディアコンテンツをネットワーク上で扱う事が容易になった．これにより著作権に対する認識も従来と比較し重要視されるようになってきた．RIAA などの音楽著作権保護団体のデータからも著作権の保護が問題になっている．著作権を保護するための，もっとも簡単な方法はコピーを不可能にする事である．しかしながら，その方法は高い互換性をもったメディアの場合現実的ではないし，デジタルデータとしての媒体を考慮すると非常に困難が予想される．実際に被害事例からの分析によって，デジタルデータからの著作権侵害が最も起こりやすく問題視されつつある事が理解できる．媒体がデジタルデータの場合はコピーが容易に可能な点や，一見して内容が一意に把握できない事から，が原因であるといわれている．どちらの場合でも対策が行われてはいるが，ハードウェア／ソフトウェア的にコピーを不可能にする事は過去の対策例からも難しい．昨今では，デジタルデータの内容を参照し把握するといった新しい見地からの対策が研究されている．

1.1.1　デジタル著作権管理技術（Digital Rights Management）

デジタルデータの内容から一意のコンテンツデータを特定する事が可能な技術のひとつにデジタル著作権管理技術（Digital Rights Management）がある．電子透かし技術や識別子を用いる方法など，さまざまな技術が研究されている．代表的なものを以下に列挙した．

DCD（識別子を用いる技術）

コンテンツ自体への事前処理が必要である．一意に判別できる識別子をデータとして，コンテンツに埋め込む事が最も基本的な技術である．また，識別子によってコンテンツを一意に識別出来なければならないので，埋め込む識別子としてのデータは比較的大きなものとなる．データの構造上埋め込む場所はコンテンツデータ信号以外の場所に限定される．よって，コンテンツとの結合の強固性が他の方法に比べ弱くなってしまう等の特徴がある．この場合コンテンツとの結合の強固性が弱めになるのでコンテンツ管理上としてはやや制約が弱くなる．コンテンツ識別時には，識別子を読み出すだけでよい．

電子透かし技術

紙媒体の透かし技術を電子データに応用したものである．当然ながら，コンテンツへの事前処理が必要になる．透かしデータだけを埋め込むので，埋め込むためのデータサイズは比較的小さくて済む．透かしの構造上，コンテンツデータを一体となることが多いので，コンテンツとの結合の強固性が強くなる，などの特徴がある．しかしながらコンテンツとの結合の強固性が強いという事はコンテンツ管理上として制約が強くなるものの，コンテンツデータとノイズの分離の不可能性も強い事を表している．コンテンツ識別時には，透かしデータを読み出すだけでよい．

フィンガープリント技術

コンテンツデータの信号自体に特徴量を見出し，これをコンテンツ ID とするため，データを埋め込むなどの必要性がない．この方法ではコンテンツへの事前処理が不要となる．しかし，コンテンツデータの信号自体に特徴量を見出す場合，特徴量を検出する計算量や外付け回路などが要求される．多くの場合，上記 2 方法よりも多くの演算や外付け検出回路が必要になる．

以上のナカから本研究では特にフィンガープリント技術に着目し，コンテンツ流通時に著作権管理を行う手法を考える．

1.2 フィンガープリント技術の特徴と問題点

フィンガープリント技術の概要は前節で述べたとおりである．フィンガープリント情報を検出するフィンガープリント技術には様々な手法が存在する．ここでは Haitsuma ら [1] の手法を用いることとした．Haitsuma らはオーディオコンテンツを対象とし，ソフトウェアによって特定の演算をコンテンツデータに施す事を行った．この結果，ある 2 つの楽曲を比較した場合，ある一定の閾値 (0.35) を基準として，その 2 楽曲が同一のものか，そうではないか，を判別できる事を実験的に証明した．礒永ら [2] はこれをハードウェアでコンテンツ ID 検出までのフローを適応的に改善することを行った．礒永の提唱するシステムでは，比較検証できる楽曲数はハードウェア制約によって 100 曲と制限されており，楽曲判定までには少なくとも 314msec が必要である．

これは 320kbps でサンプリングした 3 分程度の曲が 100Mbps でリンクしている回線を通過する時間，580msec と比較するとほぼ同じである．現在ではインターネットなどの普及により比較的高速な回線が一般的になりつつある．これによってより高速な検出と大容量の検索が要求されるはずである．

このように，フィンガープリント技術はコンテンツデータに加工が不要なので，エンドユーザーサイドでの恩恵は多きいものの，システム運用などに関しては通常の方法よりもよりコストがかかる事が考えられる．より具体的な問題としては，1 つのコンテンツ ID

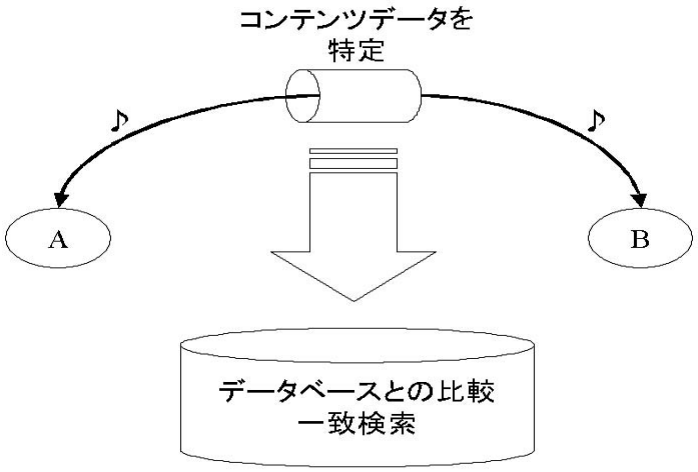

図 1.1: フィンガープリント技術を用いた楽曲の識別システム

を検出完了までの演算処理時間と，コンテンツデータを扱える容量にあるといえる．

1.3 本研究で用いるシステムの概要

本研究で用いるシステムの概要は以下のようになる．

基本的にはある合意の下で A-B 間で送受信されるデータに対し，フィンガープリント情報を検出する．これをコンテンツデータベースとの比較により，コンテンツの識別を行う．商用のある種のコンテンツデータベースとしては apple 社の ITunes における MusicSTORE や HMV 社の InternetStore，Yamaha 社における MySound など，各社各種に存在する．その収録曲数も 70 万曲〜170 万曲以上とバリエーションが豊富なのが現状である．本研究ではコンテンツデータベースの収録曲数をおよそ標準的な 100 万曲程度とし，下図のようなシステムを用いるとする．

本研究では流通する楽曲コンテンツには流行などの作用によって一時的な安定性があることに着目しデータベースの一部をハードウェア化する事で，高速な比較・検出をする事を目指した．また，現実面に即した形で，ハードウェア上で扱える楽曲数を増やし，より大容量のコンテンツを扱えるようにした．

1.4 本文の構成

第 1 章では昨今の著作権事情を踏まえ，コンテンツ管理技術の要求とその手法，ならびに問題点を述べる．この後先行研究を紹介し，本研究における背景と目的を述べる．
第 2 章ではオーディオフィンガープリントシステムに関して述べる．フィンガープリント

システムの検証とソフトウェアシミュレーションにおける結果，現実的な流通コンテンツの分析，コンテンツの流動性と高速な検索手法に関して論述する．
第 3 章では提案手法の実装に関して述べる．FPGA による判定回路の構造，ソフトウェアにおけるアルゴリズム，ハイブリッド構成における判定システムに関連して述べる．
第 4 章では評価に関して述べる．フィンガープリント検索回路の回路量と処理速度，ソフトウェアにおけるフィンガープリント検索の処理速度，システム全体の処理速度，性能評価を行う．
第五章では結論に関して述べる結論としては，コンテンツデータを扱う本システムにおいて，入力信号に依存する特性を利用する場合，ソフトウェア，ハードウェアそれぞれ単体での処理ではなく，システム全体としてハイブリッド構成にする優位性を主張する．

第2章　オーディオフィンガープリントシステムとその分析

2.1　はじめに

前節ではフィンガープリント技術に関して基本的な事項を述べた．フィンガープリントシステムは入力データとしてコンテンツ自体を用い，コンテンツ ID を検出する事でコンテンツを一意に識別することが可能である．本稿では特にコンテンツとしてオーディオコンテンツファイル（WAV ファイルや mp3 ファイル）を用いる．本文ではより具体的にフィンガープリント検出・照合システムの詳細を述べる．詳細としては照合の大容量化，つまりデータベースサイズの増加に関する従来研究を紹介する．また研究目的であるハードウェアを用いたフィンガープリント情報の高速一致検索として，FPGA デバイスを用いた情報照合の高速化手法を紹介する．実際にハードウェアで処理するアルゴリズムと同じものをプログラムし，ソフトウェアでの処理時間を算出した．これとは別に現実的なコンテンツの流通具合の分析としてオリコンチャートを参考とした，流通の偏りに関する分析も行った．

2.2　オーディオフィンガープリントシステムの処理フロー

コンテンツからコンテンツ ID を検出する方法は多数存在するが，本稿では Philips 公募における Haitsma ら [1] のオーディオフィンガープリントシステムを用いた．各オーディオファイルにおける時間，周波数上の特徴をソフトウェア処理によって得てコンテンツ ID とする手法である．得られたコンテンツ ID は源信号に印加される重畳ノイズやエフェクト効果に対し頑健性の高い事が実験的に証明されている．コンテンツ ID としては 8192 ビット（1KB）のバイナリファイルが得られるものとされており，これを被参照データベースとの比較によって一意に識別する事が可能となる．つまり**コンテンツファイルからのコンテンツ ID の検出**という部分と**得られたコンテンツ ID を被参照データベースとの比較する**と言う部分によってこのシステムは成立し，結果としてコンテンツが識別できるものである．

2.3 従来研究

2.3.1 データベースサイズの大容量化

ここでは**データベースサイズの大容量**を行う分野での従来･関連研究を紹介する．Shresthaら [3] は被参照データベースを Peer-to-Peer ネットワーク上に配置する事で 1 万曲までのオーディオファイルの種類の増加に追従可能とした．しかしながらこの手法はクエリーのヒット率を増加させたい場合は，マネージャやワーカー数などを増やさなければならず，システム構造が大規模にならざるを得ない．また，ネットワークスピードのボトルネックなどの影響をも受けてしまう．

一方 Aizawa ら [6] によって大規模データベースにおける操作，特にレコードの照合における動向が調査されていたが，これはレコードが多数あり高度に構造化された複雑なデータベース向けの操作に関してであり，本システムではもっとバイナリ数列などの，メタな部分を扱う要求があるのでこの手法はそのままでは適用できない．この手法はデータ一つ一つに直感的に理解できる意味付けや関連事象などが内包されている場合を想定してあり，本研究におけるデータとして扱うコンテンツ ID 向きではない．何故なら Haitsma らのアルゴリズムによって計算されるコンテンツ ID はその一つ一つが頑健性が高いものであると実証されているし，直感的に理解できる意味付けや関連事象などが内包されているとコンテンツ識別技術としてのフィンガープリント技術におけるコンテンツ管理上の制約力が弱まってまうためである．つまりコンテンツ識別技術としての意味がなくなってしまうからである．

2.3.2 FPGA を用いた高速化

ここでは**FPGA を用いた高速化**を行う分野での従来・関連研究を紹介する．礒永ら [2] によって提唱されたオーディオフィンガープリント検出システムは，Haitsma ら [1] のオーディオフィンガープリント検出アルゴリズムを用いて，従来手法のボトルネックであった，コンテンツファイルからコンテンツ ID を検出する過程をハードウェア化し高速な検出を行う事を目的とした．礒永らの作成したシステムでは，楽曲判定までには少なくとも 600msec が必要であり，比較検証できる楽曲数はハードウェア制約によって 100 曲と制限されている．これはハードウェアによる搭載可能な情報の制約である．そのため，検出したコンテンツ ID の検索はハードウェア上で完結するようになっており，予めハードウェア上へのコンテンツ ID を登録しておかなければならない．この手法はハードウェア上にコンテンツ ID を少量しか搭載できない制約があり実用化には難しい．

そのほかにハードウェアを用いた検索システムとしては Takamichi ら [5] の研究があり，FPGA デバイスを用いて高速に類似画像をテンプレート検索するものである．高速化としては特定のテンプレート画像専用ハードウェアを FPGA 上に作りこむ事で実現している．システムの入力に依存するデータを予め 1 つの専用ハードウェアによって順次処理することで高速化を実現している．しかしながらこれは画像処理専用に限っており，音楽

ファイルなどのコンテンツには対応していない．システムの入力依存という意味では，この手法の考え方は本システムに適用可能である．だが，しかしこの手法では比較対象が 1 つと限られていて，システムに必要とされる機能の多数のコンテンツ ID をハードウェアで高速に比較・検索すると言う機能を実現し得ない．また，その検索・照合についても，その後行われる処理は全く異なるものになるためこの手法を適用する場合には改変が必要になる．

2.4 運用を見据えたフィンガープリントシステムの検証

実際に，システム運用を考えると 100 万曲のデータベースに対応しなければならない．礒永らの作成したシステムつまりフィンガープリント情報を検出し，比較・検索するシステムの仕様の詳細を図 2.1 と比較する．礒永らのシステムでは検出完了までに少なくても 600msec は必要であると示している．しかしながら図 2.1 の A-B 間のリンクスピードが 100Mbps で接続されている状態を想定すると，本例では 576msec の転送時間中に処理が完結することが望ましいと言える．この結果は必要十分とはいえない．

320Kbps で Sampling された MP3 データのデータ転送を例にしている．320Kbps で 3min（180sec）分のデータなので，データ量は式 2.1 で計算される．

$$Data[Mbit] = SamplingRate[bps] \times time[\mathrm{sec}] \tag{2.1}$$

こえによってデータ量は 57.6Mbit となる．また，これを 100Mbps の転送速度で転送するとするので，要する理論値時間は式 2.2 で計算される．

$$TransferTime[sec] = \frac{Data[bits]}{TransferSpeed[bps]} \tag{2.2}$$

また現実的な運用を考える場合，対象データベースは 100 万曲にも及ぶ．礒永らのシステムでは検出対象が**100 曲分**と小規模のため，現実的な運用を考えると不向きである．これはフィンガープリント情報の格納場所として，デバイスのメモリ領域を指定したためである．実際の ID 検出回路を実装した段階で，全体の 90 %以上の回路面積に余裕があることが分かっており，この部分を利用する事によって更なる検索曲数の増大を狙える可能性がある．そこで礒永らのシステムを拡張し 100 万曲のデータベースにも対応できるシステムの実現を目指す．

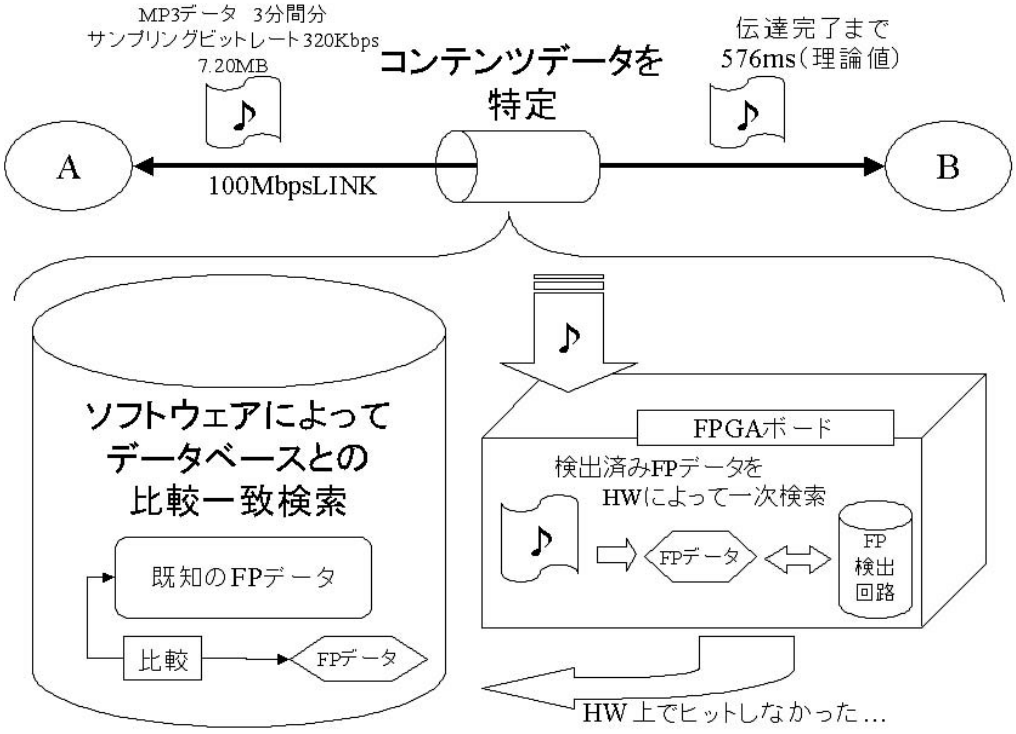

図 2.1: FP データ検出システムの概要図

2.5 ソフトウェアシミュレーションによる先行実験

本研究で用いるフィンガープリントシステムでは，最終的な目標を**得られているコンテンツ ID からコンテンツを識別する**ことにおく．コンテンツ ID を比較・検索した結果，コンテンツを識別できないと言う事態は避けねばならない．また，ハードウェアを用いたフィンガープリント情報の高速一致検索として，ソフトウェアのみで 100 万曲分のデータベースを検索する場合，どの程度の時間がかかるのか，を知る必要がある．よってソフトウェアによる検索をシミュレーションとして行った．ここでは例としてコンテンツ ID からコンテンツを識別不能であった場合などの処理を考えてプログラムを製作した．以下に詳細な仕様と目的を示す．

2.5.1 要求仕様の詳細と主な目的

作成したソフトウェアの詳細な仕様を以下に述べる．まず，検索対象のデータベースをあるコンテンツ ID で全て検索し，それにかかる時間を計測した．具体的にはあるコンテンツ ID と選択されたデータベースエントリの 1 つを 1bit づつ比較し，8192bit 分の比較終了後に選択データベースエントリとあるコンテンツ ID の BER を計算していくものである．ここで，データベースエントリ先頭から最後尾まで一致検索を行うと，ソフトウェア上の実行なので CPU キャッシュなどの外因から，実際の検索要求と同等とはいえなくなってしまう．よって，実際に検索をしている状況に近づけるために，検索対象のデータベースエントリをランダムに要求し，あるコンテンツ ID と比較する事で検索を行った．データベースの大きさは 1000，1 万，10 万，100 万曲それぞれに対して，シミュレーションを行い，時間を計測した．またその際に，実際に処理を行っている状態でなければならないので，実際のハードウェアで処理する検索アルゴリズムを同等のものを作成し，時間計測を行った．また，実際にデータベースエントリを最大 100 万曲分用意することは困難である．ここでは最も時間がかかるケースを知りたいので，ランダムに選択されるデータベースエントリの最終エントリにあるコンテンツ ID を用意しておくことにした．これによって，各データベースサイズでのワーストケースの検索時間が算出できるためである．その他のデータベースのエントリとした変数の中には何が記述されていようとも比較・検索を 8192bit 行うので，ここではプログラム作成上の簡易さをとって，ダミーのあるコンテンツ ID をコピーするようにした．図 2.2 に概要アルゴリズムと以下に詳細なステップを示す．

ビットエラーレートとは検索対象であるあるコンテンツ ID とデータベースのあるエントリ ID とが，どの程度ビットごとに異なっているか，を示す指標である．計算方法は式 2.4 で表される．これは文献 [1] の中で Haitsuma らによって実験的に明らかにされた式である．

$$BitwiseexclusiveORData[bits] = InputID[bits] \bigoplus ReferenceID[bits] \tag{2.3}$$

$$BitErrorRate = \frac{CountofBitwiseexclusiveORData[bits]}{SizeofReferenceID} \tag{2.4}$$

- Input ID：入力される ID データ (8192bits)
- Reference ID：データベースに格納されている比較のための ID データ (8192bits)
- Size of Reference ID：被参照 ID データのサイズ (8192bits)
- Bitwise exclusive OR Data： 入力された ID データと被参照 ID データとの異なり (8192bits)
- BitErrorRate：入力された ID データと被参照 ID データとの異なりが 1ID8192bit 中どの程度を占めているかの割合

BER が小さければ，ID が同一である可能性が高くなり，BER が大きければ ID が異なっている可能性が高い．BER には 0.35 という閾値が存在することが文献 [1] の中で明らかにされている．この演算をループ中に行う事は検索スピードの大幅な低下を招く．また予め被参照 ID データのサイズが分かっているため，このことを利用し閾値の算出方法を式変形する．これを式 2.5 で表す．

$$BitErrorCount = Thresholdlevel \times sizeofReferenceID \tag{2.5}$$

- Threshold level：検出判定の閾値 (0.35)
- size of Reference ID：被参照 ID データのサイズ (8192bits)
- BitErrorCount： 入力された ID データと被参照 ID データとの異なりの閾値の数 (2867bits)

これより BitErrorCount[bit 分] = 2867 と設定する．BitErrorCount が小さければ，コンテンツ ID が同一である可能性が高くなることが分かっているので，BitErrorCount[bit 分] 以上ビットが異なっていれば，そのコンテンツ ID は選択されたデータベースエントリに記録されているコンテンツ ID とは異なるといえる．これを利用する．

1. **読み込み・データベース作成部分**
   予め用意しておいた検索したいコンテンツ ID データを読み込む.
   これとは別のダミーコンテンツ ID データも読み込み,
   検索対象となるデータベースを指定曲数分作成.
2. **検索部分**
   検索したいコンテンツ ID データを，読み込み・データベース作成部分で
   用意したデータベース終端まで探索.

(a) **ID 比較部**

i. 検索時のエントリ番号をランダムに選択し，指定する.
ii. 選択されたデータベースエントリのコンテンツ ID データの先頭 1 ビットから,
   0 と 1 のバイナリが一致しているか，否かの照合を行う.
iii. 実際にはビットシフト演算を用い 1 ビット単位で 8192bit 分比較し
   それぞれにおける 0 と 1 のバイナリ不一致数をカウント.
iv. コンテンツ ID データの形式は 32bit の 256 列分を
   1 コンテンツ ID データとして扱う.
v. このコンテンツ ID データと選択されたエントリとを 32 回のループで
   1 ビットごとに照合する.

これを 256 回繰り返して選択されたデータベース 1 エントリとの比較が終了する.
8192bit 分のそれぞれにおける 0 と 1 のバイナリが不一致数をカウントした合計を
算出する.

(b) **BitErrorCount 比較部**

i. 算出された値が BitErrorCount（2867）[bit 分] 以下かを判断.
ii. BitErrorCount（2867）以下でない場合は,
   データベースエントリに格納されているコンテンツ ID と
   検索したいコンテンツ ID は異なるものであると判断.
   コンテンツ ID の BitErrorCount 判断部を完了.
   以下の処理をスキップし，検索部分先頭に戻って次のエントリを検索.
iii. BitErrorCount（2867）以下であれば,
   直近のループ中に保存された BitErrorCount と比較・判断を行う.
   保存された BitErrorCount より少ない BitErrorCount であれば,
   よりその ID に近い可能性が高いと判断できる.
   データベースエントリの番号と現在のカウント数を更新して保存する.

iv. より少ない BitErrorCount でなければ，場合は何もしない.

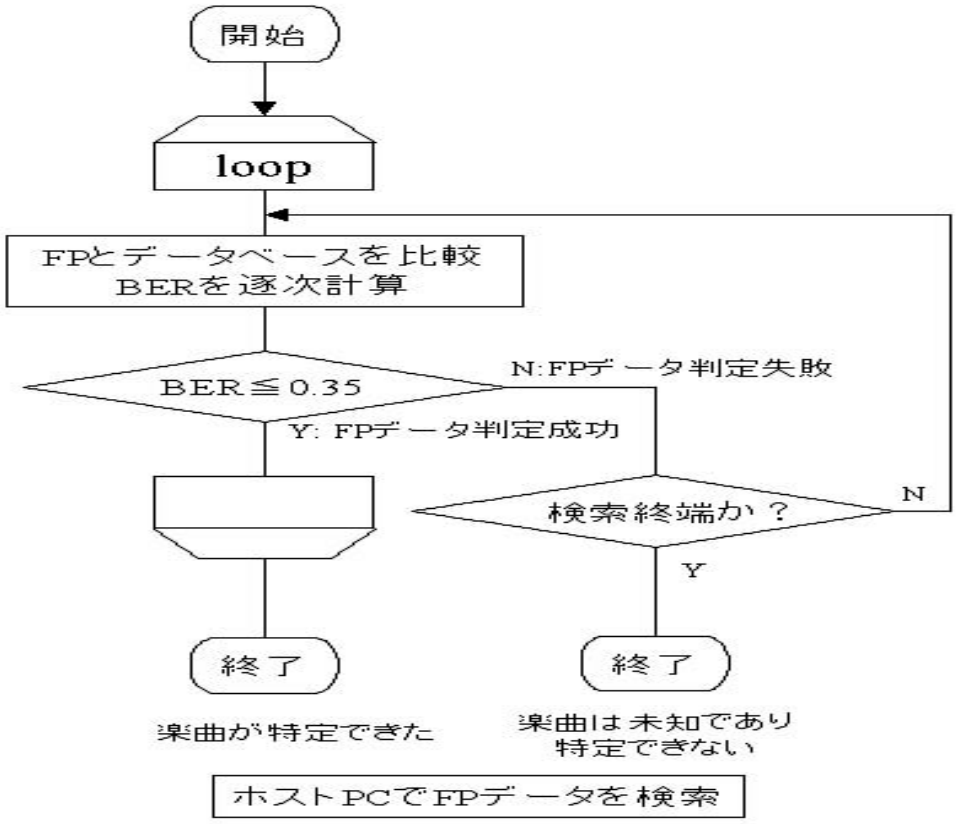

図 2.2: プログラムフロー

以上の処理で，データベース 1 エントリとの比較が終了する．
ループ部の先頭に戻りデータベースエントリの全てを検索完了するまで，
照合を行う．

3. **検索時間表示部分**
検索の部分で指定曲数分照合終了後に検索にかかった時間を表示．
最小の BitErrorCount をもつデータベースエントリ番号を表示．
同時に BitErrorRate も計算を行い表示させる．

表 2.1: シミュレーション環境

PC	Sun Blade1500
CPU	UltraSPARC III Cu 1.0GHz
搭載メモリ	512MB
OS	Sun Solaris9
開発言語	C

表 2.2: ソフトウェアによる検索時間結果

検索対象曲数	1000	10000	100000	1000000
検索時間 [sec]	0.34	3.48	34.69	353.48
メモリ消費量 [MB]	2.0	11	99	978

2.5.2 シミュレーションの環境

シミュレーションの環境は表 2.1 のとおりである．ここでは前節において解説したプログラムフローによってソフトウェアにおける検索を行った．ソフトウェアは C 言語で記述しソフトウェアに与える検索データベースのサイズを変更しながら，それぞれの検索時間とメモリの消費量を調べた．

2.5.3 結果

シミュレーションの結果を以下の表にまとめる．ハードウェア上に実現された制約のあるデータベースではないので，100 万曲分という大量のデータを扱う事が出来る．これは下表より検索対象の曲数を任意に増やせることで分かる．また同時に 1 つの ID を検索するのにかかる時間が示されている．これは msec オーダーでの転送時間中に処理が完結することが望ましいが，表より，単純な検索アルゴリズムでは大量のデータベースを検索する事に時間的な制約があることが分かる．結果としては検索対象のコンテンツ ID データベースの大きさが検索終了までにかかる時間に支配的であることが分かる．これはアルゴリズムに大きく影響するものである．今回は線形探索なので計算量のオーダーは O(n) になる．データベースの構造を工夫して与えれば 2 分探索法など各種の探索アルゴリズムが利用可能になり，理論上 O(log n) まで計算量を圧縮できる．また今回の結果はコンテンツ ID を規定のデータベースサイズ上を検索した場合にかかるワーストケースの検索時間である．当然現実の楽曲データベースを検索する場合や，ネットワーク上でやり取りされるデータを対象とすると，様々な局面や瞬間から，偏りが発生することがある．この偏りについて次節で詳しく述べる．

2.6 コンテンツ流通の分析

本節ではコンテンツの出現の偏りについての基礎的な評価を行う.

2.6.1 CD セールス数におけるオリコンチャートからの分析

シミュレーションの過程とシステムの予備調査として，実際に流通しているコンテンツファイルの流通量として妥当であると思われる，相関関係の認められるオリコン調べ CD セールス数の調査を行った．被検索対象データベースの大きさが単純なアルゴリズムでの検索時間に支配的であることが前節では示された．しかしながら，この被参照データベースの内容は当然ながら流動的であり，ある程度のスパンをもって入れ替わりが必要とされる．この入れ替わりに関しての最もわかりやすい要因は**流行**である．oricon 統計 (1 月分) によれば国内の CD（アルバム）のセールス数は 1 位から 30 位までのうち，この月のトータルセールス数のおよそ四割以上が上位 1 位から 5 位までのセールスによって占められている．この時期におけるアルバムのセールス数と言うデータは一瞬の「流行」の過程に過ぎない．常に流行は変動しており，この変化を予想する事は難しい．別の観点から見て，アーティストが売り出したあるアルバムのセールス数は（例外もあるが）リリース日からの時間経過にしたがって低下していく事が知られている．一般的なデータを参照する必要のあるシステムではこの表 2.3 を動的に入れ替えたり，予測したりして効率的な動作をする事を目的としている.

2.6.2 コンテンツ量と高速な検索の関係

ここでは，シミュレーションなどの結果からわかるコンテンツとしての楽曲ファイルを扱う量と，高速な検索を実現する方法を論ずる．本研究で用いるフィンガープリント技術の特徴と問題点として，ソフトウェアのみでこれを実現した場合，前節までの結果からコンテンツを扱う量としては十分な楽曲数が実現可能なことが分かる．しかしながら，この十分な楽曲数も単純アルゴリズムにおける検索では充分に高速な検索を行えているとはいえない．ハードウェア化のアプローチでは，高速な検索を実行できているものの，対象の楽曲ファイル数をハードウェアの制約上増やす事が出来ない点が問題になり得た．またコンテンツの分析を経て，ある一瞬において，いくつかの要因の下で検索対象の楽曲ファイルが固定化する等の減少が確認できた．これは「流行」の特徴とも言える．本研究では，このようなソフトウェア実現の特徴である柔軟性と，ハードウェア化の恩恵である高速な検索の双方の動作を踏まえて，図 2.4 のようなフローチャートを考案した．ソフトウェア実現の最大の恩恵は大量の楽曲ファイルを扱える事にある．また，予め流行の楽曲の判定回路をハードウェア化しておく事で瞬時にハードウェアによる FP データの判別が可能となり高速な動作が実現できる．各処理部における時間は図 2.4 に示すとおりである．前節の式 2.2 で計算される結果，576ms 以内に転送が完了するとすると，礒永らの構築したシ

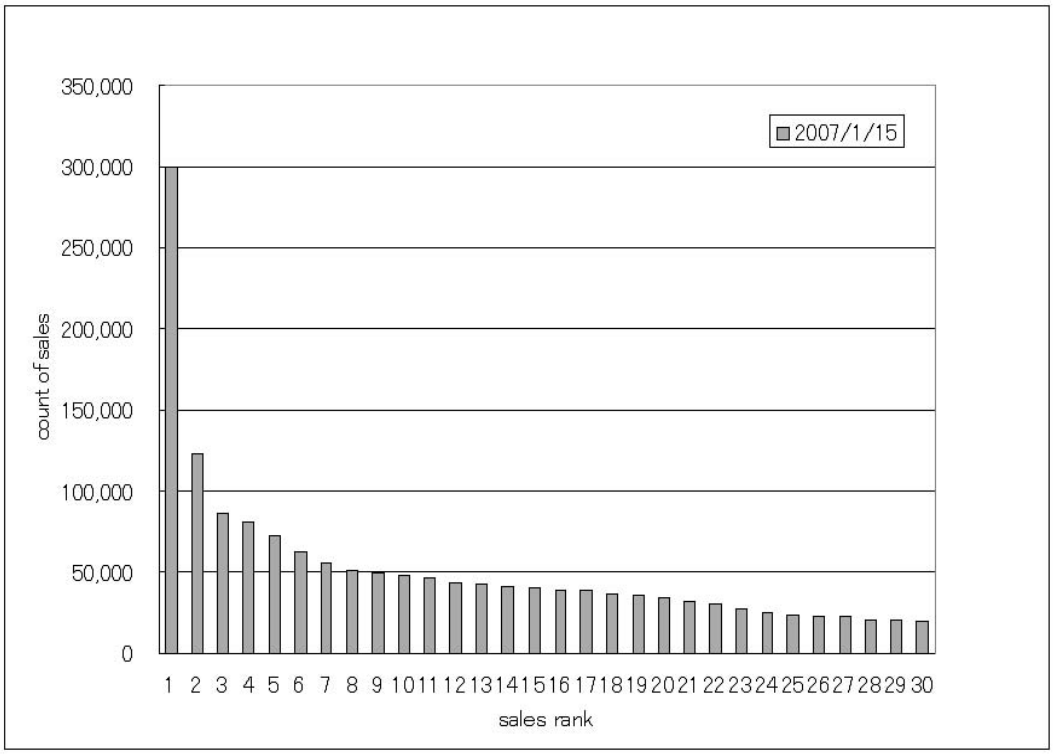

図 2.3: オリコンチャートの順位と CD アルバムのセールス数のグラフ（2007 年 1 月分）

ステムでは検出までに 314ms 必要な事が文献 [2] からわかっている．これにより，検出までの時間的余裕は 250ms 程度である事が分かる．しかしながらこれはソフトウェアとのハイブリッド動作を前提とした場合，ソフトウェアの検索時間も含まれる．次節では限定されるハードウェア面積に対し，どのように対象の楽曲検出回路を選定するかについて述べる．

2.6.3 コンテンツの流動性

提案手法としては，楽曲ファイルを扱う量と，高速な検索を実現するにあたってハードウェアとソフトウェアのハイブリッド構成を提案した．当然ながら，ハードウェア上に搭載される楽曲ファイル検出器は流行と言う要因の変化に伴ってある程度のスパンで変更可能でなければならない．この問題に対して，本提案ではターゲットデバイスとして，リコンフィギャラブルなチップを用いる事で，流行の変化時などに固定化した回路を書き換えることを可能とした．コンテンツの分析における「流行」はデータに規則性はないものの，総じてリリース直後に爆発的な売れ行きを見せ，これが多少の期間持続する，といったものであった．この「流行の期間」ではそれだけ多くの人が「流行」の楽曲を参照していることが容易に想像できる．単位時間中に参照されるデータがある程度固定化し，さら

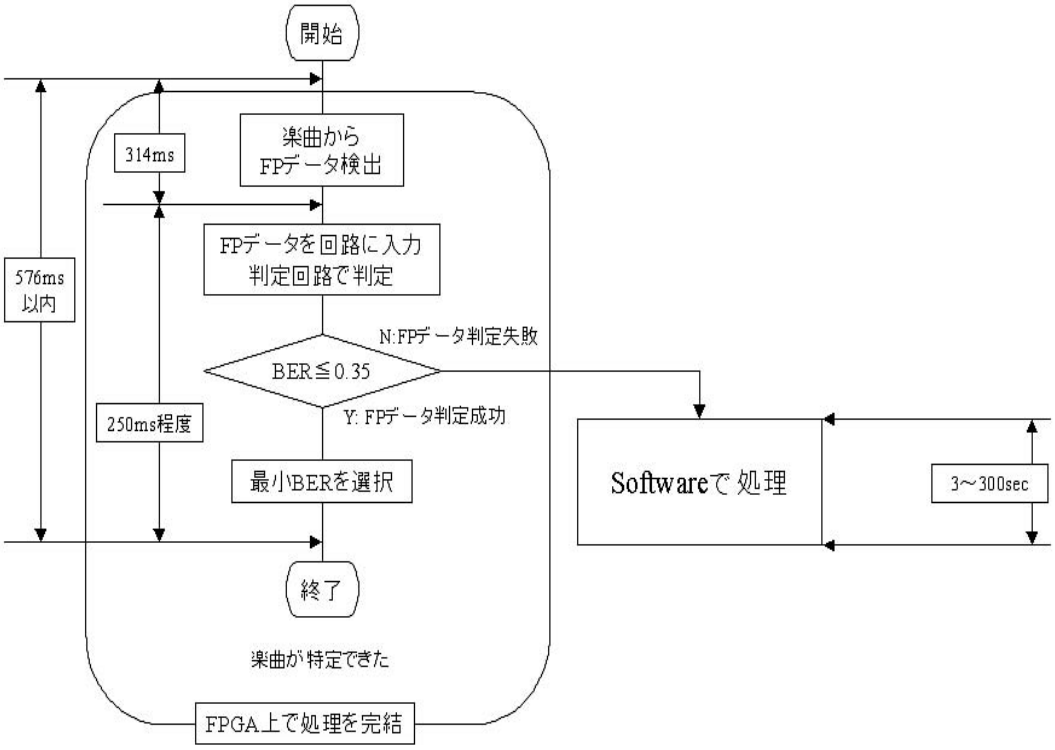

図 2.4: 提案システム概要図

に参照されるデータの頻度は高い，と言う部分に本提案の着眼点がある．本提案の処理の流れは図 2.5 のようになる．参照頻度が高いので検索時間の短縮を狙える．

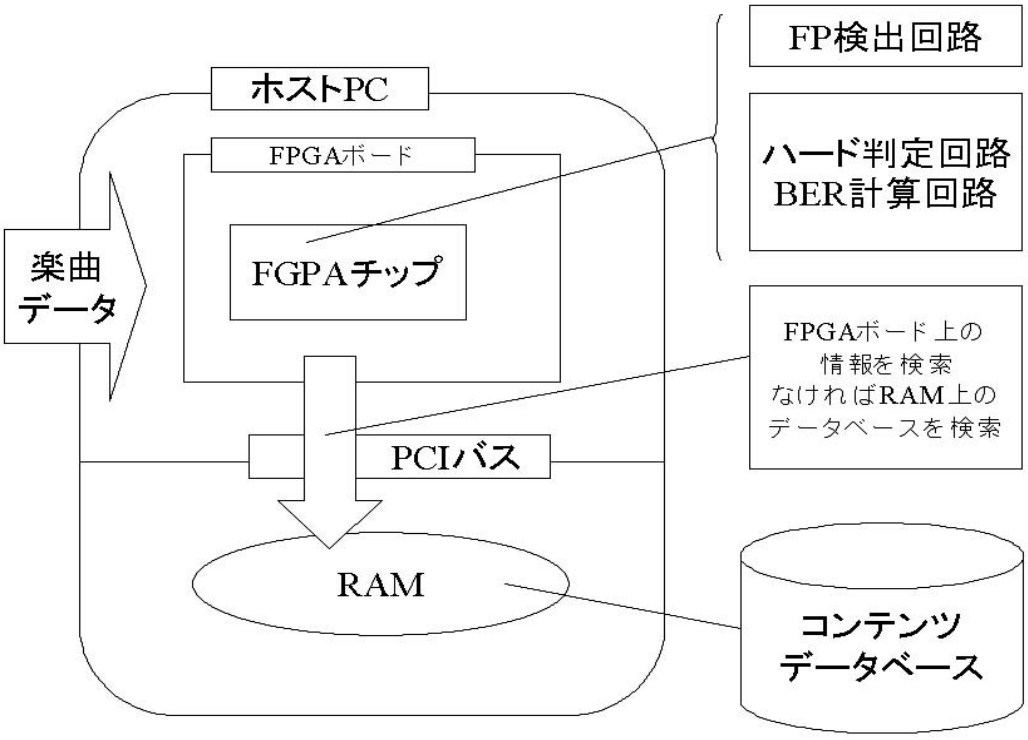

図 2.5: コンテンツ ID の処理フロー

表 2.3: 人気楽曲の順位と CD アルバムのセールス数の関係（2007 年 1 月分オリコン調べ）

2007 年 1 月 15 日	セールス数（枚）
1 位	300032
2 位	122723
3 位	86477
4 位	80441
5 位	72701
6 位	62965
7 位	55765
8 位	51521
9 位	49982
10 位	48202
11 位	46200
12 位	43522
13 位	42932
14 位	41432
15 位	40763
16 位	38546
17 位	38310
18 位	36727
19 位	36071
20 位	34238
21 位	32025
22 位	30752
23 位	27195
24 位	24771
25 位	23856
26 位	22921
27 位	22681
28 位	20772
29 位	20470
30 位	20088

2.6.4 検索システムのヒット率

本研究ではハードウェア化したフィンガープリント検出器から生成されるデータ列を利用する．礒永ら（論文引用）の研究における被参照データベースはハードウェア上の制限から少なく設定されており，現実的ではない．フィンガープリント技術の問題点はひとえにデータ読み込みのオーバヘッドであり，ソフトウェアシミュレーションによっても，この検索数の単純増加が検索時間に大きく関係する事が分かる．実際に図 2.1 の伝達猶予時間内に，ソフトウェアでの検索を行う事は難しい．

実際の検索要求に関するヒット率の試算を実際のヒットチャートから示す．アクセス率として，ヒットチャートの様子が理論的忠実に再現されるとするならば，各順位の検索リクエスト率に準じて分布する．これを式で表せば式 2.6 となる．

$$Pr(n) = \frac{s(n)}{\sum_{n=1}^{N} s(n)} \tag{2.6}$$

- n：ヒットチャートの順位
- N：ヒットチャートの下限
- s(n)：チャートの順位 n におけるセールス数
- Pr(n)：チャートの順位 n におけるヒット率

図 2.6 は 2007 年 3 月の初週における 1 位から 300 位までのセールス数から換算したものである．グラフから分かるように，楽曲の 30 位分をハードウェアで判定すると，ヒット率は全体のおよそ 6 割，100 位までででも 8 割をカバーする事になる．ヒット率における時間換算は下表のようになるが，これにおいて図 2.1 に示される目的とする処理時間を提案するシステム図 2.4 によって達成する事が最終的なゴールとなる．

2.7 まとめ

本節までにおいてオーディオフィンガープリントシステムで必要な機能の詳細を分析した．ここから検索動作で許される検索時間は 250ms 程度となる事がわかった．以降，検索を行う目標時間を 250ms とする．ここで必要とされる機能は単調な処理であり，一部をハードウェア化する事で高速化可能である．しかしながら，問題点はハードウェア上に大量のデータベースを用意できない事，さらにその内容が書き換わる事などにあり，固定的なハードウェアのみでシステムを構成すると高いパフォーマンスを維持することが困難になる．本研究ではこれを静的に再構成可能な FPGA を用いることで柔軟なシステム構成を狙った．また，書き換わるタイミングは一般的にオーディオコンテンツだと流行などの要素にとらわれやすいことなどもあり，これを書換可能な FPGA を有効活用することで問題点を解決出来ると考えられる．

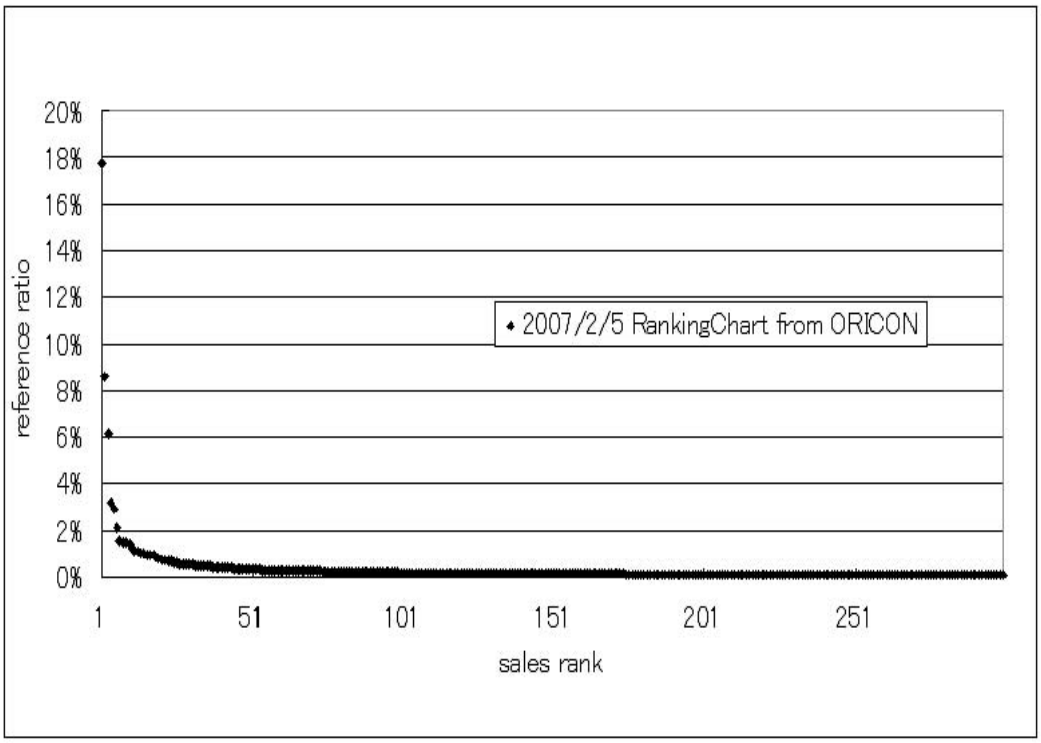

図 2.6: 単位時間当たりの楽曲順位ごとの参照頻度

従来研究としてある手法を分析し，それぞれの問題点を明らかにした．また，ハードウェアとソフトウェアそれぞれのボトルネックとなりうる部分を調べ，それらに対するアプローチを検討した．

1. ハードウェアでは高速に検索を行う事が出来るが検索対象が限定されてある．
2. また，ソフトウェアでは広大なデータベースを検索できるものの，検索速度は低速であることが問題となりうる．

ここでは Aizawa ら，Takamichi らの手法からヒントを得て，音楽ファイルを対象としたフィンガープリント検索システム全体の構成としてハードウェアとソフトウェアのハイブリッド構成を提案する．その前段階として，そのフィンガープリントシステムについての検証として，検索システム全体像を検討，ソフトウェアによる検索のシミュレーション，実際のコンテンツ流通の分析，を行った後，本システムへ対するハイブリッド構成の検証を行う．

第3章　提案するハイブリッド構成フィンガープリントシステム

3.1　はじめに

前章でソフトウェアによるシミュレーションと人気チャートとセールス数の関係から，フィンガープリントシステムの高速化に FPGA による柔軟な回路設計の面からアプローチの可能性を論述した．本稿でその有効性を主張する提案手法の実装方法について本章では述べる．ここではその具体的な方法を紹介する．

3.2　ハイブリッド構成における判定システム

ハイブリッド構成として，ソフトウェアとハードウェアの協調動作を提案する．具体的にはソフトウェア上で処理を行う上で最頻度の要求データを FPGA ハードウェア上に検出回路を作りこむ事である．これによって楽曲 ID データを FPGA 上のハードウェアで高速に検出する事が出来る．また高速な検出が可能な点を効果的に利用するために，要求回数の多い最頻度の楽曲 ID に絞ってハードウェア上に実装する．流行の変化などに柔軟に対応する目的で，ある程度の時間の経過点で任意のタイミングからハードウェア上に配置した検出回路を変更する事ができる．被参照データベースサイズがたとえば図 2.6 のように 300 曲分だとすると，150 曲エントリ分をハードウェアで検出が可能になる．これは 2007 年 2 月 12〜2 月 18 日のデータにおいて全体に占める割合の 88 %程度になる．前章の式 2.2 で計算される結果，平均検索時間を 570msec と定める．ハードウェア部分でのフィンガープリント抽出部で 314msec かかる事が礒永らによって文献 [2] で明らかにされている．これを含めれば実質的なハードウェア回路のみの検索余裕時間は 250msec 程度となる事が前章までの図??までで明らかになっている．

また，これとは別にソフトウェアでの処理時間は先行実験により時間がかかってしまうことが分かる．図 3.1 にあるように先行実験で用いたプログラムをそのまま用いると 10 万曲の検索時間は 300sec 程度と考えられる．この 2 つの条件を考えれば，ハードウェア・ソフトウェアでの分岐の回数が直感的に総検索時間を決めることが分かる．

実際のデータでの検索時では繰り返し検索要求が入力されるので，平均検索時間としての結果は少し異なったものになる．ここで，ハードウェア部分での識別確率をヒット率 Pr とするして，コンテンツ識別完了までの時間は下式で表現される．ここで，この平均

表 3.1: 実装環境

FPGA ボード	RC2000
搭載 FPGA チップ	XC2V6000
接続形態	PCI-Bus
ホスト PC の CPU スペック	AMD Athlon64 3.0GHz
ホスト PC のメモリ容量	1GB
ホスト PC の OS	WindowsXP Pro(32bit)
開発環境	Xilinx-ISE6.3.03i
開発言語	VHDL

検索時間を 250ms 以内に抑えたい．この場合，システムは一体となって動作するのでハードウェアで識別できなかったコンテンツ ID は必然的にソフトウェアで処理する事となる．これを踏まえて式を書き直すと下のようになる．

$$\text{平均検索時間} = \frac{\text{検索時間}}{\text{総検索要求回数}} \tag{3.1}$$

検索要求のあった楽曲を全て判定できたと仮定すると

$$\text{総検索要求回数} = HW \text{ での検索回数} + SW \text{ での検索回数} \tag{3.2}$$

$$\text{総検索時間} = HW \text{ での総検索時間} + SW \text{ での総検索時間} \tag{3.3}$$

$$HW \text{ での総検索時間} = HW \text{ での検索時間} \times HW \text{ での検索回数} \tag{3.4}$$

$$SW \text{ での総検索時間} = (HW \text{ での検索時間} + SW \text{ での検索時間}) \times SW \text{ での検索回数} \tag{3.5}$$

またここで，ハードウェアでの識別確率を Pr とすると，この定義はソフトウェア検索での平均的なコンテンツ識別完了までの時間は下式で表現される．

$$\text{ハードウェアでの識別確率 } Pr = \frac{HW \text{ での検索回数}}{\text{総検索要求回数}} \tag{3.6}$$

3.3 実装環境

本研究で使用する FPGA デバイスおよび実装環境を以下に示す．

評価環境としてホスト PC は Athlon64 3.0GHz，メモリ 1GB，OS は WindowsXP ProfessionalEdition を利用する．FPGA ボードとホスト PC の接続は図 3.2 のようになっている．ここでは PCI バスを解しているが，PCI バスのクロック周期は 33MHz となっている．利用するハードウェアはセロクシカの FPGA の RC2000 ボードを利用した．これは，Xilinx 社の XC2V6000-4 の FPGA を搭載しており，さらに外部メモリとして SRAM を 4Mbyte を 6 バンク接続している．これらを用いて提案手法であるハイブリッド処理の実装を行った．

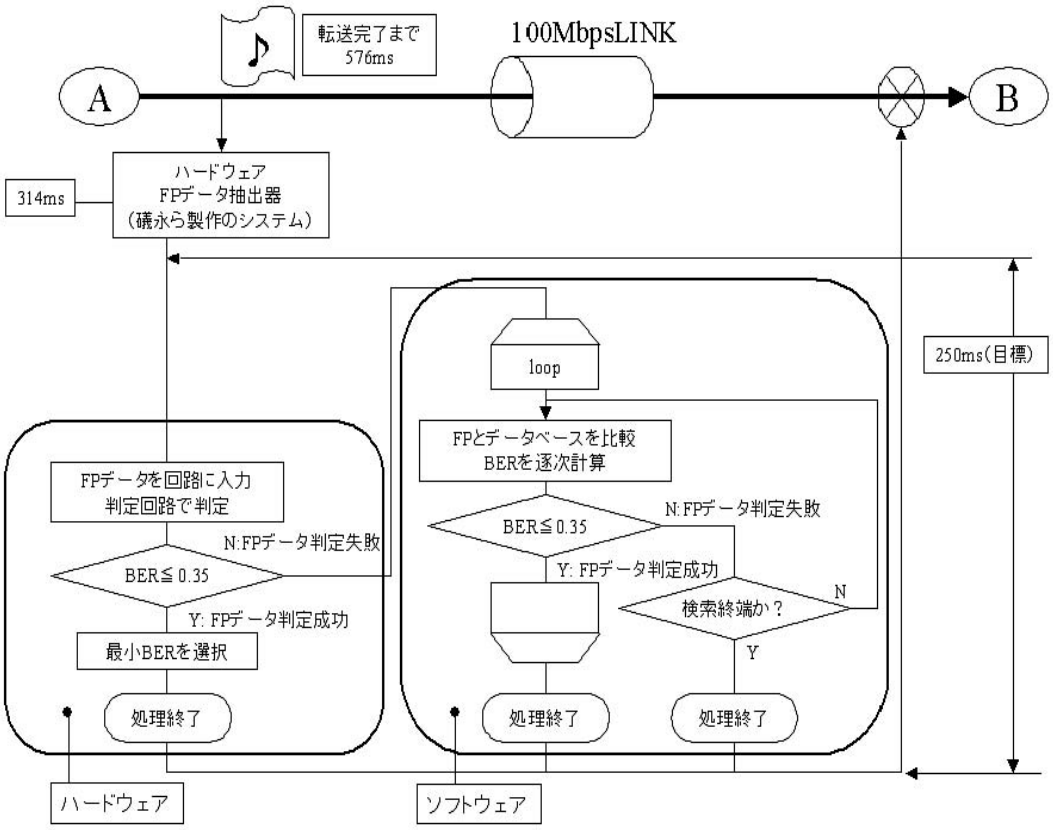

図 3.1: ハイブリッド構成における判定システム概略図

3.4 ソフトウェアにおける判定アルゴリズム

本節で述べるソフトウェア実現におけるアルゴリズムは，ハードウェアで識別できなかったコンテンツ ID を確保する事に目的がある．システムの各処理部における流れは図 4.4 に示されている．このアルゴリズム自体はソフトウェアシミュレーションで用いたものと同じである．この場合においてソフトウェア単体の動作はシステム要求を充分に満たすことができない．分岐の回数が直感的に総検索時間を決めることが分かる．検索時間を少なくするためにはソフトウェア処理に分岐する回数を減らす工夫が必要である．

ソフトウェア検索での平均的なコンテンツ識別完了までの時間と検索試行回数との関係は下式で表現される．

$$SW\text{での平均検索時間} = \frac{(SW\text{での検索試行回数} \times \text{ソフトウェアでの単位検索時間})}{\text{総検索試行回数}} \tag{3.7}$$

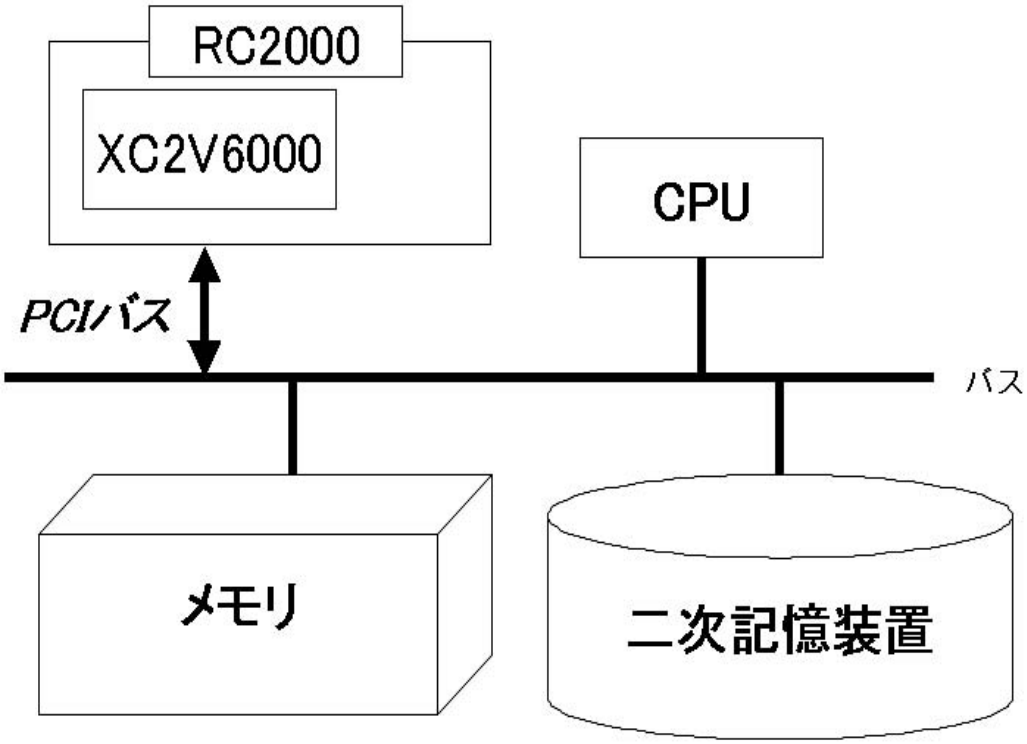

図 3.2: 実装環境の接続図

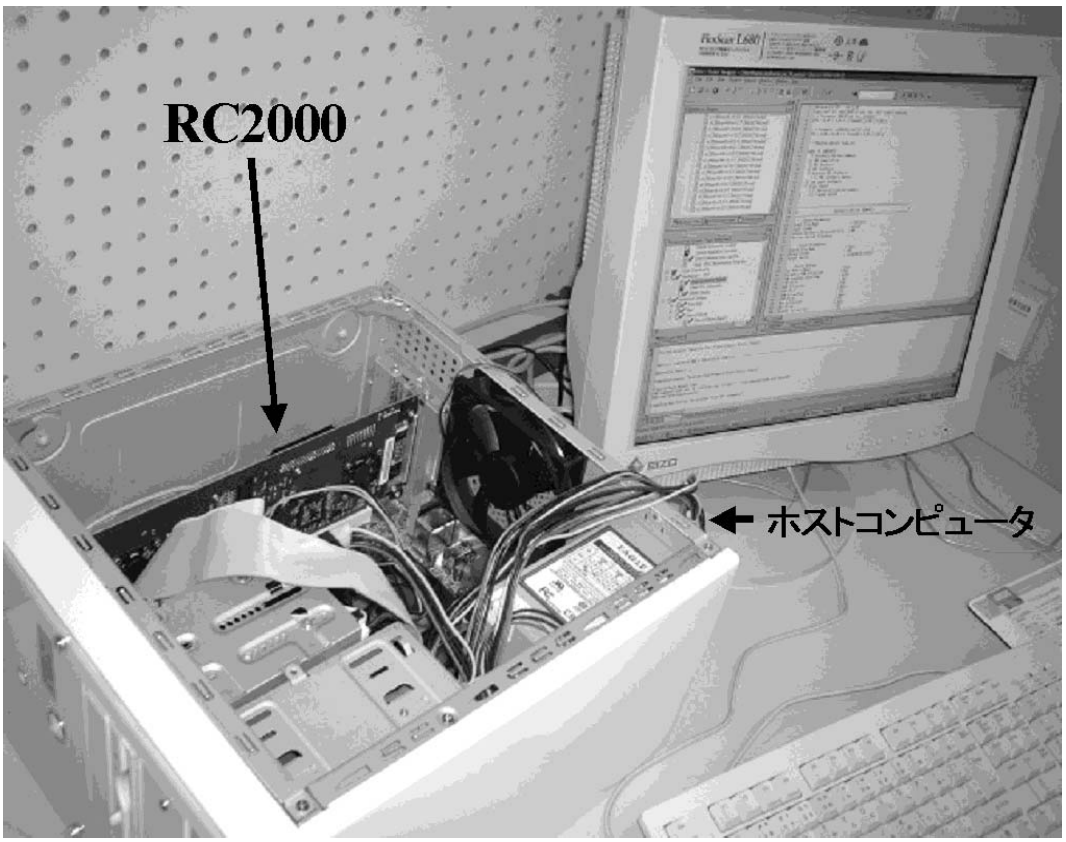

図 3.3: 実際の実装環境

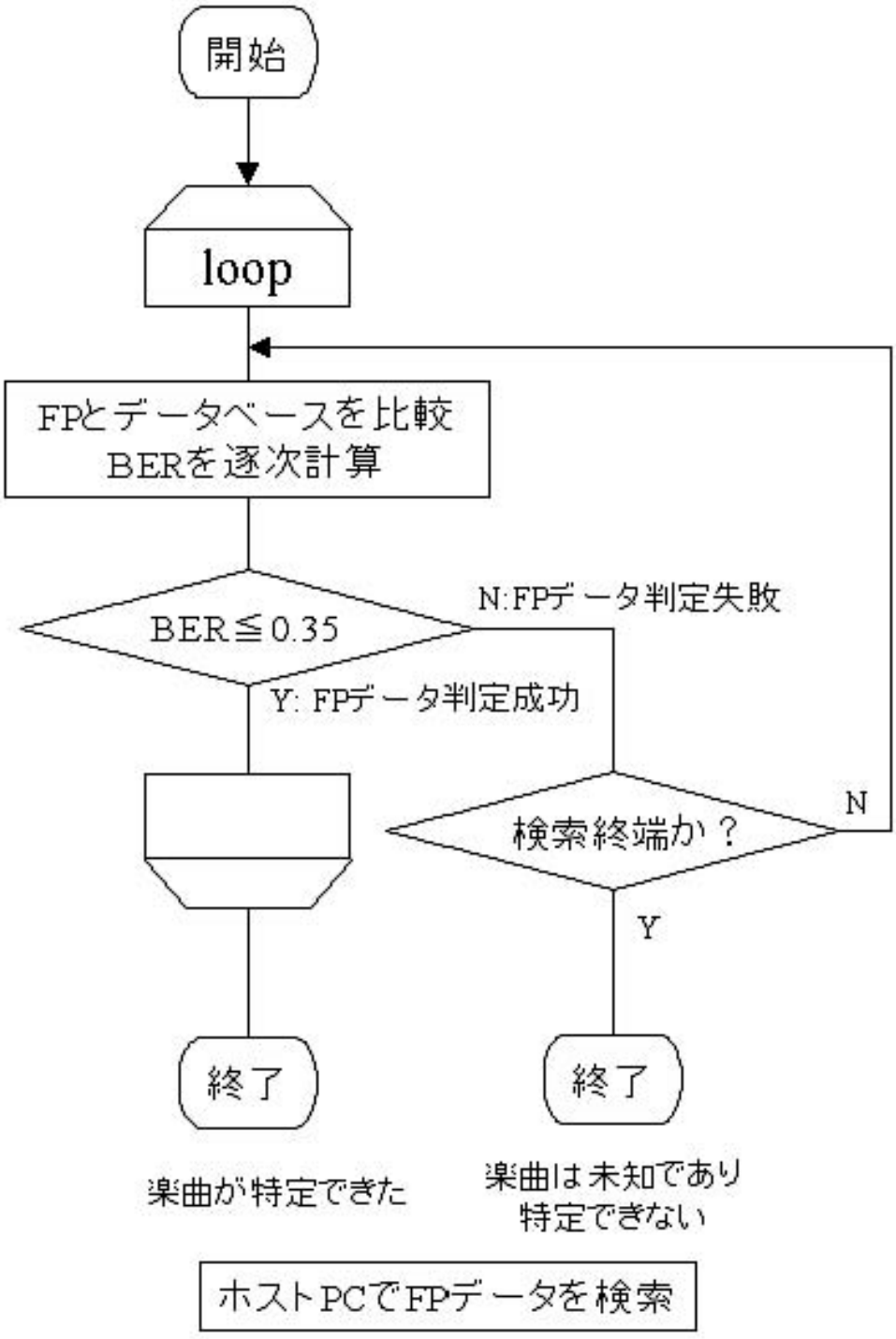

図 3.4: ソフトウェア構成における判定システム概略図

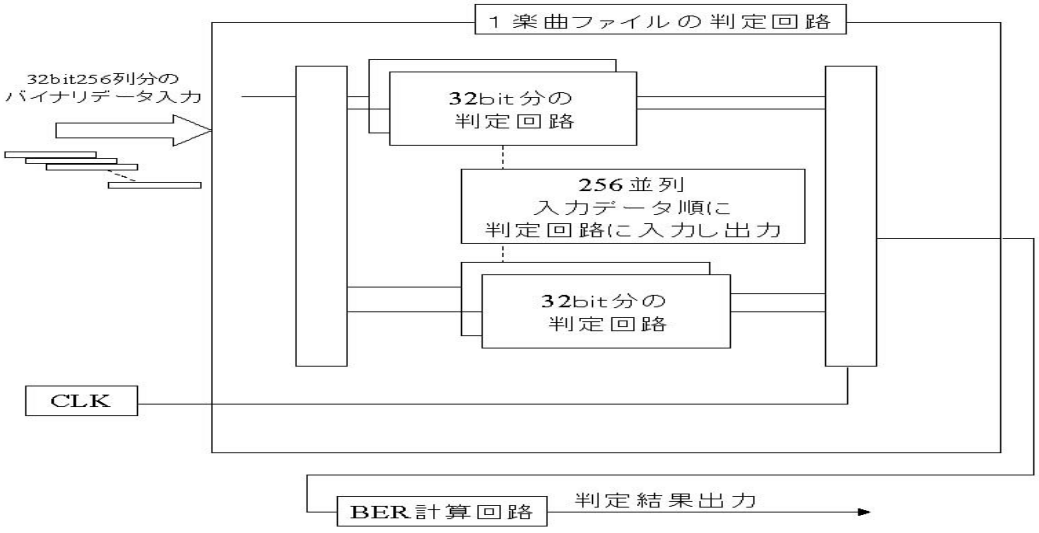

図 3.5: システムの機能ブロックと処理概要

表 3.2: 消費回路量の削減具合

パラメタ	slice	4LUTs	bonded IOBs
従来手法	18	32	96
提案手法	9	15	64

3.5 ハードウェアによる判定回路

本章では提案手法におけるハードウェア部分の実装を行う．本提案システムでは概要として図 3.5 を採用していた．楽曲判定部分と BER の計算部分に分かれる．実際に入力されるコンテンツフィンガープリント情報は単位サイクル当たり 32bit であり，これを 256 回演算が行われて 8192 ビット分の出力となる．これを並列に計算し，BER のの最小を選択肢さらにそれが閾値 (0.35) 以下であるかどうかを判定し，ハードウェア回路上での検出が終了となる．具体的には検出したい楽曲フィンガープリントデータと 1 ビットごとの比較をし，異なっているビットの総数を計算した上で，BER を計算する．以下，モジュールの詳細を示してゆく．楽曲の判定回路部分における実装の概要を示す．本提案では予め検出する楽曲をハードウェア上に作りこんでおく，とした．このため，比較対象のデータと通常の検出で用いられる比較器を使用しない．通常の比較器を用いる場合，双方に入力されたデータが同値か，そうでないかを判定するものであった．これを本提案における，ハードウェア上の判定回路として作成する場合，比較対象のフィンガープリントデータが既知であることに着目した．基本的な考え方を図 3.6 に示す．

ここでは図 3.6 でどの程度消費回路量を削減できているかをあらわす．

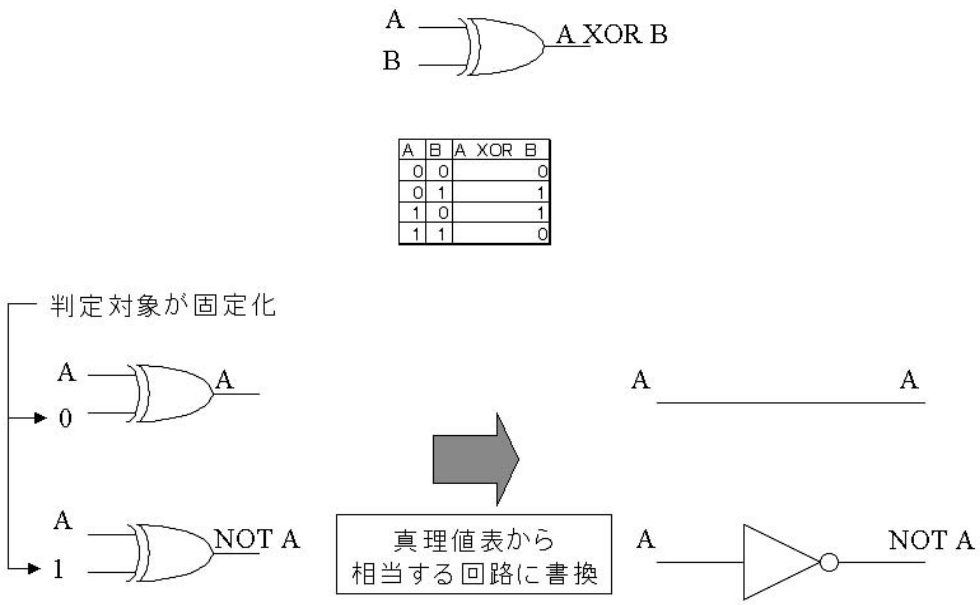

図 3.6: 判定モジュールの基本概念

3.5.1 楽曲判定回路部分

具体的な実装は図 3.7 に示す．この比較対象のフィンガープリントデータが既知であることを利用して，これを回路として予め作りこんでおく．結果的に判定回路から出力されるの 1 ビットのバイナリデータ”1”の数によって予め作成した判定回路とどの程度違うのかを計算する事が出来る．また，本モジュールでは事，通常の比較器を用いない事でゲート数の削減を図っている．ゲート数を出来るだけ抑制できれば，その分だけ判定回路を 1 つでも多く搭載する事が可能になり，並列性の向上が見込める．表 3.5.1 に回路の概要を示す.

入力 DIN 32 ビット

出力 DOUT SECT 32 ビット（n はモジュール番号）

1. 1 クロックごと DIN に 32bit の入力が行われる
2. 回路に判定したいデータを入力，異なっているかどうかをチェック
   実際には 1FP データ分の 8192bit 分を格納するために 256 回の入力が必要
3. 1 クロックごとに 32bit の入力を切り替えて 256 回の入力・判定要求に対応
4. 判定回路を通過した DIN からの入力データは DOUT SECT[n] にそれぞれ出力される

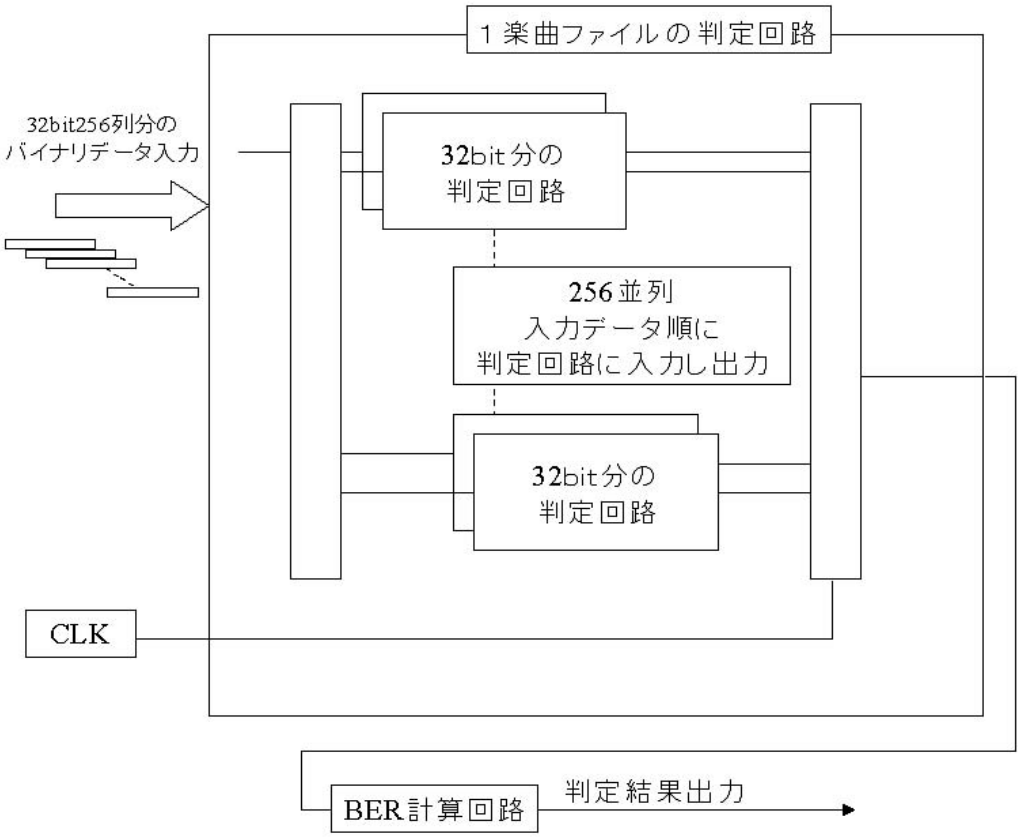

図 3.7: 楽曲判定回路の全体像

3.5.2 BER 計算回路部分

楽曲の BER 計算回路部分における実装の概要．BER 計算回路では 256 サイクルかけて出力される楽曲判定回路を用いて楽曲の比較対象のデータとどの程度ビットが違っているかと言う総数を計算する．BER の計算の概略は式 2.4 にあるように BER ＝（異なっているビット総数）／（FP データビット総数）となる．BER の閾値は Haitsuma らによって 0.35 であることが実験的に証明されている．よって，32 ビット × 256 サイクルの 8192 ビット中に 1 の数が 2867（$8192 \times 0.35 < 2868$）個以内でどの程度存在するか，を求める事を目的としている．逆に言えば 2868 個以上存在した場合，その楽曲は BER 判定条件から比較対象の楽曲とは異なると言える，ので総計カウンタの表現上限は 12 ビット以下となる．最終的な出力時に 2868 個以上存在した場合，13 ビット全てを 1 で埋める．出力としては 13 ビットの二進数表記で行い，後段の BER 最小選択判定回路へ渡す．表 3.5.2 に回路の概要を示す．具体的な実装は図 3.8 に示す．

入力 CNT DATA 13 ビット

出力 CNT SIG 13 ビット

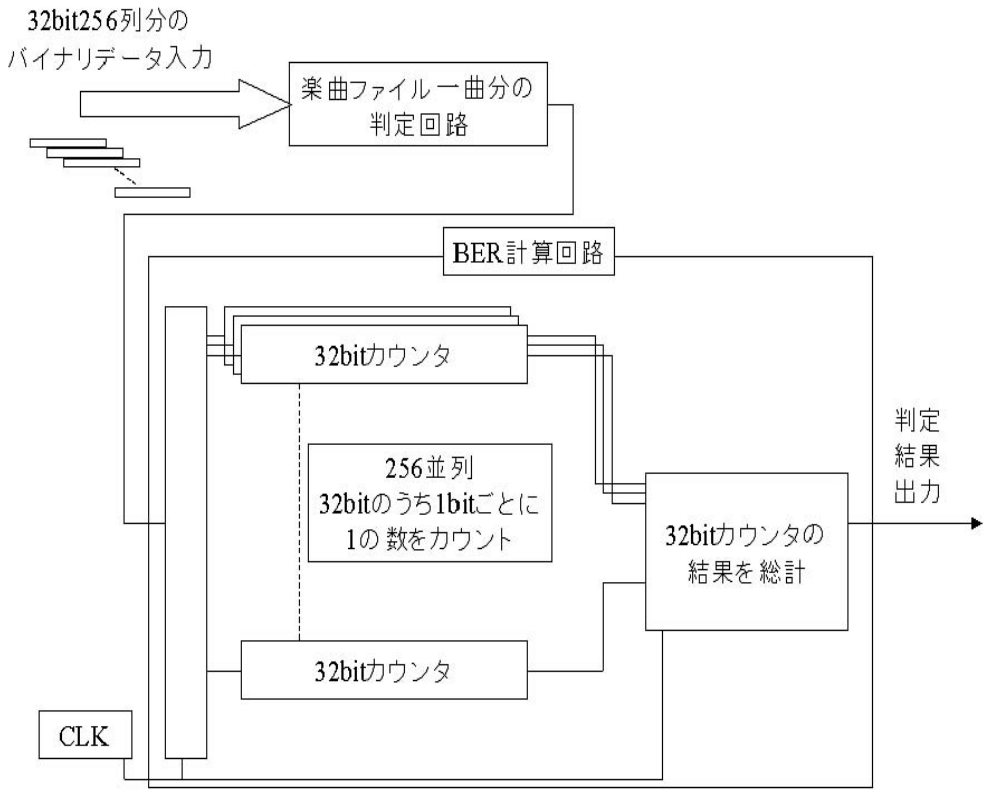

図 3.8: BER 計算回路の全体像

1. DOUT SECT[n] から入力があったデータを 1 ビットごとにチェック
2. 異なっている bit があれば CNT DATA に+1 して出力
   異なっていなければ CNT DATA には何も操作を行わない
3. CNT SIG の値に CNT DATA + CNT SIG の値を入力
4. ここまでの処理を繰り返す
5. 256 回目の DOUT SECT[n] からの入力が完了したら各判定回路の出力の CNT SIG を比較・最小を決定
6. CNT SIG の最小を出力

3.6 まとめ

オーディオフィンガープリントシステムでの大容量の楽曲を扱い，高速に検索する機能とシステムを製作した．大容量の楽曲を扱う部分ではソフトウェアでサポートし，高速な検索を行う部分ではハードウェアでそれぞれサポートをするようにした．また高速な検出が可能な点を効果的に利用するために，要求回数の多い最頻度の楽曲 ID に絞ってハードウェア実装したことで，それぞその特徴を生かしたシステムを構築する事が出来た．ハードウェアではリコンフィギャラブルデバイスである FPGA を用いることで，ハードウェア上に検出器を配置後でも変更が容易に出来る．これは流行やその他さまざまな要因からハードウェア上に配置した検出器を後から変更できる．これによって柔軟なシステム構成を達成する事が可能になる．このハイブリッドシステムによって，判別の高速化と大容量化を同時に実現する事ができた．

第4章　評価

4.1　はじめに

本章では実装した回路とソフトウェア，それぞれの性能を評価した．具体的にはハードウェアの実装における面積，速度，ソフトウェア単体での検索時の速度を評価する．次にこれらを組み合わせて構成されるシステムとしての動作を確認した．システムとしての評価としてはハイブリッドシステムとしての検索速度，時間経過におけるハードウェアでの検出確率の変化，またオリコンチャートをソースとする検出確率と従来手法である LRU 法での検出確率を比較・評価した．回路の書換にかかるオーバヘッド時間などを含めて評価を行った．

4.2　FP 検索回路の回路量と処理速度

本節では作成した回路の評価を行った．これは回路単体の評価であり，ハイブリッドシステムを提案する上ではソフトウェアの処理も重要である．ハードウェア上での検出コンテンツ ID 検出回路の評価を行った．まず検出回路がどの程度並列実装可能か，を確かめた．

消費 slice 数より，150-160 並列実装可能な事を証明．処理速度はおおむね 100MHz 前後となっている．回路の最大動作周波数も 100MHz 前後と余裕を持って動いている．1 回路で消費する slice 数が意外にも大きく並列性に限界が見えてしまった．最大動作周波数が分かっているので，回路 1 クロックの周期が式 4.1 で与えられる．

$$clockcycle = \frac{1}{maxfreq} \tag{4.1}$$

- max freq：最大動作周波数 [Hz]
- clock cycle：1 クロックの周期 [sec]

これにより，各動作周波数の時の 1 クロック分の周期が計算でき，この結果を表 4.1 に示す．しかしながら FPGA 上で実装し動作させる場合には PCI バスの動作クロックに同期させる必要がある．また動作周波数の最大以外からも様々な制約が存在する．このため本システムでは余裕を持って規格上の最大 66MHz で動作させたと仮定する．この場合，動作周波数 66.00MHz の時の 1 クロック分の周期は 15.15nsec と計算できる．

表 4.1: 並列数と最大動作可能周波数

並列数	消費 slice 数	slice 使用率 (%)	最大動作可能周波数 (MHz)	スループット (Gbps)
1	188	0.56	106.112	3.396
4	810	2.40	98.658	12.628
8	1486	4.40	98.658	25.256
10	1859	5.50	98.658	31.571
20	3725	11.02	98.658	63.141
50	9323	27.59	98.658	157.853

ここでは，回路に 256 クロックをかけてデータを入力するので，クロック周期の 256 を 1 ファイルの単位処理時間として計算した．これは式 4.2 で与えられる．並列回路の実装に関して，50 並列以上は OS のメモリ管理の仕様上合成を完了出来なかった．実験に用いた FPGA チップの slice の使用可能な量は 33792 であった．これに対しての使用率の使用率を表 4.1 に示してある．

$$Inputtime = 256 \times clockcycle \tag{4.2}$$

- Input time : 回路への入力完了に必要な時間 [sec]

データが回路を通過する速度に相当するスループットは式 4.3 で与えられる．

$$Throughput = \frac{parallelism \times sizeofIDdata}{256 \times clockcycle} \tag{4.3}$$

- parallelism : 並列度
- size of ID data : 1ID データのサイズ (8192bit)
- Throughput : 作成した回路のスループット [bps]

回路単体でのスループットが充分であることがわかる．

4.2.1 考察

以上のことから，動作周波数 66.00MHz の時の 1 クロック分の周期は 15.15nsec と算出できた．但しこれは 32 ビット分の入力から出力にかかる時間である．8192 ビット（1KB）を処理する場合，フィンガープリントデータの入力を 256 サイクル分繰り返しで行う必要がある．これにより設計した回路への入力には 256 クロック分必要である．判定に用いる BER の計算は順次行うものとしている．よって，式 4.2 から入力開始からから完了ま

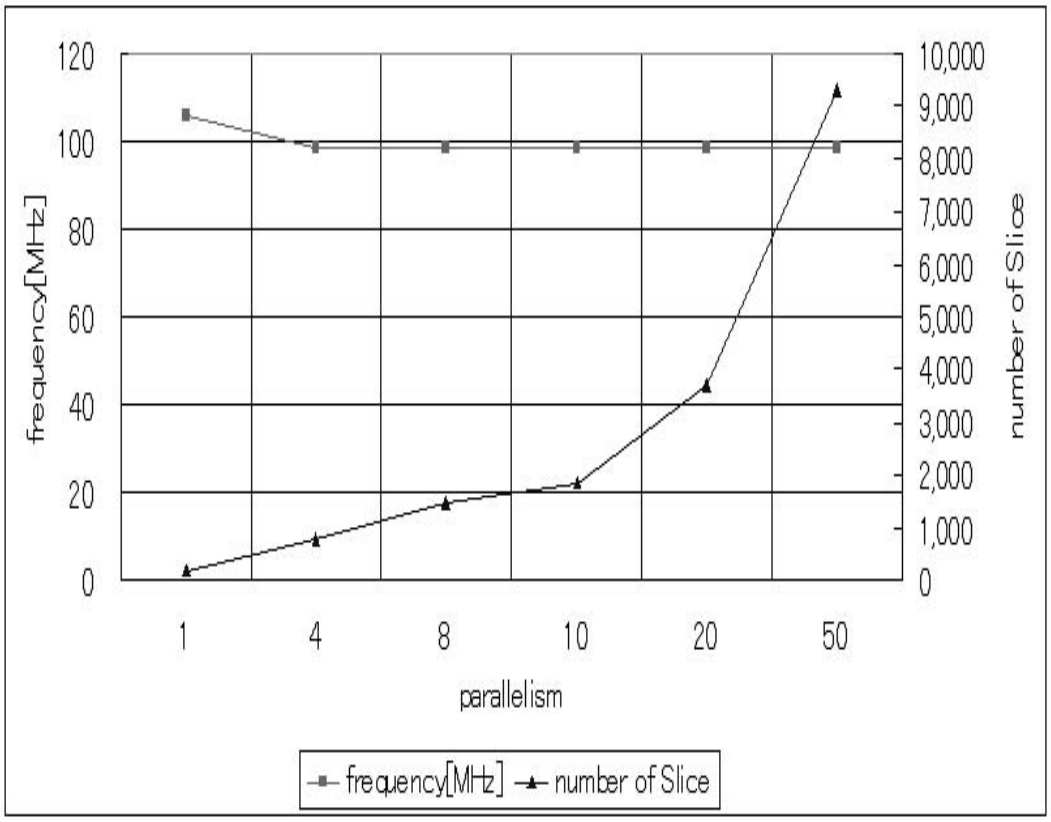

図 4.1: FP 検索回路単体での並列度，回路量と処理速度のグラフ

でに必要な時間が算出できる．これによって回路への 1 楽曲分のデータの入力に動作周波数 66.00MHz とした場合，3.88usec 必要なことが分かる．判定回路は並列に実装するので，入力のタイミングは同じになる．入力のタイミングが各回路ごとに同一であるので出力のタイミングも同じになり，回路の構成上出力時には判定が完了している．つまり，ハードウェア回路のみで検索を行う場合，検索にかかる時間は 3.88usec 必要なことが分かる．またこれにより 1 楽曲分のコンテンツ ID に対して，FPGA 上に搭載された回路の数だけ楽曲判定が行える．FPGA 上に搭載できる楽曲は FPGA チップの利用可能な領域によって変動するが，ここでは 50 楽曲までの搭載を確認した．これに対しての使用率がおおよそ線形増加である事から余裕を持って見積もりを行っても 150 並列程度の並列実装は可能であると思われる．

次節ではこのソフトウェアによるフィンガープリントデータの検索を評価する．

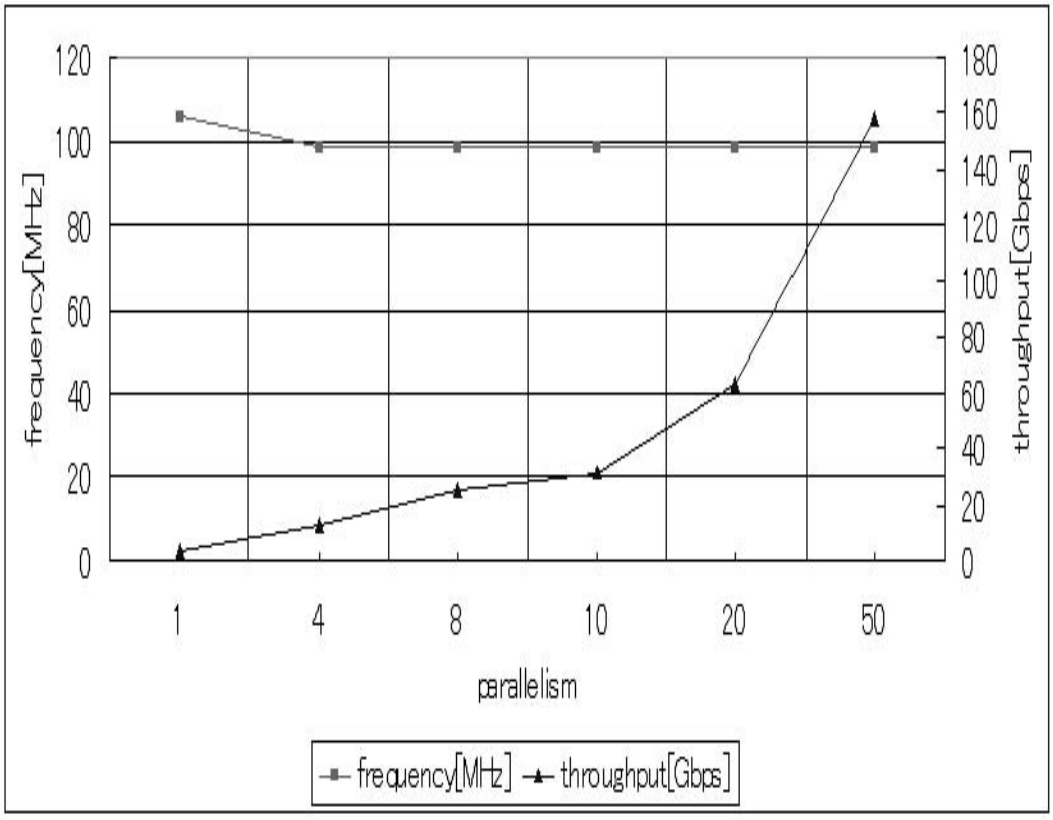

図 4.2: 並列度，動作可能周波数，スループットのグラフ

表 4.2: ソフトウェアによる検索時間

データベースサイズ (KB)	検索時間 (sec)	DB アクセスバンド幅 (Mbps)	スループット (bps)
1,000	0.07	117.03	117.03×10^3
10,000	0.68	120.47	12.05×10^3
100,000	6.78	120.83	1.21×10^3
1,000,000	73.29	111.78	111.78

4.3 ソフトウェアによる FP 検索処理速度

ハイブリッドシステムでは，ソフトウェアとハードウェアの協調動作を行う必要がある．他の部分との協調動作を目指す場合は具体的には図 4.4 のようになる．本節ではシステムのソフトウェア部分のみを評価する．アルゴリズム等は前述のとおりである．ソフトウェア部分のみでのアルゴリズムは前述のとおりで参照データベースの大きさによって処理速度が変化する．1 曲を 1KB として扱うので，データベースサイズの単位は KB になっている．経過時間でこれを実質的な処理速度としてそれぞれ算出できる．先行実験と同じアルゴリズムを用いて判定を行うプログラムではあるが，判定部分をループ展開して最適化し若干高速化してある．

同表にはソフトウェアのみで検索を行う場合のシステムとしてのスループットも併記してある．データベースアクセスバンド幅は，1 つのコンテンツ ID データを検索する場合のスループットとは異なる．これはソフトウェアにおける単位時間当たりに扱うデータ量を検索速度と表記しているだけであり，システムとしてのスループット即ち 1 クエリを検索完了するまでの時間経過とは異なる物である．1 つのコンテンツ ID データは 8192bit であり，これを検索し，完了までにかかる時間が検索時間と捉える事が出来る．ここからソフトウェアのみで構成した検索システムで 1ID の 8192bit 分を検索完了する時間との関係をスループットとして表現している．

$$1ID\text{ 検索時間} = \frac{\text{検索時間}\,(sec)}{\text{データベースサイズ}\,(KB)} \approx 70nsec \tag{4.4}$$

4.3.1 考察

判定部分をループ展開して最適化し若干高速化したこともあり，表 4.2 からも分かるように検索時間が先行実験に比べて半分以下，約五分の一程度まで短縮できている．アルゴリズム自体が線形探索のため，検索に非常に時間がかかっている．1ID あたりの検索時間は 4.4 で示される．これにより，4 種類の検索における 1ID 検索時間は平均 69.77×10^{-6}sec となる．10 万曲までは**70nsec** あれば表 4.2 の検索時間内に検索が完了する事になる．

ハードウェアとは違いソフトウェアではコンテンツ ID のデータベースサイズを比較的

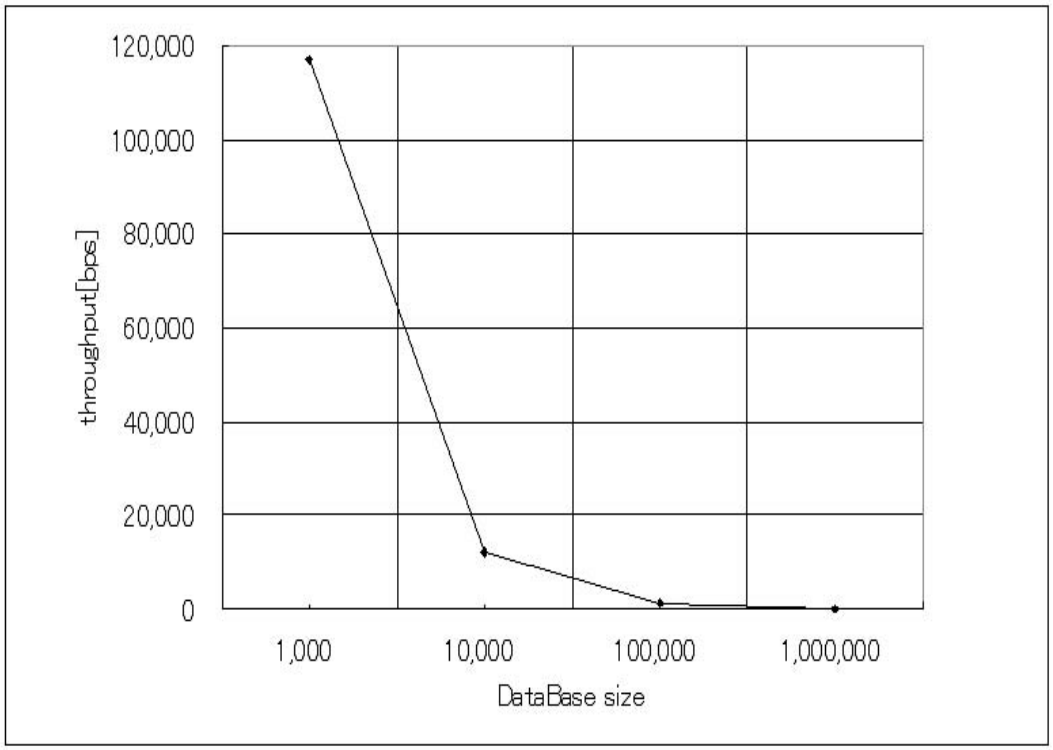

図 4.3: データベースサイズとスループットのグラフ

大容量とすることが出来る．DB アクセスバンド幅の部分を見ると非常に高速のように見受けられるが，重要な項目はスループットである．提案システムでは未知のフィンガープリントデータが繰り返し入力され，順次判定をしていくことを前提としている．プログラムではアルゴリズム上，最終端に目的のデータを配置している．この場合スループットの項目は 1 楽曲分のコンテンツ ID に対して，検索時のワーストケースになる．実験で用いたプログラムはアルゴリズムの素直さを狙ったものであり，非常に低速であった．しかしながら，未知のフィンガープリントデータが入力されることを考えれば，最良のケースは最初の 1 ステップ目でコンテンツ ID を判定できるとも言える．システム全体として考えた場合，参照頻度の高いコンテンツ ID をハードウェアで高速判定を行う事と同時に，ハードウェア検出できなかったコンテンツ ID を大規模なデータベースを持ったホスト PC で行う事で，それぞれの不足を補い合う関係であった．このソフトウェアでのプログラムにはより高速動作向きに変更を施すなどの改善しなければならない点が存在するものの，参照頻度などの概念を考慮していない．現実での運用にはやはり偏りや参照頻度が存在し，検索時間もそれにより変動する．つまり，システムの評価に関しては，偏りや参照頻度の観点から処理速度を観察する必要があることが分かる．これを次節において詳しく説明する．

4.4 システム全体のFP検索処理速度

ハードウェア部分とソフトウェア部分それぞれの評価は前節までに述べた．これらを踏まえてハイブリッド動作に関して評価を行った．ここではハイブリッド動作における各パラメータに関する評価を延べていく．

4.4.1 ハイブリッド動作の処理フロー

システム全体の動作として，ハードウェア部分とソフトウェア部分が協調動作を行う事を意識した．協調動作をする場合は，システムは図 4.4 のような挙動を示す．ハードウェアで並列処理する場合を考えると，1 並列〜150 並列までの 1 コンテンツ ID の検索時間は，前節より 3.88usec（66.0MHz 動作時）必要である事がわかっている．また，ハードウェアへの入力格納完了までには 256 クロック cycle 必要なことがわかっている．

それぞれの検索時間の算出方法は式 4.5 によってあらわされる．

図 4.4 中でソフトウェア処理での時間が 0.07-73.29sec となっているのは本研究上のシミュレーションにて使用したプログラムの最適化を施した結果を適用した場合である．実際には 1000 曲検索時に 0.07sec，100 万曲検索時には 73.29sec かかっている．ソフトウェアでの検索時間を実験より式 4.6 に表す．

$$Th = 256clockcycle \times \frac{1}{frequency} = 3.88usec \tag{4.5}$$

- frequency：動作周波数
- Th：ハードウェアでの単位検索時間 [sec]

$$Ts = 0.07〜73.29sec \approx 70nsec \times 曲数 \tag{4.6}$$

- Ts：ソフトウェアでの単位検索時間 [sec]（データベースサイズによる）

ここまででハードウェア，ソフトウェア，それぞれの検索部分の検索時間が明らかになった．システムは図 4.4 のような処理フローに従って．コンテンツ ID を処理していく．この処理方法に従って，検索完了までのモデルを数式化する．以下に検索部分の処理フローを抜き出し，示す．

1. システムにコンテンツ ID が入力
2. ハードウェアで検索
   (a) コンテンツ ID を識別できれば→処理終了
   (b) コンテンツ ID を識別できない→ソフトウェア処理に委託
3. ソフトウェアで検索
   (a) コンテンツ ID を識別できれば→処理終了

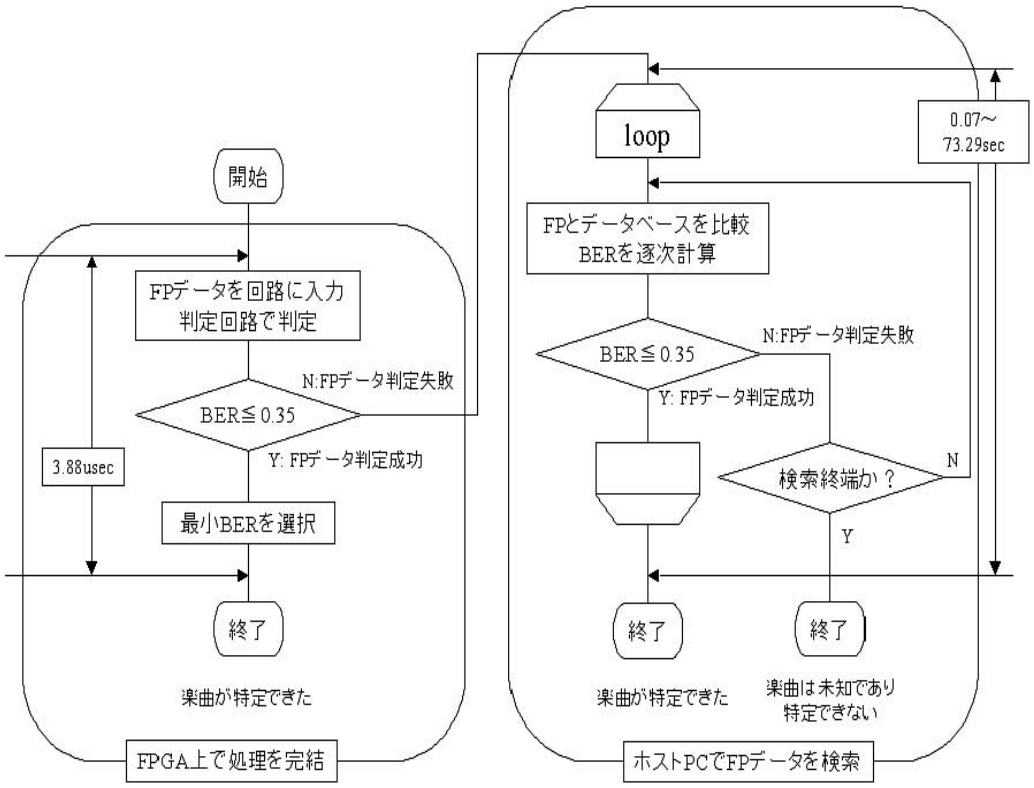

図 4.4: 検索システム全体図

(b) コンテンツ ID を識別できない→処理終了

この処理フローを踏まえてシステムの総検索時間を式 4.7 中の Ttot に示す事が出来る．

$$Ttot = Th + Ts \times (1 - Pr) \tag{4.7}$$

- Th：ハードウェア検索部分での単位検索時間 [sec]
- Ts：ソフトウェア検索部分での単位検索時間 [sec]（データベースサイズによる）
- Pr：ハードウェア検索部分での識別確率
- Ttot：システム全体での検索時間 [sec]

式 4.7 よりシステム全体での検索時間 Ttot と，ハードウェア検索時間 Th，ならびにソフトウェア検索時間 Ts の関係が明らかになった．ここまでで式 4.5，式 4.6 よりそれぞれの検索時間が前節までに明らかになっている．これらを用いて 3 章 3 節で述べているように Ttot を**250ms** 以内に行うとすれば式 4.7 より Pr の値は996.59×10^{-3} 以上必要である事が計算できる．これらパラメータの詳細な検討を以降から行う．

表 4.3: 識別確率 Pr が一定のときの Ttot

Pr	Th(sec)	検索曲数	Ts(sec)	Ttot(sec)
0.6	3.88×10^{-6}	1,000	0.07	28.00×10^{-3}
0.6	3.88×10^{-6}	1,000,000	73.29	29.32
0.9	3.88×10^{-6}	1,000	0.07	7.00×10^{-3}
0.9	3.88×10^{-6}	1,000,000	73.29	7.33

4.4.2 検索完了までの時間 Ttot の評価

100 万曲検索時に sec オーダーのソフトウェア検索部分に対し，ハードウェア検索部分は検出できる曲数が限定はされているものの usec オーダーで高速に動作する事が前節までに明らかになっている．これはハードウェアでの識別確率 Pr の値が一定のとき，ハードウェアで検出完了した場合のシステム全体の検索時間とハードウェアからソフトウェアに委託され処理が終了した場合の検索時間に大きな開きが出ることを示している．よって，本システムではハードウェアでの識別確率 Pr の値が一定であるならば，ソフトウェア検索時間の方が支配的になり得る．しかしながら式 4.10 によるパラメータ Pr によってシステム全体での検索時間は変動する．

Pr をもし仮に 0.48 と 0.89 と与えた場合，実験におけるデータから全体時間は，表 4.3 となる．この Pr の算出や変動については後述するが，オリコンチャートから得られる 300 楽曲のうち 150 楽曲をハードウェアで検出した，としたシミュレーションの結果の平均値である．なお実験から得られたハートウェア検索時間 Th とソフトウェア検索 Ts を用いて算出した．

表 4.3 中での Ts の値は表 4.2 からも分かるようにソフトウェア検索時間 Ts は，1,000 曲検索時時に 0.07sec，100 万曲検索時に 73.29sec となっている．ハードウェア検索時間 Th は式 4.5 より 3.88usec(66MHz) 動作時とした．表 4.3 からも分かるように Ttot の結果は確かにソフトウェア検索時間のほうが支配的である．同じ Pr の時に Ts が約 10 倍近い変化をする事に対して Ttot も同様に約 10 倍程度の変動があることが分かる．しかしながら異なる Pr=0.6 と 0.9 のとき，同じ Ts で検索を行った場合を観察すると，Ts の時間によらず一律で約 75 %程度検索時間を削減できている．これは Ttot に与える影響が非常に大きい事が分かる．これによって明らかになる点は以下の 2 つである．

1. 総検索時間削減のためには総検索時間を占める割合の大きい方の検索時間を削減すること．
2. 同時に Pr の改善によってこれをカバーできる可能性がある．

このうち本システムでは検索時間 Ts の割合が大きく，これを削減する事が総検索時間 Ttot を削減するひとつの手段である．しかしながら，ソフトウェアでの検索時間は検索データベースサイズと関連があり，本実験で用いたものでは検索時間が 1ID を検索する上で一定

となっており，これを削減する事は検索曲数の検証につながる．

また，Pr の改善であるが，Pr が理想的な状態であっても各検索部分でかかる時間には制約がある．これを次節では明らかにする．

4.4.3 ハードウェア検索時間 Th，ソフトウェア検索時間 Ts の制約

またシステム性能の定義のために Pr が一定と与えられる場合，ハードウェア検索部分とソフトウェア検索部分が，最低でもどの程度の時間で検索完了する必要があるかを明らかにする必要がある．提案システムはハイブリッド動作を主眼としているため，互いが互いの検索時間を補助するように動作する．これは式 4.7 でも明らかである．ここで，まずハードウェア検索部分を主眼に考えて，ソフトウェア検索時間が充分に早い状態と仮定する．充分に早い状態とは検索時間が限りなく 0 に近づく事である．つまりこれは入力に対しより高速に，瞬時に検索・判断できる状態になって行くということである．ソフトウェア検索時間が充分高速であるならば，システム総検索時間に対して支配的なパラメータはハードウェア検索時間 Th と言う事になる．

$$Th < Ttot - Ts \times (1 - Pr) \tag{4.8}$$

Pr の値を一定としたときに，式 4.8 から Ttot の値が 3 章 3 節の設定時間によって**250ms** となる．このときシステム総検索時間に対して支配的なハードウェア検索時間 Th はどのようになるか，を考えればよい．式 4.7 で Ttot 以内に検索を完了するとした場合，これは式 4.8 で表せる．ここでソフトウェア検索時間を充分に早い状態だと仮定すれば，Ts=0 となる．

$$Ts < \frac{Ttot - Th}{(1 - Pr)} \tag{4.9}$$

従って，Pr の値に対する最も時間のかかるハードウェア検索時間 Th が Ttot 以内の検索時間という制約を満たすように算出できる．式 4.8 は，検索時間 Ttot と Ts，Pr が与えられた場合のハードウェアでの検索制約時間を表すものである．

同様に，ソフトウェア検索部分を主眼に考えて，ハードウェア検索部分が充分に早い状態だと仮定する．式 4.7 から，式 4.9 と表せる．これもまた同様にハードウェア検索時間を充分に早い状態だと仮定すれば，Pr の値に対する最も時間のかかるハードウェア検索時間 Ts が Ttot の制約を満たすように算出できる．しかしながら，前節でも述べたように Pr の値によらず，ハードウェアでは検索を行うようにシステムを構築しているので，ハードウェア検索部分の実験より Th=3.88usec として計算をおこなう．式 4.9 は，検索時間 Ttot と Th，Pr が与えられた場合のソフトウェアでの検索制約時間を表すものである．

式 4.8，式 4.9 からシステム検索時間に対する最も時間がかかる検索の制約条件を算出した．以下，表 4.4，表 4.5 となる．式 4.8，式 4.9 に Pr との関係は示されているが，ここでは評価のために Pr を 0.9 と 0.6 と与えた場合を示す．

各 Pr の時に，検索時間がハードウェア検索部分とソフトウェア検索部分でどの程度までとりうる事ができるかを示した．注意すべき点は任意の Pr によってそれぞれの検索時

表 4.4: 識別確率 Pr が一定で識別時間が 250msec 制約時の Th の検索時間の制約

Pr	識別時間 (sec)	Ts(sec)	Th(sec)
0.6	0.25	0	250.00×10^{-3}
0.9	0.25	0	250.00×10^{-3}

表 4.5: 識別確率 Pr が一定で識別時間が 250msec 制約時の Ts の検索時間の制約

Pr	識別時間 (sec)	Th(sec)	Ts(sec)
0.6	0.25	3.88×10^{-6}	624.99×10^{-3}
0.9	0.25	3.88×10^{-6}	2.5

間 Ts の制約が決定すると言う事である．傾向としては Pr の上昇に伴ってソフトウェア検索時間である Ts に余裕が生まれている．この傾向はシステムのどのような検索パスを通るか，に依存する．本システムでは入力さたデータが必ずハードウェア検索部分で検索される．Pr の変化に対して Th の変化がなく一定なのはこのためである．Pr の変動により，特に Ts に関しては影響を受けやすい事が分かった．提案システムのコンセプトとして，より高速な検索部分への検索回数を高めるように行った．ハードウェアでの検索が充分に高速であることはハードウェア検索部分の評価より明らかである．これはもとより図 4.4 のように，検索時間がより支配的な検索部分へ，余裕の発生の大きいほうを割り付ける作業に他ならない．従って，本システムではソフトウェア検索部分を余裕の発生の大きいほうへ割り付けていることがわかる．これはソフトウェアにおける検索時間が支配的なことに他ならないが，このソフトウェア検索部分の改善によってより効果的なシステム全体としての検索時間の削減を達成する事ができるようになっていると言える．ここで検索時間を表 4.5 のように与えれば，実験で用いたソフトウェアによる検索の場合，1ID の検索時間が 70×10^{-6}sec と分かっているので，**Ts=624.99×10^{-3}sec のときに検索可能な曲数は 8928 曲分，Ts=2.5sec のときに検索可能な曲数は 35714 曲分**と言う事がわかる．次に式 4.7 にあるように，Pr と識別時間である Ttot の関連に関して評価をおこなっていく．

4.4.4 識別率 Pr と識別時間 Ttot の関連性

ハードウェア検索時間 Th と，ソフトウェア検索時間 Ts が実験のとおりのオーダー，値で与えられるとするならば，式 4.7 からシステムの高速動作のためには出来うる限りハードウェアで ID を検出する事が好ましい事がわかる．ここでは主に式 4.7 中の Pr はどのように定義されるべきであるか，を論ずる．実装を行ったシステムでは，主にハードウェアでの検索を行い，そこで発見できなかったコンテンツ ID をソフトウェアで検索する，というものであった．少なくともハードウェアで一度は検索を必ず行い，ハードウェア上で検索・判定できる ID の数には限りがあった．ここでは 2 章で行ったオリコンチャートの

分析を踏まえてPrについて評価を行ってゆく．オリコンチャートは例として表2.3（2007年1月分オリコン調べ）のように与えられ，左列に順位，タイトル名，右列にセールス数が記録されている．分析からを踏まえて，ハードウェアで検索するとした場合のPrの定義は式4.10のようになっている．

$$Pr = \frac{\sum_{n=1}^{N_h} s(n)}{\sum_{n=1}^{N_h} s(n) + \sum_{n=N_h+1}^{\infty} s(n)} \tag{4.10}$$

- N_h：ハードウェアで検出できるとするタイトル数
- s(n)：各タイトル数のセールス数
- $\sum_{n=1}^{N_h} s(n)$：システムに入力されるIDのうちハードウェアで**検出可能**なセールス数の合計
- $\sum_{n=N_h+1}^{\infty} s(n)$：システムに入力されるIDのうちハードウェアで**検出不可能**なセールス数の合計
- $\sum_{n=1}^{N_h} s(n) + \sum_{n=N_h+1}^{\infty} s(n)$：システムに入力されるタイトル数の総計
- Pr：ハードウェアでの識別確率

これはハードウェアで検出できる曲数の割合がシステムに入力される検索要求全てのうちどの程度の割合を占めているのか，を表す．ここではハードウェア上に実装する検出回路はオリコンチャートを参照するようにしてある．オリコンチャート上のデータを選択して，ハードウェア上に検索部分として実現する．よってオリコンチャート内のデータのうちハードウェアに搭載できるものとできないものに分けられる．また，オリコンチャートでは集計データをローカライズされたものがデータとして配布されているので，必ずリストの下限が存在する．従ってオリコンチャートを実装に意識する場合，Prの値は4.11で算出する事ができる．

$$Pr = \frac{\sum_{n=1}^{N_h} s(n)}{\sum_{n=1}^{N_h} s(n) + \sum_{n=N_h+1}^{N} s(n) + \alpha} \tag{4.11}$$

$$\alpha = \sum_{n=N+1}^{\infty} s(n) \tag{4.12}$$

- N_h：ハードウェアで検出できるとするタイトル数
- N：オリコンチャートの下限のタイトル数
- s(n)：各タイトル数のセールス数
- $\sum_{n=1}^{N_h} s(n)$：ハードウェア上に実装できて**検出可能**セールスの合計

表 4.6: Pr を変化させたときの識別時間 Ttot(Th=3.88usec,Ts=72.29sec)

Pr	Ttot(sec)
0	73.29
0.6	29.32
0.7	21.99
0.8	14.66
0.9	7.33
996.50×10^{-3}	256.52×10^{-3}
996.51×10^{-3}	255.79×10^{-3}
996.52×10^{-3}	255.05×10^{-3}
996.53×10^{-3}	254.32×10^{-3}
996.54×10^{-3}	253.59×10^{-3}
996.55×10^{-3}	252.85×10^{-3}
996.56×10^{-3}	252.12×10^{-3}
996.57×10^{-3}	251.39×10^{-3}
996.58×10^{-3}	250.66×10^{-3}
996.59×10^{-3}	249.92×10^{-3}
996.60×10^{-3}	249.19×10^{-3}
1	3.88×10^{-6}

- $\sum_{n=N_h+1}^{N-N_h} s(n)$：オリコンチャート上のハードウェア上で**検出不可能**なセールス数の合計
- α：オリコンチャート下限以下のセールス数の合計
- Pr：ハードウェアでの識別確率

式中の α はオリコンチャート下限以下のセールス数の合計である．これについては後述する．

ここで Pr を変化させたときの識別時間について表 4.6，図 4.5 に示す．

同様のシステム全体での検索時間に対し，ハードウェアでの識別確率 Pr が異なる場合，前項で提示したように，ハードウェア部分での検索時間 Th と，ソフトウェア部分での検索時間 Ts，それぞれに与えられる余裕時間は全く異なる結果となる．出来うる限り高速なハードウェアでどれだけコンテンツ ID を識別できるか，が提案システムでの高速動作へのポイントになる．

使用したオリコンチャートの表では下限が 300 位となっていたので，式 4.11 では N=300 となる．ここで Pr の意味は順位 n までのセールス数 s(n) の合計が順位 300 位までのセールス数の合計に対してどの程度の割合を持っているかどうか，である．この場合ハード

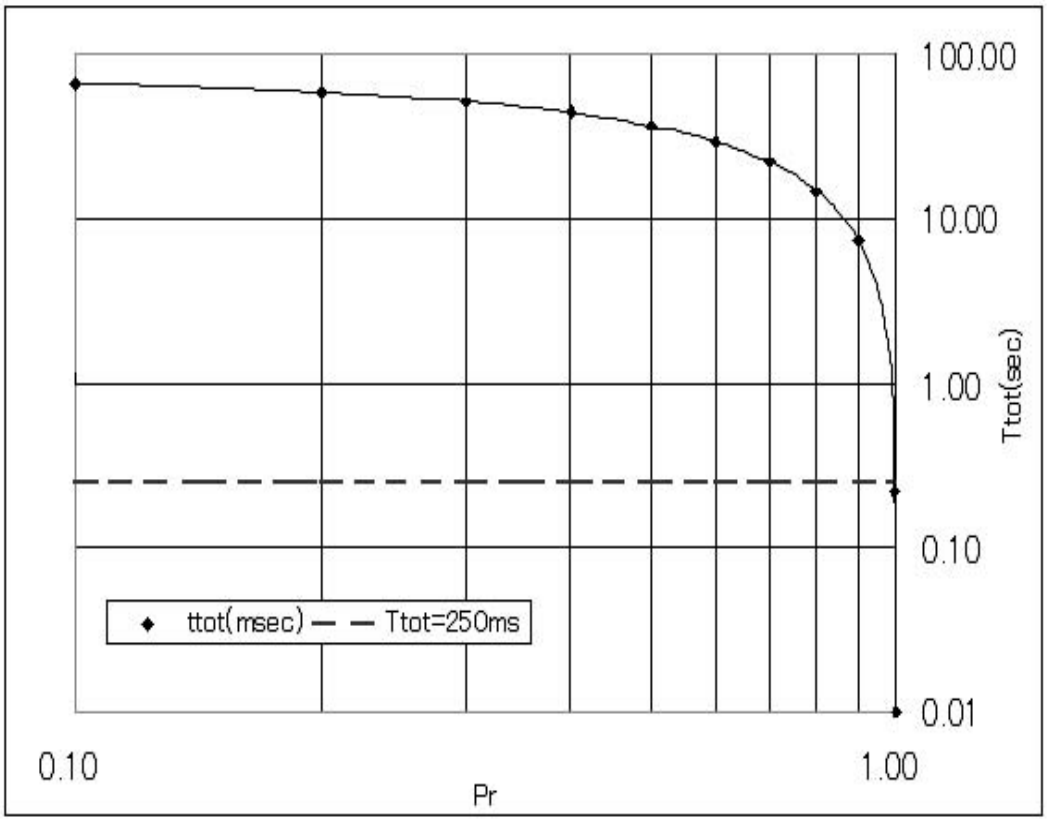

図 4.5: 識別確率 Pr と識別時間 Ttot の関係 (Th=3.88usec,Ts=72.29sec)

ウェア上で検出を行う事が出来るチャート順位はハードウェアの合成に対する結果表 4.1 からみて 150 並列分である．オリコンチャート順位の上位からハードウェア上に実現する検出回路を選択する事によって，出来うる限り Pr の向上を狙わねばならない．チャート順位は上位が全体のセールスの合計に対してのより大きな割合を占める傾向があるからである．既存のオリコンチャートの順位が 300 位までなので，これを用いて試算すると現実のオリコンチャートのデータから 150 位までをハードウェア上で検出可能となる．2007 年 3 月 12〜2007 年 3 月 18 日におけるオリコンチャート集計データで Pr の値は最良でも 934.50×10^{-3} になっている．前述のように式 4.7 よりシステム全体での検索時間 Ttot と，ハードウェア検索時間 Th，ならびにソフトウェア検索時間 Ts の関係が明らかになった．よって Ttot を**250ms** 以内に行うとすれば式 4.7，表 4.6 より Pr の値は996.59×10^{-3} 以上必要である事を本項で示した．Pr の値は996.59×10^{-3} 以上必要だとすれば，2007 年 3 月 12〜2007 年 3 月 18 日におけるオリコンチャート集計データだと 1〜290 位分をハードウェアに実装する必要がある．別の観点として，検索対象の曲総数によって Pr の値は変動する事が考えられ，この点については検討の余地がある．次節では時間経過に対する Pr の変化などを評価しつつシステムとしてのパフォーマンス見極めてゆく．

4.4.5 時間経過とハードウェアでの識別率

本節ではオリコンチャート 300 位分によって与えられた Pr の値について時間経過とともにどのような変化が起こるのか，について評価を行う．一般的に流行は常に変動するものであり，多くの人々が求める楽曲には様々な方向性がある．このなかからオリコンチャート 300 位分によって与えられた，2007 年 2 月 5 日の週分のデータを基点として，経過時間によってハードウェアにおける 150 曲分のカバー範囲からどの程度ヒットチャートが変更されていくか，を観察する．日付的には 2007 年 1 月 22 日から，2007 年 2 月 18 日までの期間をまとめたものである．2007 年 1 月 22 日から 2007 年 1 月 28 日，2007 年 1 月 29 日から 2007 年 2 月 4 日，2007 年 2 月 5 日から 2007 年 2 月 11 日，2007 年 2 月 12 日から 2007 年 2 月 18 日，同様に 2007 年 4 月 1 日までを 1 週間ごとに 4.6 にまとめた．理論上ではハードウェアで識別できる確率 Pr の値が低下すると，ソフトウェア検索処理に時間がとられてしまい目的とした性能を発揮できない．この場合は計測基点を 1/22〜28 とした場合この 7 日経過後の 1/29〜2/4 までの集計期間にハードウェアでの識別確率が 88.03 %から 64.43 %まで低下している．この評価においてはハードウェアでの識別確率は時間の経過に伴って低下する傾向が見て取れる．実質これはシステム性能の劣化を示すものである．

ハードウェアに実現する識別回路の量を 150 位分から 300 位分まで変化させた場合の識別確率 Pr の変動を図 4.7，図 4.8，図 4.9，図 4.10 と示した．集計に用いた入力データはオリコンチャートから 300 位までのデータしか得られないため，実質図 4.10 が最良値と言う事になる．これらによって，識別確率 Pr の値の経過時間に関する変動が明らかになった．データとして，やや変動があるが概ね時間経過に対し下降を示す事がグラフより理解できる．これは分析の項でも述べたように，ランキングの上位と下位がわずか一週間の間に大きく変動する事による．システムの性能評価として一般には識別確率 Pr を可能な限り低下させない事が重要になる．よって，ここから FPGA 上に実現したハードウェア上に実現した識別回路を書き換えることによって Pr の値の低下を防ぐ事が出来ると考えられる．最も識別確率が低下している場合，図 4.10 の中において，2/19〜2/25 の項で 100 %から 7 日間経過の後 30.41 %まで低下が確認できる．これは一時間当たり 0.4 %の減少ということが言える．また，提案システムの中で，計測した 100 万曲のソフトウェア検索時間で目標検索時間である 250msec を満たすためには Pr が 996.59×10^3 以上である必要があることが分かっている．ここでは，識別確率 Pr が 1 時間当たり 0.4 %の減少が確認された．

実験で使用した FPGA デバイスでは 150 曲分の楽曲 ID 識別回路を構成することが出来ている．この場合 290 曲までの楽曲 ID 識別回路を搭載する必要があることも判っている．従って，このための解決策としては次の 2 つを行う必要がある．

1. 使用する FPGA デバイスにおいて大容量のものを用いる
2. Pr の値を限りなく高く保ち続ける

このうち Pr の値を限りなく高く保ち続けると言う事は，提案手法ではハードウェア上に

実現した検出回路を最新のチャートのデータに従って書き換えてゆく，と言う事に他ならない．このため，どの程度のタイミングで識別回路を書き換える事が必要なのか，また識別回路の書換えの方針について，を次節より検証していく．

経過日数	基準にした集計期間											
	1/22–1/28	1/29–2/4	2/5–2/11	2/12–2/18	2/19–2/25	2/26–3/4	3/5–3/11	3/12–3/18	3/19–3/25	3/26–4/1		
1/22–1/28	88.03											
1/29–2/4	64.43	88.32										
2/5–2/11	39.50	46.54	89.17									
2/12–2/18	40.87	47.35	68.23	86.88								
2/19–2/25	21.84	25.51	33.89	40.60	87.37							
2/26–3/4	9.86	11.91	14.89	16.71	26.74	91.71						
3/5–3/11	9.48	11.74	13.49	14.76	20.72	41.69	89.78					
3/12–3/18	5.88	7.18	8.22	9.30	11.69	19.49	33.46	93.45				
3/19–3/25	9.56	11.20	12.18	13.18	14.81	21.64	36.06	66.34	87.72			
3/26–4/1	7.45	8.44	9.05	9.65	11.04	15.38	23.49	40.80	47.72	89.52		

図 4.6: 経過時間におけるハードウェアでの識別率の表 (チャート 150 位までを HW で検出した場合)

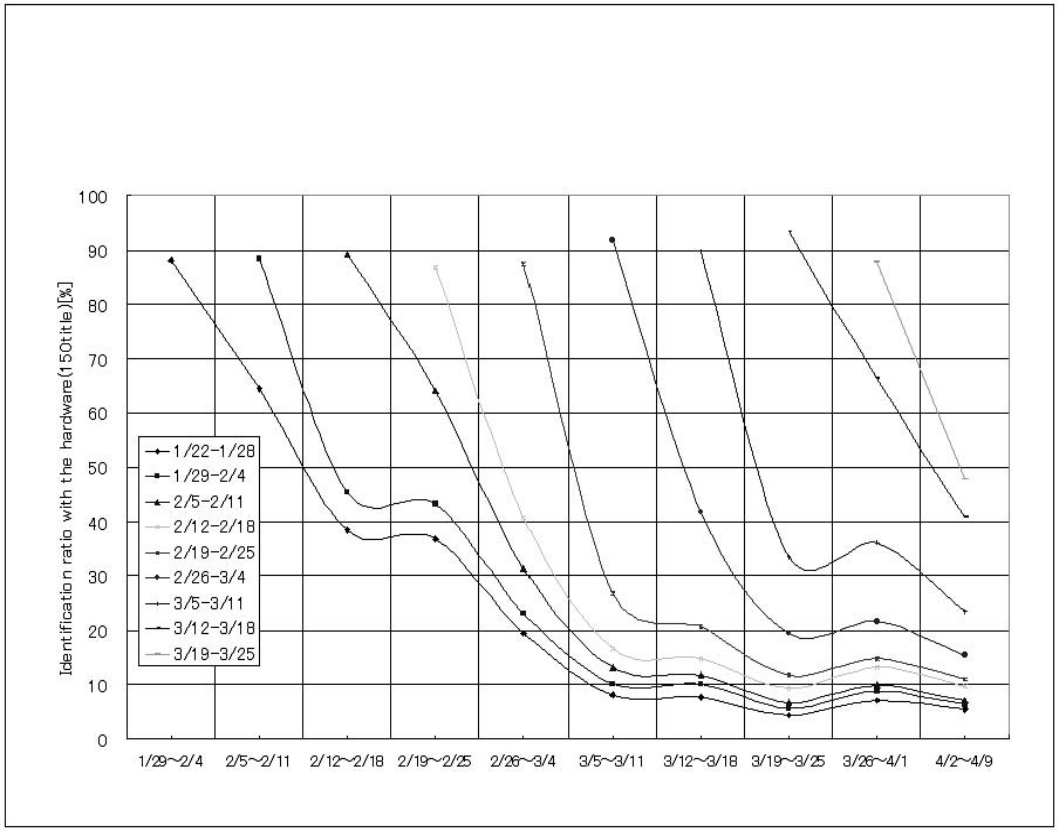

図 4.7: 経過時間におけるハードウェアでの識別率 (チャート 150 位までを HW で検出した場合)

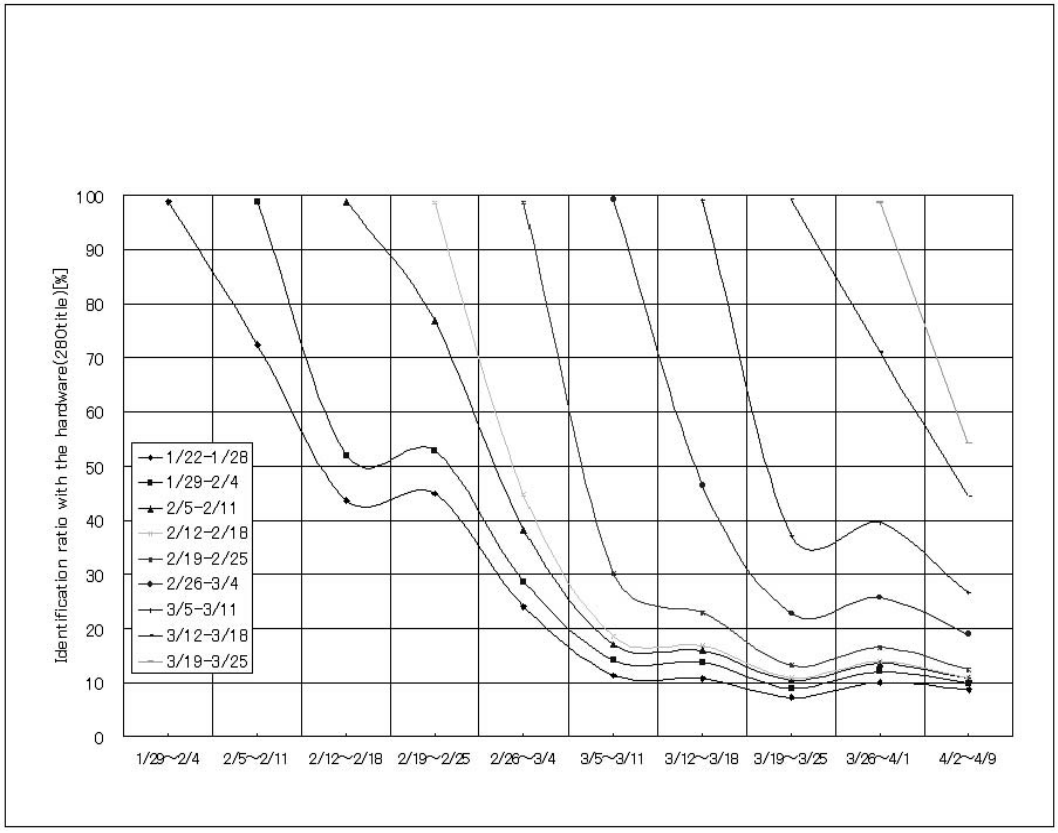

図 4.8: 経過時間におけるハードウェアでの識別率 (チャート 280 位までを HW で検出した場合)

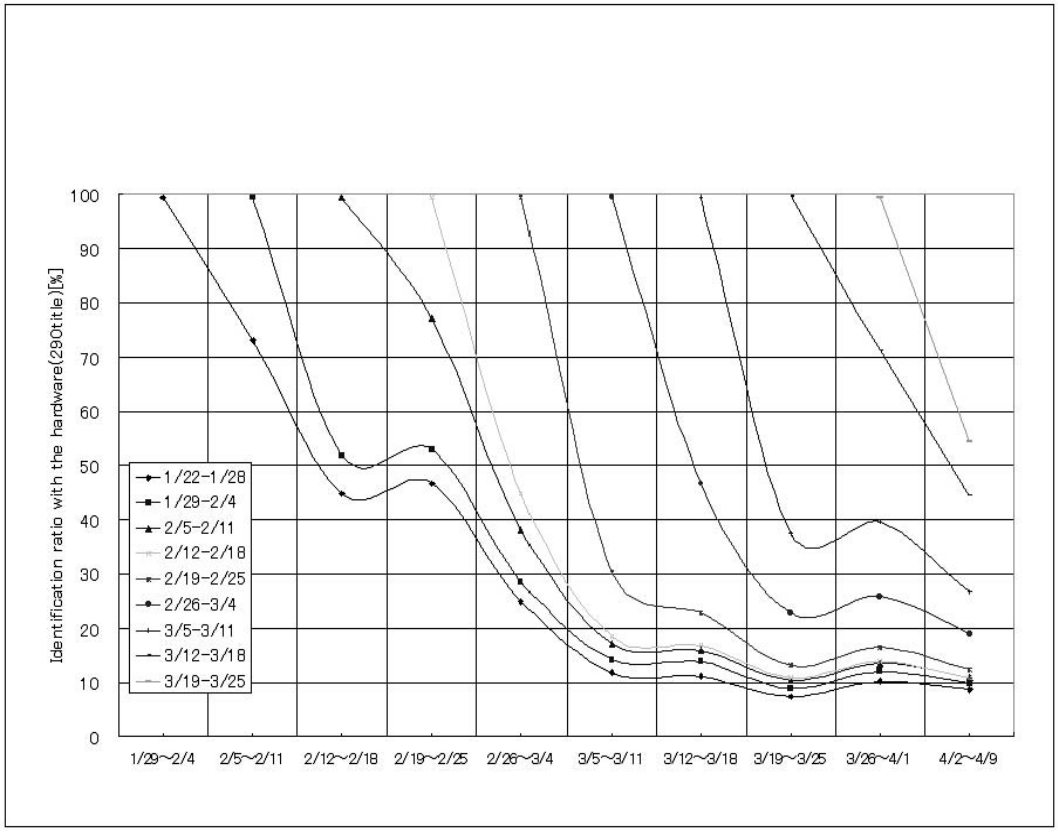

図 4.9: 経過時間におけるハードウェアでの識別率 (チャート 290 位までを HW で検出した場合)

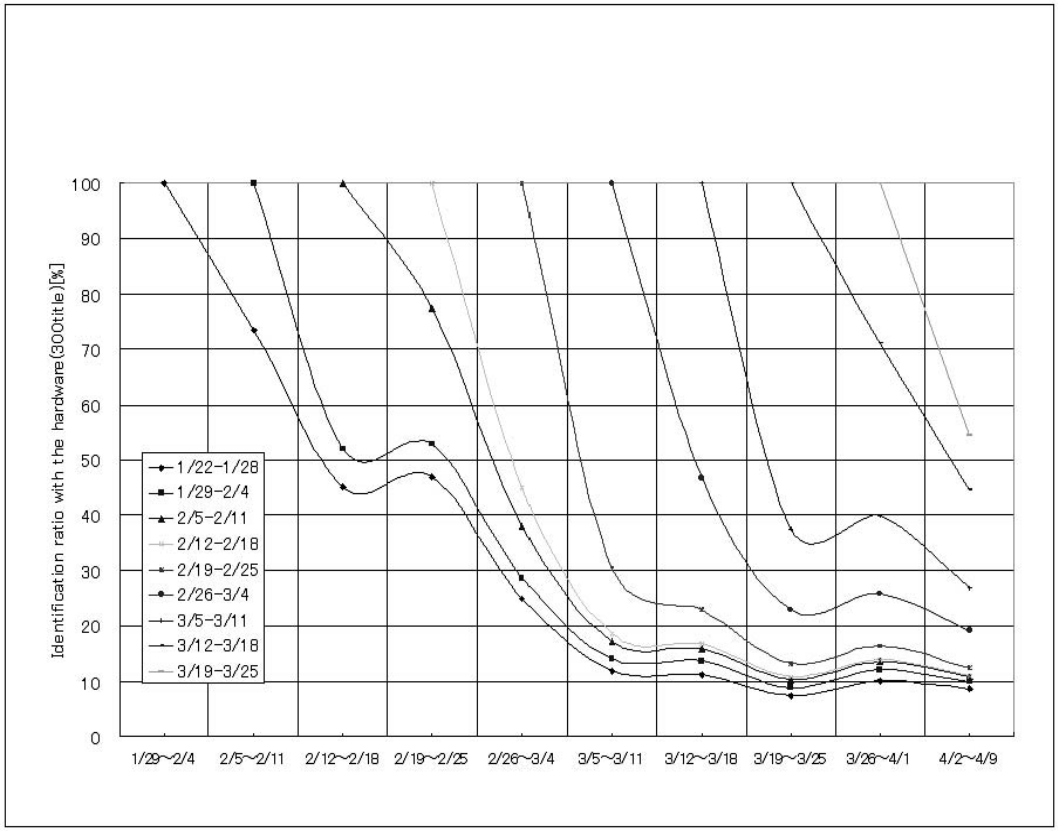

図 4.10: 経過時間におけるハードウェアでの識別率 (チャート 300 位までを HW で検出した場合)

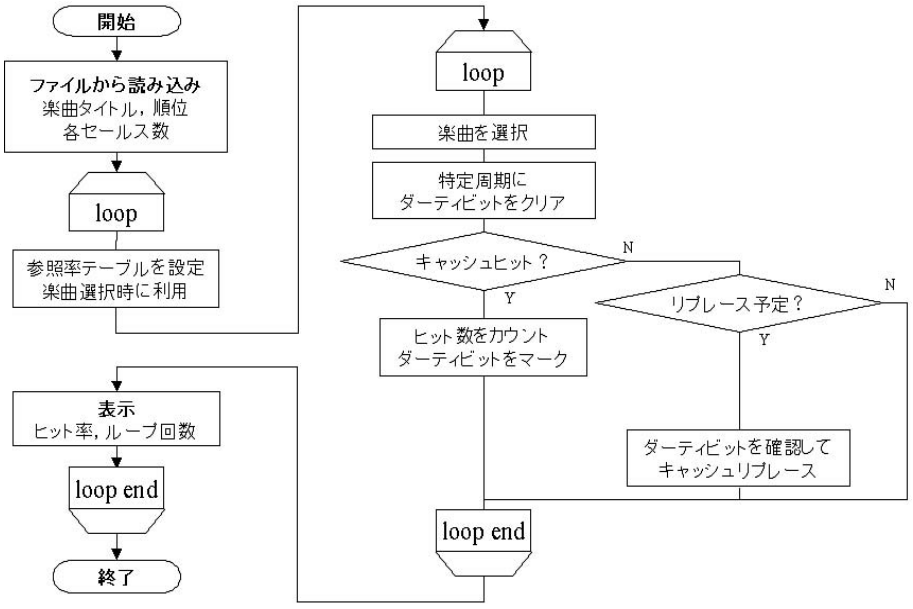

図 4.11: LRU 手法のアルゴリズム

4.4.6 識別率の評価

システム全体として捉えたときにハードウェアでの楽曲検出部分はソフトウェア検索のキャッシュに相当すると見なす事が出来る．ハードウェア処理部分として用いるデバイスの性格上，楽曲の入れ替えには書換が必要な事がわかっている．キャッシュとしてのリプレース法としてはLRU（Least Recently Used）と言われる手法が広く知られている．この手法は最も最近使われていないデータをキャッシュ領域から追い出し，新たに参照が発生したデータをキャッシュ領域に組み込むものである．オリコンチャートをもとに，オリコンチャートを参照し書換を行った場合の時間経過と識別率の関係と，LRU 手法での識別率の関係を下に示す．なお LRU 手法の識別率を算出するためのプログラムフローチャート図 4.11，プログラム内部の処理フローを以下に示す．フローの中でキャッシュにリプレース領域を決定する要素は最も直近の時間内に参照があったかどうか，であり，これをプログラム中にダーティビットなどで処理する事が LRU の性能に近似的な値を与えることがわかっている．

表 4.7: 2 手法における時間経過と識別率の関係

時間経過	2/5〜2/11	2/12〜2/18	2/19〜2/25	2/26〜3/4
LRU でのヒット率	99.54 %	98.62 %	97.69 %	96.81 %
LRU でのリプレース回数	5416	12520	28543	59316

1. **読み込み部分**
   予め用意しておいたファイルから検索したいコンテンツ ID データ，順位，タイトル，セールス数の合計などの情報を読み込む．
   読み込みを行ったファイルを全てにおいて参照率のテーブルを準備する．
2. **楽曲選択部分**
   ランダムに選択された番号で参照テーブルで分類を行い選択楽曲を決定する
   特定のループ回数でキャッシュ領域内のダーティビットを 0 でクリアする

   (a) **キャッシュヒット判定**

      i. 選択されたが楽曲のデータが，キャッシュとして用意した領域内で，
         発見できるかを判定．
      ii. 発見できた場合はキャッシュヒットとしてダーティビットを立てる．

      iii. 発見できない場合，調べたキャッシュの領域のどこかがリプレース対象かどうかを判定．
      iv. リプレース対象（ダーティビットが 0）に相当するのであれば
         キャッシュ領域の内容を選択された楽曲のデータとリプレース．
         同時にダーティビットを 1 にする．

   以上の処理で，選択された 1 楽曲分のデータの処理が終了する．
   読み込みで用意した情報のうちセールス数の合計回数分だけ楽曲の選択を行う．

3. **ヒット率表示部分**
   最初に読み込みを行ったファイルの処理が完了したら，
   検索時のループ総数と総ヒット回数からヒット率を計算し，
   リプレースの回数とともにヒット率を表示．
   ファイルの全てを処理完了するまで先頭に戻り繰り返す．

基準として用いたデータは前節で掲載した 2/5〜3/4 までの期間の集計と同一である．対象とした期間に対して，LRU 手法では最近最もアクセスされていない（参照頻度）の低いデータをリプレース対象とした．全検索回数の 1,183,881（オリコンチャート 2/5〜

2/11 のセールス数 300 位までの合計）に対し，リプレースの回数は 5416 回になっている．集計のデータが 7 日間の出来事となるので，これはリプレース 1 回を 1.8 分以内に行う必要があることを示している．この手法を提案したハードウェアで行う事は，Pr の増加につながるが，FPGA デバイスにおける書換の時間はゼロではなく，ある程度の時間がかかる，よって逆にシステムの効率的な動作を妨げる恐れがある．次節では書換の時間を考慮し，提案するシステムでの書換のタイミングを評価する．

表 4.8: 実装環境

並列回路数	書換にかかる時間 (sec)
1 並列	36
4 並列	133
8 並列	259
10 並列	325
20 並列	723
50 並列	8399

4.4.7 書換頻度

本提案におけるシステムでは回路の再合成によって判定する楽曲を切り替える事を想定しており，ここではその書換にかかる時間を示す．並列回路数を増やすほど，合成に時間がかかることが分かる．50 並列を合成した場合にはおよそ 2.3 時間程度かかる．提案するシステムでは識別する楽曲をハードウェア回路として作りこむ事を行う．ここでハードウェア上に実現した識別対象の楽曲を入れ替えたい場合は必ず回路の再合成が必要になる．提案システムの運用と同時に識別する楽曲は流行などの要素で絶えず変化している．

50 並列以上の場合ホスト PC に用いた OS の制限からメモリ領域が不足し，合成が完了できなかった．また並列数を上昇させる事でハードウェア内の配線量も増加する傾向にある．これはシミュレーション時間に密接に関わる事が分かっている．並列数を上昇させる場合，構築するハードウェアの構造にも注意する必要がある．

また，提案システムの中で，計測した 100 万曲のソフトウェア検索時間 73.29sec で目標検索時間である 250msec を満たすためには Pr が 996.59×10^3 以上である必要があることが分かっていた．前節におけるリプレース面とアルゴリズムの 1 つである LRU を用いると，1 回の書換をおよそ 1.8 分に一度必ず行う必要があることが分かる．これに対し，図 4.10 から最悪の場合，識別確率 Pr は 1 時間 0.4 %の低下が認められる．このことからオリコンチャートを用いる書換え手法では 1 回の書換を 50 分に一度以上行うことで解決できる．

これを利用すれば最新チャートを入手しハードウェアに反映させる事で性能の劣化を防ぐ事ができる．しかしながら，2.3 時間以上をシステムのセットアップ時間として運用する場合，本提案システム 1 つだけではあまりにも無駄になる部分が多い．そこでこれを回避するために主に 2 つの方法が挙げられる．

1. ハードウェアとして用いる FPGA に動的再構成デバイスを用いて書換時間を隠蔽
2. チップ，ホスト PC 問わず 2 台以上並行稼動によってシステムとしてのダウンタイムを隠蔽

このうち動的再構成デバイスに関しては本提案で用いた手法が静的再構成デバイスをター

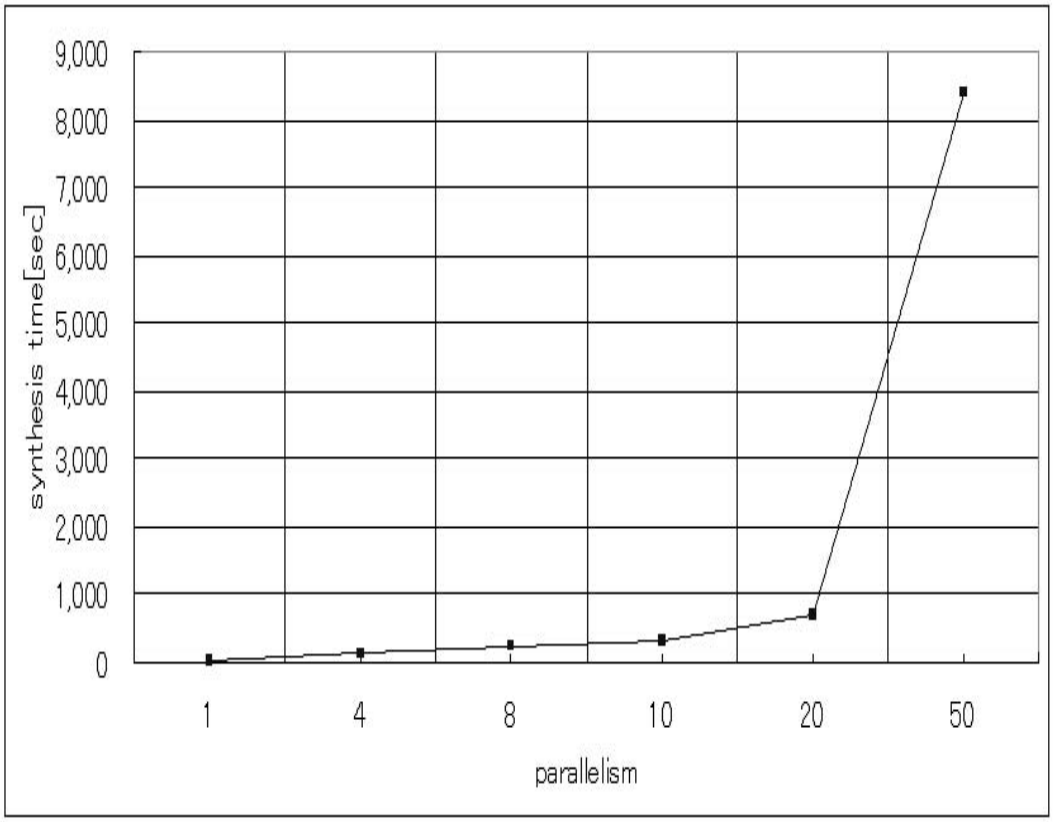

図 4.12: 並列度，再合成時間のグラフ

ゲットにしていることもあり若干の手直しが必要になる可能性がある．この場合 2 台以上のシステムを用いてダウンタイムを隠蔽する方法が入手製，コスト共に優位になるのではなかろうか，と考えられる．

4.4.8 考察

本章によって提案したシステムの評価を行った．システムコンポーネントとしてハードウェア，ソフトウェアそれぞれの速度，並列性を確認した．ハードウェア検索部分に関しては余裕を持った見積もりでも 3.88usec と充分な速度を得る事が出来た．しかしながらソフトウェア検索部分に関しては最適化を施したものの検索時間が 73.29sec と若干の改良・検討の余地が残っている．それらを用いてハイブリッド動作をするシステム全体のシステムを構築した際の評価として，速度，ハードウェアでの識別確率などのパラメタについても評価を行った．これらについては式 4.7 中にもあるようにソフトウェア検索部分が主な検索時間を占めており，直近かつ効果的な改善余地があると言える．しかしながらソフトウェア検索部分の改善についてはアルゴリズムやデータ構造と密接に関わる部分があり，本稿では扱わない．性能向上のための別の方法としてハードウェア検索部分での識別確率 Pr の改善などがあることを確認した．主に評価によるところであるが，ハードウェア検索部分での識別確率 Pr によってシステム全体の性能が大きく変化する事も確認できた．これは主にソフトウェア検索時間が支配的な事もあり，重要なパラメタと言える．ソフトウェア検索の時間と曲数のデータはほぼ線形に推移していると見なせる．Pr の値は996.59×10^{-3} 以上必要だとすれば，2007 年 3 月 12〜2007 年 3 月 18 日におけるオリコンチャート集計データだと**1〜290 位分をハードウェアに実装する必要**がある．また，実際の一致検索に用いるソフトウェアは前節でも述べたように実用的な面からもう少し高速なものが好ましい．システム全体としてはハードウェア検索部分はソフトウェア検索部分からみてキャッシュと見なせる特性がある．これをハードウェアでの識別確率 Pr に関して時間経過との関係を調べ，従来手法である LRU 手法との比較によって本手法における優位性を認められると言える．この部分において更に，LRU を用いると 1.8 分に一度以上の書換を行う必要があることが分かった．対して提案手法では最低でも 50 分に一度以上の書換を行う必要性があることが明らかになった．対象が楽曲データと言う事もあり，流行など様々な要因がある中特にオリコンチャートを用いて最新の流行に追従しようとする手法については本提案の有効性を確認できたと言える．

4.5 まとめ

ハイブリッドシステムにおける様々なパラメータの評価を行った．ハードウェア，ソフトウェア単体での検索時間なども評価を行った．システム全体の検索時間 Ttot を構成する要素として，ハードウェア検索時間 Th，ソフトウェア検索時間 Ts とそれぞれの関係を式 4.7 によって明らかにした．またハードウェアでの識別確率 Pr がシステムの検索時間に及ぼす影響と，4.3 にまとめた．これによると Pr の変動によって一意ではあるが 75 %の検索時間の削減を可能としていた．また表 4.4，表 4.5 によって，ハードウェア検索部分と，ソフトウェア検索部分がそれぞれ充分に早い状態で動作している仮定を行う事によって，ハードウェア検索部分とソフトウェア検索部分のそれぞれの制約条件が明らかになった．これは，Pr がある一定の値のときに，システムが 3 章で定義されている要求検索時間内**250msec 以内**に検索完了するためのハードウェア検索部分と，ソフトウェア検索部分の検索時間の制約が一意に与えられる事を示していた．例として Pr が 0.9 のときにハードウェア検索時間 Th が 131.58×10^{-3}sec 以上かかると要求検索時間**250msec 以内**に検索が完了しない事を示していた．同様にソフトウェア検索時間も 2.5sec 以上かかると要求検索時間**250msec 以内**に検索が完了できない．ソフトウェア検索時間が 2.5sec 以内だとすれば，実験から得られたデータを基にすれば，ソフトウェア検索時間がほぼ線形であると見なせば 1 曲検索時に 70usec かかっているので，**約 35700 曲**に相当する．

最後に上に挙げた全ての要素における重要なパラメータ，ハードウェア識別率 Pr について評価を行った．オリコンチャートから得る事のできる 150 位までをハードウェアに搭載した状態で得られる Pr ではシステム設計目標である検索時間**250ms 以内**には足りないことが分かった．Pr の値は996.59×10^{-3} 以上必要だとすれば，2007 年 3 月 12〜2007 年 3 月 18 日におけるオリコンチャート集計データだと**1〜290 位分をハードウェアに実装する必要**がある事からも分かるように，これは 300 位近くまでをハードウェアに実装することで解決できる事が示された．面積的にハードウェアに実装できる検索回路は 150 曲分としていたが，ここでは**290 曲分を搭載する**ことによってシステムの設計目標である要求検索時間内を満たせる事が分かった．デバイスの面積の問題に関してはデバイスの集積度の向上によって新しいデバイスを利用する事で解決できると思われる．本提案では，1 チップでの実装を行っているので，これを複数チップ構成などにし面積的により多くの識別回路をハードウェアで実現できる事で，この問題は解決できる．LRU 法では書換えの要求回数が増加し，高頻度での書換を要求する事が分かった．これに対し，オリコンチャートを用いた本提案手法では書換頻度が最悪でも 50 分につき 1 回と従来手法のそれらよりも少なくて済むことが分かる．書換時間に対しては一回につき 50 並列の場合 2.3 時間かかることが明らかになった．本提案手法を適用する場合，50 分につき 1 回の書換が必要と分かっているので，2 ホスト PC，またはマルチチップ構成などを用いる事で書換時間を隠蔽しこれを実現する事が出来る．

第5章　結論

5.1　まとめと今後の課題

楽曲ファイルを対象とした，本稿では，コンテンツ識別技術のフィンガープリントをテーマに，大規模かつ高速な一致検索をハードウェアとソフトウェアのハイブリッドシステムを用いて実現した．

現実的なシステム運用を踏まえ現在流通している楽曲を分析した結果コンテンツの流通具合は「流行」という要素に左右される事が分かった．流行の開始は話題の楽曲ファイルのリリースによって始まり，時間経過によって緩やかに変動する．この「流行」という要素を利用し，高速に検索を行うシステムを製作し，挙動を確認できた．これに着目して，再構成可能なデバイスをハードウェアとして用いる事で高速に一致検索を行う手法を確立できた．最も最近リリースされた話題の楽曲を「流行」の楽曲と予想して一部分を再構成可能なハードウェア上に搭載するようにし，流行の変化などにハードウェアも柔軟に対応できるようにした．これによってその時期に最頻度の参照データをハードウェアで高速に検出する事が可能である．参照頻度を利用する手法としてLRUが挙げられるが，これを本システムでハードウェア実装することは得策では無い事も評価から分かった．

通常，ハードウェア上に実装する一致検索システムではハードウェア上の制限から検索データベースサイズが制限されるので，大容量にすることが出来ないが，本稿ではこれをソフトウェアによる柔軟なサポートで解決をした．これにより，単位時間当たりシステムに対する入力のうち多数の割合を占める流行の楽曲の検出を可能にした．

速度の面から言えば，ハイブリッドシステムであるので，処理時間の最もかかる方へシステム全体の処理時間が引きずられる．しかしながらPrの値が996.59×10^{-3}以上で，ソフトウェア検索時間が100万曲分を検索するために73.29sec以内であるならば設計目標であるシステムの総検索時間を式4.7より**検索時間250ms以内**に収める事が出来る．Prの値が996.59×10^{-3}以上必要だとすれば，2007年3月12〜2007年3月18日におけるオリコンチャート集計データだと**1〜290位分をハードウェアに実装する必要**がある．よって本システムでは検索タイトル数が**290曲目以上**であるならば，実装を行ったハードウェアとソフトウェアによるハイブリッドリアルタイム検索システムが有用といえる．

逆に本実験より得られたソフトウェア検索時間，ハードウェア検索時間を利用し，検索目標時間を式4.7より**250msec**以内とするならば，オリコンチャートの分析より得られたハードウェアでの識別確率Pr=0.9を用いるとソフトウェア検索時間は**2.5sec以内**でなければならない．このことから，実験によって得られたソフトウェア検索時間について検

討すれば，検索可能な楽曲数は実験よりおよそ**35700 曲分**とわかる．

以上をまとめると提案するシステムの有効範囲は以下のようになる．

- ハードウェアでの検索時間 Th=3.88×10^{-6}sec,
  ソフトウェアでの検索時間 Ts=73.9sec(100 万曲検索時) のとき
  識別確率 Pr は 996.59×10^{-3} 以上なら,
  検索目標時間 T が 250msec 以内で検索完了する．
  このときハードウェアで識別すべき曲数はオリコンチャートで換算すれば 290 位以上となる．

- ハードウェアでの検索時間 Th=3.88×10^{-6}sec,
  識別確率 Pr は 0.9(オリコンチャート分析より) のとき
  検索目標時間 T が 250msec 以内で検索完了するなら,
  ソフトウェアでの検索時間 Ts は 2.5sec 以内でなければならない．
  このときソフトウェアでの検索時間内で検索できる楽曲数は実験よりおよそ 35700 曲分となる．

ハードウェア上に実装可能かどうかは FPGA の面積的な問題とも関わってきている．

また書換の時間について従来手法である LRU との比較を行った．その結果，**LRU では 1.8 分に 1 回以上**と高頻度での書換を要求することに対し，本提案手法では**最低でも 50 分に 1 回以上**の書換頻度の要求であることが明らかになった．これは従来手法に対し，少ない書換時間で性能維持が可能と言う事が明らかになった，といえる．書換時間に関しては並列数が増加するほど増える傾向にあることが分かった．これらはチップの集積度の向上，マルチチップ運用，ホスト PC の並行稼動など，様々な方法で実現可能といえる．

また本実験で用いたソフトウェアでの検索において，速度やデータベースの格納方法などに現実的ではない部分があり，これは今後の課題となりうる．

参考文献

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

[2] Hisashi ISONAGA,Yasushi INOGUCHI, "Implimentation of high speed audiofingerprint system using FPGAs" VLD2005-88,CPSY2005-44,RECONF2005-77,pp.1-6

[3] Prarthana Shrestha,Ton Kalker, "Audio Fingerprinting In Peer-to-peer Networks" ISMIR 2004 Conference Committee.

[4] Akiko AIZAWA,Atsuhiro TAKASU,Keizo OYAMA,Jun ADACHI, "Record Linkage of Multi-source Databases: Research Trends" NII Journal No.8 (2004.2)

[5] Etsuko TAKAMICHI,Koji NAKANO , "An Image Retrieval System Using FPGAs" IEICE technical report. Circuits and systems Vol.102, No.426(20021101) pp. 37-42 CAS2002-89

[6] P.J.O.Doets,R.L.Lagendijk , "Theoretical Modeling of a Robust Audio Fingerprinting system" Proc.SPS 2004 (the 2004 IEEE Benelux Signal Processing System)

[7] Yasushi Inoguchi,Hisashi Isonaga , "High-speed detection system of Audiofingerprint with hardware" Proc. of the International Workshop on High Performance and Highly Survivable Routers and Networks (HPSRN 2007) ISBN: 978-4-9900330-6-4, 1-8, Mar. 14, 2007

[8] 安田 浩, 安原 隆一, "コンテンツ流通" アスキー,2003

[9] エ・ビスコム・テック・ラボ , "ブロードバンドコンテンツ配信実践" 毎日コミュニケーションズ,

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

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

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ともに，ここに御礼申し上げます.

適切な御意見，御助言を賜りました本学の松澤照男教授，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，御意見を賜りました佐藤 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皆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長瀬 文彦氏,Li Qi 氏,Li Bo 氏, に深く感謝致します．ポストドクターの Yu Chen 氏，井口研究室の先輩である Sun Wei 氏，礒永 久史氏，近藤 裕貴氏，清水 昭尋氏，高畠 義和氏，松山 周平氏には様々なアドバイスを頂き，後輩である荒木 光一氏，伊藤 徹也氏，高橋 宏幸氏，中尾 哲也氏には日々の愚痴を聞いて頂きました.
また、プログラムに関して貴重なご意見を頂きました田中研究室の請園 智玲氏，日比野研究室の矢澤 慶樹氏，松澤研究室の太田 理氏にも感謝いたします.
本研究を進めるにあたり，素晴らしい開発環境を提供して頂いた北陸先端科学技術大学院大学及び xilinx 社に感謝いたします.

最後に，長い間、私を支えて下さった福島の家族に深く感謝致します.