平成１５年度　研究開発成果報告書

研究開発課題：

「ＰＣなどオープンアーキテクチャーデジタル放送受信機に対応する
権利保護システムの研究開発」

目次

１　研究課題の背景

１９９９年末のＢＳデジタル放送開始を皮切りに、２００２年には１１０°ＣＳデジタル放送が開始され、図１－１のように２００３年は出荷台数が順調であり、２００４年も特に薄型ＰＤＰテレビの出荷増が見込める状況となっている（社団法人電子情報技術産業協会調べ)。

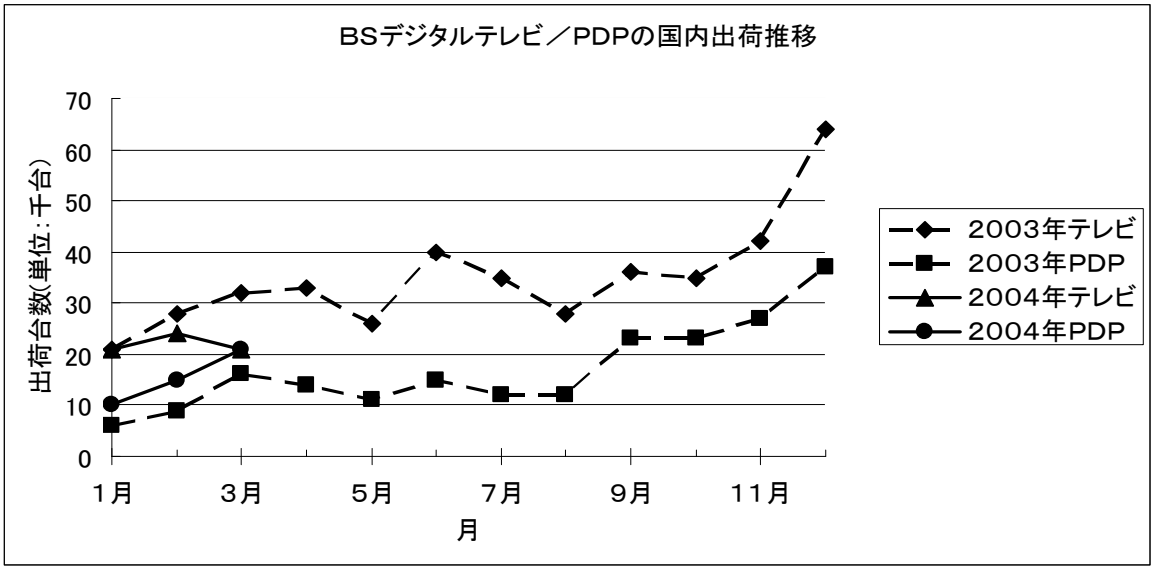


図１－１

また、ＢＳデジタル放送の普及率は２００４年４月末で約５６４万世帯（ＮＨＫ調べ）となり、受信機器の総出荷台数も２００４年４月末で約３３６万台（同）と、順調に普及している。更に、２００３年１２月より一部地域で開始された地上デジタル放送への対応受信機器の出荷台数は、表１－１のように順調に推移し、２００４年３月末までの累計出荷台数は約７２万２千台となった（社団法人電子情報技術産業協会調べ)。

年	月	累計出荷台数
２００３年	１１月まで	３１万１千台
	１２月まで	４８万台
２００４年	１月まで	５３万５千台
	２月まで	６０万６千台
	３月まで	７２万２千台

表１－１

なおこれには、ＣＡＴＶ経由の視聴形態での契約者数が入っていないため、実際の視聴者数は累計出荷台数を上回ることが十分に考えられる（総務省の予想では、約７１０万世帯が２００４年１２月からに視聴可能となっている）。

ちなみに、総務省が２００４年１月に行った調査では、約４人に３人が地上デジタルの放送の開始を認知している、という結果が得られている。

デジタル放送の開始・普及と共にもう一方で、これらデジタル放送受信機にハードディスクを搭載させ、受信したデジタルＡＶコンテンツをハードディスクに蓄積、コンテンツ課金やタイムシフト再生など行う機器・サービスが普及している。特にハードディスク内蔵録画装置は２００４年３月現在で、前年比約２４０％以上という驚異的な伸びを示している（社団法人電子情報技術産業協会調べ）。また受信機でのハードディスクの蓄積媒体の存在を前提としたサーバー型放送の規格化も策定されている。これに伴いデジタルＡＶコンテンツ複製や移動を制限するコンテンツ権利保護システム研究開発や導入が急務となっており、ＮＨＫと民放各社は、２００４年４月にＢＳデジタル・地上デジタルの各放送で、「１回だけ録画可能」という意味での「コピーワンス」制御方式を導入した。

過去の一例ではあるが、ＰＣから音楽をインターネット上に配信するナプスターは、コンテンツ複製や移動を制限する機能がないため著作権者からクレームが付き、サービスの停止を余儀なくされた。また最近では、Ｐ２Ｐファイル共有ソフト“Winny”の開発者や利用者が逮捕されている。ＰＣなど内部が公開されたオープンアーキテクチャー機器上での権利保護機構の困難さを示した例と言える。

現状のＰＣ上ソフトウェアベースのシステムでは、本来的にコンテンツなどの情報の複製／移動、その他制御が可能であることが前提で、これを制限することはＰＣシステム全体の構想に合わないといったこともある。

しかし、今後更にデジタル放送が一般化し、ＰＣ上でもデジタルＴＶ視聴機能が実現されるようになると、デジタルＡＶコンテンツもＰＣ上に保存されるようになり、ますますＰＣ上のコンテンツ権利保護が重要となる。デジタルＡＶコンテンツの著作権者は、ＰＣ上に強固なコンテンツ権利保護機構が構築されない限り、ＰＣ上でのデジタルＡＶコンテンツ保存を容認しないものと思われる（現在販売中のＢＳ・地上デジタル受信機能を持つＰＣであっても、デジタルＡＶコンテンツのハイビジョンサイズでの保存機能を有していない）。

一方、ＰＣ上で保存されたデジタルＡＶコンテンツの権利保護が有効な方法で実現すると、インターネット経由で世界中のＰＣにデジタルＡＶコンテ

ンツが流通する市場（いわゆるインターネット放送など）が開かれる可能性がある。

本研究開発は、ＰＣでのデジタル放送受信とデジタルＡＶコンテンツ保存を前提に、ＰＣソフトウェア上でのコンテンツ権利保護を実現する技術に関するものである。現状ＰＣソフトウェア上でのコンテンツ権利保護は、基本的に保護アルゴリズムを秘匿化することで行われている。そのため、ソフトウェアの解析を困難にする「難読化」などが中心である。しかし、これでは一旦アルゴリズムが解析されてしまうとコンテンツ権利保護が困難になるため、セキュリティレベルが低くなり、その強化が必要である。本研究は、ＰＣなど（１）ソフトウェア処理を主体とし、かつ（２）オープンアーキテクチャー構成のデジタル放送受信機のコンテンツ権利保護の研究開発に関するものである。

２　研究開発分野の現状

デジタルコンテンツの権利保護や流通に関しては、これまでにも各社、各団体により提案がいくつもされてきている。例えば，ＤＶＤvideo における著作権保護システムであるＣＳＳ方式や、リムーバブルメディア向けＣＰＲＭ方式、ＩＥＥＥ１３９４を用いた方式(DTCP)など存在する。

- CSS 方式(DVD における著作権保護システム)　詳細は非公開
- CPRM(Content Protection for Removable Media)方式　詳細は非公開だが、コピーワンス放送の「ムーヴ」にも使用されている方式。
- DTCP(Digital Transmission Content Protection)
  http://www.dtcp.com/data/dtcp_tut.pdf など

これらの方式においては、ハードウェアやソフトウェアの耐タンパー性が重要とされている。ハードウェアの耐タンパー技術はある程度確立されてきた技術であり、コンテンツ提供者サイドにもある程度理解されているものと考えている。このため、ハードウェアによる専用装置（デジタルＴＶ，ハードディスク内蔵録画装置など）の世界においては耐タンパー性を利用したコンテンツ保護システムの構築は可能となってきている。

しかし、ＰＣのようなオープンアーキテクチャー上でソフトウェアが動作するような世界では、全てをハードウェアで処理することへの抵抗や、全てをハードウェアで処理するアーキテクチャーが受け入れられ難い状況である。この為、ソフトウェアの耐タンパー性が強く求められている。

一般的にＰＣのようなオープンアーキテクチャー上でのソフトウェアの耐タンパー性は、ハードウェアよりも低いレベルであり、ソフトウェアの解析から完全に守ることは難しいとされている。現在、各社で実施されているソフトウェア耐タンパーの技術には大きく分けると以下の２つの技術が用いられている。

【難読化】

ソフトウェアに余分なコードを付加したり、簡単な計算処理をわざと複雑な計算ルーチンを用いて計算させることで、逆アセンブルによるコード解析に時間がかかるようにすることで耐タンパー性を持たせようとする方式。

【暗号化手段】

ソフトウェアを暗号化してハードディスク上に格納しておく方式。また、

ソフトウェアを細かいモデュールに分割して実行させ、モデュール間の相互認証などをさせる方式も存在する。内容は一般的には公開されておらず（方式を未公開にしないと安全性を保てないため）内容は不明だが、公開されている技術論文としては、

- 「逆解析や改変からソフトを守る」(日経エレクトロニクス 1998.1.5(no706)) 耐タンパなソフトウェア構造の提案論文
  が存在する。ただし、暗号化技術を用いた場合には、対象となるソフトウェアの攻撃だけでは解析することはできないが、ソフトウェアを復号するための別のソフトウェアが存在するわけで、これを含めて攻撃されるとソフトウェアの解析から守ることは困難となる。

更に近年になって、ＰＣそれ自体のアーキテクチャーに手を入れ、ＰＣとその上で動作するＯＳ(Operating System)をセットにして、耐タンパー性を高めようという動きがある。

米国 Microsoft 社は、“NGSCB: Next-Generation Secure Computing Base”を提唱している。

- http://www.microsoft.com/presspass/features/2002/jul02/0724palladiumwp.asp

簡単に書けば、NGSCB は既存のＰＣ内に、もう一つ、高度なセキュリティを持ったＰＣを設け、例えば内部のセキュアなＰＣの隠しメモリにソフトウェアを置くなどの方式で、解析や攻撃からソフトウェアを守るといった方式である。

同様に米国 Intel 社も、“LaGrande Technology”を提唱している。

- http://www.intel.com/technology/security/downloads/LT_overview_fall_idf03.pdf

しかし、どちらの方式でも、ＰＣそのもののアーキテクチャーの変更が必要であり、既存のＰＣでは対応できず、かつ、ＰＣのオープンアーキテクチャー性が失われていく可能性が大きい。

今回の研究では、上記のようなソフトオンリーのセキュリティーでは、現在の技術水準においての十分な安全性（耐タンパー性）の確保は困難であり（方式が分かってしまうと、ソフトウェアの安全性を保てない)、また現行のＰＣアーキテクチャーの全面的な変更をしないという方針のもと、詳細な検

討の上に抜き出した最小限のセキュリティー機能を搭載したハードウェア（ＴＲＭ:Tamper Resistant Module:化した「セキュアハード」）をベースに、ソフトウェアの安全性を確保する手段を研究開発したものである。

セキュアハードのコストを低くするため回路規模を小さくするよう検討すると同時に、セキュアハードの普及が容易なように、ＰＣＩバス等の公開されたバスにセキュアハードを接続することを前提に研究開発を進めてきた。

なお当社では、数年に渡って超流通やデジタルコンテンツ保護の研究開発を進めてきたという実績がある。

- 鳥居直哉，長谷部高行，武仲正彦，木島裕二：“超流通システムの試作”，電子情報通信学会技術研究報告，Vol.96，No.71(OFS96-10)，pp.1--5 (1996)
- 長谷部高行，鳥居直哉，武仲正彦：“超流通システムの試作（課金サーバ型）”，電子情報通信学会　基礎・境界ソサイエティ大会講演論文集, SA-5-6, pp.283--284 (1996)
- 木島裕二，長谷部高行，鳥居直哉：“超流通におけるコンテンツ流通のための課金機構の開発”，創造的ソフトウェア育成事業及びエレクトロニック・コマース推進事業 最終成果 発表会論文集 創造的ソフトウェア育成事業編，pp.701--704 (1998)
- 長谷部高行，木島裕二，鳥居直哉：“超流通における課金機構の開発”，情報処理学会研究報告, Vol.98, No.85 (98-EIP-2),pp.15--19 (1998)
- 田平孝彦、瀬野尾晴海、小川清隆、小檜山清之、秋山良太，“MPEG-TS（デジタルビデオ／オーディオ）セキュリティー方式の開発”、テレビジョン学会年次大会、No. 23-3（1996）

これらの研究がベースとなり、既に実際に製品運用されている以下の権利保護関連システムがある。それは、ＤＤＩポケット社が提供する Sound Market と呼ばれるサービスである。そのサービスで採用されている、ケータイ de ミュージック方式と呼ぶ方式は当社等により開発された方式である。

- http://www.keitaide-music.org/index_j.html

（２００３年３月１３日に、ＵＤＡＣコンソーシアムと改称、活動範囲を拡大している。http://www.udac-consortium.org/index_j.html）

これは、主に音声の権利保護が目的でＰＣのようなソフト主体のオープンアーキテクチャーシステムではないが、権利保護やコンテンツ課金に必要な以下の全ての機能が揃っている。

- ライセンスとコンテンツの分離技術

- ライセンス管理技術（メディアベースランセンス管理）
- 暗号化実装技術（ハードウェア、ソフトウェア）
- 相手認証や再送防止等といったプロトコル技術
- ハードウェア耐タンパー技術、など

今回の研究開発における課題は、さまざまなソフトウェアが動作するＰＣのようなオープンアーキテクチャー上において、ＰＣのアーキテクチャーの変更無しに、詳細な検討の上に抜き出した最小限のセキュリティー機能を搭載したハードウェアを搭載した形のソフトウェアベースのコンテンツ保護が、いかなる形式ならば可能なのかを研究し、その実装技術を開発すること。また、実際のデジタル放送にこれらの技術を適用した場合にＰＣなどのオープンアーキテクチャー上での動作が可能かを実証することである。

３　研究開発の全体計画

３-１　研究開発課題の概要

１９９９年末のBSデジタル放送開始を皮切りに２００２年は１１０°CSデジタル放送、２００３年１２月には地上デジタルが開始された。デジタル放送開始と共にもう一方で、受信機にハードディスクを搭載し、受信したデジタル AV コンテンツをハードディスクに蓄積しコンテンツ課金やタイムシフトなど行う新サービスが普及しつつある。受信機でのハードディスクの蓄積媒体の存在を前提としたサーバー型放送規格化も策定されている。これに伴いデジタル AV コンテンツ複製や移動を制限する権利保護システム研究開発が急務になる。

本研究は、PC でのデジタル放送受信を前提に PC ソフトウェア上で権利保護を実現する技術に関するものである。現状 PC ソフトウェア上での権利保護は、基本的に保護アルゴリズムを秘匿化することで権利保護している。そのためソフトウェアの解析を困難にする「難読化」などが中心である。しかし、これではアルゴリズムが解析されると権利保護が困難になるため、セキュリティレベルが低く強化が必要である。本研究は、PC など（１）ソフトウェア処理主体かつ（２）オープンアーキテクチャー構成デジタル受信機の権利保護の研究開発に関するものである。

過去の例では、PC で音楽をインターネット配信するナプスターは、コンテンツ複製や移動を制限する機能がないため著作権者からクレームが付きサービスを停止された。PC など内部が公開されたソフトウェア処理主体機器での権利保護機構実現の困難さを示した例と言える。これは現在のソフトウエアベースのシステムでは、本来的に情報の複製／移動、その他制御が可能であることが前提でこれを制限することはシステム全体の構想に合わないためである。

今後デジタル放送が一般化し、PC 上でデジタル TV 機能が実現されるようになると音楽のみならずデジタル AV コンテンツも PC 上に存在するようになり、ますます PC 上のコンテンツ権利保護が重要になる。デジタル AV コンテンツの著作権者は、強固な権利保護機能を持たない限り、PC 上でのデジタル AV コンテンツを容認しないものと思われる。

一方で PC 上でのデジタル AV コンテンツ権利保護が一旦実現すると PC からブロードバンドインターネット経由で世界中にデジタル AV コンテンツが流通する可能性が開ける。

図 3-1 は一般的な既存ＢＳデジタル放送受信機能搭載 PC の機能ブロック図であり、デジタル AV コンテンツをハードディスク蓄積すると想定して

いる。信号の流れを概説すると以下のようになる。放送波は「デジタルチューナモジュール」より入力され、デジタル AV コンテンツを時分割多重の形で持つ MPEG - TS デジタルストリームが出力される。MPEG - TS デジタルストリーム中の視聴者が選択した番組（特定デジタル AV コンテンツ）は MULTI2 暗号化されており、「MULTI2 暗号復号モジュール」で復号される。復号されたデジタル AV ストリームはハードディスク蓄積のため再暗号化され、再暗号化デジタル AV コンテンツがハードディスク等の蓄積媒体に PCI バス等を経由し蓄積される。MULTI2 暗号復号に必要な暗号鍵等のライセンス情報は「B-CAS カード等ライセンス保持モジュール」から出力される。また、再暗号化に必要なライセンス情報は、「ライセンス生成モジュール」で生成される。

HDD 蓄積された暗号化デジタル AV コンテンツを再生する場合は、「暗号復号モジュール」に転送され復号され、復号デジタル AV 情報は、「MPEG 等デコードモジュール」で伸長処理される。伸長デジタル AV 情報は、「グラフィックモジュール」でグラフィックがオーバレイされ、その後アナログ出力の場合はアナログ著作権保護情報が付加され出力される。デジタル出力の場合は、デジタル著作権保護情報が付加され出力される。

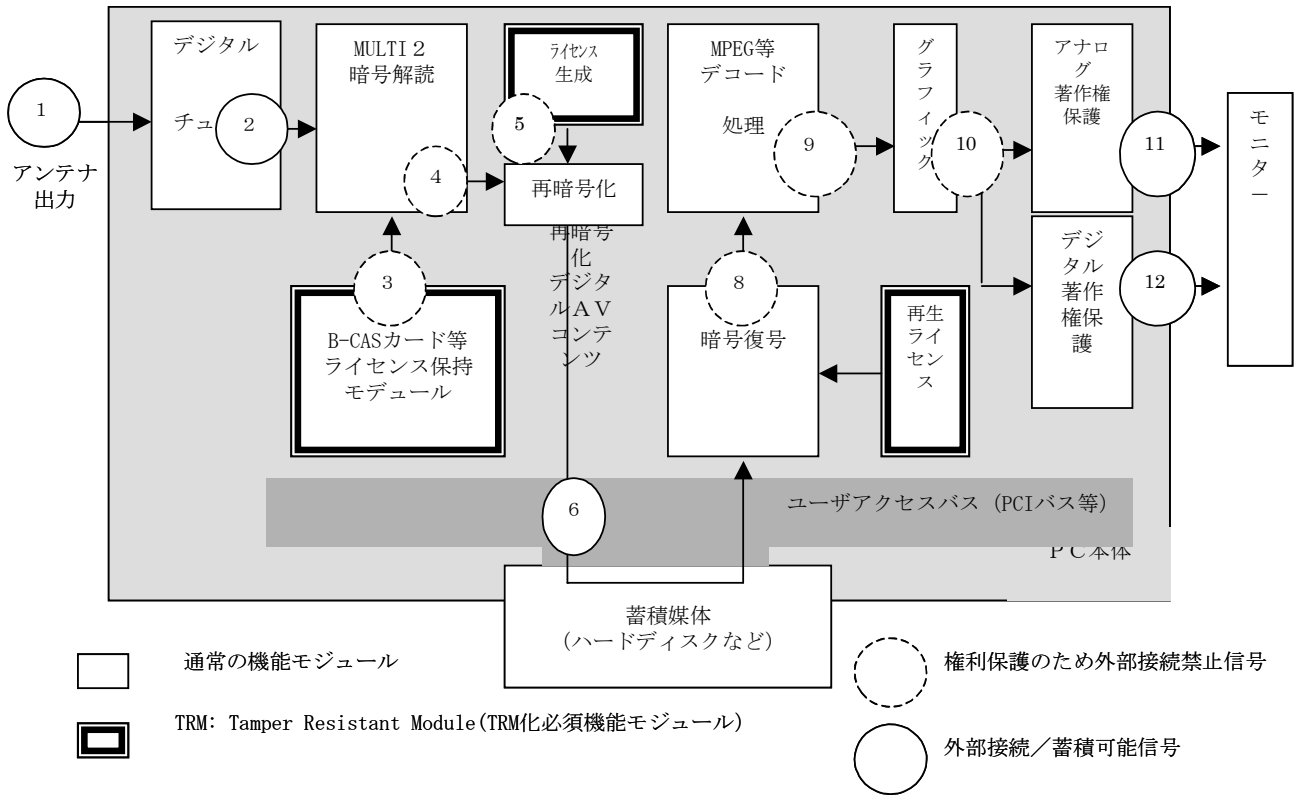


図 3-1　ＰＣ等ユーザアクセスバスを有するデジタル放送受信機の機能ブロック図（例）

図 3-1 で暗号解読鍵等のライセンス情報を持つ機能モジュールは TRM（Tamper Resistant Module）である必要がある。また、その他のモジュールは TRM である必要はないが図 3-1 で点線の信号（信号４、５、８、９、１０など）は、権利保護上、外部流出禁止信号である。例えば、暗号復号後の信号８などが外に漏れると著作権者の権利を保護することができなくなる。

また、図 3-1 は機能モジュールであり、各モジュールともソフトウェアで実現してもハードウェアで実現しても機能的には同様に動作する。

図 3-2 は、LSI（ハードウェア主体）で図 3-1 の機能を実現した場合のブロック図である。太線内が LSI であり、ほとんどの機能が LSI 内に搭載され、LSI 全体を TRM（Tamper Resistant）化することで外への信号流出が防げ、比較的容易に権利保護が実現する。しかし、ハードウェアを搭載することは PC のコストアップに繋がる。特に MPEG デコード機能は回路規模が大きく高価格である。

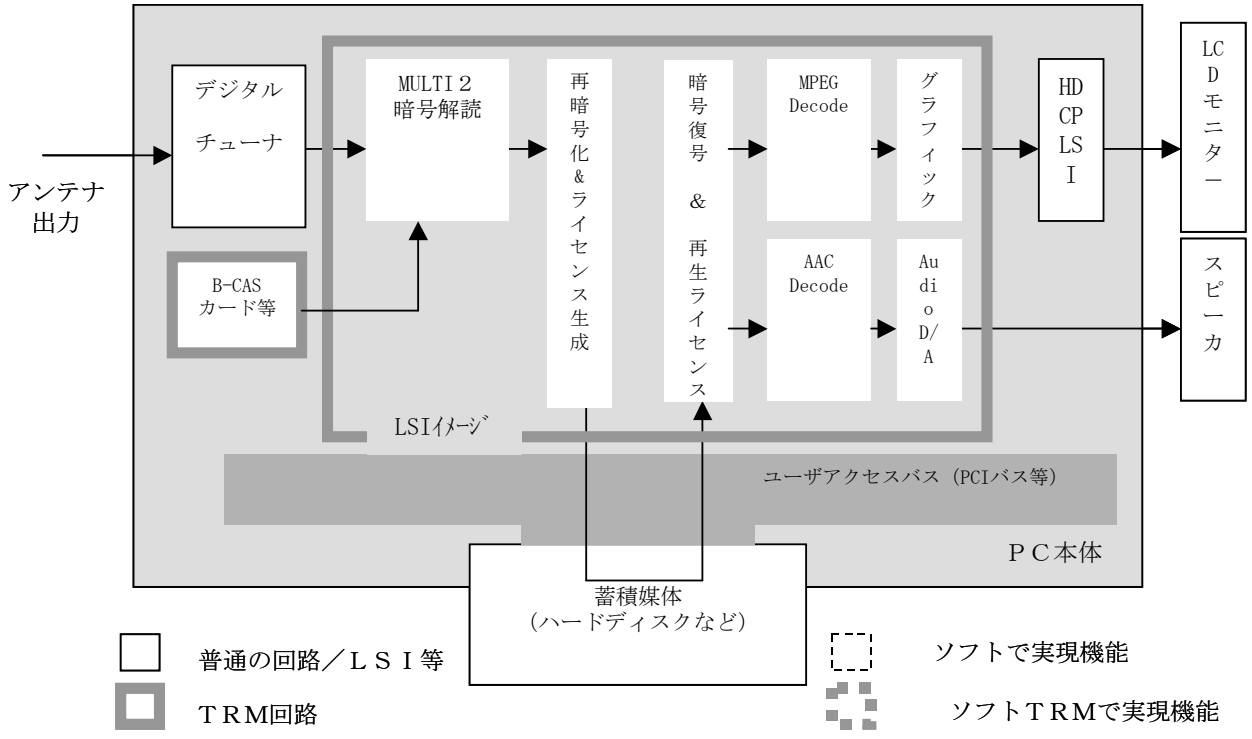


図 3-2 想定されるハード処理主体の実施例

また PC はソフトウェア主体の装置であるが、この構成ではソフトウェア／プロセッサー等の資源が有効利用されていると言い難い。さらに、既に DVD プレイヤー（MPEG　MP@ML 圧縮デジタル情報のデコードが必要）が一般の PC 上でソフトウェア処理されていることを考えるとプロセッサー

の処理能力が今後増大した場合、デジタル放送（MPEG　MP@HL 圧縮デジタル情報のデコードが必要。MP@HL は MP@ML の 6 倍の処理能力が必要）でも一般 PC でソフトウェア処理可能になるのは時間の問題と考えられる。

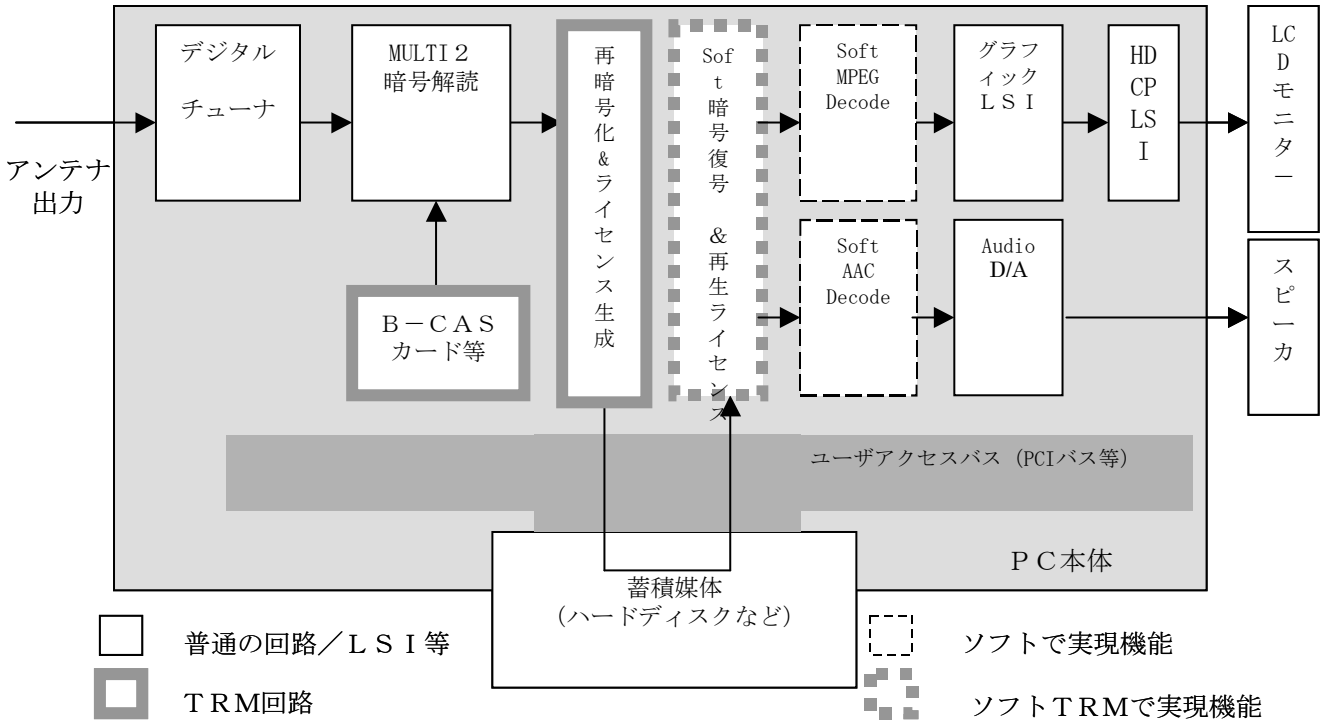


図　3-3　想定されるソフト処理主体の実施例

図　3-3 は、ソフトウェア処理を主体とした場合のシステム構成例である。ハードウェア処理した場合に回路規模が大きくコストアップにつながる「MPEG デコード」、「AAC 音声デコード」や「暗号復号」処理などがソフトウェア処理されている。従って、ソフトウェア処理が実現した場合、ハードウェアで実現したときと比較してかなりのコストダウンが図られると考えられる。

ソフトウェア処理した場合の課題は、「暗号復号」に必要な暗号解読鍵の保護や「MPEG デコード」など、各ソフトウェア処理モジュールからの出力信号の保護である。

以上の検討で PC のようなソフトウェア主体装置の場合、ハードウェア処理を持ち込むことはソフトウェア主体装置の持つ本来の可能性が大幅に制約されることが分かる。

従って、本研究方針は上記検討で「PC などソフトウェア処理主体の装置の可能性を損なわないデジタル AV コンテンツの権利保護システム」に絞った方がより本質的であることも分かる。しかし、一方ですべてソフトウェア

のみの処理では、強固なデジタル AV の権利保護は困難と思われる。すべてソフトウェア処理とした場合、最終的に暗号鍵等の機密保持が非常に困難と考えられるためである。

現状の技術水準を考慮した場合、処理の大半をソフトウェア処理するが最低限の信号処理はハードウェア化（セキュアハード）したデジタル放送対応の権利保護システムの研究開発が現実的であると考える。

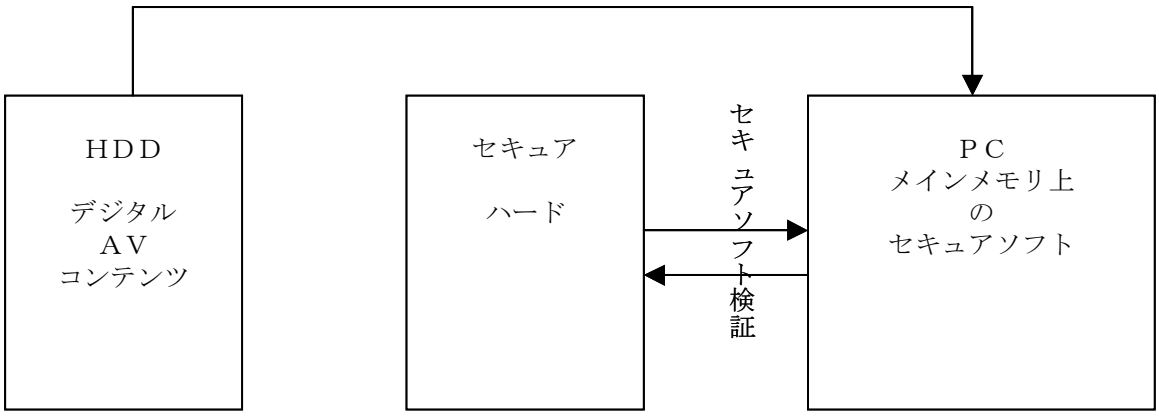


図 3-4 想定されるセキュアソフトとハード

本研究で想定するデジタル TV 機能のシステム構成ブロック図を図 3-4 に示す。このシステム構成における課題は、以下のようなものである。

ハードディスク内のデジタル AV コンテンツは、PC のメインメモリ上のセキュアなソフトウェアで処理される。一方でセキュアなソフトウェアの安全性を保障するのはセキュアハードになる。最終的な信頼点としてのセキュアハードを活用として、権利保護アルゴリズムを構築するという点が大きな研究開発課題になる。

またこの権利保護システム全体にリアルタイム処理性が要求される。処理するコンテンツは、基本的にハイビジョン動画像（MPEG MP@HL レベル）であり、PC 性能をギリギリまで使用しないと画像のリアルタイム処理は困難と考えられる。この中にセキュアなための処理を PC から見てあまり負担のかからない範囲でどう盛り込むか課題になる。安全性と PC 性能のトレードオフをどう考えるか、ハードディスク／セキュアハード／セキュアソフトウェア間の機能分担をどう捕らえるか、リアルタイム性というセキュリティーと別の視点から検討が必要である。

さらに別の視点としてコストがあげられる。２００４年以降店頭販売の一般 PC で活用されることを前提とした場合、コストを押さえるため、権利保護システム内の特殊部品はセキュアハードのみとして検討する必要がある。

また、普及が容易なようにセキュアハードはPCカード搭載を前提とし、PC接続するときのインターフェースは PCI バスなどユーザーに公開された汎用なものとする必要がある。

研究手法として、まずセキュアハード／ソフトの概略構成を初年度に検討し、大まかに権利保護システムの構想を固める。この時、図 3-4 の（1）ハードディスク／（2）セキュアハード／（3）PC 上のセキュアソフトの機能分担や想定されるセキュリティーホールを、特定し対策を検討する。１４年度は概略構成に従って、セキュアハード、セキュアソフトの基本仕様の検討を行い、機能試作を行う。

３-２　研究開発目標

３-２-１　最終目標（１５年度３月）

PC 上、ソフト処理主体でデジタル TV の権利保護を実現する基盤技術を完成させることを最終目標とする。

具体的にはセキュアソフト実現のための（1）セキュアハード コア LSI、（2）セキュアハードを利用したセキュア MPEG ビデオ処理ソフト／セキュアオーディオ処理ソフトなど、（3）「権利保護 PC カード」などを完成させる計画とする。「権利保護 PC カード」の中にセキュア コア LSI 以外に地上デジタル放送受信機能も搭載を計画する。また、旧 PC から新 PC に買い替えた時に必要な蓄積コンテンツのセキュアな視聴権移動機能など PC 上でデジタルＴＶ受信機を構成するための基本機能を揃えるものとする。セキュリティー強度は、基準となる物差しがないので言及困難であるが、基本的にオールソフト処理より強い、放送など公共性の高い網で適用可能なレベルを目指す。またリアルタイム処理性能は、２００４年の PC 上で MPEG MP@HL レベルの画像を権利保護しつつ一般の視聴者が違和感なく動画像を視聴出来るレベルとする。

３-２-２　中間目標（１４年度３月）

基本の機能レベルの完成目標を平成１５年３月とする。セキュアハードの回路、セキュアソフトを機能レベルで完成させ、実際に PC 上でデジタルテレビ画像を見られるようにすることを計画する。

但し、FPGA 搭載実験ボードで製作されたハードウェアは物理的なサイズが大きく、PC カード等に実装出来るレベルでない。また FPGA は高価であり、サイズ的にもコスト的にも製品展開は不可能なレベルに留まる。製品展開が期待出来るレベルにするためには、ＦＰＧＡ搭載実験ボードのＬＳＩ化が不可欠あり、またリアルタイム性もこの段階では十分なレベルに到達して

いないと想定する。リアルタイム性の目標として毎秒５フレーム（最終目標は一般視聴者が違和感なく動画像を視聴出来るレベル）の処理性能を想定する。

３-３研究開発の年度別計

（金額は非公表）

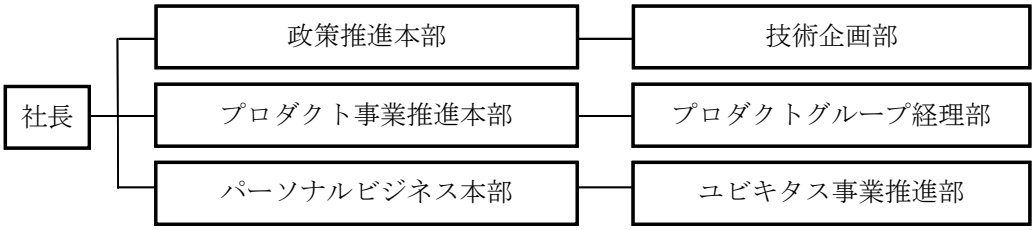
研究開発項目	１３年度	１４年度	１５年度	計	備　考
アーキ検討、実験ボード開発 設計、試作（ＦＰＧＡ）、 ソフトウェア開発 改良、ＬＳＩ化、ＰＣカード開発	→	→	→		
間接経費					
合　　計					

注）１　経費は研究開発項目毎に消費税を含めた額で計上。また間接経費は直接経費の３０％で計上（消費税含む。）。
　　２　備考欄に再委託先機関名を記載。

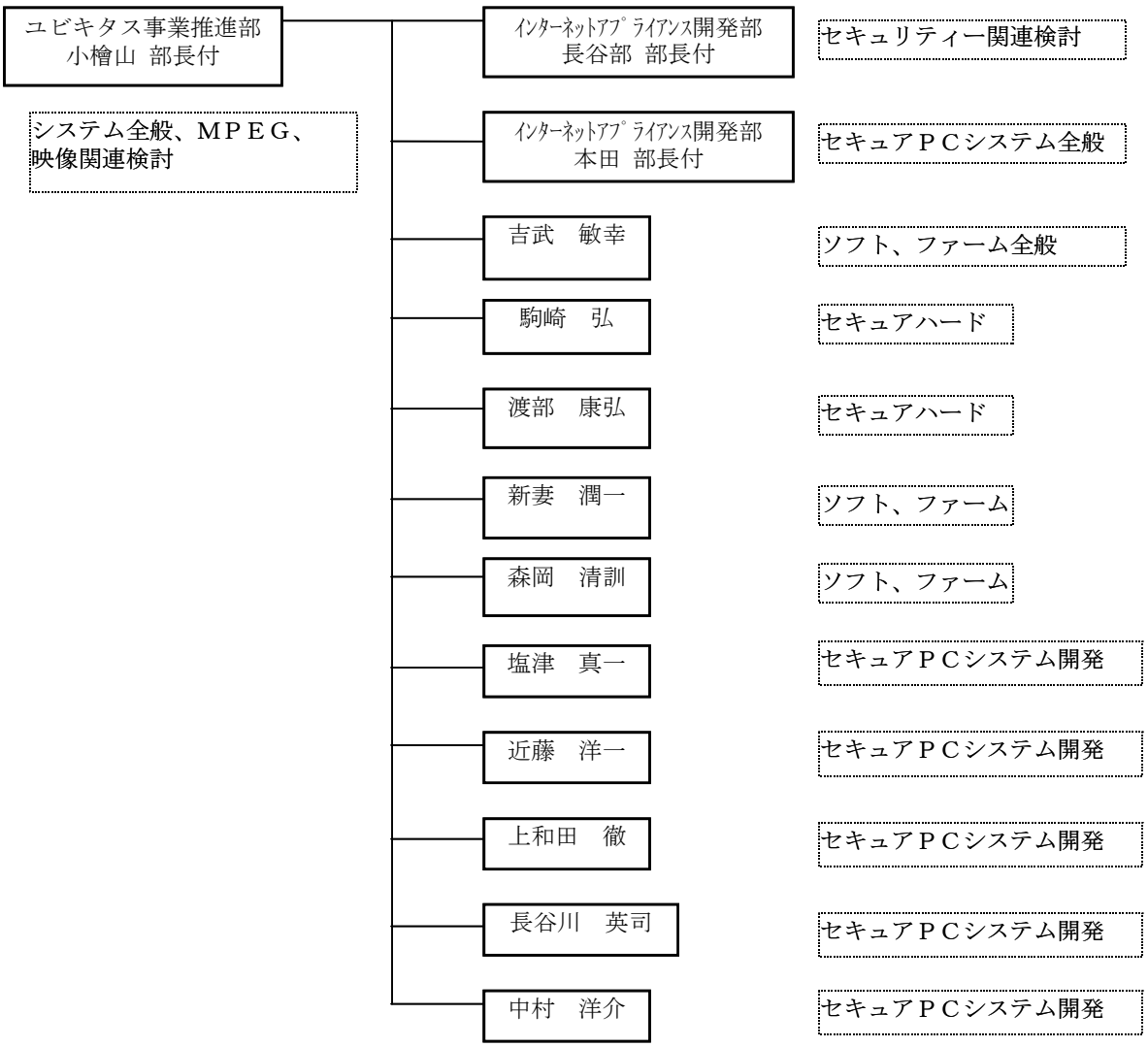
３-４　究開発体制

３-４-１　研究開発管理体制

（注　受託者の経理部門の体制、経理責任者（所属、氏名、電話、ＦＡＸ、Ｅ－メールの連絡先）を含む。）



３-４-２　研究開発実施体制



４　研究開発の概要

４-１研究開発実施計画

４-１-１　研究計画内容

４-１-１-１　平成１３年度

初年度の１３年度は、PC 型のセキュアデジタル放送受信機のセキュアハード／ソフトの概略構成を検討し、大まかに権利保護システムの基本アーキテクチャを固める。またこの構想に従い、FPGA（Field Programmable Gate Array）を使用した構成でセキュアハード実験ボードの基本部分を製作する。この実験ボードは１４年度以降のセキュアデジタル放送受信機の性能や機能評価などを行うための実験プラットフォームという位置付けで製作する。１４年度以降の実験などで改良点や問題が発覚した場合、その時点でハードウエアの設計を容易に変更出来るように FPGA（Field Programmable Gate Array）構成とする。

なお、実験ボードは、各種改良を経たのち、本研究開発の最終年度には LSI 化し PC カード搭載を目標とし、本研究開発が目標とする権利保護システムの「核」になる部分とする。

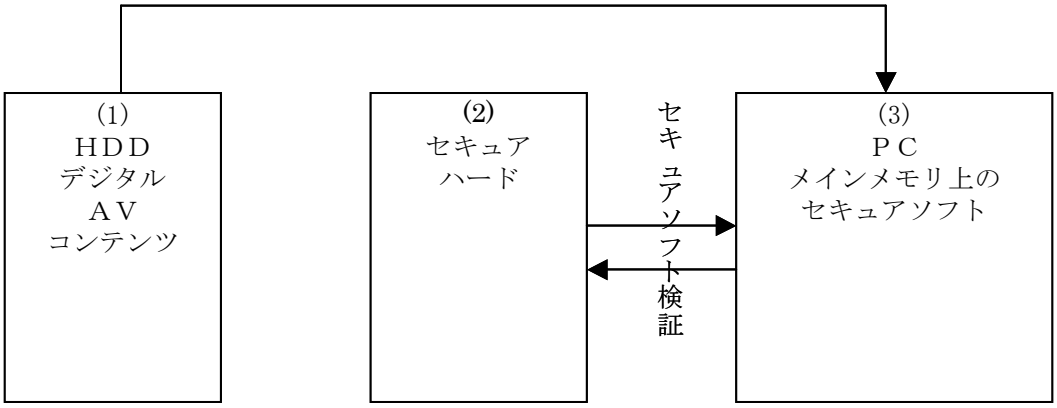


図 4-1　想定されるセキュアソフトとハード

PC 型のソフトウェア処理主体のセキュアデジタル放送受信機は、セキュリティーという側面から見ると図 4-1 に示すブロック図のような構成が考えられる。本図に従い、平成１３年度はセキュアデジタル放送受信機の基本アーキテクチャの決定など今後の方向付けを行う。

図 4-1 でデジタル放送受信機は、（1）ハードディスク、（2）セキュアハード、（3）PC のメインメモリ上のセキュアソフトという基本要素から構成される。ハードディスク内に放送されたデジタル AV コンテンツが蓄積されており、これが PC メインメモリ上のセキュアソフトでデコード処理され、視聴される。一方でセキュアソフトの安全性（デコード処理後のデジタル AV コン

テンツの著作権など権利保護機能）をセキュアハードが保証する。以上の基本構成を前提に１３年度研究課題として以下を検討する。

１）最終的な信頼点としてのセキュアハードの活用法の検討。

２）リアルタイム性とセキュリティーのトレードオフの検討。システム全体にリアルタイム性が要求される。処理コンテンツ（デジタル AV コンテンツ）は、基本的にハイビジョン動画像（MPEG　MP@HL レベル）であり、PC 性能をギリギリまで使用しないとリアルタイムソフト処理は困難と考えられる。この中にセキュア処理を PC から見てあまり負担のかからない範囲でどう盛り込むか。セキュリティー（安全性）と PC 性能のトレードオフをどう考えるか、ハードディスク／セキュアハード／セキュアソフト間の機能分担をどう捕らえるか、リアルタイム性というセキュリティーと別の視点から検討する。

３）さらに別の視点としてのコストがある。本技術を２００４年以降店頭販売の一般 PC で活用されることを前提とした場合、コスト削減のためシステム内の特殊部品はセキュアハードのみとすることが望ましい。またセキュアハード普及を容易にするためセキュアハードは PC カード搭載とし、PC 接続時のインタフェースは PCI バスなどユーザーに公開された汎用なものが望ましい。

以上のようなことを前提とし、セキュアハード／ソフトの概略構成を１３年度に検討し、大まかなシステム構想を固める。

また本システム構想に従い、１３年度は FPGA（Field Programmable Gate Array）構成でセキュアハード実験ボードを製作する。この実験ボードの最終目標はこれを LSI 化し、図 4-1 のセキュアハードとすることにある。実験ボードにより１４年度以降、上記で検討したセキュアハードとセキュアソフト間で安全性検証のための特別な通信などを具体化し、検証実験を行う。実験ボードの基本部分を FPGA 構成とすることでセキュア化の研究が進み問題が発覚した時点でのセキュアハードの設計変更を容易にする。従って、このボードはセキュリティーシステム試作／実証のための基本プラットフォームとなるよう構成する。

なお１３年度はセキュアシステム構想を固めることが主眼であり、セキュアハード実験ボードの中の FPGA の具体設計は、１４年度以降とする。但し、セキュアハードには FPGA 以外にプロセッサー、プロセッサー周辺回路、メモリ、PCI インタフェース回路などの搭載するため、プロセッサー周辺、PCI インタフェース回路の動作確認などは１３年度に行うものとする。

4-1-1-2　平成１４年度

１３年度の成果を踏まえ、基本機能レベル（基本デジタル放送受信機能）の完成目標を平成１４年度とする。セキュアハードの回路、およびセキュアソフトを機能レベルで完成させ、実際に PC 上でデジタルテレビ画像を見られるようにする。

但し、セキュアハードは FPGA（Field Programmable Gate Array）搭載実験ボードで試作されるため、物理的なサイズが大きく、PC カード等に実装

出来るレベルでなく、また高価なものとなる。従って、サイズ的にもコスト的にも製品展開にはさらに開発が必要なレベルに留まる。製品展開が期待出来るレベルにするには、FPGA 搭載実験ボードの LSI 化が必要であり、これは１５年度の目標となる。またセキュアソフトなどのリアルタイム性も１４年度の段階ではさらに開発が必要なレベルと想定する。具体的なリアルタイム性の目標として毎秒約５フレーム（最終目標は一般視聴者が違和感なく動画像を視聴出来るレベル）の処理性能を想定する。

以上の計画を以下のような手順で実施する。１３年度に決定した基本アーキをベースに研究要素を特定し、それぞれの要素ごと検討を行う。

まず、図 4-2 に示すような基本アーキテクチャをセキュアシステムとして展開していく。

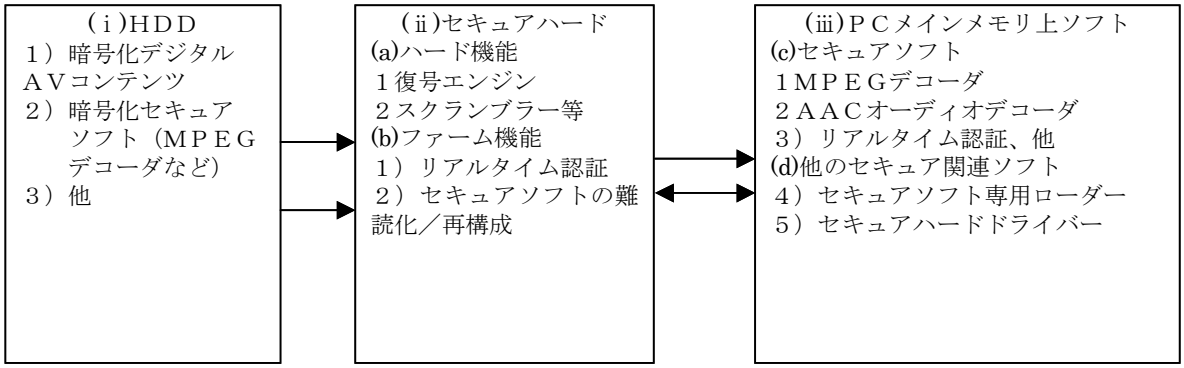


図 4-2　１３年度の研究開発で想定されたセキュアソフトとハードの関係

全体基本アーキテクチャは以下のように考える。HDD 内に（1）暗号化デジタル AV コンテンツ、（2）暗号化セキュアソフト（MPEG デコーダや AAC オーディオデコーダなど）が存在する。デジタル AV コンテンツ再生には、暗号化セキュアソフトをセキュアハードに読込み、復号する。復号セキュアソフトに対し、セキュアハード内で難読化／ソフト再構成などセキュア処理を加えた上で PC メインメモリ上に転送する。転送はセキュアソフト専用ローダーが行う。

ロードされたセキュアソフトは、セキュアハードがリアルタイムに認証する。常に同じ手順で認証したり、セキュアソフトが常に同じプログラムコードになっていたりすると PC 上のセキュアソフトや認証アルゴリズムを解析され、成りすましにより処理途中でデジタル AV コンテンツを盗まれる危険がある。これを防ぐため、セキュアハード内で毎回セキュアソフトを再構成する。再構成された毎回違うプログラムコードでセキュアソフトを実現し、さらに毎回（例えば１時間ごと）違う手順で認証することで、クラッカーに解析に必要な時間を与えないようにする。これにより、従来の難読化主体のソフトウェアと比較して、大幅に安全度の向上したソフトウェアを提供することを考える。セキュアハード上のファームウェアは、ソフトを半ば自動的に再構成し、毎回違う認証手順を半ば自動発生させる処理を行う。ここが、研

究開発のキーポイントとなると考える。

また、HDD 内に暗号化された状態で蓄積されているデジタル AV コンテンツは、セキュアハードで復号され、さらにスクランブルされて PC メモリ上のセキュアソフトに転送される。PC メモリ上のセキュアソフト（MPEG ビデオデコーダなど）は、デジタル AV コンテンツのデスクランブルや伸張処理などを行う。

１４年度は、上記のようにして図 4-2 に示す基本アーキテクチャをベースにして実際にセキュアソフト、セキュアハード等を試作し、検証を行う。この目標レベルは、最初に記載したとおり、基本機能レベルで完成を目指す。

１４年度で研究開発を要する項目は、図 4-2 で示した全体の基本アーキテクチャ、セキュアハードの（a）ハード機能、（b）ファーム機能、PC メインメモリ上の（c）セキュアソフト、（d）その他のセキュア関連ソフトなどである。また、上で挙げた項目以外にも「基本デジタル放送受信機能」を実現するために必要な付属的機能は存在する。例えば GUI（Graphical User Interface）による簡単な全体制御機構が必要と考えられるが、それらも必要に応じ開発する。

実際の研究開発は、以下のように行う予定である。まず、全体基本アーキテクチャを前述のように想定し、想定に沿ってセキュアハード、PC メインメモリ上のソフトなどセキュアシステムを実験的に構築する。最初に構築されたシステムは、必ずしも完全なものでなく、一部機能のみ持つ暫定的なものでも良いこととし、順次機能を追加していくことにより完成度を高める。つまり、上述の要素の中の幾つかのみが実装されている状態であり、各要素が未完成であっても、その要素に関してある程度の評価が可能であれば良いと考える。例えばセキュアソフトの中の MPEG ビデオデコーダは、当初リアルタイムで動作しなくても、高速化のためのチューニングなど実験可能なレベルであれば良いとする。このようにして出来るだけ、各要素を個別に評価実験出来るような体制にして研究開発を進める予定としている。評価・検証実験の過程で問題が発覚した場合は、基本的には要素ごとに改良研究を行う。また、基本アーキテクチャに関連した問題が発覚した場合は、セキュアハード等の構成まで戻って再検討を行う必要があるが、セキュアハード回路は FPGA 構成になっているため迅速な設計変更が可能である。より具体的には、以下のような手順で開発を進める。

１）まず、セキュアハードの暫定版を開発し、セキュアハードの機能が確認できるレベルのファームやソフトを立ち上げる。

２）その後、ソフトやファームの最低限の機能が評価できるようなレベルで立ち上げる。セキュアハード上のファーム（セキュアソフト再構成機能など）と PC メインメモリ上のソフト（MPEG ビデオデコード機能など）は、必ずしも同時に立ち上げる必要はなく個別に立ち上げることも可能である。すなわち、セキュアソフト再構成機能は、MPEG ビデオデコーダが動作していなくとも対象となる単純な評価用ソフトがあれば、再構成機能の評価実験は可能と考えられる。また、セキュアソフト（MPEG ビデオデコーダなど）は、セキュアハードがなくともある程度の評価実験は可能であるし、セキュ

アハードの機能の一部を使用しながら検証を行うことが可能である。

３）最終的に、各要素を組み合わせて、機能レベル（具体的な１４年度のリアルタイム性目標は毎秒約５フレーム）で「基本デジタル放送受信機能」を実現させる。

上記のように本研究開発は、図 ４-2 の（ｉ）HDD,（ii）セキュアハード、（iii）セキュアソフトの３基本要素をベースに如何にセキュアなシステム（セキュア MPEG ビデオデコードソフトなどを含む基本デジタル放送受信システム）を構築するかが一つの課題となる。

一方で研究開発要素ごとの研究開発も必要である。

１）セキュアハードは、地上デジタル放送関連規格にどう対応するか検討する必要がある。さらに Multi2 復号、ローカル暗号／復号回路など、平成１５年度の LSI 化を前提に開発や機能レベルの実証を行う。

２）また、セキュアハード内のファームはセキュアソフトを再構成する機能やセキュアソフトの保護を行う機能を持ち、本権利保護システムの大きなポイントと位置付けられる。セキュアソフトにリアルタイム認証アルゴリズムを挿入するなど、「セキュアソフト再構成手法」を検討する予定である。ここでのポイントは、（１）「成りすまし」攻撃を防ぐような解析困難なレベルの再構成の実現、（２）セキュアハード内のメモリなど限られた資源の中での再構成機能の実現だと考えられる。セキュアソフトの保護や、リアルタイム認証の具体手順など詳細検討を行っていく。詳細検討において、或いは評価実験の過程でセキュリティーホールなどの検討を行い、より安全な手法を追求する。そのため、前述のように可能な限り研究要素ごとに独立に研究開発できる体制を敷いて検討を進める。

３）セキュアソフトなど PC メインメモリ上の関連ソフトに関しても、以下のような検討が必要である。（１）地上デジタル放送対応高速 MPEG ビデオデコーダ、（２）オーディオデコーダ、（３）リアルタイム認証などのセキュア機能、（４）セキュアソフト専用ローダー（５）セキュアハードドライバーである。また、伸張されたビデオ映像、オーディオの同期処理を含めた統合システムとユーザーが操作するための GUI（Graphical User Interface）が必要である。対象となる OS については、１４年度も引き続き検討するが、目標はできる限り OS に依存しないシステムを考える。OS を調査し、図 ４-2 の専用ローダーやセキュアハードドライバーなどでできる限り OS 依存部を全面的にカバーし、ソフト本体は OS に依存しないような手法を考える。ローダー、ドライバに関しては対象 OS を Microsoft Windows として検討を進める。MPEG デコーダは、セキュアハードで再構成した上での、処理速度の向上がポイントとなる。今後の検討に依存するが、まず MPEG ビデオデコードの機能的な動作を確認し、その後、再構成したことによる影響などを検討し、最後に処理性能向上を図ることを目標とする。１４年度は、５フレーム／秒の処理速度が目標であり、本格的な高速化は次年度以降とする予定である。再構成によりなるべく処理性能に影響が出ないようにすることが研究開発の一つのポイントとなると考えられる。オーディオデコードに関しても、一定の開発は必要だが、MPEG ビデオデコードに準じた開発が可能であると判断す

る。

４-１-１-３　平成１５年度

PC 上で、ソフト処理主体でデジタル TV の権利保護を実現する基盤技術を完成させることが１５年度（最終年度）目標である。

１４年度で機能レベル試作したハードウェア及びソフトウェアは、まだ製品化を期待出来るレベルにない。これを製品コアとして利用が期待出来るレベルにまでするのが平成１５年度の目標となる。このため、具体的に以下のような研究開発を行う。

（１）LSI 化研究開発

平成１４年度試作の FPGA（Field Programmable Gate Array）搭載実験ボードは、試作が目的であり、物理的サイズやコストなどさまざまな要素のため、製品展開困難なレベルである。これを、製品展開が期待出来るレベルにするため、FPGA 回路、外付けプロセッサー、周辺回路などの LSI 化を図る。出来る限り最終目標であるデジタル放送対応のセキュア PC カード（セキュアハード）を意識し、その製品展開に必要と考えられる周辺回路を検討し、その検討にもとつき、内蔵すべき回路は LSI 内蔵し、インタフェースを用意すべき回路はインタフェースを用意する。LSI 化に際し、コスト、汎用性も意識した場合、必ずしも LSI 内蔵がベストでない場合もあり、その辺を十分に考慮し検討する。

（２）ソフト・リアルタイム化（改良）研究開発

セキュアハードを利用したセキュア MPEG ビデオ処理ソフト／セキュアオーディオ処理ソフトについては、以下のような研究開発を行う。即ち、平成１４年度の研究でソフトウェアは、機能試作レベルであり、リアルタイム性は持っていない。平成１５年度では、リアルタイム性を意識した改良を加える。具体的には、平成１４年度は、MPEG ビデオ処理に関しては、MP@HL（ハイビジョン画像）で毎秒５フレームの処理性能を達成している。これを平成１５年度は、一般視聴者が見て違和感のないレベルにまで高速化を図る。さらに、ソフトのセキュア化には、ソフトの「再構成」、「リアルタイム認証」、「リアルタイム・コード・スキャン」などの要素を組み込む必要があるが、リアルタイム性を意識し、処理性能の劣化を極力抑えながら、前述要素の組み込みを行う。これらの要素は、当然ながら、セキュアハード（１５年度 LSI 化予定）、セキュアハード内のファームウェアとも連動している（セキュアハードがセキュアソフトの動作を監視する）ため、必要となれば、セキュアハード、ファームウェアに対し、可能な範囲でフィードバックをかける。

なお、セキュリティー強度は、ソフトの「再構成」、「リアルタイム認証」、「リアルタイム・コード・スキャン」の頻度などに依存すると考えられる。セキュリティー強度に関しは、基準となる一般的な物差しがないので言及困難ではあるが、基本的にオールソフトウェア処理によるソフトセキュア化（一

般的に難読化と呼ばれる）より強い、放送など公共性の高い網で適用可能なレベルを目指す。

（３）PC カード化

権利保護 PC カードの中に、上記セキュアハード（LSI）以外に地上デジタル放送を受信するときに必要な機能を搭載する予定である。従って１５年度は、カード内に搭載すべき回路を検討し、具体的に実装する方法などを検討する。また、カード形状などの具体検討も行う。さらに、実際の製品を考えたときに必要と考えられる「旧 PC から新 PC に買い替えた時に必要な蓄積コンテンツのセキュアな視聴権移動機能（コア機能)」などの実装方法を検討する。

４-１-２　研究計画内容

平成１３年度　　（金額は非公表）

研究開発項目	第１四半期	第２四半期	第３四半期	第４四半期	計	備　考
アーキ検討、実験ボード開発				→		
間接経費						
合　　計						

単位：百万円

注）１　経費は研究開発項目毎に消費税を含めた額で計上。また、間接経費は直接経費の３０％を上限として計上（消費税を含む。）。
（合計の計は、「３－１の研究開発課題必要概算経費」の総額と一致）
２　備考欄に再委託先機関名を記載。

平成１４年度　　（金額は非公表）

研究開発項目	第１四半期	第２四半期	第３四半期	第４四半期	計	備　考
１）セキュアハードの研究開発 ２）セキュアハード上のファームウエアの研究開発 ３）ＰＣメインメモリ上のソフトの研究開発	→	→	→	→		
間接経費						
合　　計						

単位：百万円

注）１　経費は研究開発項目毎に消費税を含めた額で計上。また、間接経費は直接経費の３０％を上限として計上（消費税を含む。）。
（合計の計は、「３－１の研究開発課題必要概算経費」の総額と一致）
２　備考欄に再委託先機関名を記載。

平成１５年度　　（金額は非公表）

研究開発項目	第１四半期	第２四半期	第３四半期	第４四半期	計	備　考
１）改良 ２）LSI 化 ３）PC カード開発	→	→	→	→		
間接経費						
合　　計						

単位：百万円

注）１　経費は研究開発項目毎に消費税を含めた額で計上。また、間接経費は直接経費の３０％を上限として計上（消費税を含む。）。
（合計の計は、「３－１の研究開発課題必要概算経費」の総額と一致）
２　備考欄に再委託先機関名を記載。

４-２　研究開発実施計画

４-２-１　平成１３年度

初年度の１３年度は、PC 型のセキュアデジタル放送受信機のセキュアハード／ソフトの概略構成を検討し、大まかに権利保護システムの基本アーキテクチャを固める。またこの構想に従い、FPGA（Field Programmable Gate Array）を使用した構成でセキュアハード実験ボードの基本部分を製作する。この実験ボードは１４年度以降のセキュアデジタル放送受信機の性能や機能評価などを行うための実験プラットフォームという位置付けで製作する。１４年度以降の実験などで改良点や問題が発覚した場合、その時点でハードウエアの設計を容易に変更出来るように FPGA（Field Programmable Gate Array）構成とする。

なお、実験ボードは、各種改良を経たのち、本研究開発の最終年度には LSI 化し PC カード搭載を目標とし、本研究開発が目標とする権利保護システムの「核」になる部分とする。

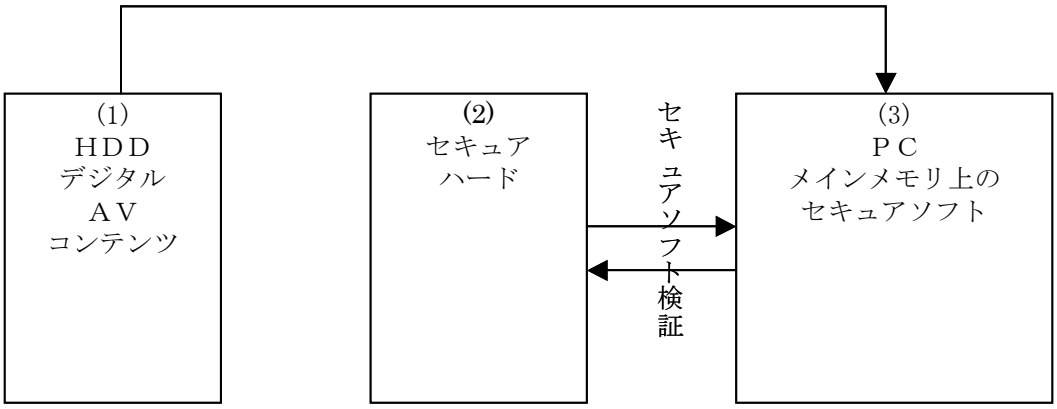


図 4-3　想定されるセキュアソフトとハード

PC 型のソフトウェア処理主体のセキュアデジタル放送受信機は、セキュリティーという側面から見ると図 4-3 に示すブロック図のような構成が考えられる。本図に従い、平成１３年度はセキュアデジタル放送受信機の基本アーキテクチャの決定など今後の方向付けを行う。

図 4-3 でデジタル放送受信機は、（1）ハードディスク、（2）セキュアハード、（3）PC のメインメモリ上のセキュアソフトという基本要素から構成される。ハードディスク内に放送されたデジタル AV コンテンツが蓄積されており、これが PC メインメモリ上のセキュアソフトでデコード処理され、

視聴される。一方でセキュアソフトの安全性（デコード処理後のデジタル AV コンテンツの著作権など権利保護機能）をセキュアハードが保証する。以上の基本構成を前提に１３年度研究課題として以下を検討する。

１）最終的な信頼点としてのセキュアハードの活用法の検討。

２）リアルタイム性とセキュリティーのトレードオフの検討。システム全体にリアルタイム性が要求される。処理コンテンツ（デジタル AV コンテンツ）は、基本的にハイビジョン動画像（MPEG　MP@HL レベル）であり、PC 性能をギリギリまで使用しないとリアルタイムソフト処理は困難と考えられる。この中にセキュア処理を PC から見てあまり負担のかからない範囲でどう盛り込むか。セキュリティー（安全性）と PC 性能のトレードオフをどう考えるか、ハードディスク／セキュアハード／セキュアソフト間の機能分担をどう捕らえるか、リアルタイム性というセキュリティーと別の視点から検討する。

３）さらに別の視点としてのコストがある。本技術を２００４年以降店頭販売の一般 PC で活用されることを前提とした場合、コスト削減のためシステム内の特殊部品はセキュアハードのみとすることが望ましい。またセキュアハード普及を容易にするためセキュアハードは PC カード搭載とし、PC 接続時のインタフェースは PCI バスなどユーザーに公開された汎用なものが望ましい。

以上のようなことを前提とし、セキュアハード／ソフトの概略構成を１３年度に検討し、大まかなシステム構想を固める。

また本システム構想に従い、１３年度は FPGA（Field Programmable Gate Array）構成でセキュアハード実験ボードを製作する。この実験ボードの最終目標はこれを LSI 化し、図 4-3 のセキュアハードとすることにある。実験ボードにより１４年度以降、上記で検討したセキュアハードとセキュアソフト間で安全性検証のための特別な通信などを具体化し、検証実験を行う。実験ボードの基本部分を FPGA 構成とすることでセキュア化の研究が進み問題が発覚した時点でのセキュアハードの設計変更を容易にする。従って、このボードはセキュリティーシステム試作／実証のための基本プラットフォームとなるよう構成する。

なお１３年度はセキュアシステム構想を固めることが主眼であり、セキュアハード実験ボードの中の FPGA の具体設計は、１４年度以降とする。但し、セキュアハードには FPGA 以外にプロセッサー、プロセッサー周辺回路、メモリ、PCI インタフェース回路などの搭載するため、プロセッサー周辺、PCI インタフェース回路の動作確認などは１３年度に行うものとする。

４-２-２　平成１４年度

１３年度の成果を踏まえ、基本機能レベル（基本デジタル放送受信機能）の完成目標を平成１４年度とする。セキュアハードの回路、およびセキュアソフトを機能レベルで完成させ、実際に PC 上でデジタルテレビ画像を見られるようにする。

但し、セキュアハードは FPGA（Field Programmable Gate Array）搭載実験ボードで試作されるため、物理的なサイズが大きく、PC カード等に実装出来るレベルでなく、また高価なものとなる。従って、サイズ的にもコスト的にも製品展開にはさらに開発が必要なレベルに留まる。製品展開が期待出来るレベルにするには、FPGA 搭載実験ボードの LSI 化が必要であり、これは１５年度の目標となる。またセキュアソフトなどのリアルタイム性も１４年度の段階ではさらに開発が必要なレベルと想定する。具体的なリアルタイム性の目標として毎秒約５フレーム（最終目標は一般視聴者が違和感なく動画像を視聴出来るレベル）の処理性能を想定する。

以上の計画を以下のような手順で実施する。１３年度に決定した基本アーキをベースに研究要素を特定し、それぞれの要素ごと検討を行う。

まず、図 ４-4 に示すような基本アーキテクチャをセキュアシステムとして展開していく。

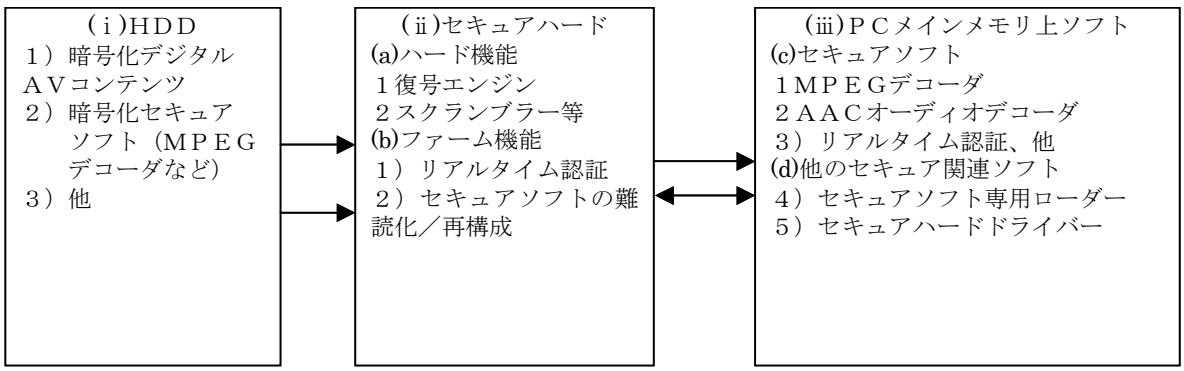


図 ４-4　１３年度の研究開発で想定されたセキュアソフトとハードの関係

全体基本アーキテクチャは以下のように考える。HDD 内に（１）暗号化デジタル AV コンテンツ、（２）暗号化セキュアソフト（MPEG デコーダや AAC オーディオデコーダなど）が存在する。デジタル AV コンテンツ再生には、暗号化セキュアソフトをセキュアハードに読込み、復号する。復号セキュアソフトに対し、セキュアハード内で難読化／ソフト再構成などセキュア処理を加えた上で PC メインメモリ上に転送する。転送はセキュアソフト専用ローダーが行う。

ロードされたセキュアソフトは、セキュアハードがリアルタイムに認証する。常に同じ手順で認証したり、セキュアソフトが常に同じプログラムコードになっていたりすると PC 上のセキュアソフトや認証アルゴリズムを解析され、成りすましにより処理途中でデジタル AV コンテンツを盗まれる危険がある。これを防ぐため、セキュアハード内で毎回セキュアソフトを再構成する。再構成された毎回違うプログラムコードでセキュアソフトを実現し、さらに毎回（例えば 1 時間ごと）違う手順で認証することで、クラッカーに解析に必要な時間を与えないようにする。これにより、従来の難読化主体のソフトウェアと比較して、大幅に安全度の向上したソフトウェアを提供することを考える。セキュアハード上のファームは、ソフトを半ば自動的に再構成し、毎回違う認証手順を半ば自動発生させる処理を行う。ここが、研究開発のキーポイントとなると考える。

また、HDD 内に暗号化された状態で蓄積されているデジタル AV コンテンツは、セキュアハードで復号され、さらにスクランブルされて PC メモリ上のセキュアソフトに転送される。PC メモリ上のセキュアソフト（MPEG ビデオデコーダなど）は、デジタル AV コンテンツのデスクランブルや伸張処理などを行う。

１４年度は、上記のようにして図 4-4 に示す基本アーキテクチャをベースにして実際にセキュアソフト、セキュアハード等を試作し、検証を行う。この目標レベルは、最初に記載したとおり、基本機能レベルで完成を目指す。

１４年度で研究開発を要する項目は、図 4-4 で示した全体の基本アーキテクチャ、セキュアハードの（a）ハード機能、（b）ファーム機能、PC メインメモリ上の（c）セキュアソフト、（d）その他のセキュア関連ソフトなどである。また、上で挙げた項目以外にも「基本デジタル放送受信機能」を実現するために必要な付属的機能は存在する。例えば GUI（Graphical User Interface）による簡単な全体制御機構が必要と考えられるが、それらも必要に応じ開発する。

実際の研究開発は、以下のように行う予定である。まず、全体基本アーキテクチャを前述のように想定し、想定に沿ってセキュアハード、PC メインメモリ上のソフトなどセキュアシステムを実験的に構築する。最初に構築されたシステムは、必ずしも完全なものでなく、一部機能のみ持つ暫定的なものでも良いこととし、順次機能を追加していくことにより完成度を高める。つまり、上述の要素の中の幾つかのみが実装されている状態であり、各要素が未完成であっても、その要素に関してある程度の評価が可能であれば良いと考える。例えばセキュアソフトの中の MPEG ビデオデコーダは、当初リアルタイムで動作しなくても、高速化のためのチューニングなど実験可能な

レベルであれば良いとする。このようにして出来るだけ、各要素を個別に評価実験出来るような体制にして研究開発を進める予定としている。評価・検証実験の過程で問題が発覚した場合は、基本的には要素ごとに改良研究を行う。また、基本アーキテクチャに関連した問題が発覚した場合は、セキュアハード等の構成まで戻って再検討を行う必要があるが、セキュアハード回路は FPGA 構成になっているため迅速な設計変更が可能である。より具体的には、以下のような手順で開発を進める。

１）まず、セキュアハードの暫定版を開発し、セキュアハードの機能が確認できるレベルのファームやソフトを立ち上げる。

２）その後、ソフトやファームの最低限の機能が評価できるようなレベルで立ち上げる。セキュアハード上のファーム（セキュアソフト再構成機能など）と PC メインメモリ上のソフト（MPEG ビデオデコード機能など）は、必ずしも同時に立ち上げる必要はなく個別に立ち上げることも可能である。すなわち、セキュアソフト再構成機能は、MPEG ビデオデコーダが動作していなくとも対象となる単純な評価用ソフトがあれば、再構成機能の評価実験は可能と考えられる。また、セキュアソフト（MPEG ビデオデコーダなど）は、セキュアハードがなくともある程度の評価実験は可能であるし、セキュアハードの機能の一部を使用しながら検証を行うことが可能である。

３）最終的に、各要素を組み合わせて、機能レベル（具体的な１４年度のリアルタイム性目標は毎秒約５フレーム）で「基本デジタル放送受信機能」を実現させる。

上記のように本研究開発は、図 4-4 の（ⅰ）HDD,（ⅱ）セキュアハード、(ⅲ) セキュアソフトの３基本要素をベースに如何にセキュアなシステム（セキュア MPEG ビデオデコードソフトなどを含む基本デジタル放送受信システム）を構築するかが一つの課題となる。

一方で研究開発要素ごとの研究開発も必要である。

１）セキュアハードは、地上デジタル放送関連規格にどう対応するか検討する必要がある。さらに Multi2 復号、ローカル暗号／復号回路など、平成１５年度の LSI 化を前提に開発や機能レベルの実証を行う。

２）また、セキュアハード内のファームはセキュアソフトを再構成する機能やセキュアソフトの保護を行う機能を持ち、本権利保護システムの大きなポイントと位置付けられる。セキュアソフトにリアルタイム認証アルゴリズムを挿入するなど、「セキュアソフト再構成手法」を検討する予定である。ここでのポイントは、(１)「成りすまし」攻撃を防ぐような解析困難なレベルの再構成の実現、(２) セキュアハード内のメモリなど限られた資源の中での再構成機能の実現だと考えられる。セキュアソフトの保護や、リアルタイム

認証の具体手順など詳細検討を行っていく。詳細検討において、或いは評価実験の過程でセキュリティーホールなどの検討を行い、より安全な手法を追求する。そのため、前述のように可能な限り研究要素ごとに独立に研究開発できる体制を敷いて検討を進める。

3）セキュアソフトなど PC メインメモリ上の関連ソフトに関しても、以下のような検討が必要である。（1）地上デジタル放送対応高速 MPEG ビデオデコーダ、（2）オーディオデコーダ、（3）リアルタイム認証などのセキュア機能、（4）セキュアソフト専用ローダー（5）セキュアハードドライバーである。また、伸張されたビデオ映像、オーディオの同期処理を含めた統合システムとユーザーが操作するための GUI（Graphical User Interface）が必要である。対象となる OS については、１４年度も引き続き検討するが、目標はできる限り OS に依存しないシステムを考える。OS を調査し、図 4-4 の専用ローダーやセキュアハードドライバーなどでできる限り OS 依存部を全面的にカバーし、ソフト本体は OS に依存しないような手法を考える。ローダー、ドライバに関しては対象 OS を Microsoft Windows として検討を進める。MPEG デコーダは、セキュアハードで再構成した上での、処理速度の向上がポイントとなる。今後の検討に依存するが、まず MPEG ビデオデコードの機能的な動作を確認し、その後、再構成したことによる影響などを検討し、最後に処理性能向上を図ることを目標とする。１４年度は、５フレーム／秒の処理速度が目標であり、本格的な高速化は次年度以降とする予定である。再構成によりなるべく処理性能に影響が出ないようにすることが研究開発の一つのポイントとなると考えられる。オーディオデコードに関しても、一定の開発は必要だが、MPEG ビデオデコードに準じた開発が可能であると判断する。

4-2-3　平成１５年度

PC 上で、ソフト処理主体でデジタル TV の権利保護を実現する基盤技術を完成させることが１５年度（最終年度）目標である。

１４年度で機能レベルで試作したハードウェア及びソフトウェアは、まだ製品化を期待出来るレベルにない。これを製品コアとして利用が期待出来るレベルにまでするのが平成１５年度の目標となる。このため、具体的に以下のような研究開発を行う。

（1）LSI 化研究開発

平成１４年度試作の FPGA（Field Programmable Gate Array）搭載実験

ボードは、試作が目的であり、物理的サイズやコストなどさまざまな要素のため、製品展開困難なレベルである。これを、製品展開が期待出来るレベルにするため、FPGA 回路、外付けプロセッサー、周辺回路などの LSI 化を図る。出来る限り最終目標であるデジタル放送対応のセキュア PC カード（セキュアハード）を意識し、その製品展開に必要と考えられる周辺回路を検討し、その検討にもとつき、内蔵すべき回路は LSI 内蔵し、インタフェースを用意すべき回路はインタフェースを用意する。LSI 化に際し、コスト、汎用性も意識した場合、必ずしも LSI 内蔵がベストでない場合もあり、その辺を十分に考慮し検討する。

（２）ソフト・リアルタイム化（改良）研究開発

セキュアハードを利用したセキュア MPEG ビデオ処理ソフト／セキュアオーディオ処理ソフトについては、以下のような研究開発を行う。即ち、平成１４年度の研究でソフトウェアは、機能試作レベルであり、リアルタイム性は持っていない。平成１５年度では、リアルタイム性を意識した改良を加える。具体的には、平成１４年度は、MPEG ビデオ処理に関しては、MP@HL（ハイビジョン画像）で毎秒５フレームの処理性能を達成している。これを平成１５年度は、一般視聴者が見て違和感のないレベルにまで高速化を図る。さらに、ソフトのセキュア化には、ソフトの「再構成」、「リアルタイム認証」、「リアルタイム・コード・スキャン」などの要素を組み込む必要があるが、リアルタイム性を意識し、処理性能の劣化を極力抑えながら、前述要素の組み込みを行う。これらの要素は、当然ながら、セキュアハード（１５年度 LSI 化予定）、セキュアハード内のファームウェアとも連動している（セキュアハードがセキュアソフトの動作を監視する）ため、必要となれば、セキュアハード、ファームウェアに対し、可能な範囲でフィードバックをかける。

なお、セキュリティー強度は、ソフトの「再構成」、「リアルタイム認証」、「リアルタイム・コード・スキャン」の頻度などに依存すると考えられる。セキュリティー強度に関しは、基準となる一般的な物差しがないので言及困難ではあるが、基本的にオールソフトウェア処理によるソフトセキュア化（一般的に難読化と呼ばれる）より強い、放送など公共性の高い網で適用可能なレベルを目指す。

（３）PC カード化

権利保護 PC カードの中に、上記 LSI 以外に地上デジタル放送を受信するときに必要な機能を搭載する予定である。従って１５年度は、カード内に搭載すべき回路を検討し、具体的に実装する方法などを検討する。また、カ

ード形状などの具体検討も行う。さらに、実際の製品を考えたときに必要と考えられる「旧 PC から新 PC に買い替えた時に必要な蓄積コンテンツのセキュアな視聴権移動機能（コア機能)」などの実装方法を検討する。

図 4-5 に PC カードとして実現するセキュアハードと HDD およびセキュアソフトの概要を示す。セキュアハードには LSI として実現する暗号エンジン、スクランブラーの他に、地上デジタル放送を受信するためのチューナおよび OFDM 復調回路等を実装する。

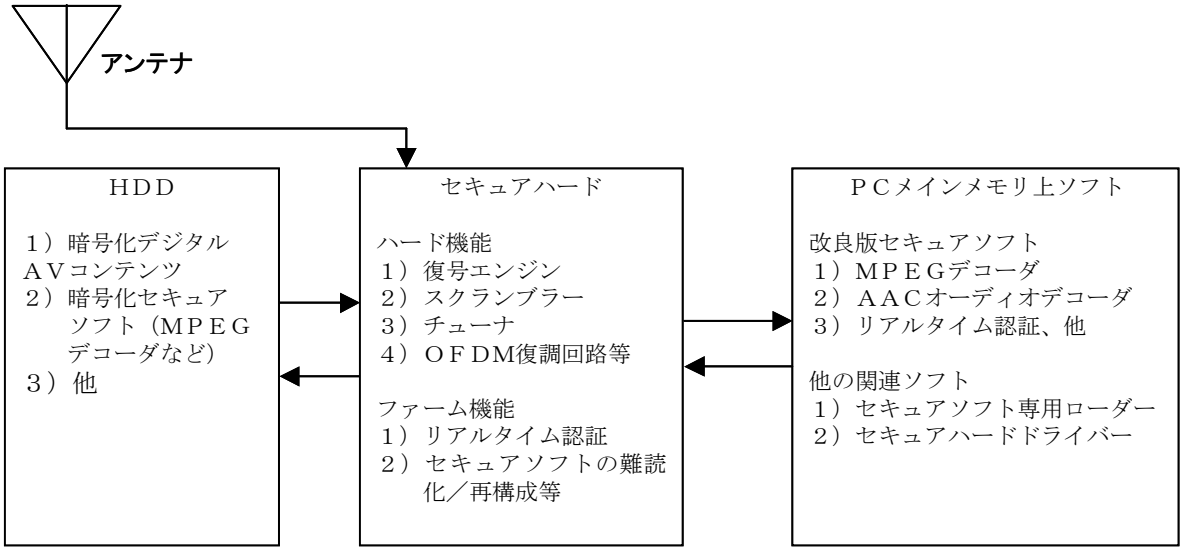


図 4-5 PC カードにおけるセキュアハードとセキュアソフトの概要

５　研究開発実施状況

５-１　セキュアLSI

AVコンテンツやPC上で実行されるソフトウェアのセキュア化を実現するには、外部から「改ざん」や「覗き見」を行なうことができない耐タンパモジュールが必要不可欠である。本システムにおいては、この耐タンパモジュールを「セキュアハード」と呼ぶ。平成 14 年度の研究開発においては、この「セキュアハード」の機能をＦＰＧＡ（Field Programmable Gate Array）で実現した。本年度の研究開発においては、システム全体としての性能や製品適用の実現性を評価するために、昨年度開発したFPGA版セキュアハードの機能に加え、公開鍵暗号マクロやハッシュ値計算マクロなどを１チップに集積した「セキュアLSI」の開発を行った。本項では、開発したセキュアLSIの機能について解説する。

５-１-１　機能概要

SECURE_LSIは、FR70EをCPUコアとして、MPEG TS信号の権利保護を伴う暗号化/復号化処理、B-CASカードインターフェイス、PCIコントローラ等を搭載したデジタル放送向けのLSIです。

本LSIは、入力されたMPEG TSパケットデータをコンテンツの権利を保護した状態で自由に視聴し、ハードディスクへの保存など行う事を実現するためにTSパケットデータの暗号化/復号化やRouting､PCIとのデータ転送を行います。

また、本LSI内アプリケーションへの改ざん監視機能や、物理的に別のデバイスにおけるコンテンツ視聴機能を実現します。

５-１-２　特徴

■FR70Eコア

- 富士通FRシリーズ32ビット命令互換RISCコア搭載、5段パイプライン、1命令/1サイクル実行
- 命令キャッシュ 4kB、データ RAM2kB～搭載
- アドレス/データバス 32ビット
- DMAC 5ch、UART 2ch、16 ビットリロードタイマー 3ch、割り込みコントローラの周辺機能内蔵

■MPEG TS信号処理

- TSパケットデータの暗号化/復号化処理

・ Routing 機能

■PCI コントローラ
・ PCI ローカル・バス使用 Rev2.2 準拠
・ 5V 及び 3.3V 信号環境で 33MHz をサポート
・ マスタ・モード及びターゲット・モードをサポート
・ 独立したマスタ DMA チャネル
・ ユーザ使用可能な 3 つのターゲット・コンポーネント
・ PCI パワーマネージメント仕様 Rev1.1 に準拠したレジスタを実装
・ マスタ FIFO(8dword～32dword)及びターゲット FIFO(8dword～32dword)を双方向に内蔵
・ マスタ・モード及びターゲット・モードでノン・ウェイト・バースト転送可能
(最大転送レート 33MHz:133MB/s, 66MHz:266MB/s)
・ ビット情報の CONFIG ROM 回路により、コンフィグレーション・レジスタ情報及びターゲット転送モードのカスタマイズが可能
・ ターゲット遅延トランザクションのサポート
・ ターゲット・メモリ空間として FIFO マッピングをサポート(FIFO モード)

■公開鍵マクロ
・楕円暗号を用いた公開鍵暗号演算処理をハードウェアにて高速に実現
・ FR70E からの入力信号を、公開鍵マクロの仕様に合わせて変換(WRAP_DAENC)

■Hash 計算マクロ
・認証アルゴリズム：HMAC-MD5-96 及び HMAC-SHA-1-96 の鍵付きハッシングによるメッセージ認証処理をハードウェアにて高速に実現
・モードにより、HMAC 構造をとらない MD5、SHA-1 だけの処理をすることも可能
・FR 系 32bit バスに接続できる HOST I/F 部を搭載
・FR70E からの入力信号を、Hash 計算マクロの仕様に合わせて変換 (WRAP_HMAC)

■I^2C
・Inter IC BUS をサポートするシリアル I/O ポートで、I^2C バス上のマスター/スレーブデバイス動作
・FR70E からの入力信号を、I^2C の仕様に合わせて変換 (WRAP_I2C)

■B-CAS カードインターフェース

■SDRAM データの XOR 処理

・SDRAM 内に外部から読み出して意味のわかるデータ(鍵等)がそのまま見えてしまうのは問題があるため、起動のたびにファームから設定する値を使って SDRAM アクセス時にデータを XOR 処理する。

・FR70E から LSI 内部マクロへのアクセス時には、データを LSI 外部へ出力しない処理をする。

・上記以外のデータは LSI 外部へそのまま出力する。

■リセットマクロ

・PCI のターゲット転送を利用し、FR70E よりレジスタ設定されると、本 LSI に内蔵されている IP を個別に初期化することが可能。

■LSI 全体

・ 0.18μ CMOS Al5 層テクノロジ
・ FBGA-288pin パッケージ(Ball Pitch 0.75mm、18 × 18 mm)
・ 電源:2 電源
　VDDI(LSI 内部電源)　:1.65～1.95V
　VDDE(I/O 電源)　:3.0～3.6V
・ FR70E コア動作クロック : 40～50MHz

５-１-３　ブロック構成

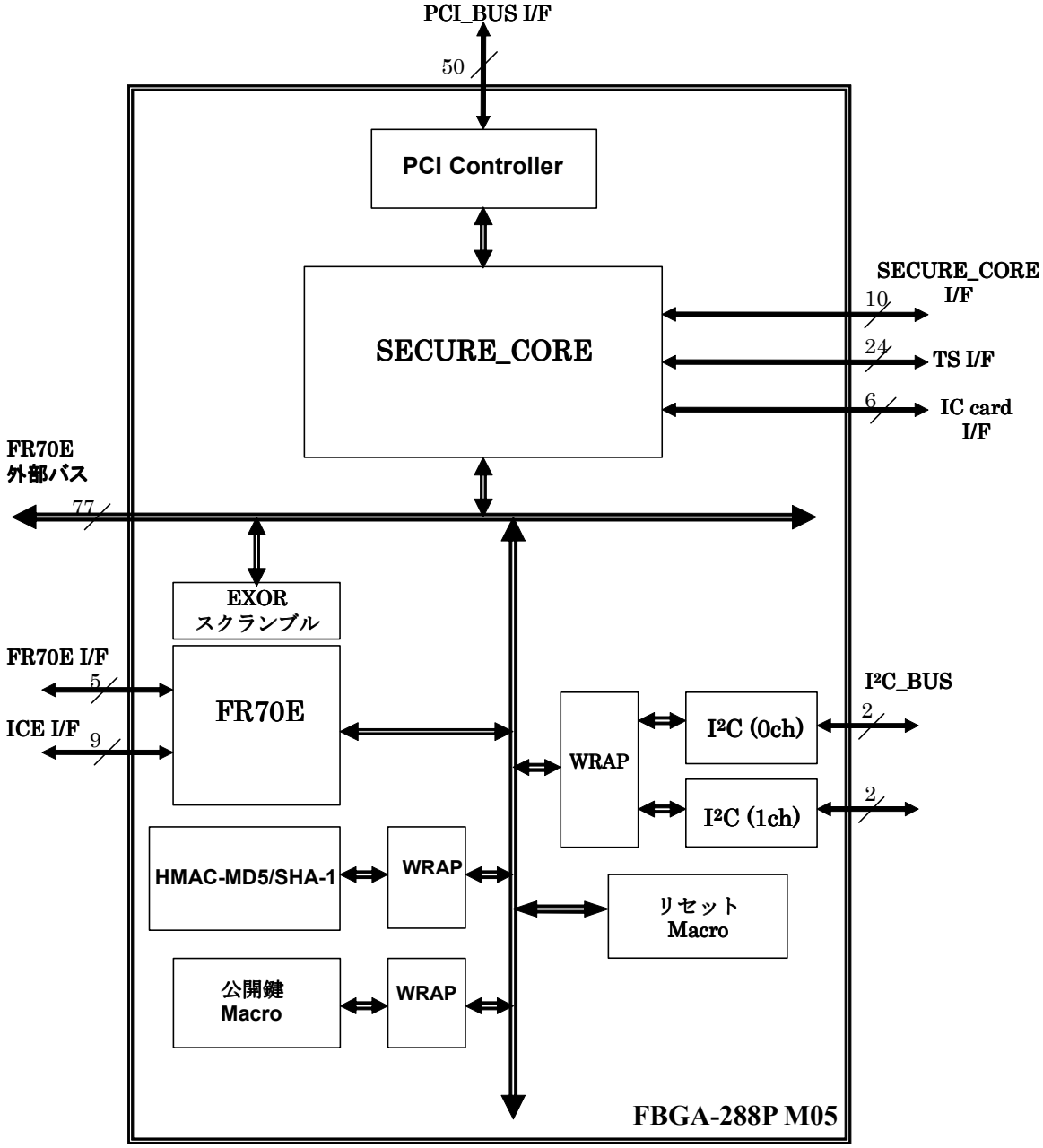


図　5-1-1　セキュア LSI ブロック図

５-１-４　入出力端子

一部の端子において、SCAN I/F 端子とマルチプレクサになっています。
下表の端子表示にて、I/F 端子名／Scan I/F 端子名 で記載されています。

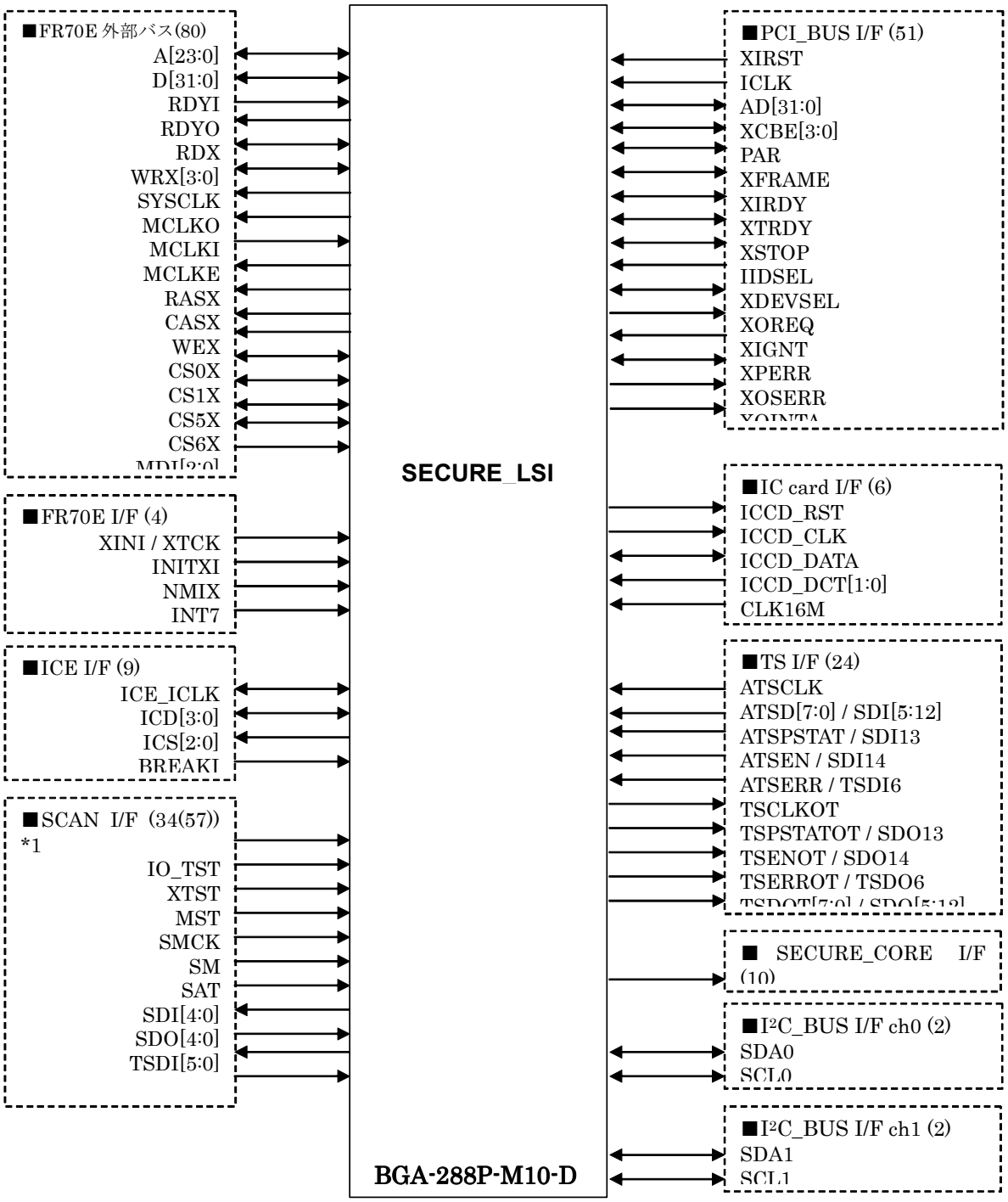


図 5-1-1　セキュア LSI 入出端子図

■ 端子機能表

１：FR70E インターフェース

端子名	IO 抵抗	機能名	属性	説明
D[31:0]	U/D (※1)	DO[7:0]	out	外部データバス bit7～0 の出力（外バス 32bit 時） 外バス 32bit モード以外の時は機能しません。
		DI[7:0]	in	外部データバス bit7 ～0 の入力(外バス 32bit 時) 外バス 32bit モード以外の時は機能しません。
		DC[3]	out	外部データバス bit7 ～0 のアウトプットイネーブル(外バス 32bit 時) 外バス 32bit モード以外の時は常にディセーブル状態になります。
		DO[15:8]	out	外部データバス bit15～8 の出力（外バス 32bit 時） 外バス 32bit モード以外の時は機能しません。
		DI[15:8]	in	外部データバス bit15 ～8 の入力(外バス 32bit 時) 外バス 32bit モード以外の時は機能しません。
		DC[2]	out	外部データバス bit15 ～8 のアウトプットイネーブル(外バス 32bit 時) 外バス 32bit モード以外の時は常にディセーブル状態になります。
		DO[23:16]	out	外部データバス bit23～16 の出力（外バス 16,32bit 時） 外バス 8bit モード時は機能しません。
		DI[23:16]	in	外部データバス bit23～16 の入力（外バス 16,32bit 時） 外バス 8bit モード時は機能しません。
		DC[1]	out	外部データバス bit23～16 のアウトプットイネーブル(外バス 16,32bit 時) 外バス 8bit モード時は常にディセーブル状態になります。
		DO[31:25]	out	外部データバス bit31～24 の出力
		DI[31:25]	in	外部データバス bit31～24 の入力
		DC[0]	out	外部データバス bit31～24 のアウトプットイネーブル
A[23:0]	U	AO[7:0]	out	外部アドレスバス bit7～0 の出力

		AI[7:0]	in	外部アドレスバスbit7～0の入力 ただし、テスト時以外は機能しません。
		AC[2]	out	外部アドレスバスbit7～0のアウトプットイネーブル 外部バス開放時に、ディセーブル状態となります。
		AO[15:8]	out	外部アドレスバス bit15 ～8 の出力
		AI[15:8]	in	外部アドレスバスbit15～8の入力 ただし、テスト時以外は機能しません。
		AC[1]	out	外部アドレスバスbit15～8のアウトプットイネーブル 外部バス開放時に、ディセーブル状態となります。
		AO[23:16]	out	外部アドレスバス bit23～16 の出力
		AI[23:16]	in	外部アドレスバスbit23～16の入力 ただし、テスト時以外は機能しません。
		AC[0]	out	外部アドレスバスbit23～16のアウトプットイネーブル 外部バス開放時に、ディセーブル状態となります。
RDYI	D	RDYI	in	外部 RDY 入力。外部バスサイクルが完了しないとき、0 を入力してバスサイクルを延長します。
RDYO	―	RDYO	out	外部RDY 出力。 ただし、テスト時のみ機能します。
RDX	U	RDXO	out	外部バスリードストローブ出力
		RDXI	in	外部バスリードストローブ入力 ただし、テスト時のみ機能します。
		RDXC	out	外部バスリードストローブ・アウトプットイネーブル 外部バス解放時に H を出力します。

端子名	IO 抵抗	機能名	属性	機能
WRX[3:0]	U	WRXO[3]	out	外部データバス bit7～0 に対するライトストローブ出力 SDRAM/FCRAM 領域に対しては、DQM として機能します。
		WRXI[3]	in	外部データバス bit7～0 に対するライトストローブ入力 ただし、テスト時のみ機能します。
		WRXO[2]	out	外部データバス bit15～8 に対するライトストローブ出力 SDRAM/FCRAM 領域に対しては、DQM として機能します。
		WRXI[2]	in	外部データバス bit15～8 に対するライトストローブ入力 ただし、テスト時のみ機能します。
		WRXO[1]	out	外部データバス bit23～16 に対するライトストローブ出力 SDRAM/FCRAM 領域に対しては、DQM として機能します。
		WRXI[1]	in	外部データバス bit23～16 に対するライトストローブ入力 ただし、テスト時のみ機能します。
		WRXO[0]	out	外部データバス bit31～24 に対するライトストローブ出力 SDRAM/FCRAM 領域に対しては、DQM として機能します。
		WRXI[0]	in	外部データバス bit31～24 に対するライトストローブ入力 ただし、テスト時のみ機能します。
		WRC	out	外部バスライトストローブ・アウトプットイネーブル 外部バス解放時に H を出力します。
SYSCLK	－	SYSCLK	out	外部バスクロック出力 MCLKO と同じタイミングで出力されます。
MCLKO	－ (※2)	MCLKO	out	メモリクロック出力 SYSCLK と同じタイミングで出力されます。
		MCLKI	in	メモリクロック入力 MCLKO を外部に放出した後の信号を接続します。 データの取り込み、RDY のサンプリング等は、この信号で行われます。
MCLKE	－	MCLKE	out	メモリクロックイネーブル出力
RASX	－	RASX	out	アドレスストローブ出力 外部バスのサイクル開始を意味します。 SDRAM/FCRAM 領域では、RAS として機能します。

CASX	－	CASX	out	バーストアクセスストローブ出力 バーストモード時の PageRead 毎のサイクルにアクティブとなります。 SDRAM/FCRAM 領域では、CAS として機能します。
WEX	－	WEX	out	外部バスライトストローブ出力 バスサイズによらず、出力されます。 SDRAM/FCRAM に対するライトストローブとしても機能します。
CS0X	U	CS0XO	out	チップセレクト 0 出力
		CS0XI	in	チップセレクト 0 入力 ただし、テスト時のみ機能します。
		CS0XC	out	チップセレクト・アウトプットイネーブル 0 出力
CS1X	U	CS1XO	out	チップセレクト 1 出力
		CS1XI	in	チップセレクト 1 入力 ただし、テスト時のみ機能します。
		CS1XC	out	チップセレクト・アウトプットイネーブル 1 出力
CS5X	U	CS5XO	out	チップセレクト 5 出力
		CS5XI	in	チップセレクト 5 入力 ただし、テスト時のみ機能します。
		CS5XC	out	チップセレクト・アウトプットイネーブル 5 出力
CS6X	U	CS6XO	out	チップセレクト 6 出力
		CS6XI	in	チップセレクト 6 入力 ただし、テスト時のみ機能します。
		CS6XC	out	チップセレクト・アウトプットイネーブル 6 出力

端子名	IO 抵抗	機能名	属性	機能
MDI[2:0]	U	MDI[2:0]	in	モード端子 本端子により、コアの基本動作モードを設定します。 ノイズによる誤動作を防ぐため、入力バッファと１対１で接続し、 外部で VDD は VSS に直接接続してください。
XINI	－	XINI	in	クロック入力
INITXI	S	INITXI	in	設定初期化リセット入力
NMIX	U	NMIX	in	NMI 入力
INT7	D	INT7	in	外部割り込み要求入力 ch7

２：FR70E　ICE インターフェース

端子名	IO 抵抗	機能名	属性	機能
ICE_ICLK	U	ICLKO	out	エミュレータ用クロック出力
		ICLKB	in	I/O バッファの出力に接続してください。
ICD[3:0]	D	ICDO[3:0]	out	エミュレータ用データ出力
		ICDI[3:0]	in	エミュレータ用データ入力
		ICDC	out	エミュレータ用データ出力のアウトプットイネーブル
ICS[2:0]	－	ICS[2:0]	out	エミュレータ用チップステータス出力
BREAKI	D	BREAKI	in	ブレーク要求入力

３：SECURE_CORE　TS Stream インターフェース

端子名	IO 抵抗	機能名	属性	機能
ATSCLK	－	ATSCLK	in	TS クロック入力
ATSD[7:0]	U	ATSD[7:0]	in	TS ストリームデータ入力（1packet=188byte）
ATSPSTAT	U	ATSPSTAT	in	TS ストリーム入力の 1byte 目を示すスタート信号
ATSEN	U	ATSEN	in	Valid Data Enable
ATSERR	U	ATSERR	in	TS ストリームのエラー入力
TSCLKOT	－	TSCLKOT	out	TS クロック出力
TSPSTATOT	－	TSPSTATOT	out	TS ストリーム出力の 1byte 目を示すスタート信号
TSENOT	－	TSENOT	out	Valid Data Enable
TSERROT	－	TSERROT	out	TS ストリームのエラー出力
TSDOT[7:0]	－	TSDOT[7:0]	out	TS ストリームデータ出力（1packet=188byte）

4 ：SECURE_CORE　IC Card インターフェース

端子名	IO 抵抗	機能名	属性	機能
ICCD_RST	－	ICCD_RST	out	IC Card 用リセット出力
ICCD_CLK	－	ICCD_CLK	out	IC Card 用クロック出力
ICCD_DATA	U	ICCD_DATA	inout	IC Card 用データ
ICCD_DCT[1:0]	U	ICCD_DCT[1:0]	in	IC Card 挿入信号
CLK16M	－	CLK16M	in	ICCD_CLK の源振クロック入力

5 ：SECURE_CORE インターフェース

端子名	IO 抵抗	機能名	属性	機能
MON[9:0]	－	MON[9:0]	out	DMA Enable モニタ信号

6 ：I2C インターフェース

端子名	IO 抵抗	機能名	属性	機能
SDA1	U	SDAO	out	I2C データ出力（ch1）
		SDAI	in	I2C データ入力（ch1）
SCL1	U	SCLO	out	I2C クロック出力（ch1）
		SCLI	in	I2C クロック入力（ch1）
SDA0	U	SDAO	out	I2C データ出力（ch0）
		SDAI	in	I2C データ入力（ch0）
SCL0	U	SCLO	out	I2C クロック出力（ch0）
		SCLI	in	I2C クロック入力（ch0）

7 : SCAN インターフェース

端子名	IO 抵抗	機能名	属性	機能
XTST	D	XTST	in	テストモード信号 ’0’ : 通常モード、’1’ : テストモード
MST	D	MST	in	メモリテストモード信号 ‘0’ : 通常モード、’1’ : メモリテストモード
XTCK	U	XTCK	in	テスト用クロック信号／SCAN 用クロック信号
SMCK	U	SMCK	in	Memory SCAN モード用クロック
SM	U	SM	in	SCAN モード信号 ‘0’ : 通常モード、’1’ : SCAN テストモード
SAT	D	SAT	in	メモリ BIST 端子 通常使用時は’0’入力
SDI[14:0]	U	SDI[14:0]	in	Logic SCAN スキャンデータ入力
SDO[14:0]	－	SDO[14:0]	out	Logic SCAN スキャンデータ出力
TSDI[6:0]	U	TSDI[6:0]	in	Memory SCAN スキャンデータ入力 TSDI[0]は PLL の RST 端子としても使用します。通常モードで使用する場合は、『6.2.1 : リセット』に説明されているシーケンスに乗っ取って使用してください。
TSDO[6:0]	－	TSDO[6:0]	out	Memory SCAN スキャンデータ出力
MMS[5:0]	U	MMS[5:0]	in	メモリテストモジュール選択端子
IO_TST	－ （※３）	IO_TST	in	I/O の VPD 用端子 通常使用時は’0’入力

８：PCI_Controller PCI バスインターフェース

端子名	IO 抵抗	機能名	属性	機能
XIRST	S	XIRST	in	リセット入力
ICLK	－	ICLK	in	PCI バス・クロック入力(33MHz) PCI インタフェースおよび CONFIG ROM インタフェースは、CLK 同期で動作します。
AD[31:0]	U	OAD[31:0]	out	アドレスおよびデータ出力
		IAD[31:0]	in	アドレスおよびデータ入力
		XOE_AD[7:0]	out	アドレスおよびデータのアウトプットイネーブル出力 OAD４本につき、１本ずつ接続してください。
XCBE[3:0]	U	XOCBE[3:0]	out	バス・コマンドおよびバイト・イネーブル出力
		XICBE[3:0]	in	バス・コマンドおよびバイト・イネーブル入力
		XOE_XCBE[3:0]	out	バス･コマンドおよびバイトイネーブルのアウトプットイネーブル出力 各 XOCBE 出力に１本ずつ接続してください。
PAR	U	OPAR	out	パリティ出力
		IPAR	in	パリティ入力
		XOE_PAR	out	パリティのアウトプットイネーブル出力
XFRAME	U	XOFRAME	out	サイクル・フレーム出力
		XIFRAME	in	サイクル・フレーム入力
		XOE_XFRAME	out	サイクル・フレームのアウトプットイネーブル出力
XIRDY	U	XOIRDY	out	イニシエータ・レディ出力
		XIIRDY	in	イニシエータ・レディ入力
		XOE_XIRDY	out	イニシエータ・レディのアウトプットイネーブル出力
XTRDY	U	XOTRDY	out	ターゲット・レディ出力
		XITRDY	in	ターゲット・レディ入力
		XOE_XTRDY	out	ターゲット・レディのアウトプットイネーブル出力
XSTOP	U	XOSTOP	out	ストップ出力
		XISTOP	in	ストップ入力
		XOE_XSTOP	out	ストップのアウトプットイネーブル出力
IIDSEL	U	IIDSEL	in	イニシャル・デバイス・セレクト入力
XDEVSEL	U	XODEVSEL	out	デバイス・セレクト出力

		XIDEVSEL	in	デバイス・セレクト入力
		XOE_XDEVSEL	out	デバイス・セレクトのアウトプットイネーブル出力
XOREQ	－	XOREQ	out	PCI バス・リクエスト出力
		XOE_XREQ	out	リクエストのアウトプットイネーブル出力 XIRST の反転信号。REQ は３ステート I/O バッファを使用し、リセット中の REQ は３ステート I/O バッファはハイ・インピーダンスになります。
XIGNT	U	XIGNT	in	グラント入力
XPERR	U	XOPERR	out	パリティ・エラー出力
		XIPERR	in	パリティ・エラー入力
		XOE_XPERR	out	パリティ・エラーのアウトプットイネーブル出力
XOSERR	－	XOSERR	out	システム・エラー出力 SERR 信号はオープン・ドレイン信号です。使用テクノロジにオープン・ドレインの I/O バッファがない場合は、３ステート I/O バッファを使用し、この信号をアウトプットイネーブルに、入力を GND に接続します。
XOINTA	－	XOINTA	out	インタラプト A 出力 INTA 信号はオープン・ドレイン信号です。使用テクノロジにオープン・ドレインの I/O バッファがない場合は、３ステート I/O バッファを使用し、この信号をアウトプットイネーブルに、入力を GND に接続します。

５－２　セキュア LSI 評価ボード

前項で解説したセキュア LSI を搭載した「セキュア LSI 評価ボード」を製作した。本ボードには、チューナーモジュール、OFDM 復調 LSI、CAS カード用の IC カードインターフェース等が搭載されており、地上波デジタル放送の受信が可能となっている。本評価ボードが本セキュアシステムにおける「セキュアハード」にあたり、本ボードを用いることにより、

1. 放送コンテンツのセキュアな状態での視聴
2. 放送コンテンツのセキュアな状態での記録
3. 記録した放送コンテンツのセキュアな状態での視聴

が可能となる。

本項では、開発したセキュア LSI 評価ボードの機能について解説する。

５-２-１　システム概要

セキュアボードは、デジタル TV により供給されるコンテンツの権利保護を PC にて行うための PCI ボードである。本ボードに搭載されたセキュア LSI とファームウェアおよびＰＣ上のアプリケーションにて権利侵害をほぼ不可能とする環境を構築している。

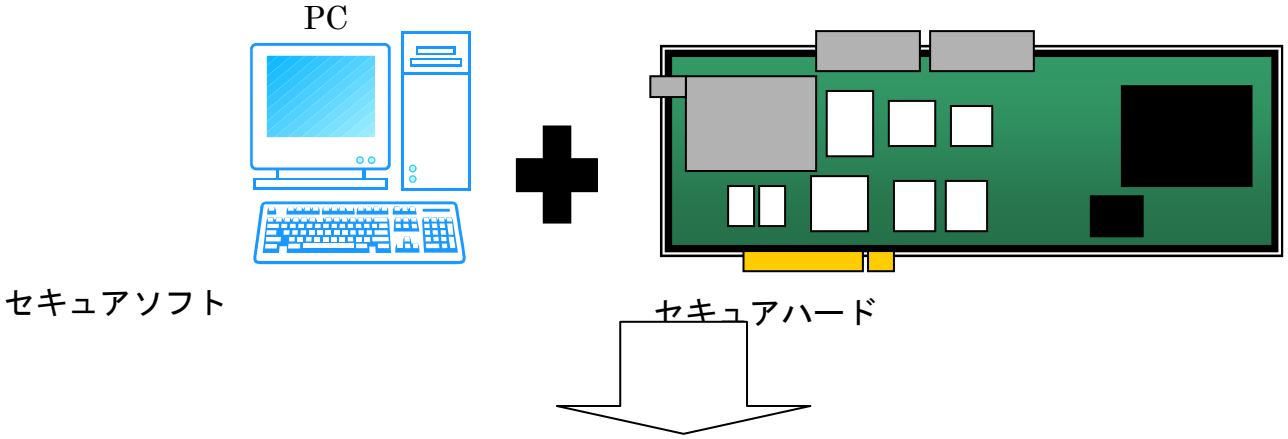


図　5-2-1　セキュア PC 構成概念図

５-２-２　機能概要

本ボードの機能を以下に示します。

- 富士通製セキュアＬＳＩを搭載。
- TUNER モジュール、OFDM LSI を搭載し、RF 信号を受信・復調可能である。
- 外部インタフェースの TS パラレル入出力ポート(Dsub25pin x2)を持っている。
- TS データのルート制御を PLD にて構成した切替回路にて行うことが可能。(切替指示は DIP SW)
- デバッグ用ロジックアナライザコネクタを搭載。
- ＦＲ７０用 ICE コネクタを搭載。
- Ｉ^{2}Ｃコネクタ２ch 搭載。
- ＩＣカードインタフェースを搭載。
- Flash ROM 2MB　2 個搭載。(ソケット実装)
- SDRAM 16MB 2 個搭載。
- デバッグ用に LED モジュールを搭載。
- 電源供給は PCI バスからと Stand Alone と選択可能。

■ セキュア LSI ボードブロック図

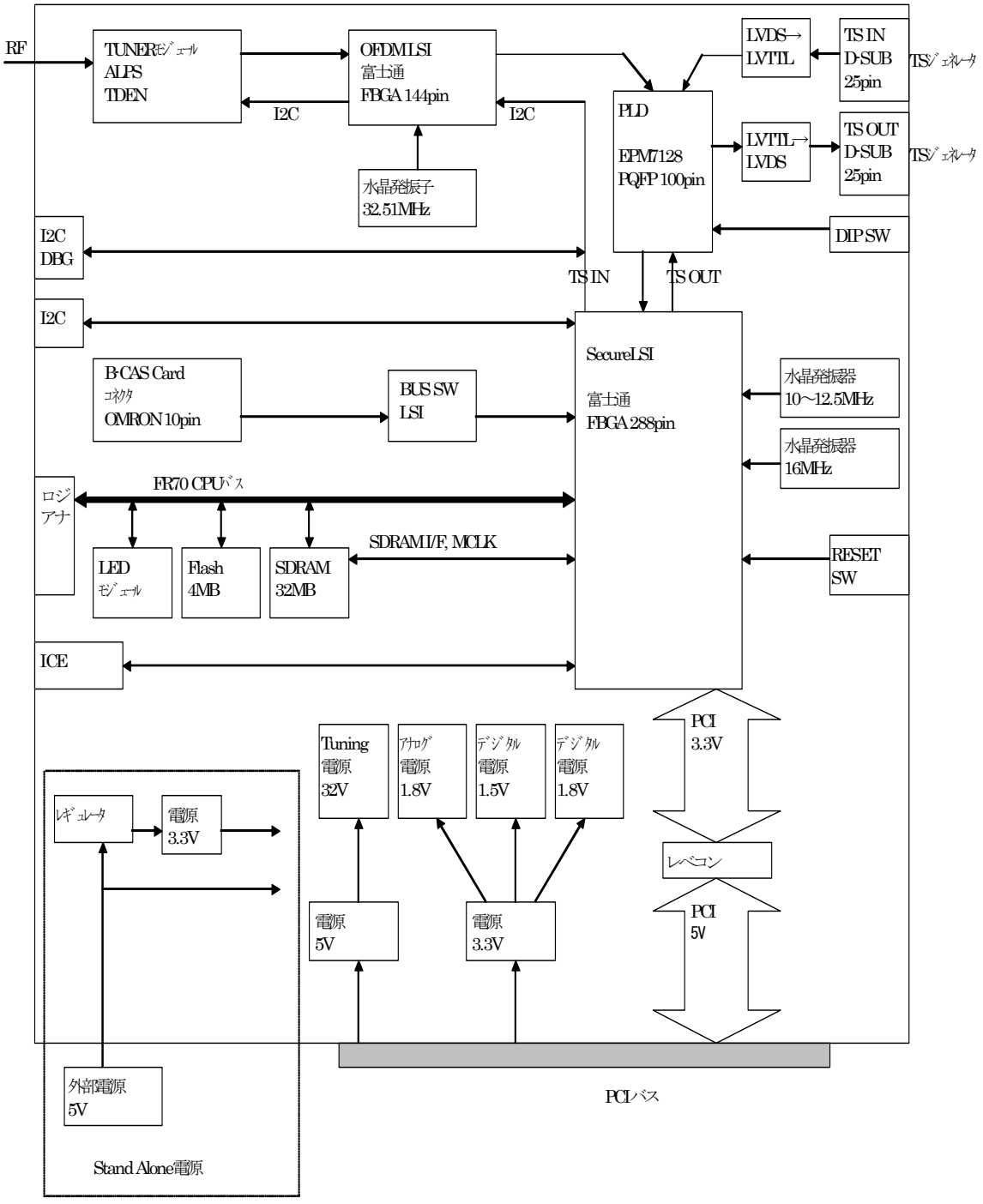


図 5-2-2 セキュア LSI ボードブロック図

■ セキュア LSI ボード部品配置図

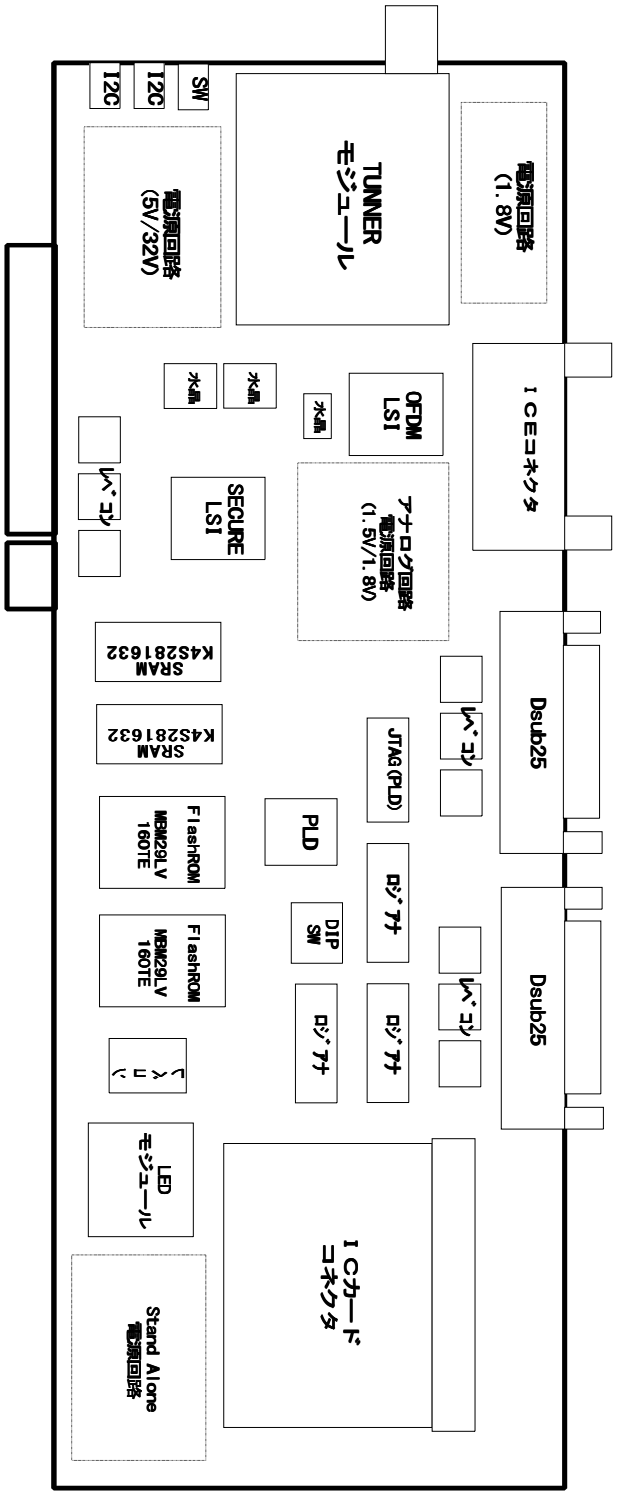


図 5-2-2 セキュア LSI ボード部品配置図

５-２-３　機能仕様

本ボードの機能仕様および本ボードに搭載されるデバイス機能仕様を以下に示す。

■ セキュア LSI

本ボードにおけるメインデバイスであるセキュア LSI。仕様に関しては、5-1 項を参照。

■リセットシーケンス

本ボードはハードリセット機能を２種類、ソフトリセット機能を１種類もっている。

(1)ハードリセット

●PCI バスリセット

PC のパワーON 時に発生するリセットであり、ボード内全てのデバイスにリセットが発生する。

●RESET SW

外部 RESET ボタン SW によるリセットであり、セキュア LSI の PCI コントローラ部以外のマクロおよびボード内デバイスにリセットが発生する。

(2)ソフトリセット

セキュア LSI 内部のレジスタ等をソフトウェア上(アプリケーション、ドライバ) からクリアさせたい場合などに使用する。ただし、セキュア LSI 内蔵の PCI コントローラが正常動作していることが前提条件である。

■クロックツリー

本ボードに使用するクロックおよび供給方法を以下に示す。

(1)OFDM LSI

必要スペック：32.51MHz

供給方法　　：水晶発振子から 32.51MHz を供給

⑵SecureLSI

●内部クロック用

供給方法　　：水晶発振器から 10MHz～12.5MHz を供給
　　内部にて 4 逓倍し、内部クロックは 40MHz～50MHz

●IC カード用

供給方法　　：水晶発振器から 16MHz を供給

●PCI クロック

供給方法　　：PCI バスから 33MHz を供給

●TS 入力クロック

供給方法　　：TS をセレクトする PLD から供給
　　このクロックに同期して TS データが入力される。

⑶B-CAS Card

必要スペック：8MHz/4MHz
供給方法　　：SecureLSI から 8MHz もしくは 4MHz を供給

⑷SDRAM

供給方法　　：SecureLSI から 4 逓倍した内部クロック出力
　　(40MHz～50MHz)を供給

■ TS データ入出力

本ボードに搭載している PLD にて行っている TS データ入出力制御の機能および仕様を以下に示す。

セキュア LSI に関して入出力している TS データのパスは、以下のようになっている。この TS データパスの制御を PLD にて行う。パスの選択は外部 DIP SW にて行う。

①OFDM LSI からの TS 入力
②TS ジェネレータからの TS 入力
③TS ジェネレータへの TS 出力
④SECURE LSI への TS 入力

⑤SECURE LSI からの TS 出力

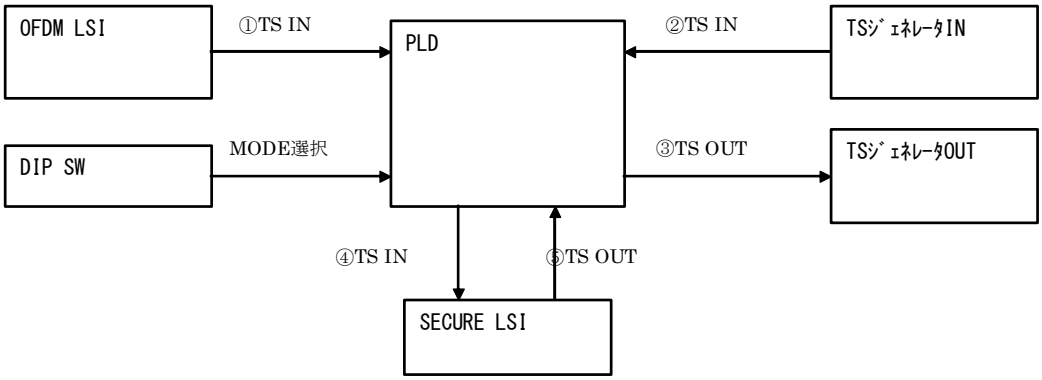


図 5-2-4 TS データ入出力パス構成図

●PLD 仕様

DIP SW の MODE を以下に示す。

(1) ＴＳ入力選択

DIP_SW0 = 0 ：①→④

OFDM LSI から SECURE LSI への TS データ入力

DIP_SW0 = 1 ：②→④

S ジェネレータから SECURE LSI への TS データ入力

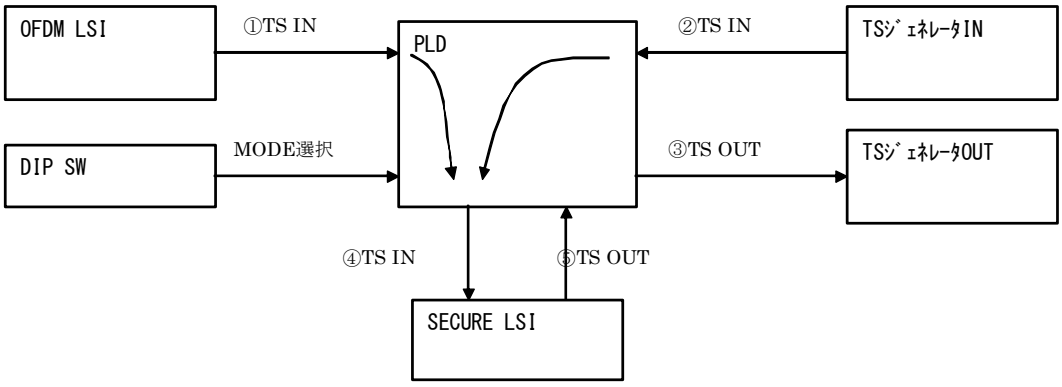


図 5-2-5 TS データ入力パス切り替え図

(2) ＴＳ出力選択

DIP_SW1＝0　かつ DIP_SW2＝0　　：①→③

OFDM LSI から TS ジェネレータへの TS データ出力

DIP_SW1＝1　かつ DIP_SW2＝0　　：⑤→③

SECURE LSI から TS ジェネレータへの TS データ出力

DIP_SW1＝X　かつ DIP_SW2＝1　　：②→③

TS ジェネレータから TS ジェネレータへの TS データ出力

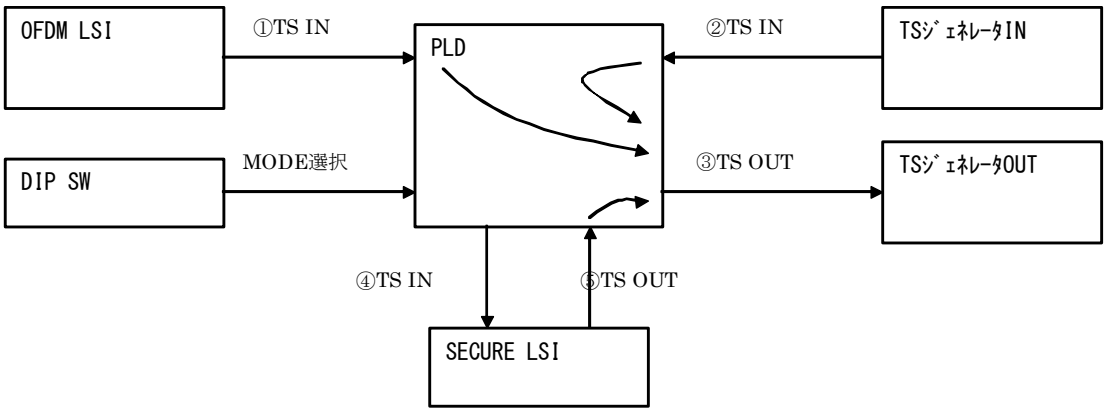


図　5-2-6　TS データ出力パス選択切り替え図

■ TUNER モジュール

地上波デジタル放送用のチューナーユニットです。

■ OFDM LSI

日本の地上波デジタルテレビジョン放送(ISDB-T)の規格に準拠した OFDM-IF 信号復調用 LSI です。

A/D, 8K-FFT 、デインタリーバ（周波数、時間、バイト、ビット）、誤り訂正(ビタビ、リードソロモン)、多重フレーム処理、TMCC デコーダ、必要なすべてのメモリをワンチップに内蔵しています。

■ I^2C コネクタ

セキュア LSI には、I^2C インタフェースを２系統持っている。１つ(I^2C0)は OFDM LSI および TUNER モジュールの制御用であり、もう１つ(I^2C1)は予備である。

２系統とも４ピンヘッダコネクタと接続されている。

■ Flash ROM

FlashROM は 2MB の容量をもったデバイスを 2 個搭載している。
この 2 つの FlashROM に対して CS 信号を切り替えることで、2 つの領域を使い分けることが可能となっている。

■ PLD データダウンロード

PLD の回路データダウンロードは ALTERA 製 MasterBlaster を使用する。
PLD の JTAG ポートに接続されている 10 ピンヘッダに MasterBlaster のソケットを装着してダウンロードを実施する。

5-3　ソフトの高速化

５-３-１　はじめに

2002 年度に MPEG-2 のデコード機能を持つセキュアソフトの開発(以下 Secure decoder)を行った。性能面では MP@ML(標準 TV サイズ)データの秒間 30 フレームデコード機能および、MP@HL データ(ハイビジョンサイズ)の秒間 10 フレームデコード機能を達成した。

本章は、本年度の目標である、Secure decoder において、MP@HL データの秒間 30 フレームデコードできる性能を実現する為の性能向上検討についてまとめたものである。MP@HL データのストリームのデコードを実現する為にはセキュアソフトの性能を向上する必要がある５-３-２では、性能測定及び高速化を実現するためのツール及びコンパイラの検討について記載する。また、５-３-３では性能測定の結果、セキュアソフトの中で最も負荷の高かったビデオデコーダの高速化の検討案について記載する。

５-３-２　開発ツール・コンパイラの検討

５-３-２-１　開発ツールについて

５-３-２-１-１　DevPartner Studio 7.0

http://www.xlsoft.com/jp/shopping/xlshop/index.html

- ✓ Bounds Checker Visual C++ Edition － 自動エラー検出と診断
- ✓ TrueTime Visual C++ Edition － パフォーマンス分析と最適化
- ✓ TrueCoverage Visual C++ Edition － コード・カバレッジ分析

Visual Studio.NET 2003 には対応していないため、注意すること。

５-３-２-１-２　VTune Performance Analyzer 7.0

単体で良いならこちら.

５-３-２-１-３　Thread Checker 1.0 for Windows

http://www.xlsoft.com/jp/products/intel/ithread.html

スレッドチェッカとスレッドプロファイラにより構成される。
スレッドチェッカの主な機能で、以下のエラーを検出する。

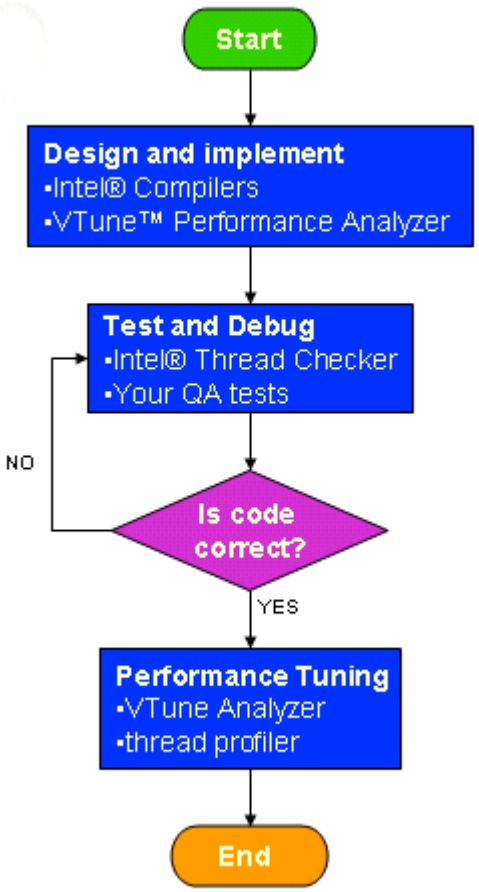
- data races
- deadlocks
- stalled threads
- lost signals
- abandoned locks

全メモリを監視するため Thread Checker 自身のオーバヘッドは大きいが、テストの負荷を大幅に軽減することが可能なようである。

スレッドプロファイラは、VTune analyzer data collector の一種である。OpenMP でスレッド化されたアプリケーションのパフォーマンスをチューニングする。OpenMP API によるプログラミングが前提となる。
スレッドプロファイラにより、以下のことが可能である。

- スレッドスケジューリング、分割スレッド数、などの構成パラメータ変更によるパフォーマンス比較
- 不均衡な負荷によるスレッドの効率性破壊かどうかの決定
- 過度のロック競合の発見
- プロセッサ数を増加させた場合の総実行時間の予測

プログラム開発フローにおけるスレッドチェッカの位置付けを以下に示す。



MTAlpha を使用して簡単な SSE2 のマルチスレッドによる試験を行った。
その結果、MTAlpha の SSE2 スレッド x 2 を実行した場合、1896*1380 ピクセルで約 0.047 秒が Thread Checker を使用すると約 3.4 秒となった。つまり、3.4 / 0.047 ≧ 72 倍となり、Secure Decoder での適用は無理と判断できる。

５-３-２-１-４　Intel C++ Compiler 7.1 for Windows

http://www.xlsoft.com/jp/products/intel/icppwin.html

C マガジン 2003 年 6 月号に詳細なレポートがあり。
2003 年 6 月に W_CC_PC_7[1].1.013.exe がリリースされている。しかし、最新リリースでは、プロジェクトが新規、既存両方でオープンできなくなる、等の障害が出て、テスト不可となったため、実績がある W_CC_P_7.1.005.exe を使用した。

５-３-２-１-５　Intel Integrated Performance Primitives V3.0

http://www.xlsoft.com/jp/products/intel/ipp.html

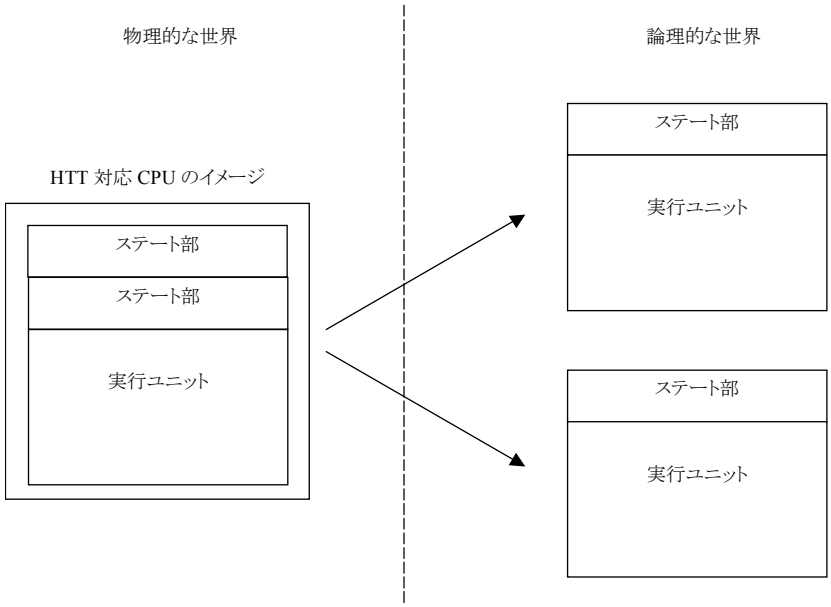
Inter IA-32 系 Pentium 4/Xeon/M CPU、および IA-64 系 ItaniumⅡ CPU 向けに最適化された信号処理、イメージ、スピーチ、画像およびオーディオ処理、ベクタ操作、マトリックス、各種 CODEC などに対応したソフトウェアライブラリである。
ライブラリで提供されているメソッドとして、

```
ippsIIR_32f_I(
    Ipp32f* pSrcDst,
    int len,
    IppsIIRState_32f* pState);
ippiBlur_8u16s_C1R(
    const Ipp8u* pSrc,
    int srcStep,
    Ipp16s* pDst,
    int dstStep,
    IppiSize* pROI,
    IppConvState* state,
    int stage);
ippmSub_vac_32f_5x1_PS2(
    const Ipp32f** pSrc,
    Ipp32s srcROIShift,
    Ipp32s srcStride2,
    Ipp32f val,
    Ipp32f** pDst,
    Ipp32s dstROIShift,
    Ipp32s dstStride2,
    Ipp32u count);
```

などがあるが、このライブラリのためのメソッドのため、関連部分のソースを変更する必要がある。

５-３-２-２　Hyper-Threading Technology

Hyper-Threading Technology（以降 HTT）は，スレッドに効果的と言われている。ただ、調べてみると、それほど単純ではなく適用をうまく考えないと逆に足を引っ張る場合も考えられる。HTT はひとつの物理的な CPU を 2 つの論理プロセッサで物理実行ユニットを共有する技術である。



各論理 CPU は以下の独立したレジスタ、その他を有する。

- ✓ IA-32 general-purpose registers
- ✓ IA-32 segment registers
- ✓ IA-32 control registers
- ✓ IA-32 debug registers
- ✓ IA-32 most of the machine specific registers (MSRs)
- ✓ Advanced programmable interrupt controller (APIC)

従って、同時に二つのアーキテクチャステートを保持する事が可能である。
BIOS から見た場合、各論理 CPU は完全に独立した Dualprocessor/Multiprocessor として認識される。
しかし、NetBurst Microarchitecture、FPU Registers、MMX Registers、SSE/SSE2 Registers などは共有しているため、その使い方を注意する必要がありそうである。

<u>性能向上が期待できるケース</u>

- ✓ 異なる実行ユニット（固定小数点演算＋浮動小数点演算の組み合わせなど）を使うスレッドが複数ある場合.

- ✓ 同じ論理ユニットを使うが，ミスキャッシュが比較的多い場合（ミスキャッシュにより実行ユニットが待ちに入るのを有効に使えるようになるため）

性能向上が期待できないケース

- ✓ 同じ論理ユニットを隙間無く使うスレッドが複数ある場合．（ミスキャッシュが少ないこと）
- ✓ スレッドの排他制御がうまく行っていない場合

５-３-２-３　試験環境

以降の試験は、次の環境にて試験を行った。

- Processor
  Intel Pentium 4 1.7Ghz (SystemBus　400Mhz)

- Memory
  512MB

- OS
  Microsoft Windows XP

- Compiler
  Intel C++ Compiler 7.0（以降、Intel C++7.1 と記載する）
  Microsoft Visual Studio.NET 2003（以降、VC++2003 と記載する）

５-３-２-４　最適化に関連するコンパイラオプション

プログラム実行速度の最適化に関連するコンパイラオプションを以下に列記する。設定時に有効となるように、VC++プロジェクトのプロパティページの構成順に記載する。
それぞれの意味は、ヘルプ等を参照すること。また、文字はVC++2003のみのオプション、*文字*はIntel C++7.1のみのオプションである。

【全般】
　プログラム全体の最適化　　/GL
【Intel Specific】　Intel C++ Compiler 用設定
　Compiler and Environment Setting
　Profile-Guided Optimization (PGO)
　　PGO Phase　Phase 1：Instrumentation Build (/Qprof_gen)
　　Phase 1 : Enhanced Instrumentation Build (/Qprof_genx)
　　Phase 3：Feedback Build (/Qprof_use)
　Whole Program Optimization
　　Enable WPO　　Yes (/Qipo)　/GL オプションと等価
【C/C++】
　最適化、または *Optimization*
　　最適化、または *Optimization* /O2 (実行速度)、または
　　/O3 (/O2 + High Level Optimization)
　　/O2 を指定すると、
　　/Og　　グローバルの最適化
　　/Oi　　組み込み関数の生成
　　/Ot　　実行速度の優先
　　/Oy　　フレームポインタの省略
　　/Ob1　関数の inline 化（Intel C++の場合）
　　/Ob2　関数の inline 化（Visual C++の場合）
　　/Gs　　スタックチェック呼び出しの制御
　　/GF　　同一文字の削除（Visual C++の場合）
　　/Gf　　同一文字の削除（Intel C++の場合）
　　/Gy　　関数レベルのリンクの有効化
　　を個別に指定した場合と等価である。

/O3 を指定すると、/O2 の指定に加えて、データのプリフェッチ、ループ変形、スカラーデータ置換、などの最適化を行う。

プロセッサの最適化　　/G7 (Pentium 4 以上)

Use Processor Extensions　　/QaxW　　(Pentium 4 instructions)

Require Processor Extensions/QxW (Pentium 4 instructions)

Loop Unrolling　　/Qunroll[n]

通常は未設定

Parallelization　　/Qparallel　　(Enable Parallelization)

Hyper Threading Technology 等の場合有効

コード生成

拡張命令セットの有効化　　/arch:SSE, /arch:SSE2

VC++2003 には、プロファイラフィードバックによる最適化機能、/O3 によるキャッシュ管理最適化機能が見当たらない。

５-３-２-５　MMX, SSE, SSE2 による性能の差

同一条件（PC、コンパイラ、オプション）で同一処理の処理を行った場合の MMX、SSE、SSE2 を使用したときの性能の差を実験で確かる。サンプルソースは、C マガジン等で入手できるものを使用、または転用して評価用プログラムを作成する。
コンパイラは VC++2003、および Intel C++7.1 評価版を使用し、最も最適化するコンパイルオプションを設定する。ただし、Intel C++7.1 の Profile-Guided Optimization (PGO)は適用しない。

- テスト用プログラム
  Cマガジン 2002 年 9 月号添付サンプルプログラムの AlphaMMX と AlphaSSE2 を元に拡張を行い、マルチスレッド対応の MTAlpha を作成した。
  それぞれの処理は、Cマガジン 2002 年 9 月号添付サンプルプログラムを参照されたい。
- 試験内容
  画像サイズ 1896 x 1380 ピクセルを有する画像間のアルファブレンディング演算を行う。
  サンプルソースに SSE 命令を使用したソースが無かったため、今回は SSE による性能の差は確認していない。
- VC++2003 試験結果

```
// シングルスレッドにて実行
Test 1 : Integer. CPU time = 0.194922 秒
Test 2 : Floating Point. CPU time = 0.202069 秒
Test 3 : MMX. CPU time = 0.0302251 秒
Test 4 : SSE2. CPU time = 0.0250771 秒

    C / MMX = 0.194922 / 0.0302251 = 6.44 倍
    C / SSE2 = 0.194922 / 0.0250771 = 7.77 倍
    MMX / SSE2 = 0.0302251 / 0.0250771 = 1.20 倍

// マルチスレッドにて実行
Test 1 : Integer. CPU time = 0.531293 秒
Test 3 : MMX. CPU time = 0.0320763 秒

    C / MMX = 0.531293 / 0.0320763 = 16.56 倍
```

Test 1 : Integer. CPU time = 0.335517 秒
Test 4 : SSE2. CPU time = 0.0492752 秒

C / SSE2 = 0.335517 / 0.0492752 = 6.80 倍

Test 3 : MMX. CPU time = 0.0690364 秒
Test 4 : SSE2. CPU time = 0.0306305 秒

MMX / SSE2 = 0.0690364 / 0.0306305 = 2.25 倍

Test 4 : SSE2. CPU time = 0.0433365 秒
Test 3 : MMX. CPU time = 0.0585859 秒

MMX / SSE2 = 0.0585859 / 0.0433365 = 1.35 倍

Test 1 : Integer. CPU time = 0.380077 秒
Test 1 : Integer. CPU time = 0.418146 秒

マルチスレッド - シングルスレッド =
(0.380077 + 0.418146) / 4 - 0.194922 = 0.004633 秒

Test 3 : MMX. CPU time = 0.0558096 秒
Test 3 : MMX. CPU time = 0.0587813 秒

マルチスレッド - シングルスレッド =
(0.0558096 + 0.0587813) / 4 - 0.0302251 = -0.056614 秒

Test 4 : SSE2. CPU time = 0.0488909 秒
Test 4 : SSE2. CPU time = 0.0478616 秒

マルチスレッド - シングルスレッド =
(0.0488909 + 0.0478616) / 4 - 0.0250771 = -0.000889 秒

- Intel C++7.1 試験結果

// シングルスレッドにて実行

Test 1 : Integer. CPU time = 0.175819 秒
Test 2 : Floating Point. CPU time = 0.182707 秒
Test 3 : MMX. CPU time = 0.0306106 秒
Test 4 : SSE2. CPU time = 0.0250654 秒

C / MMX = 0.175819 / 0.0306106 = 5.74 倍
C / SSE2 = 0.175819 / 0.0250654 = 7.01 倍
MMX / SSE2 = 0.0306106 / 0.0250654 = 1.22 倍

// マルチスレッドにて実行
Test 1 : Integer. CPU time = 0.310755 秒
Test 3 : MMX. CPU time = 0.0575895 秒

C / MMX = 0.310755 / 0.0575895 = 5.39 倍

Test 1 : Integer. CPU time = 0.307807 秒
Test 4 : SSE2. CPU time = 0.0510067 秒

C / SSE2 = 0.307807 / 0.0510067 = 6.03 倍

Test 3 : MMX. CPU time = 0.0561503 秒
Test 4 : SSE2. CPU time = 0.0449803 秒

MMX / SSE2 = 0.0561503 / 0.0449803 = 1.24 倍

Test 4 : SSE2. CPU time = 0.0322821 秒
Test 3 : MMX. CPU time = 0.0720131 秒

MMX / SSE2 = 0.0720131 / 0.0322821 = 2.23 倍

Test 1 : Integer. CPU time = 0.399716
Test 1 : Integer. CPU time = 0.345083

マルチスレッド - シングルスレッド =
(0.399716 + 0.345083) / 4 - 0.175819 * 2 = 0.010381 秒

Test 3 : MMX. CPU time = 0.0552065
Test 3 : MMX. CPU time = 0.0552358

マルチスレッド - シングルスレッド =
(0.0552065 + 0.0552358) / 4 - 0.0306106 = -0.055626 秒

Test 4 : SSE2. CPU time = 0.0413937
Test 4 : SSE2. CPU time = 0.0472232

マルチスレッド - シングルスレッド =
(0.0413937 + 0.0472232) / 4 - 0.0250654 = -0.061126 秒

- 考察

C の記述は、アルゴリズムが最適化されていないため、単純に比較するのは危険と考えられる。

VC++2003 では、シングルスレッドで処理した場合、処理時間で C が MMX に対し 6.44 倍、SSE2 に対し 7.77 倍の差が実測された。MMX 対 SSE2 でも MMX が 1.20 倍程度の差が実測された。
Intel C++7.1 では、シングルスレッドで処理した場合、処理時間で C が MMX に対し 5.74 倍、SSE2 に対し 7.01 倍の差が実測された。MMX 対 SSE2 でも MMX が 1.22 倍程度の差が実測された。

マルチスレッドで同時に演算を行った場合、VC++2003、Intel C++7.1 とも単独に処理した時よりも、かえって処理時間の高速が認められた。
これについては、キャッシュミスヒット率の計測で改めて、検討する。

VC++2003 と Intel C++7.1 の性能の比較は、次節で行う。

５-３-２-６　コンパイラの違いによる性能の差

VC++2003、Intel C++7.1 による実行時性能の差を実験で確認する。
C、MMX、SSE2 の比較で使用した MTAlpha.exe での実績では、

```
C   VC++2003 : Intel C++7.1 = 0.194922 : 0.175819 = 1.10 倍
MMXVC++2003 : Intel C++7.1 = 0.0302251 : 0.0306106 = 0.99 倍
SSE2        VC++2003 : Intel C++7.1 = 0.0250771 : 0.0250654 = 1.00 倍
```

である。
MMX、および SSE2 はアセンブラで記述しているため、処理時間に差は認められない。しかし、C で記述した部分は、コンパイラの差が出ており、明らかに Intel C++7.1 が優秀と言える。

別の C のプログラムにより、VC++2003 と Intel C++7.1 のコンパイラの違いによる性能の差を確認する。使用するプログラムは MTIntFloat.exe を使用する。コンパイルオプションは、それぞれで最も最適化されるオプションを設定する。

- 試験項目

  MTIntFloat は以下の試験をマルチスレッドで実行する機能を有する。

  Test 1 : FFT (Floating Point)
  Test 2 : HUFFMAN Encode(Integer)
  Test 3 : HUFFMAN Decode(Integer)
  Test 4 : Pollute Cache
  Test 5 : Streaming Store
  Test 6 : CopyMemory

- VC++2003 コンパイルオプション

  /O2 /Ot /GL /G7 /I "../common" /D "WIN32" /D "NDEBUG" /D "_CONSOLE" /D "_MBCS" /FD /EHsc /MT /GS /arch:SSE2 /FAs /Fa"Release/" /Fo"Release/" /Fd"Release/vc70.pdb" /W3 /nologo /c /Zi /TP

- VC++2003 試験結果

```
Test 4 : Pollute Cache. 結果 = 43.4321 秒
    Pollute/Streaming Store Method call count = 1048575
```

43.4321 / 1048575 = 4.14201e-5 秒

Test 5 : Streaming Store. 結果 = 42.2369 秒
    Pollute/Streaming Store Method call count = 65535

    42.2369 / 65535 = 6.44493e-4

Test 6 : CopyMemory. 結果 = 42.265 秒
    Pollute/Streaming Store Method call count = 65535

    42.265 / 65535 = 6.44922e-4

Test 1 : FFT (Floating Point). 結果 = 0.503012 秒

Input FIRST thread test number : 1
Input SECOND thread test number : 4
    Test 1 : FFT (Floating Point). 結果 = 0.988852 秒
    Test 4 : Pollute Cache. 結果 = 0.948959 秒
      Pollute/Streaming Store Method call count = 11373

    -4.14201e-5 * 11373 + 0.988852 = 0.51778
    0.51778 / 0.503012 = 1.03 倍

Input FIRST thread test number : 1
Input SECOND thread test number : 5
    Test 1 : FFT (Floating Point). 結果 = 0.969397 秒
    Test 5 : Streaming Store. 結果 = 0.930268 秒
      Pollute/Streaming Store Method call count = 718

    -6.44493e-4 * 718 + 0.969397 = 0.50665
    0.50665 / 0.503012 = 1.01 倍

Input FIRST thread test number : 1
Input SECOND thread test number : 6

```
    Test 1 : FFT (Floating Point). 結果 = 0.974873 秒
    Test 6 : CopyMemory. 結果 = 0.943131 秒
      Pollute/Streaming Store Method call count = 727

    -0.00064 * 727 + 0.974873 = 0.50601
    0.50601 / 0.503012 = 1.01 倍


Test 2 : HUFFMAN Encode(Integer). 結果 = 0.107235 秒

Input FIRST thread test number : 2
Input SECOND thread test number : 4
    Test 2 : HUFFMAN Encode(Integer). 結果 = 0.208732 秒
    Test 4 : Pollute Cache. 結果 = 0.207318 秒
      Pollute/Streaming Store Method call count = 2352

    -4.14201e-5 * 2352 + 0.208732 = 0.11131
    0.11131 / 0.107235 = 1.04 倍

Input FIRST thread test number : 2
Input SECOND thread test number : 5
    Test 2 : HUFFMAN Encode(Integer). 結果 = 0.22885 秒
    Test 5 : Streaming Store. 結果 = 0.221836 秒
      Pollute/Streaming Store Method call count = 185

    -6.44493e-4 * 185 + 0.22885 = 0.10961
    0.10961 / 0.107235 = 1.02 倍

Input FIRST thread test number : 2
Input SECOND thread test number : 6
    Test 2 : HUFFMAN Encode(Integer). 結果 = 0.229653 秒
    Test 6 : CopyMemory. 結果 = 0.225079 秒
      Pollute/Streaming Store Method call count = 178

    -6.44922e-4 * 178 + 0.229653 = 0.11485
    0.11485 / 0.107235 = 1.07 倍
```

```
Test 3 : HUFFMAN Decode(Integer). 結果 = 0.52388 秒

Input FIRST thread test number : 3
Input SECOND thread test number : 4
    Test 3 : HUFFMAN Decode(Integer). 結果 = 1.06238 秒
    Test 4 : Pollute Cache. 結果 = 1.04654 秒
      Pollute/Streaming Store Method call count = 11719

    -4.14201e-5 * 11719 + 1.06238 = 0.57697
    0.57697 / 0.52388 = 1.10 倍

Input FIRST thread test number : 3
Input SECOND thread test number : 5
    Test 3 : HUFFMAN Decode(Integer). 結果 = 1.08351 秒
    Test 5 : Streaming Store. 結果 = 1.07621 秒
      Pollute/Streaming Store Method call count = 817

    -6.44493e-4 * 817 + 1.08351 = 0.55695
    0.55695 / 0.52388 = 1.06 倍

Input FIRST thread test number : 3
Input SECOND thread test number : 6
    Test 3 : HUFFMAN Decode(Integer). 結果 = 1.02707 秒
    Test 6 : CopyMemory. 結果 = 1.02051 秒
      Pollute/Streaming Store Method call count = 754

    -6.44922e4 * 754 + 1.02707 = 0.54079
    0.54079 / 0.52388 = 1.03 倍
```

単純な for ループによるメモリコピーを行った場合の VC++2003 のコンパイル結果を確認した。
オリジナルの C ソースとアセンブリング結果を以下に示す。

```
// オリジナル
for(j=0;j<SZ;j++)
    b[j]=a[j];

// アセンブリング結果
;for(j=0;j<SZ;j++)
;    b[j]=a[j];
mov ecx, 102400                    ; 00019000H
mov esi, OFFSET FLAT:?a@@3PANA          ; a
mov edi, OFFSET FLAT:?b@@3PANA          ; b
rep movsd
```

```
// オリジナル
CopyMemory(b,a,sizeof(b));

// アセンブリング結果
; CopyMemory(b,a,sizeof(b));
mov    ecx, 102400                 ; 00019000H
mov    esi, OFFSET FLAT:?a@@3PANA          ; a
mov    edi, OFFSET FLAT:?b@@3PANA          ; b
rep movsd
```

最適化パラメータや前後の状況から、VC++2003 ではアセンブリング結果が for ループ、および CopyMemory（memcpy）で単純な movs 命令を使用して、キャッシュ汚染が発生する場合があるため、注意が必要である。

- Intel C++7.1 コンパイルオプション

  プロファイラフィードバックを行い、最適化を行った。

  /c /Qprof_use /Qprof_dir "Release" /GL /I "../common" /Zi /nologo /W3 /O3 /Og /Ob2 /Oi /Ot /Oy /G7 /QaxW /QxW /D "WIN32" /D "NDEBUG" /D "_CONSOLE" /D "_MBCS" /GF /FD /EHsc /MT /GS /Gy /Fo"Release/" /Fd"Release/vc70.pdb" /Gd /TP
- Intel C++7.1 試験結果

Test 4 : Pollute Cache. 結果 = 23.7288 秒
Pollute/Streaming Store Method call count = 1048575

23.7288 / 1048575 = 2.26295e-5 秒

Test 5 : Streaming Store. 結果 = 24.3393 秒
Pollute/Streaming Store Method call count = 65535

24.3393 / 65535 = 3.71393e-4 秒

Test 6 : CopyMemory. 結果 = 36.0948 秒
Pollute/Streaming Store Method call count = 65535

36.0948 / 65535 = 5.50771e-4 秒

Test 1 : FFT (Floating Point). 結果 = 0.263464 秒

Test 1 : FFT (Floating Point). 結果 = 0.51834 秒
Test 4 : Pollute Cache. 結果 = 0.485923 秒
Pollute/Streaming Store Method call count = 10651

-2.26295e-5 * 10651 + 0.51834 = 0.27731
0.27731 / 0.263464 = 1.05 倍

Test 1 : FFT (Floating Point). 結果 = 0.505539 秒
Test 5 : Streaming Store. 結果 = 0. 475649 秒
Pollute/Streaming Store Method call count = 645

-3.71393e-4 * 645 + 0.50553 = 0.26598
0.26598 / 0.263464 = 1.01 倍

Test 1 : FFT (Floating Point). 結果 = 0.484948 秒
Test 6 : CopyMemory. 結果 = 0.456483 秒
Pollute/Streaming Store Method call count = 401

-5.50771e-4 * 401 + 0.484948 = 0.26408
0.26408 / 0.263464 = 1.00 倍

Test 2 : HUFFMAN Encode(Integer). 結果 = 0.0745275 秒

Test 2 : HUFFMAN Encode(Integer). 結果 = 0.154233 秒
Test 4 : Pollute Cache. 結果 = 0.14073 秒
Pollute/Streaming Store Method call count = 3465

-2.26295e-5 * 3465 + 0.154233 = 0.07582
0.07582 / 0.0745275 = 1.02 倍

Test 2 : HUFFMAN Encode(Integer). 結果 = 0.154906 秒
Test 5 : Streaming Store. 結果 = 0.143216 秒
Pollute/Streaming Store Method call count = 212

-3.71393e-4 * 212 + 0.154906 = 0.07617
0.07617 / 0.0745275 = 1.02 倍

Test 2 : HUFFMAN Encode(Integer). 結果 = 0.133948 秒
Test 6 : CopyMemory. 結果 = 0.122985 秒
Pollute/Streaming Store Method call count = 107

-5.50771e-4 * 107 + 0.133948 = 0.07501
0.07501 / 0.0745275 = 1.01 倍

Test 3 : HUFFMAN Decode(Integer). 結果 = 0.439917 秒

Test 3 : HUFFMAN Decode(Integer). 結果 = 0.886829 秒
Test 4 : Pollute Cache. 結果 = 0.874162 秒
Pollute/Streaming Store Method call count = 19358

-2.26295e-5 * 19358 + 0.886829 = 0.44876
0.44876 / 0.439917 = 1.02 倍

Test 3 : HUFFMAN Decode(Integer). 結果 = 0.86573 秒
Test 5 : Streaming Store. 結果 = 0.858528 秒
Pollute/Streaming Store Method call count = 1132

-3.71393e-4 * 1132 + 0.86573 = 0.44531
0.44531 / 0.439917 = 1.01 倍

Test 3 : HUFFMAN Decode(Integer). 結果 = 0.885354 秒
Test 6 : CopyMemory. 結果 = 0.880422 秒
Pollute/Streaming Store Method call count = 805

-5.50771e-4 * 805 + 0.885354 = 0.44198
0.44198 / 0.439917 = 1.00 倍

- 考察

FFT VC++2003 : Intel C++7.1 = 0.503012 : 0.263464 = 1.91 倍
HUF_E VC++2003 : Intel C++7.1 = 0.107235 : 0.0745275 = 1.44 倍
HUF_D VC++2003 : Intel C++7.1 = 0.52388 : 0.439917 = 1.19 倍

VC++2003 と Intel C++では、/G7 コンパイルオプションや、インストール時の libmmt.lib の問題から、内部では共通のテクノロジを持っていると思われるが、試験結果には大きな差が現れた。

C による記述の場合は、Intel C++7.1 が明らかに有利である。
しかし、ここで使用した Intel C++7.1 コンパイラオプションは、プロファイラフィードバックを使用しているため、最適化に必要なステップは、１）プロファイルコード組み込みビルド、２）実行しプロファイルデータ取得、３）プロファイルデータフィードバックビルドというステップが必要なため、相当面倒である。

マルチスレッドにおける考察は、キャッシュミスヒット率計測で行う。

５-３-２-７　コンパイルオプションの違いによる性能の差

Intel C++7.1 を使い、同一のテストプログラムでのコンパイルオプションの違いによる性能の差を確認する。テスト用プログラムは MTIntFloat.exe を使用する。

- 試験結果

/Qprof_use /GL /O3 /Og /Ob2 /Oi /Ot /Oy /G7 /QaxW /QxW /Gy

プロファイラフィードバック有り、SSE2 有効

FFT (Floating)	0.262111 **秒**
HUFFMAN Encode(Integer)	0.0728665 **秒**
HUFFMAN Decode(Integer)	0.449666 **秒**

/GL /O2 /Ob1 /Oy /G7 /QaxW /QxW /Gy

Pentium4 指定、SSE2 **有効**

FFT (Floating)	0.391848 秒
HUFFMAN Encode(Integer)	0.0778351 秒
HUFFMAN Decode(Integer)	0.447826 秒

/GL /O2 /Ob1 /Oy /G6 /Gy

通常のリリースビルド時オプション

FFT (Floating)	0.559495 秒
HUFFMAN Encode(Integer)	0.0816882 秒
HUFFMAN Decode(Integer)	0.44935 秒

- 考察

今回の試験では、浮動小数点演算でプロファイラフィードバック最適化による効果を確認できた。

しかし、整数演算では大きな違いは確認出来なかった。逆に整数演算ではアルゴリズムから考察した SIMD 化の対象になる可能性が高い。

５-３-２-８　Hyper-Threading Technology 使用の有無による性能の差

Hyper-Threading Technology 対応の試験環境がある場合に実施する。

整数演算処理と浮動小数点演算処理を実行する独立したスレッドを生成する実験用のプログラムを作成する。これにより

- 整数演算スレッドのみ
- 浮動小数点演算スレッドのみ
- 整数演算スレッド　＋　整数演算スレッド
- 浮動小数点演算スレッド　＋　浮動小数点演算スレッド
- 整数演算スレッド　＋　浮動小数点演算スレッド

の実行時の Hyper-Threading Technology の効果を確認する。
テスト用プログラムは MTIntFloat.exe を使用する。

５-３-２-９　キャッシュ汚染防用命令の効果確認

Hyper-Threading Technology の有効性確認用に開発したプログラムを拡張し、Streaming Store（prefetch、movntps、movntdq、sfence 等を使用）の有効性を確認する。
具体的には、MTIntFloat を使用して、演算処理スレッドとキャッシュ汚染スレッド、キャッシュ汚染対処スレッドを同時に実行して、その演算速度を計測する。また、VTune を使用し、キャッシュにヒット率等を確認する。
結果は、先の「コンパイラの違いによる性能の差」で取得した Intel C++7.1 実行データを使用する。

- 整数演算（FFT）スレッド ＋ キャッシュ汚染スレッド
  FFT + Pollute ： FFT 単独 = 0.27731 / 0.263464 = 1.05 倍

- 整数演算（FFT）スレッド ＋ Streaming Store スレッド
  FFT + StreamingStore ： FFT 単独 = 0.26598 / 0.263464 = 1.01 倍

- 整数演算（FFT）スレッド ＋ CopyMemory
  FFT + CopyMemory ： FFT 単独 = 0.26408 / 0.263464 = 1.00 倍

- 浮動小数点演算スレッド（HUFFMAN Encode） ＋ キャッシュ汚染スレッド
  HUF_E + Pollute ： HUF_E 単独 = 0.07582 / 0.0745275 = 1.02 倍

- 浮動小数点演算スレッド（HUFFMAN Encode） ＋ Streaming Store スレッド
  HUF_E + StreamingStore ： HUF_E 単独 = 0.07617 / 0.0745275 = 1.02 倍

- 浮動小数点演算スレッド（HUFFMAN Encode） ＋ CopyMemoy
  HUF_E + CopyMemory ： HUF_E 単独 = 0.07501 / 0.0745275 = 1.01 倍

- 浮動小数点演算スレッド（HUFFMAN Decode） ＋ キャッシュ汚染スレッド
  HUF_D + Pollute ： HUF_D 単独 = 0.44876 / 0.439917 = 1.02 倍

- 浮動小数点演算スレッド（HUFFMAN Decode） ＋ Streaming Store スレッド
  HUF_D + StreamingStore ： HUF_D 単独 = 0.44531 / 0.439917 = 1.01

倍

- 浮動小数点演算スレッド（HUFFMAN Decode） ＋ CopyMemory

  HUF_D + CopyMemory : HUF_D 単独 = 0.44198 / 0.439917 = 1.00 倍

- 考察

  実行結果だけを見ると、キャッシュ汚染スレッド、キャッシュ汚染対策スレッドの有無により、演算スレッドの実行時間に有意な差があるとは言えない。

  VTune による解析データを見る限りでは、WindowsXP のコンテキストスイッチによるスレッド間の影響は、予想外に低いことが分かった。

  今回作成した MTIntFloat.exe は、今回のテストの目的からは不適当であったと思われる。

５-３-２-１０　キャッシュミスヒット率計測

Intel C++7.1 で最適化を行った MTIntFloat を使用して、キャッシュのミスヒット率を VTune を使用して計測を行った。

- 考察

  Intel C++7.1 コンパイラで最適化を行い、FFT、HUFFMAN Encode、HUFFMAN Decode を単独で実行すると、プロセス単位でのキャッシュヒットミスは少ないようである。
  しかし、キャッシュ汚染スレッド（Pollute Cache）とともに実行すると、いずれもプロセス単位ではキャッシュヒットミスが極めて高い数値を示している。しかし、スレッド単位で見ると、メインスレッドは大きな影響を受けていないことが分かる。

  Streaming Store スレッドとともに実行しても、プロセス単位ではキャッシュヒットミスが極めて高い数値を示している。アセンブリングされた結果を見ると、movntpd 命令に変換されているが、prefetch 命令、sfence 命令等は見当たらない。しかし、スレッド単位で見ると、キャッシュ汚染スレッドの時と同様に、メインスレッドは大きな影響を受けていないことが分かる。

  Streaming Store スレッドと WIN32 API である CopyMemory（memcpy も同様）のテスト結果をみると、マルチスレッドで実行したにもかかわらず、キャッシュヒットミスは抑えられたままである。一方、CopyMemory（または memcpy）は単独で実行した場合、速度が遅い。これらことから CopyMemory（または memcpy）は内部では Streaming Store 機能を実装していると思われる。
  変数のコピーや移動では、直後に再利用する変数等を CopyMemory（または memcpy）を使用してセットアップする場合、アクセス時にキャッシュミスが発生し、ペナルティが発生することが予想される。したがって、キャッシュラインサイズ、その他を考慮して、変数名による直接の代入命令等の使用を検討すべきである。（コンパイラの最適化により CopyMemory がストアフォワーディングとなるように実装される可能性はある。未確認。）
  一方、Source Filter 等でバッファのコピーを行う場合、CopyMemory（または memcpy）を使用しても差し支えないと考えられる。

５-３-２-１１　DirectX9 の適用確認

DirectX9.0a が 2003 年 5 月にリリースされた。
VC++2003 を使用して、DirectX9.0a を既存 Secure Decoder にリンク後、実行できることを確認した。

bs-j.ts を処理ルート1で視聴中の Secure Decoder を VTune によりコールグラフ解析、サンプリング解析を行った。
シンボル情報を適切に入力することにより、ブレイクダウンしながら、各種調査が可能である。

DirectX9.0a は、DirectX8.1b より安定しているように感じる。GUI の視聴開始ボタン、視聴停止ボタンを押下した時の応答がスムーズである。

その後、2003 年 7 月に DirectX9.0b がリリースされた。

最新のグラフィックスボードは、Windows XP ＋ DirectX9 に最適化しているため、それらのリソースを活かすためには DirectX9 の正式な採用を検討するべきと考える。

５-３-２-１２　Secure Decoder によるコンパイラ性能比較

DevPartner7.0 を考慮すると、VC++2002 となるため、Secure Decoder の Intel C++7.1 によるビルドの効果検討では、今までの環境である VC++2002 と Intel C++7.1 の比較を行う。

５-３-２-１２-１　Intel C++7.1 による Secure Decoder のビルドについて

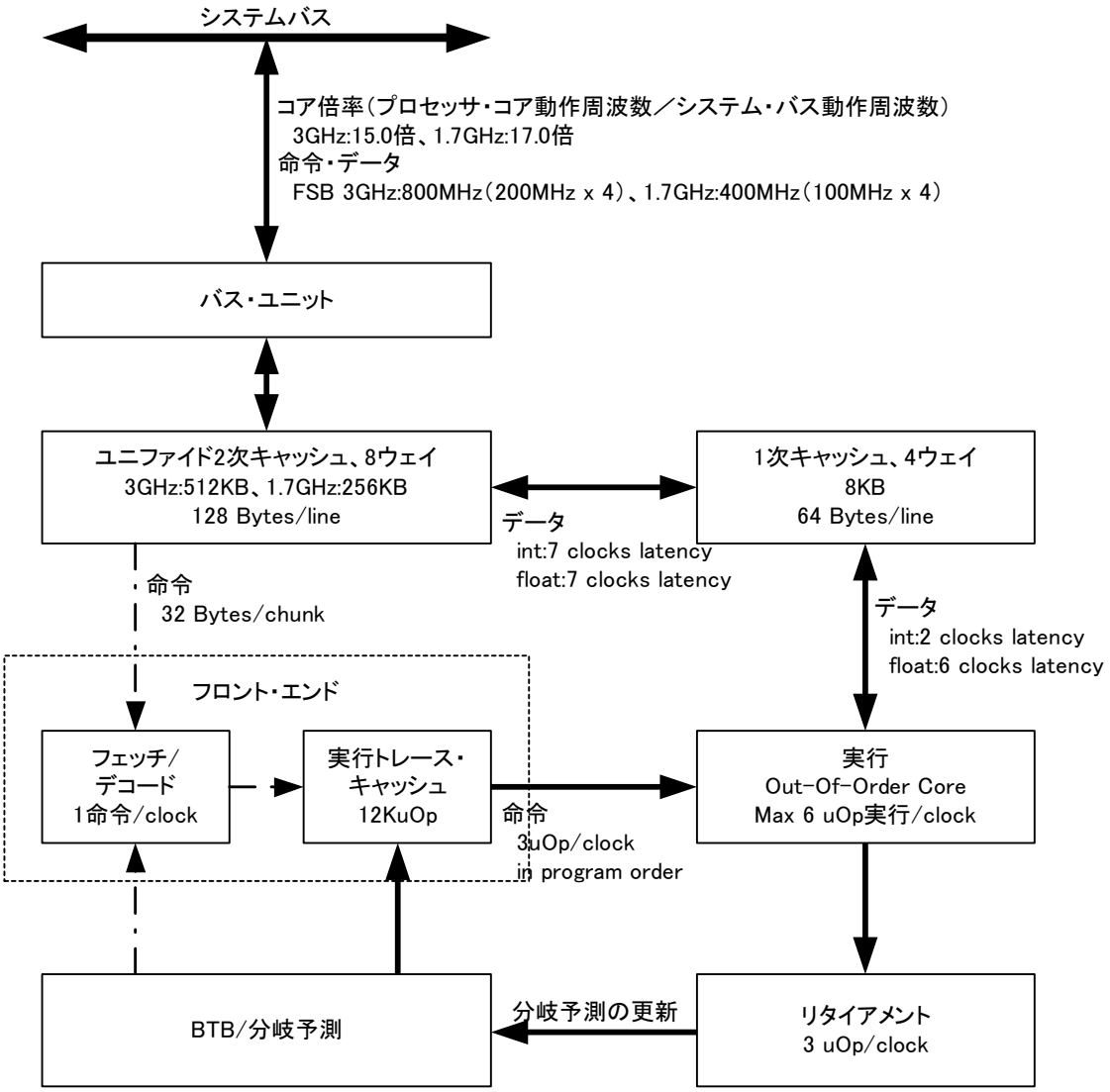
2002 年 5 月 23 日版 VS.NET 2002 用 Secure Decoder ソリューションに含むプロジェクトの内、ActiveX プロジェクト、および COM オブジェクトは、Intel C++7.1 ではそのままビルドすることはできない。
Intel C++7.1 ドキュメントを確認したところ、VC++が ATL 用プロジェクト登録時に自動生成するソースの IDL 属性フォーマットに対応していない、との記載が有った。
従って、エラーとなるプロジェクトは、VC++のプロジェクトウィザードにより、ATL オプションの属性チェックボタンをオフにして、再作成を行う必要がある。
従って、TAOSecureControllerProject 、ProgramStreamProject 、ProgramHDDProject 、TAORecOut StreamController、TAOHDDInStreamController の各プロジェクトの再作成を行った。

しかし、フィルタグラフ関連のプロジェクトで原因不明のエラーが発生するため、Interl C++7.1 による Secure Decoder のビルドは中断せざるを得なくなった。

５-３-２-１３　Intel Pentium 4 Processor について

５-３-２-１３-１　Intel NetBurst マイクロアーキテクチャ

Intel Pentium4 1.7GHzと3.0GHzの場合のNetBurstマイクロアーキテクチャブロック図を以下に示す。各ブロックの機能は、「インテル Pentium4 プロセッサ最適化リファレンス・マニュアル」を参照すること。



整数データが 1 次キャッシュにヒットした場合のレイテンシは 2 クロックである。
1 次キャッシュにミスし、2 次キャッシュでヒットした場合のレイテンシは 7 クロックである。
2 次キャッシュにミスし、外部メモリまでリードする場合、コア倍率に大きく依存する。コア

比率が 15 倍の場合、システムバス 1 クロックがプロセッサクロックの 15 クロックに相当する。

実行 Out-Of-Order コアは、ポート 0 からポート 3 までの 4 個の実行ユニットを有する。ポート 0 とポート 1 の ALU は 2 uOp/clock のスケジュールが可能のため、最大 6 uOp/clock を同時に実行することが可能である。
アセンブリ言語でコーディングする場合、後述する IA-32 命令のレイテンシとスループット、および使用する実行ユニットを勘案して、命令の効率的な実行や、リソースの競合の回避、等を行うことができる可能性がある。

５-３-２-１３-２　Intel Pentium4 プロセッサのパフォーマンス指標

Pentium4 プロセッサ最適化マニュアルにパフォーマンス指標が規定されている。日本語訳された「インテル Pentium 4 プロセッサおよびインテル Xeon プロセッサ最適化リファレンス・マニュアル」（資料番号：248966J-004）の「付録 B.インテル Pentium 4 プロセッサのパフォーマンス指標」と、原文である「IA-32 Intel Architecture Optimization Reference Manual」（資料番号：248966-008）の「Appendix B Intel Pentium 4 Processor Performance Metrics」では、その項目に若干差異があるため注意すること。

VTune でレポートされる各種統計データは、最適化マニュアルで規定している指標をベースに作成されるため、適宜参照すること。

５-３-２-１３-３　IA-32 命令のレイテンシとスループット

前述リファレンス・マニュアルの「付録 C. IA-32 命令のレイテンシシとスループット」に一般的な IA-32 命令、全 MMX テクノロジ命令、全 SSE テクノロジ命令、全 SSE2 テクノロジ命令のレイテンシとスループットが表示されている。

リファレンス・マニュアルには、レイテンシは「IA-32 命令を構成しているすべてのマイクロオペレーション(μ OP) の実行が実行コアで完了するのに要するクロック・サイクル数。」、スループットは「発行ポートが同じ命令を再度自由に受け入れられるようになるまで待たなければならないクロック・サイクル数。」のように規定している。

表に記載の IA-32 はオペランドにレジスタを指定した場合である。従って、オペランドに

メモリを指定した場合、キャッシュヒット時アクセスレイテンシ、キャッシュミスヒット時のメモリアクセスレイテンシが加算されることに注意すること。

５-３-２-１２-１-１シフト関連のレイテンシとスループット

シフトを行うアセンブリ命令には、以下のものがある。

32 ビットシフト命令

SAL/SAR/SHL/SHR	Latency 4 / Throughput 1	

MMX64 ビットシフト命令

PSLLQ/PSRLQ	Latency 2 / Throughput 1	MMX_SHIFT unit

SSE2 128 ビットシフト命令

PSLLDQ/PSRLDQ	Latency 4 / Throughput 2	MMX_SHIFT unit

シフトデータの補填処理が、実際のシフト処理中にどの程度発生するかに依存するが、データだけをみると、PSLLQ/PSRLQ 命令を使用するのが妥当と思われる。

スワップ関連のレイテンシとスループット

スワップを行うアセンブリ命令には、以下のものがある。

16 ビットスワップ命令

XCHG	Latency 1.5 / Throughput 1	ALU

32 ビットスワップ命令

BSWAP	Latency 7 / Throughput 1	ALU

SSE2 128 ビットスワップ命令

PSHUFD	Latency 4 / Throughput 2	MMX_SHFT
PSHUFHW	Latency 2 / Throughput 1	MMX_SHFT
PSHUFLW	Latency 2 / Throughput 1	MMX_SHFT

最適化マニュアルの「例 4-14 命令を 3 つ使用したリバース」。

	7	6	5	4	3	2	1	0
PSHUFLW (0,1,2,3)	7	6	5	4	0	1	2	3	
PSHUFHW (0,1,2,3)	4	5	6	7	0	1	2	3	
PSHUFD (1,0,3,2)	0	1	2	3	4	5	6	7	

MMX 関連のレイテンシとスループット

MMX テクノロジ命令を使用した場合、EMMS 命令を最後にコールする必要がある。
EMMS 命令のレイテンシとスループットを以下に示す

EMMS	Latency 12 / Throughput 12

EMMS 命令は、アプリケーションから見ると全くのオーバヘッドであるため、EMMS 命令を必要としない SSE 命令、または SSE2 命令への移行を検討すべきである。

５-３-２-１４　Intel C++ Compiler 7.1 について

VC++ 2003 と Intel C++7.1 の性能比較の結果から、本プロジェクトでは Intel C++7.1 の採用が妥当と考えられる。

５-３-２-１４-１　Visual Studio.NET 2002 との親和性

Intel C++7.1 をインストールする手順によって、様々な障害の発生を経験した。
開発環境は、C ドライブ HDD をフォーマットし、Windows XP、Visual Studio.NET 2002、Intel C++7.1、VTune7.0 の順にインストールすること。

５-３-２-１４-２　インラインアセンブラと組み込み関数

VC++では、MMX テクノロジ命令、ストリーミング SIMD 拡張命令（SSE）、ストリーミング SIMD 拡張命令２（SSE2）を使用する場合、インラインアセンブラ言語命令として記述する必要があるが、Intel C++では、MMX、SSE、SSE2 命令に対応した組み込み関数が提供されている。
Intel C++コンパイラ組み込み関数については、Intel C++コンパイラのヘルプから「リファレンス情報－インテル C++組み込み関数リファレンス」、または「IA-32 インテルアーキテクチャ・ソフトウェア・デベロッパーズ・マニュアル、中巻：命令セット・リファレンス」を参照すること。

５-３-２-１４-２-１　組み込み関数使用上の注意

組み込み関数名には、以下の規則がある。（インテル C++コンパイラヘルプより）

ほとんどの組込み関数名は、次の表記規則に従います。

_mm_<intrin_op>_<suffix>

<intrin_op>	組込み関数の基本操作を示します。例えば、加算の場合は add、減算の場合は sub になります。
<suffix>	命令の操作対象となるデータの型を示します。各サフィックスの最初の 1 文字または 2 文字は、データがパックドデータ(p)、拡張パックドデータ(ep)、またはスカラデータ(s)であることを示します。その他の文字

	は、次のとおりデータ型を示します。 • s 単精度浮動小数点値 • d 倍精度浮動小数点値 • i128 符号付き 128 ビット整数 • i64 符号付き 64 ビット整数 • u64 符号なし 64 ビット整数 • i32 符号付き 32 ビット整数 • u32 符号なし 32 ビット整数 • i16 符号付き 16 ビット整数 • u16 符号なし 16 ビット整数 • i8 符号付き 8 ビット整数 • u8 符号なし 8 ビット整数

変数名を付加した数字は、パックされたオブジェクトの要素を示します。例えば、r0 は r の最下位ワードです。一部の組込み関数は、2 つ以上の命令で実行するため、「複合組込み関数」と呼ばれます。

コードの中で組込み関数を使用するには、次の構文の行を挿入します。

data_type intrinsic_name (parameters)

各項の意味は次のとおりです。

data_type	void、int、_m64、_m128、_m128d、_m128i、_int64 のうちのいずれかです。すべてのインテル アーキテクチャでサポートする組込み関数は、組込み関数の構文の定義に従って、その他のデータ型を返す場合があります。
intrinsic_name	この名前は、実際の命令をインライン展開する代わりに C++ コード内で使用できる関数として機能します。
parameters	各組込み関数が要求するパラメータを表します。

MMX テクノロジ命令用組み込み関数

MMX テクノロジ組み込み関数を使用する場合、

```
#include <mmintrin.h>
```

をインクルードすること。

MMX テクノロジ命令のほとんどは組み込み関数として提供されている。
MMX テクノロジ命令用組み込み関数には、

一般的組み込み関数	12 種類
パックド算術演算組み込み関数	17 種類
シフト組み込み関数	16 種類
論理演算組み込み関数	4 種類
比較組み込み関数	6 種類
設定組み込み関数	10 種類

がある。

SSE テクノロジ命令用組み込み関数

SSE テクノロジ組み込み関数を使用する場合、

```
#include <xmmintrin.h>
```

をインクルードすること。

SSE テクノロジ命令のほとんどは組み込み関数として提供されている。
SSE テクノロジ命令用組み込み関数には、

算術演算組み込み関数	18 種類
論理演算組み込み関数	4 種類
比較操作組み込み関数	35 種類
変換操作組み込み関数	13 種類
ロード操作、設定操作、ストア操作、メモリ操作、初期化操作組み込み関数	23 種類
整数演算組み込み関数	13 種類
その他の組み込み関数	10 種類

がある。

SSE2 テクノロジ命令用組み込み関数

SSE2 テクノロジ組み込み関数を使用する場合、

```
#include <emmintrin.h>
```

をインクルードすること。

SSE2 テクノロジ命令のほとんどは組み込み関数として提供されている。
SSE2 テクノロジ命令用浮動小数点演算組み込み関数には、

浮動小数点算術演算組み込み関数	14 種類
論理演算組み込み関数	4 種類
比較操作組み込み関数	36 種類
変換操作組み込み関数	13 種類
ロード操作組み込み関数	7 種類
設定操作組み込み関数	6 種類
ストア操作組み込み関数	7 種類
その他の操作組み込み関数	4 種類

がある。

SSE2 テクノロジ命令用整数演算組み込み関数には、

整数算術演算組み込み関数	31 種類
整数論理演算組み込み関数	4 種類
整数シフト操作組み込み関数	18 種類
整数比較操作組み込み関数	9 種類
変換操作組み込み関数	5 種類
シャッフル用マクロ関数	1 種類
キャッシュ操作組み込み関数	7 種類
その他の組み込み関数	20 種類
ロード操作組み込み関数	3 種類
設定操作組み込み関数	13 種類
ストア操作組み込み関数	4 種類

がある。

メモリ割り当て関数

組み込み関数をサポートする機能として、データのアラインメント用拡張構文、メモリブロックの割り当て関数が提供されている。

データのアラインメントのサポートするための拡張構文は、

```
align(n)
```

である。
コンパイラに対する命令では、

```
__declspec( align(n) )
```

を使用する。

アラインメントに有ったメモリブロック割り当て関数は、

```
void* _mm_malloc (int size, int align);
void _mm_free (void *p);
```

である。
_mm_malloc を使用して割り当てられたメモリは、_mm_free を使用して解放すること。

IA プロセッサ命令用組み込み関数

後述する SIMD 命令用組み込み関数の他に、全ての IA プロセッサで使用できる 84 種類の組み込み関数が提供されている。それらの組み込み関数を使用することにより、アセンブラ命令を C 言語内で直接記述することができる。

５-３-２-１４-３　SIMD 用クラス・ライブラリ

Intel C++コンパイラは、組み込み関数に加えて、SIMD 演算用クラス・ライブラリが提供されている。
提供されているクラスは、

- 整数ベクトル(Ivec)クラス
- 浮動小数点ベクトル(Fvec)クラス

がある。

先に説明した組み込み関数は、アセンブラを意識した手続き型関数である。
しかし、SIMD 用クラス・ライブラリでは、operator &等の論理演算子、operator +等の算術演算子、operator >>等のシフト演算子、cmpeq 等の比較演算子のクラスへの隠蔽、select_eq 等の条件付選択演算子、pack_sat 等のパック演算子、unpack_high 等のアンパック演算子、F64vec2ToInt 等の変換演算子、等のサポート関数を準備しているため、SIMD 用クラス・ライブラリによるコーディングでは、特にアセンブラを意識した記述を行う必要がなくなる。

５-３-２-１６-２-１　ハードウェアとソフトウェアの要件

SIMD 用クラス・ライブラリを使用するためのハードウェアとソフトウェア条件を転記する。

各クラス・ライブラリを使用するときはインテル® C++ コンパイラ バージョン 4.0 またはそれ以上をインストールしてください。インテル C++ クラス・ライブラリとは、次の表に示すように、各種インテル・プロセッサで使用できるすべての拡張命令の中からいくつか選び出した関数のことです。

クラス・ライブラリを使用するプロセッサの必要条件

ヘッダファイル	拡張命令セット	対応プロセッサ
ivec.h	MMX® テクノロジ	MMX テクノロジ Pentium®、Pentium II プロセッサ、Pentium III プロセッサ、Pentium 4 プロセッサ、インテル® Xeon™ プロセッサおよび Itanium® プロセッサ
fvec.h	ストリーミング SIMD 拡張命令	Pentium III プロセッサ、Pentium 4 プロセッサ、インテル Xeon プロセッサ、および Itanium プロセッサ
dvec.h	ストリーミング SIMD 拡張命令 2	Pentium 4 プロセッサおよびインテル Xeon プロセッサ

５-３-２-１４-４　処理の並列化

Intel C++コンパイラは、OpenMP による並列化と自動並列化をサポートしている。
ここで言う並列化と通常のマルチスレッドとは意味が異なる。通常は、プログラマがスレッドの起動や終了、他スレッドとの共有データの排他制御、等は全て行う必要がある。

Intel C++コンパイラが提供する並列化では、単一のプロセスとして実行を開始（マスタスレッド）後、マスタスレッドは、シーケンシャルに実行するが、最初の並列構造が検出されると、ワーカスレッドを自動的に起動し、マスタスレッドと協調したマルチスレッドで処理を実行する。しかし、並列対象の処理が終わるとワーカスレッドは終了し、再びマスタスレッドのみによるシーケンシャル処理に戻る。

OpenMP は、プログラマが明示的に並列化すべき部分に OpenMP 用ディレクティブを挿入する。OpenMP ディレクティブはシリアル・アプリケーションを素早く並列アプリケーションに変換できる一方、並列処理を含み、適切なコンパイラ・ディレクティブを追加するアプリケーション・コードの特定部分を、プログラマは明示的に識別する必要がある。
-Qparallel オプションで起動された自動並列化は、並列処理を含むループ構造を自動

的に識別する。コンパイル中、コンパイラは、並列処理のためにコード・シーケンスを別々のスレッドに自動的に分解しようと試みる。他にプログラマにかかる負荷はない。

５-３-２-１４-２-１　OpenMP

OpenMP については、Intel C++コンパイラの「最適化 - 並列化 - OpenMP による並列化」を参照すること。

(1)ディレクティブ・フォーマットと構造

OpenMP モードで Intel C++コンパイラを起動するには、-Qopenmp オプションを使用する。このコンパイラオプションにより、マルチ・スレッド・コードを生成できるようになる。

OpenMP ディレクティブフォーマットは以下の通りである。

```
#pragma omp directive-name [clause, ...] newline
```

各アイテムの意味は次の通り。

# pragma omp	全ての OpenOP ディレクティブに必須
directive-name	有効な OpenMP ディレクティブ pragma の後および clause（節）の前に出力する必要がある
clause	節。オプション clause は順番に関係なく指定でき、制限されていない限り必要に応じて繰り返すことができる。
newline	必須。このディレクティブに囲まれた構造ブロックを続行する。

(2)環境変数

標準環境変数

変数	説明	デフォルト
OMP_SCHEDULE	ランタイム・スケジュールの型とブロックサイズを設定します。	STATIC（ ブロックサイズの指定なし）
OMP_NUM_THREADS	実行時に使用するスレッド数を設定します。	プロセッサの数
OMP_DYNAMIC	スレッド数の動的な調整を有効(TRUE)または無効(FALSE)にします。	FALSE

OMP_NESTED	ネストされた並列処理を有効(TRUE)または無効(FALSE)にします。	FALSE

インテル拡張環境変数

環境変数	説明	デフォルト
KMP_LIBRARY	OpenMP ランタイム・ライブラリ・スループットを選択します。この変数の値には、実行モードを示す serial、turnaround または throughput があります。この変数が指定されなかった場合、デフォルト値の throughput が使用されます。	throughput (実行モード)
KMP_STACKSIZE	各並行スレッドがプライベート・スタックとして使用するバイト数を設定します。オプションの b、k、m、g または t サフィックスを使用して、確保するバイト数をバイト、キロバイト、メガバイト、ギガバイトまたはテラバイトで指定してください。	IA-32: 2m Itanium® コンパイラ: 4m

(3)並列処理モデルの擬似コード

一般的な OpneMP ディレクティブをいくつか使用した擬似プログラムのサンプルを以下に示す。各ディレクティブの意味は Intel C++コンパイラのヘルプを参照すること。

```
main()
{ // Begin serial execution
  ... // Only the master thread executes
  #pragma omp parallel // Begin a Parallel Construct, form a team.
  {
     ...  //  This is Replicated Code
     ...  //  (each team member executes the same code)
     #pragma omp sections    // Begin a Worksharing Construct
     {
        #pragma omp section  // One unit of work
           {...}  //
        #pragma omp section  // Another unit of work
           {...} //
     } // Wait until both units of work complete
```

```
        ... // More Replicated Code
        #pragma omp for nowait // Begin a Worksharing Construct;
        for(...) { // each iteration is unit of work
            ...  // Work is distributed among the team members
        } // End of Worksharing Construct;

        // nowait was specified, so threads proceed
        #pragma omp critical  // Begin a Critical Section
        {
            ... // Replicated Code, but only one thread can execute it
            ... // at a given time
        }
        ...  // More Replicated Code
        #pragma omp barrier // Wait for all team members to arrive
        ...  // More Replicated Code
    } // End of Parallel Construct;
    // disband team and continue serial execution
    ...  // Possibly more Parallel constructs
} // End serial execution
```

自動並列化

インテル C++ コンパイラの自動並列化機能は、入力プログラムのシーケンシャル部分を同等のマルチスレッド・コードに自動的に変換する。自動パラレライザは、プログラムのループのデータフローを分析して、安全かつ効率的に並列実行可能なループに対するマルチスレッド・コードを生成する。これにより、SMP(対称型マルチプロセッサ)システムの並列アーキテクチャを活用できるようになる。

自動並列化を有効にするには、-Qparallel オプションを使用する。-Qparallel オプションは、並列で安全に実行できる並列ループを検出して自動的にこれらのループのマルチ・スレッド・コードを生成する。

(1)自動並列化オプション

-O2 (または-O3)最適化オプションがオン(デフォルトは -O2)の場合、-Qparallel オプションは自動並列化を有効にする。-Qparallel オプションは、並列で安全に実行できる

並列ループを検出して自動的にこれらのループのマルチ・スレッド・コードを生成する。

オプション	説明
-Qparallel	自動パラレライザを有効にします。
-Qparallel_threshold{1-100}	自動並列化に必要な作業しきい値を制御します。詳細は、後のサブセクションを参照してください。
-Qpar_report{1|2|3}	自動並列化の診断メッセージを制御します。詳細は、後のサブセクションを参照してください。

(2)自動並列化の環境変数

変数	説明	デフォルト
OMP_NUM_THREADS	使用されるスレッド数を制御します。	実行ファイルを生成する際にシステムに現在搭載されているプロセッサ数
OMP_SCHEDULE	ランタイム・スケジューリングのタイプを指定します。	スタティック

C 言語とインラインアセンブラの性能比較

MTIntFloat で用意した Streaming Store 用ソースコードを以下に示す。

```
// C 言語
for(int j=0;j<n;j++)
        b[j]=a[j];


// インラインアセンブラ
_asm {
        mov             eax, DWORD PTR [a]
        add             eax, -64
        mov             ebx, DWORD PTR [b]
        add             ebx, -64
        mov             edi, DWORD PTR [n]
        xor             esi, esi
L0:
```

```
        add             eax, 64
        add             ebx, 64
        movapd          xmm3, XMMWORD PTR [eax]
        movapd          xmm2, XMMWORD PTR [eax+16]
        movapd          xmm1, XMMWORD PTR [eax+32]
        movapd          xmm0, XMMWORD PTR [eax+48]
        movntpd         XMMWORD PTR [ebx], xmm3
        movntpd         XMMWORD PTR [ebx+16], xmm2
        movntpd         XMMWORD PTR [ebx+32], xmm1
        movntpd         XMMWORD PTR [ebx+48], xmm0
        add             esi, 8          ; 64/sizeof(double)
        cmp             esi, edi
        jl              L0

        sfence
}
```

インラインアセンブラは、Streaming Store 機能を確認するためであるため、最適化は考えていない。

今回の C とアセンブラを Intel C++7.1 のプロファイルフィードバックによるビルドを行った場合、VTune の解析結果では、特に有意な差異は認められなかった。どちらかと言うと、C の方が優れたコードを生成しているように思われる。
このことから、コーディングを行う場合、最初からアセンブラで記述するのではなく、まず C でビルドし、その解析結果からアセンブラ化するかを判断しても良いだろう。

５-３-２-１５　VTune Performance Analyzer について

５-３-２-１５-１サンプリングの使用

サンプリング・データ・コレクタは、システム全体の割り込み的でないパフォーマンス・データを収集する。これにより、OS、デバイスドライバ、アプリケーション・ソフトウェアを含む、システム上で稼働中のソフトウェアをすべてモニタすることができる。
サンプリングとしては、

- タイム・ベース・サンプリング（TBS）
  命令アドレス・サンプリング を使用することにより、OS、デバイスドライバ、アプリケーション・ソフトウェアを含む、システム上で稼働中のソフトウェアをすべてモニタすることができる。サンプリング・メカニズムにタイム・ベース・サンプリング（TBS）を選択した場合、VTune™ パフォーマンス・アナライザはオペレーティング・システム・タイマを使用して、一定の間隔（デフォルトは 1ms）で割り込み、すべてのアクティブな命令アドレスのサンプルを収集する。収集したサンプルは、システム上で実行しているすべてのプロセスに関するパフォーマンス・データを提供する。実行するのに最も時間を費やしたプロセスは、最も多い数のサンプルがある。
- イベント・ベース・サンプリング（EBS）
  イベント・ベース・サンプリング （EBS）を使用して、Cache Misses や Misdirected Branches などのプロセッサ・イベントが原因となるシステム全体におけるソフトウェアのパフォーマンス問題を識別する。EBS データから、プログラムで最も多くのプロセッサ・イベントを生成したプロセス、スレッド、モジュール、関数およびソース行を特定することができる。また、どのイベントがプログラムのパフォーマンスに影響を与えたのか特定することもできる。

がある。
今回は、キャッシュミス等の解析を中心に行うため、イベント・ベース・サンプリング（EBS）を使用して解析を行う。従って、これ以降は、主に EBS について説明する。

５-３-２-１４-２-１　サンプリングによる解析

サンプリングを行い、アプリケーションのチューニングを行う場合、VTune では以下のステップを推奨している。

- Pentium 4 プライマリ・パフォーマンス・チューニング・イベント
- Pentium 4 セカンダリ・パフォーマンス・チューニング・イベント

- Pentium 4 詳細イベント

Pentium 4 プライマリ・パフォーマンス・チューニング・イベント

Pentium 4 プライマリ・パフォーマンス・チューニング・イベントは、以下のイベントを含む。

【命令 Mix イベント】

- Packed Single-precision Floating-point Streaming SIMD ExtensionInstructions Retired(リタイアしたパックド単精度の浮動小数点ストリーミング SIMD 拡張命令)
- Packed Double-precision Floating-point Streaming SIMD Extension 2 (SSE2) Instructions Retired (リタイアしたパックドダブル精度の浮動小数点ストリーミング SIMD 拡張 2 (SSE2) 命令)
- Scalar Single-precision Floating-point Streaming SIMD Extension (SSE) Instructions Retired (リタイアしたスカラ単精度の浮動小数点ストリーミング SIMD 拡張命令)
- Scalar Double-precision Floating-point Streaming SIMD Extension (SSE) Instructions Retired (リタイアしたスカラダブル精度の浮動小数点ストリーミング SIMD 拡張 (SSE) 命令)
- 128-bit MMX® Instructions Retired(リタイアした 128 ビットの MMX® 命令)
- 64-bit MMX® Instructions Retired (64-bit SIMD Integer Instructions Retired (リタイアした 64 ビットの MMX® 命令 (リタイアされた 64 ビット SIMD 整数演算命令))
- x87 Instructions Retired (リタイアした x87 命令)
- x87 and SIMD Register and Memory Moves Retired (x87 と SIMD のリタイアしたレジスタおよびメモリ移動命令数)

【鍵となる CPI 制限イベント】

- 2nd-Level Cache Load Misses Retired (リタイアした二次キャッシュのロードミス)
- Branches Retired (リタイアした分岐)
- Clockticks (Non-sleep Clockticks)
- Counting Clock Cycles
- Instructions Retired (リタイアされた命令)
- Loads Retired (リタイアしたロード)
- Micro-ops Retired

- Mispredicted Branches Retired (リタイアした分岐予測ミス)
- Non-halted Clockticks (停止されていない Clockticks)

【バスイベント】
- Bus Data Ready from the Processor (プロセッサからのバス・データ・レディ)

【ストールイベント】
- Machine Clear Count (マシン・クリア・カウント)

【コードの落とし穴を示すイベント】
- Split Loads Retired (リタイアした分割ロード)
- MOB Load Replays Retired (リタイアした MOB ロードのリプレイ) (Blocked Store-to-Load Forwards Retired)
- Streaming SIMD Extensions Input Assists (ストリーミング SIMD 拡張命令の入力支援)
- x87 Input Assists (x87 入力支援)
- x87 Output Assists (x87 出力支援)
- 64K Aliasing Conflicts (64K エイリアシング競合)

Pentium 4 セカンダリ・パフォーマンス・チューニング・イベント

Pentium 4 セカンダリ・パフォーマンス・チューニング・イベントは、プライマリ・パフォーマンス・チューニングのサンプリング解析からの結果を評価した後に使用する。
Pentium 4 セカンダリ・パフォーマンス・チューニング・イベントは、以下のイベントを含む。

【フロント・エンド・イベント】
- Trace Cache Deliver Mode (トレース・キャッシュの転送モード)
- Trace Cache Misses (トレース・キャッシュのミス)
- Trace Cache Build Mode

【メモリイベント】
- Loads Retired (リタイアしたロード)
- 1st-Level Cache Load Misses Retired (リタイアした一次キャッシュのロードミス)
- Split Stores Retired (リタイアした分割ストア)
- DTLB Page Walks (DTLB ページウォーク)

- ITLB Page Walks(ITLB ページウォーク)

【バスイベント】
- Reads from the Processor (プロセッサからの読み出し)
- Writes from the Processor (プロセッサからの書き込み)
- Write WC Partial (BSQ) (WC パーシャルへの書き込み)

【ストールイベント】
- Memory Order Machine Clear (メモリ・オーダー・マシン・クリア)
- Self-Modifying Code Clear (自己修正コードクリア)

Pentium 4 詳細イベント

このカテゴリのイベントは、大半のパフォーマンス・チューニングの使用範囲を超えていることから、高度なイベントと考えらる。経験豊かなパフォーマンス・エンジニアにより使用されるリファレンスとして含まれている。
従って、ここではこれ以上は言及しない。詳細は、VTune ヘルプを参照すること。

イベント比率

イベント比率は、複数のイベントの比率である。イベント比率を使用することで、複数の Activity の結果を理解し、比較することができる。

VTune にはあらかじめ、
- プライマリ比率
- セカンダリ比率
- 詳細比率

がイベント比率グループして設定されている。

プライマリ比率

【命令 Mix のイベント比率】
- Scalar Single-Precision Floating-Point Streaming SIMD Extensions Instructions (スカラ単精度の浮動小数点ストリーミング SIMD 拡張命令)
  (Scalar SPFP Streaming SIMD Extensions Instructions Retired (リタイアされたスカラ SPFP ストリーミング SIMD 拡張命令数) / Instructions Retired (リタイ

アされた命令数))*100

- Packed Double-Precision Floating-Point Streaming SIMD Extensions (パックドダブル精度の浮動小数点ストリーミング SIMD 拡張命令)
  (Packed DP FP Streaming SIMD Extension Instructions Retired (リタイアされたパックド DP FP ストリーミング SIMD 拡張命令数) / Instructions Retired (リタイアされた命令数))*100
- Packed Single-Precision Floating-Point Streaming SIMD Extensions (パックドダブル精度の浮動小数点ストリーミング SIMD 拡張命令)
  (Packed Single-Precision Floating-Point Streaming SIMD Extensions Instructions Retired (リタイアされた単精度の浮動小数点ストリーミング SIMD 拡張命令数) / Instructions Retired (リタイアされた命令数))*100
- Floating-point Computation Instructions (浮動小数点演算命令)
  (x87 Instructions Retired (リタイアされた x87 命令数) / Instructions Retired (リタイアされた命令数))*100
- Scalar Double-Precision Floating-Point Streaming SIMD Extensions Instructions (スカラダブル精度の浮動小数点ストリーミング SIMD 拡張命令)
  (Scalar DPFP Streaming SIMD Extensions Instructions Retired (リタイアされたスカラ DPFP ストリーミング SIMD 拡張命令数) / Instructions Retired (リタイアされた命令数))*100
- 64-bit MMX Instructions (64 ビット MMX(R) 命令)
  (64-bit MMX® Instructions Retired (リタイアされた 64 ビット MMX® 命令) / Instructions Retired (リタイアされた命令数))*100
- 128-bit MMX(R) Instructions per Instructions Retired (リタイアされた命令ごとの 128 ビット MMX(R) 命令数)
  (128-bit MMX® Instructions Retired (128 ビット MMX® 命令 のリタイアされた命令数) / Instructions Retired (リタイアされた命令数))*100
- x87 SSE and SSE2 Register and Memory Move Instructions (x87 SSE と SSE2 のレジスタおよびメモリ移動命令)
  x87 and SIMD Register and Memory Moves Retired (x87 と SIMD のリタイアされたレジスタおよびメモリ移動命令数) / Instructions Retired (リタイアされた命令数)

【鍵となる CPI 制限のイベント比率】

- Clockticks per Instructions Retired (CPI) (リタイアされた命令ごとの Clocktick の回数)
  Clockticks (Non-sleep) (スリープモードではない Clocktick の回数) /

Instructions Retired (リタイアされた命令数)

- Non-Halted Clockticks (停止されていない Clocktick の回数)/Instructions Retired (リタイアされた命令数) (Non-Halted CPI (停止されていない CPI))

  Non-Halted Clockticks (停止されていない Clocktick の回数)/ Instructions Retired (リタイアされた命令数)
- Cycles per Retired Micro-op (リタイアされたマイクロ・オペレーションごとのサイクル数)

  Clockticks (Non-sleep) (スリープモードではない Clocktick の回数) / Uops Retired (リタイアされた Uops 数)
- 2nd-Level Cache Load Misses per Instructions Retired (リタイアされた命令ごとの二次キャッシュのロードミスの回数)

  (2nd Cache Load Misses Retired (リタイアされた二次キャッシュのロードミスの回数) / Instructions Retired (リタイアされた命令数))
- 2nd Level Cache Load Hit Rate (二次キャッシュのロードヒットの比率)

  (100-((2nd Cache Load Misses Retired (リタイアされた二次キャッシュのロードミスの回数) / Loads Retired (リタイアされたロード数))*100))
- Branch Prediction Rate (分岐予測の比率)

  (100-((Mispredicted Branches Retired (リタイアされた分岐予測ミスの数) / Branches Retired (リタイアされた分岐数))*100))
- Branch Mispredict Performance Impact (パフォーマンスに与える分岐予測ミスの影響)

  ((Mispredicted Branches Retired (リタイアされた分岐予測ミスの数) * 20) / Clockticks (Clocktick の回数))*100
- Mispredicted Branches per Instructions Retired (リタイアされた命令ごとの分岐予測ミスの数)

  (Mispredicted Branches Retired (リタイアされた分岐予測ミスの数) / Instructions Retired (リタイアされた命令数))

【コードの落とし穴を示すイベント比率】

- 64K Aliasing Conflicts (64K エイリアシング競合)

  ((64k Aliasing Conflicts (64K エイリアシング競合の数)*12) / Clockticks (Clocktick の回数))*100
- x87 Input Assists Performance Impact (パフォーマンスに与える x87 入力アシストの影響)

  ((x87 Input Assists (x87 入力アシスト数)*650) / Clockticks (Clocktick の回数))*100

- x87 Output Assists Performance Impact (パフォーマンスに与える x87 出力アシストの影響)

  ((x87 Output Assists (x87 出力アシスト数)*650) / Clockticks (Clocktic の回数))*100
- Split Loads Performance Impact (パフォーマンスに与える分割されたロードの影響)

  ((Split Loads Retired (リタイアされた分割ロード数)*30) / Clockticks (Clocktick の回数) * 100
- Store Forward Performance Impact (パフォーマンスに与えるストア・フォワードの影響)

  ((MOB Loads Replays Retired (リタイアされた MOB ロードのリプレイ)*50) / Clockticks (Clocktick の回数))*100
- Streaming SIMD Extensions (SSE) Input Assists per Instructions Retired (リタイアされた命令ごとのストリーミング SIMD 拡張命令の入力アシスト数)

  (Streaming SIMD Extensions (SSE) Input Assists (ストリーミング SIMD 拡張命令の入力アシスト数)) / (Instructions Retired (リタイアされた命令数))
- Streaming SIMD Extensions (SSE) Input Assists Performance Impact (パフォーマンスに与えるストリーミング SIMD 拡張命令の入力アシストの影響)

  ((Streaming SIMD Extensions Input Assists (ストリーミング SIMD 拡張命令の入力アシスト数)*1300) / Clockticks (Clocktick の回数))*100
- Branch Misprediction per Branch Retired Instructions (リタイアされた分岐命令ごとの分岐予測ミスの数)

  Mispredicted Branches Retired (リタイアされた分岐予測ミスの数)/Branches Retired (リタイアされた分岐数)

セカンダリ比率

【ストールイベント比率】

- Memory Order Machine Clear Performance Impact (パフォーマンスに与えるメモリの配列によるマシンクリアの影響)

  ((Memory Order Machine Clear (メモリの配列によるマシンクリアの回数)*500) / Clockticks (Clocktick の回数))*100
- Self-Modifying Code Clear Performance Impact (パフォーマンスに与える自己修正コードによるクリアの影響)

  ((Self-Modifying Code Clear (自己修正コードによるクリア数)*1000) / Clockticks (Clocktick の回数))*100

【メモリイベント比率】

- 1st Level Cache Load Hit Rate (一次キャッシュのロードヒットの比率)
  (100-((1st-Level Cache Load Misses Retired (リタイアした一次キャッシュのロードミスの回数) / Loads Retired (リタイアしたロード数))*100))
- 1st Level Cache Load Miss Performance Impact (パフォーマンスに与える一次キャッシュのロードミスの影響)
  ((1st-Level Cache Load Misses Retired (リタイアされたキャッシュのロードミスの数)*10) / Clockticks (Clocktick の回数))*100
- Split Stores Performance Impact (パフォーマンスに与える分割されたストアの影響)
  ((Split Stores Retired (リタイアされた分割ストア数)*50) / Clockticks (Clocktick の回数)) * 100

【フロント・エンド・イベント比率】

- TC Delivery Rate (トレース・キャッシュの転送比率)
  (Trace Cache Deliver Mode (トレース・キャッシュの転送モード)*100) / Clockticks (Clocktick の回数)
- Trace Cache (TC) Miss Performance Impact (パフォーマンスに与えるトレース・キャッシュ・ミスの影響)
  ((Trace Cache Misses (トレース・キャッシュ・ミスの数)*20) / Clockticks (Clocktick の回数))*100

詳細比率

【バス比率】

- Percentage Prefetches (比率プリフェッチ)
  Non-prefetch Bus Accesses from the processor (プロセッサからのノンプリフェッチ・バス・アクセス数) / Bus Accesses from the Processor (プロセッサからのバスアクセス数)

Pause/Resume API

アプリケーションからサンプリングの停止、開始を制御することが可能である。以下の手順で機能を組み込む。

アプリケーションは、*vtuneapi. h* ヘッダファイルをソースコードにインクルードする。
コード内でサンプリングを一時停止したい箇所に、*VtPauseSampling()* の呼び出しを挿入する。
サンプリングを再開したい箇所に、*VtResumeSampling()* の呼び出しを挿入する。
vtuneapi.lib ファイルへのリンクを作成して、静的なこれらの関数にアクセスする。

VTune パフォーマンス・アナライザがアプリケーションのサンプリングを開始するタイミングを制御したい場合は、[Advanced Activity Configuration (Activity の詳細設定)] ダイアログ・ボックスから [Start with data collection paused (データ収集の再実行)] オプションを選択する。
[Start with data collection paused (データ収集の再実行)] オプションを選択して、VTPauseSampling() および VTResumeSampling() API を使用するアプリケーションで Activity を実行する場合、VTune パフォーマンス・アナライザはアプリケーションを起動するが、コード内の VTResumeSampling() API が実行されるまでサンプリング・データを収集しない。

なお、ツールバーの [Pause/Resume Activity (Activity の一時停止/再実行)] ボタンは、Pause/ Resume API と同じ機能を提供している。

カウンタモニタの使用

カウンタ・モニタ・データ・コレクタは、指定した期間に渡る、ハードウェアおよびソフトウェアを含むパフォーマンス カウンタをモニタし、システム・パフォーマンスを解析する。
カウンタ・モニタ用コンフィギュレーション・ウィザードを使用して Activity を生成すると、解析を行うアプリケーションの種別(データベース系、グラフィックス系、ネットワーク系、その他一般)を指定することができる。

Windows 2000 および Windows XP では、以下の OS カウンタが提供されている。

- System: Processor Queue Length
- System: Context Switches/sec
- Memory: Available Bytes
- Processor (_Total): Processor Time
- Processor (_Total): Privileged Time
- Redirector: Network Errors/sec

５-３-２-１５-２　スタティック・モジュール／アセンブリ解析

スタティック・モジュール解析は、実行可能なモジュールの解析とモジュール内の関数とクラス情報を表示できる。解析を行うと、VTune アナライザは、コードの各種パフォーマンス情報を表示する。

ホットスポットやスタティックな関数を［assembly（アセンブリ）］ビューに表示する場合、VTune™ パフォーマンス・アナライザは、ホットスポットや関数に関連付けられているスタティックな .exe、.obj、または .dll ファイルを解析し、逆アセンブルを実行する。

VTune パフォーマンス・アナライザは、コードの基本ブロックを解析し、コードとデータが既にキャッシュ内にあると仮定して、アセンブリ言語の命令を表示する。注釈として、パフォーマンスについての情報も表示される。

５-３-２-１４-２-１　スタティック・アセンブリ・ペナルティについて

「ペナルティ」では、特定の問題を指摘し、それがパフォーマンスにどのように影響するかを説明する。たとえば、ある問題が原因で増えたクロックサイクルの数を示す。
「警告」は、特定の状況においてパフォーマンスを低下させる可能性のある問題を示す。

解析はスタティックなため、ランタイム時の実際の影響は不明である。ペナルティおよび警告の実際の影響を確認するには、イベント・ベース・サンプリングを特定のイベントで実行する必要がある。

Pentium 4 に関連するペナルティ、または警告には以下のものがある。

- シリアル化 - PPro_Serialized（警告）
  PPro_Serialized のインジケータが付いている命令によって、シリアル化が起こる。シリアル化が起こると、前のすべての命令が実行、リタイアされ、その結果がメモリに書き込まれるまで、命令の実行が延期される。
  次のような命令によってシリアル化が起こる。
  ✓ロック・プリフィックス付きの読み込み - 変更 - 書き込み命令
  ✓メモリ・オペランド付きの XCHG

✓レジスタを制御またはデバッグする MOV 命令

- メモリストール – PPro_Mem_Stall(警告)

  PPro_Mem_Stall のインジケータが付いている命令は、前の命令が書き込みを行った後の、重複するメモリ空間を読み込む。次のいずれかが該当する場合、ストールが発生する。

  ✓読み込み命令と書き込み命令のターゲット・メモリ・アドレスが異なる場合。

  ✓読み込み命令と書き込み命令のターゲット・メモリ・アドレスが同じアドレスにアライメントされているが、読み込みデータが書き込みデータより大きい場合。

- レイテンシ

  レイテンシは、Pentium® 4 プロセッサで Pentium® III プロセッサより多くのサイクルがかかる命令を表す。

  この命令を、同じ操作を少ないサイクルで実行する他の命令のシーケンスに置き換えてみる事。

  次の表に、**レイテンシ** ペナルティが発生する可能性のある命令およびペナルティを最小限に抑えるためのアドバイスをリストする。

命令	レイテンシの説明	アドバイス
Inc および dec	inc および dec 命令は、フラグレジスタ内のビットのサブセットのみ変更します。この操作により、以前に行われたすべてのフラグレジスタの書き込みに依存性を作成します。これらの命令は、多くの命令が依存するロードのアドレスを変更するため、命令がクリティカル・パスにある場合特に問題になります。	inc および dec 命令を add または sub 命令で置き換えてください。add および sub 命令は、すべてのフラグを上書きします。 **注意** Pentium® II プロセッサの最適化は、命令ごとにバイトを追加するため効果はありません。
Shift	shift 命令のレイテンシは、以前のプロセッサより Pentium 4 プロセッサのほうが長いです。固定および変数 shift 命令のレイテンシは同じです。 add 命令シーケンスのレイテンシは、左 shift 命令のレイテンシより短く 3 かそれ以下です。	shift 命令がクリティカル・パスにある場合、shift 命令を最大 3 つの add 命令に置き換えてください。レイテンシがクリティカルではない場合、少ないループを使用する shift 命令を使用してください。

Rotate	rotate 命令のレイテンシは、以前のプロセッサより Pentium 4 プロセッサのほうが長いです。	rotate の即値またはレジスタ命令は、shift 命令より大きいです。rotate 一命令と shift 命令のレイテンシは同じです。 rotate のレジスタまたは即値命令は避けてください。可能であれば、rotate 一命令に置き換えてください。
Mul および imul	mul および imul 整数乗算命令は、浮動小数点ユニットで実行されるため、浮動小数点命令と並行に実行してはいけません。また、これらの命令を浮動小数点ユニット上で実行しているため、余計なレイテンシを被る場合があります。直前のサイクルで浮動小数点乗算命令（fmul）を実行した場合、fmul は 1 サイクル遅れます。乗算器は、1 サイクルおきにしか新しいオペランドのペアを受け入れることができません。	整数乗算を 2 つ以上の add および lea 命令を使用した小さい定数で置き換えてください。特にこれらの乗算が依存チェーンの一部の場合置き換えてください。

５-３-２-１５-３　コールグラフ・プロファイル

コール グラフ・データ・コレクタは、どの親関数によりどの子関数が呼び出されるかを表示することにより、アプリケーションのプログラム・フローについての情報を提供する。
以下の情報を取得することが出来る。

- アプリケーションのパフォーマンスの評価
- 潜在的なパフォーマンスのボトルネックの検索
- [Edge Time（エッジ時間：特定の呼び出し元の関数から呼び出された際に、呼び出し先関数の合計時間に加わる時間。関数に対するすべての着信エッジのエッジ時間は、この関数の合計時間と同じである。)] が最大であるクリティカル・パスを探索

リリースビルドでシンボルファイルを生成すると、オリジナルのソースファイルとリンクしたプロファイルデータの取得が可能である。
コールグラフ・データ・コレクタは、データ収集ルーチンを実行時に組み込むため、プロ

グラムの機能に変更は無いが、実装速度は低下する。従って、関数単位の処理時間計測等を行うには有効だが、実時間処理を必要とするアプリケーションを評価する場合は注意が必要である。
なお、VTune では、関数単位の時間計測等は可能であるが、ソースステップ単位の計測は出来ないようである。（DevPartner は可能）

５-３-２-１６　VTune による Secure Decoder の解析

VTune 評価版を使用し、Visual Studio 2003 でビルドを行った Secure Decoder のコールグラフ解析、およびサンプリング解析を行った。

５-３-２-１６-１　基本データ

- 計測用ストリーム　bs-j.ts（GOP サイズ 12, I/P/B=1/3/8）
- 処理ルート　1
- 計測時間　コールグラフ　約 120 秒
  サンプリング　60 秒
- クオリティコントロール　オフ
- IDE　Visual Studio 2002、および Visual Studio 2003
  以下の解析結果は Visual Studio 2003 を使用する。

５-３-２-１６-２　サンプリング解析結果

サンプリングは、プライマリサンプリングイベントとセカンダリサンプリングイベント、および対応する比率を取得した。

５-３-２-１４-２-１　モジュール単位の解析結果

Secure Decoder を構成するモジュールが使用する Clockticks 率をワースト順に示す。

module	Clockticks %
fjMpeg2v.ax	91.28877785
mpgaudio.ax	0.832812825
NEMPEGSP.ax	0.745367479
TAOSourceFilter.ax	0.116593796
ProgramStreamProject.dll	0
TAOSecureControllerProject.dll	0

表をみて分かるように、全体の 91%の Clockticks を fjMpeg2v.ax が消費している。従って、最適化を行う場合、そのまま fjMpeg2v.ax の最適化と考えるべきである。
処理ルート 1 での試験のため、TAORecOutController.dll 、TAOHDDInStreamController.dll は表にはレポートされていない。しかし、処理の性格か

ら使用する Clockticks は TAOSourceFilter.ax に若干負荷が増えた程度と想定して良い。

クラス単位の解析結果

fjMpeg2v.ax を構成するクラスが使用する Clockticks 率をワースト順に示す。

class	Clockticks %
CVDecoder Class	93.83296082
main	5.62879168
CMpegVideoCodec	0.346667883
CPesParser	0.132281166
CBaseInputPin	0.027368517
CTransformInputPin	0.027368517

表をみて分かるように、全体の 93%の Clockticks を CVDecoder クラスが消費している。従って、最適化を行う場合、そのまま fjMpeg2v.ax の最適化と考えるべきである。

メソッド単位の解析結果

CVDecoder クラスを構成するメソッドが使用する Clockticks 率をワースト順に示す。

class	Clockticks %
CVDecoder:: Format_YUY2P3	34.95415773
CVDecoder:: MotionCompensationMpeg2P3	28.85097842
CVDecoder:: MotionVector1Direct16P3	7.421429549
CVDecoder:: MotionVector1Direct8P3	5.788441363
ClearBlock	5.62423026
CVDecoder:: Decode_MPEG2_Inter_Block_ALL	4.921771655
CVDecoder:: DecodeMPEG2Macroblock	3.731241162
CVDecoder:: MotionVector2Direct16P3	1.546321215
CVDecoder:: FramePredictions	1.445969986
CVDecoder:: Slice	1.400355791
CVDecoder:: Decode_MPEG2_Intra_Block_ALL	1.386671532
CVDecoder:: MotionVector2Direct8P3	1.026319391
CVDecoder:: motion_vector2	0.930529581

表から、Format_YUY2P3 と MotionCompensationMpeg2P3 で全体の 60%を消費していることが分かる。
メソッド単位にサンプリング比率を確認すると、Format_YUY2P3 や MotionCompensationMpeg2P3 では、64K Aliasing Conflicts Performance Impact が高い数値を示している。

また、MotionCompensationMpeg2P3 では Store Forward Performance Impact も高い数値を示している。
これらのことから、メモリ配置等の最適化が必要なことが分かる。

その他のメソッドでも、個別の指標では高い数値を示しているものもあるが、Clockticks 比率が引くため、その部分のみの最適化を行っても、全体に対する影響は小さい。
Format_YUY2P3 と MotionCompensationMpeg2P3 を中心に、アルゴリズムからの見直しとメモリ配置の最適化を並行して行うべきと考える。

５-３-２-１６-３　コールグラフ解析結果

コールグラム解析では、ビデオデコーダーフィルタ（fjMpeg2v.ax）を中心に解析を行う。

計測時間は、スットプウォッチにより 120 秒であるが、実行開始時に計測用コードの組み込み等を行うため、Secure Decoder 処理開始の明確なタイミングは不明である。ここでは、画面に映像の表示を開始した時点を起点としている。

デコード総フレーム数　　869
- 内 I ピクチャ　　73
- 内 P ピクチャ　　217
- 内 B ピクチャ　　579

コールグラフのレポートでは、I/P/B ピクチャの総処理時間（Edge Time）は、それぞれ
- Slice_I　4,043,192　μ 秒／73 フレーム
- Slice_P　11,712,733　μ 秒／217 フレーム
- Slice_B　28,178,424　μ 秒／549 フレーム

である。
フレーム単位では、
- Slice_I　55,386　μ 秒／フレーム
- Slice_P　53,976　μ 秒／フレーム
- Slice_B　51,327　μ 秒／フレーム

である。
画像フォーマット YCbCr を YUY2 に変換する処理は、28,105　μ 秒／フレームである。

これを GOP 単位に変換すると、
55,386 + 53,976 * 3 + 51,327 * 8 + 28,105 * 12 = 965,200　μ 秒

である。

サンプリング解析では、ビデオデコーダは全体の 91%の Clockticks を消費している。残りの Clockticks を全て Secure Decoder が消費出来ると仮定すると、
　965,200 * 100 / 91 = 1,060,659　μ秒
が実際には必要である。

毎秒 30 フレームを表示する場合、30 / 12 = 2.5 GOP/秒の処理能力が必要である。
従って、bs-j.ts をフルモーションで再生する場合、
　1,060,659 * 2.5 = 2,651,648　μ秒
の時間が必要と考えられる。

単純に考えると、本テストで使用した PC の CPU は Pentium4 1.7GHz である。同じ条件で CPU のみを早くすることを考えると、
　1.7 * 2.651 = 4.5 GHz
のクロックを実現する必要がある。

逆に、現在の Pentium4 1.7GHz では、
　1,000,000 / 1,060,659 * 12 = 11.3 フレーム/秒
の処理能力を有することになる。
コールグラフ解析結果から得られた処理能力は、現実の実測データとよく合致している。

５-３-２-１７　総括

VTune は本プロジェクトでは必須。
Thread Checker は使い方を研究する必要がある。現時点、本プロジェクトへの適用は難しいと判断する。

C、MMX、SSE2 の比較を行った結果、MMX、SSE2 の効果は大きい。また、MMX に対し SSE2 は 20%程度の高速化を達成している。
しかし、アセンブラで記述する場合、コンパイラによる最適化の対象外となり、適用するアルゴリズムやプログラマの技量によって、処理能力は大きく左右されるため、注意が必要である。

今回は、Microsoft の Visual Studio.NET 2003 と Intel C++ Compiler 7.1 の比較を行った。
その結果、Active Template Library (ATL)等で Intel C++7.1 でエラーが発生し、Secure Decoder のコンパイルが出来なかった。
しかし、コンパイラの性能は Intel C++7.1 の方が優秀である。
C で記述する部分の効率をみると、VC++2003 より Intel C++7.1 の方が実測で 20%程度高速である。
また、本文では記載していないが、マルチスレッド化のオプションを有効にすると、依存性がない異なるループ等は、その部分がマルチスレッド化されていた。
その分、ロードモジュールのサイズは大きくなる。実測では、80KB が 120KB となっていた。

また、Intel C++7.1 ではプロファイラフィードバック機能により、非常に効率の良いコードを生成することが可能である。
実測では、浮動小数点演算の場合、通常のリリースビルドより約 50%の高速化を達成している。しかし、整数演算ではそれほどの効果は得られなかった。

今回は Hyper-Threading Technology の効果は確認できなかった。
残念である。

データブロックの複写等によるキャッシュ汚染を防止するためには、Intel C++7.1 コンパイラを使用する場合、CopyMemory、memcpy 等の WIN32 API を使用すべきである。しかし、VC++2003 では必ずしも最適化されるかどうか分からないため、アセンブリングされた結果を確認する必要がある。
一方、再利用を考えた局所的なデータ複写に CopyMemory、memcpy 等は使用すべき

ではない。キャッシュサイズ等を考慮し、コーディングする必要がありそうである。

Secure DecoderをVC++2003でビルドした場合について解析を行った。その結果、試験で使用したPC（Pentium4 1.7GHz）ではか、毎秒17フレーム程度の処理能力があることが分った。また、単純にCPUのクロックだけの変更では、2.94 GHz以上のCPU クロックが必要と見積もれた。
実際にPentium4 3.0Ghzの実験機で毎秒30フレーム再生を達成できる事を確認した。

５-３-３　ビデオデコーダ高速化方針

セキュアソフトの性能計測の結果、ビデオデコーダが最も処理負荷が高い事が判明した。本章ではビデオデコードフィルタにおいて負荷の高いメソッドを中心に、以下の項目について高速化検討を記載する。そして、この検討案をセキュアソフトに適用する。

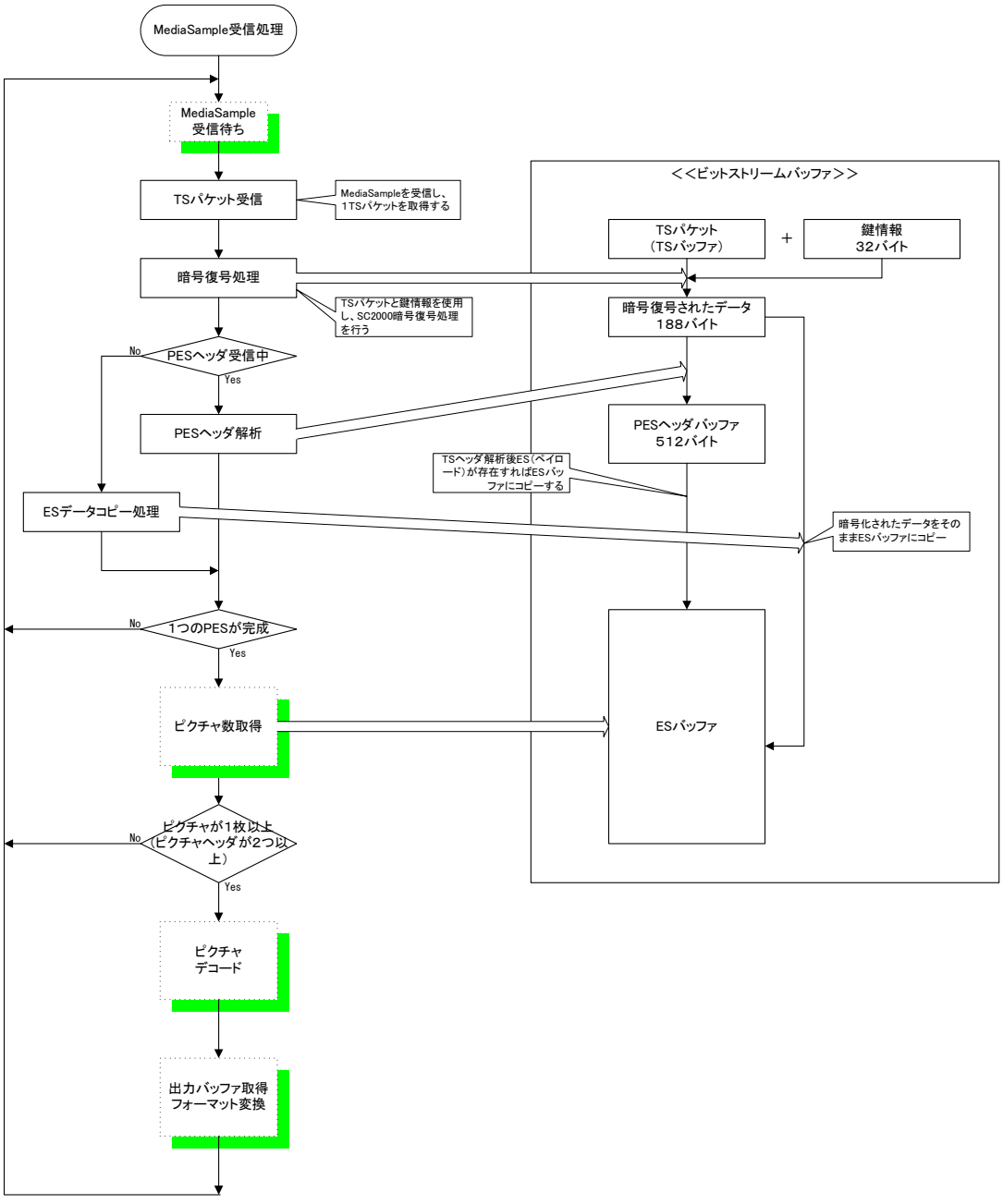
- インラインアセンブラ(MMX/SSE/MMX2/SSE2)使用による高速化
- メモリ転送における高速化
  - ・各バッファの構造変更よるキャッシュ汚染防止。
  - ・明示的なプリフェッチの使用
  - ・可変長テーブルの最小化
- デコード演算アルゴリズム修正
- Hyper-Threading Technology 利用による高速化
- 非効率な C 言語記述の最適化

５-３-３-１　高速化検討

５-３-３-１-１　MediaSample 受信処理

TAO Splitter Filter のビデオ用アウトプットキュースレッドから起動される。MediaSample に付加された TS パケットおよび鍵情報をもとに暗号復号処理を行い、ES データを作成する。

現状の MediaSample 受信処理



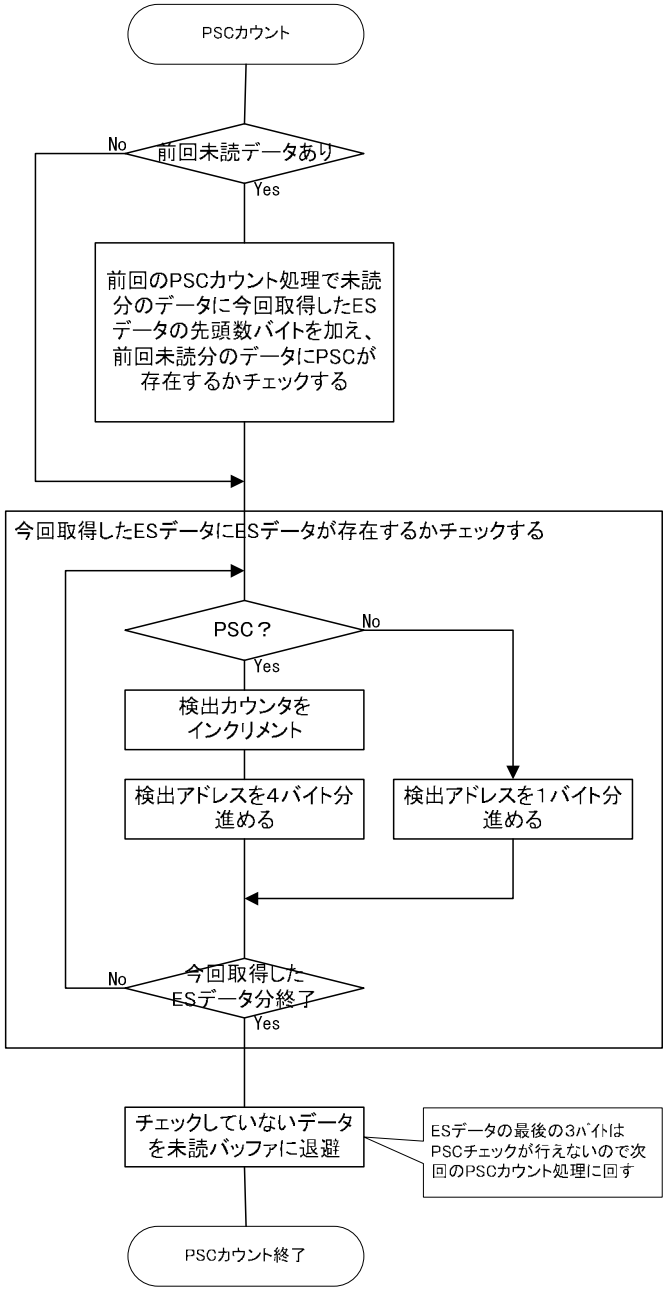
この MediaSample 受信処理で高速化対象となるメソッドは存在しないが、メモリ転送高速化の観点から以下の改善方法が考えられる。

- 暗号化復号されたデータを ES バッファに転送する際にキャッシュを使用しないことによるキャッシュ汚染の削減。

５-３-３-１-２　ピクチャ開始コードカウント処理

１ピクチャ以上データを受信したかどうか判断するため、1 PES パケット受信毎にピクチャ開始コード（以下 PSC）の個数を調べる。

現状のピクチャ開始コードカウント処理



全ビットストリームから PSC を検出するため、負荷が高い処理である。

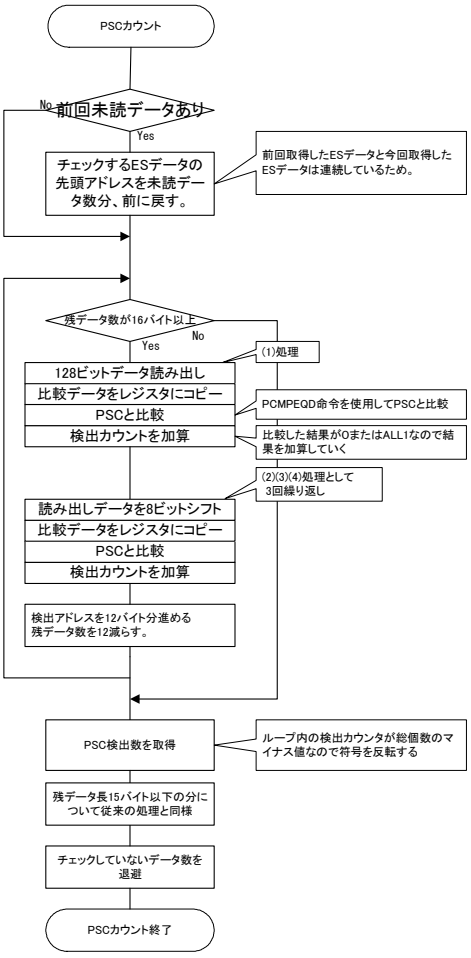
現状のピクチャ開始コードカウント処理から以下の高速化案が考えられる。

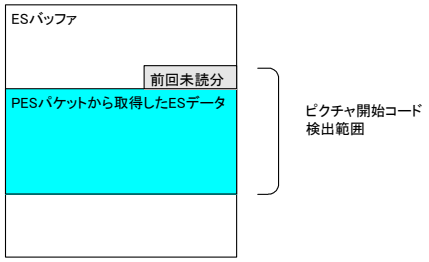
- PSC 検出が行われないデータは未読バッファに退避されるが、ES バッファ

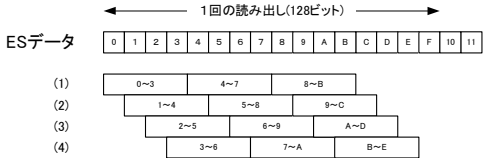
に格納されるデータの連続性は保たれているため、未読バッファに退避せず、未読データバイト数のみ記憶する。
また、上記処理により、未読バッファにおける PSC 検出を行わない。

・ MMX 機能を使用し、1 度に複数の PSC 検出を行うようにする。

ピクチャ開始コードの検出方法の改善案







メモリからの読み出しはリトルエンディアンになるので、
右シフト演算を使用します。

(2)(3)(4)をビットシフト

(1)	0~3	4~7	8~B	対象外	XMM1
(2)	1~4	5~8	9~C	対象外	XMM1の8ビットシフト
(3)	2~5	6~9	A~D	対象外	XMM1の16ビットシフト
(4)	3~6	7~A	B~E	対象外	XMM1の24ビットシフト

これらの値をPSCと比較する

	00_00_01_00	00_00_01_00	00_00_01_00	対象外

PCMPEQD命令使用しPSCと比較
PSCと同じ場合：all 1,不一致 all 0。

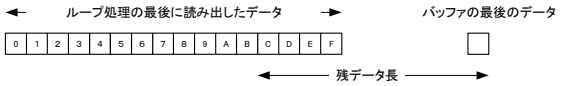
結果を加算していく

(1)	0	0	0
(2)	0	FFFFFFFF	0
(3)	0	0	0
(4)	0	0	0

0	FFFFFFFF	0

ESデータ指定領域の最後の部分の処理について

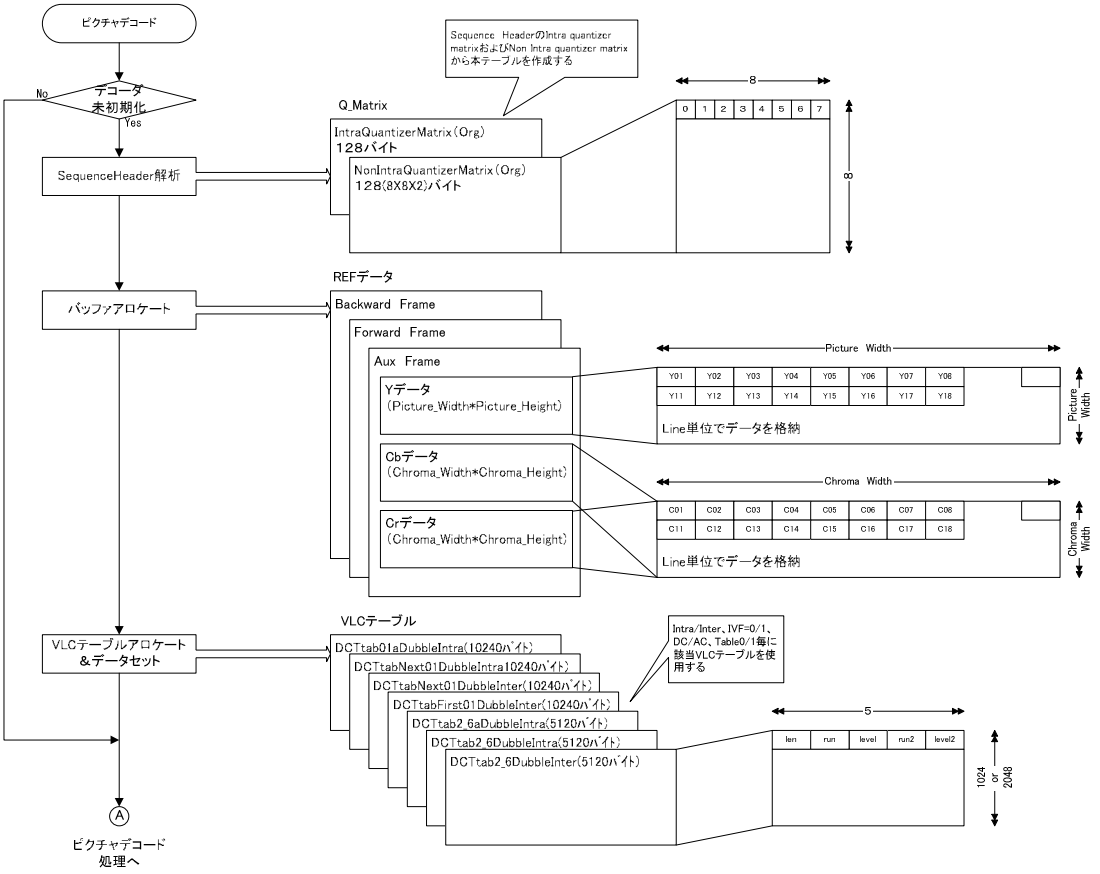
ピクチャカウント取得のループは残データ長が16バイト以上の場合である。残データ長が15バイト以下の場合は従来通りの処理を行うことにする。



５-３-３-１-３　ピクチャデコード（初期化）

ピクチャデコードの初期化処理は受信したシーケンスヘッダをもとに各バッファの獲得、および内部変数の初期化を行う。

現状のピクチャデコード初期化処理



このピクチャデコード初期化処理で高速化対象となるメソッドは存在しないが、メモリ転送高速化の観点から以下の改善方法が考えられる。

- REF データ（Backward Frame/Forward Frame/Aux Frame）のフォーマットを変更することによるキャッシュ汚染を削減する。（ただし、フォーマット変更はデコード処理全体に影響があるため、ソース変更量多）
- 現状の VLC テーブル構成を再検討し、テーブルサイズを縮小することによるキャッシュ汚染を削減する。

現状の REF データフォーマット

現状の REF データは YCbCr 毎に管理されているため、1つのマクロブロックをデコードするために必要な REF データが離れた場所に格納されている。

例えば Y111 と Y121 では画面幅ﾊﾞｲﾄ分離れている。

また、Y111 と B111 では画面幅×画面高さﾊﾞｲﾄ分離れている。

Y

Y111	Y112	Y113	Y114	Y115	Y116	Y117	Y118	Y211	Y212	Y213	Y214	Y215	Y216	Y217	Y218
Y121	Y122	Y123	Y124	Y125	Y126	Y127	Y128	Y221	Y222	Y223	Y224	Y225	Y226	Y227	Y228
Y131	Y132	Y133	Y134	Y135	Y136	Y137	Y138	Y231	Y232	Y233	Y234	Y235	Y236	Y237	Y238
Y141	Y142	Y143	Y144	Y145	Y146	Y147	Y148	Y241	Y242	Y243	Y244	Y245	Y246	Y247	Y248
Y151	Y152	Y153	Y154	Y155	Y156	Y157	Y158	Y251	Y252	Y253	Y254	Y255	Y256	Y257	Y258
Y161	Y162	Y163	Y164	Y165	Y166	Y167	Y168	Y261	Y262	Y263	Y264	Y265	Y266	Y267	Y268
Y171	Y172	Y173	Y174	Y175	Y176	Y177	Y178	Y271	Y272	Y273	Y274	Y275	Y276	Y277	Y278
Y181	Y182	Y183	Y184	Y185	Y186	Y187	Y188	Y281	Y282	Y283	Y284	Y285	Y286	Y287	Y288
Y311	Y312	Y313	Y314	Y315	Y316	Y317	Y318	Y411	Y412	Y413	Y414	Y415	Y416	Y417	Y418
Y321	Y322	Y323	Y324	Y325	Y326	Y327	Y328	Y421	Y422	Y423	Y424	Y425	Y426	Y427	Y428
Y331	Y332	Y333	Y334	Y335	Y336	Y337	Y338	Y431	Y432	Y433	Y434	Y435	Y436	Y437	Y438
Y341	Y342	Y343	Y344	Y345	Y346	Y347	Y348	Y441	Y442	Y443	Y444	Y445	Y446	Y447	Y448
Y351	Y352	Y353	Y354	Y355	Y356	Y357	Y358	Y451	Y452	Y453	Y454	Y455	Y456	Y457	Y458
Y361	Y362	Y363	Y364	Y365	Y366	Y367	Y368	Y461	Y462	Y463	Y464	Y465	Y466	Y467	Y468
Y371	Y372	Y373	Y374	Y375	Y376	Y377	Y378	Y471	Y472	Y473	Y474	Y475	Y476	Y477	Y478
Y381	Y382	Y383	Y384	Y385	Y386	Y387	Y388	Y481	Y482	Y483	Y484	Y485	Y486	Y487	Y488

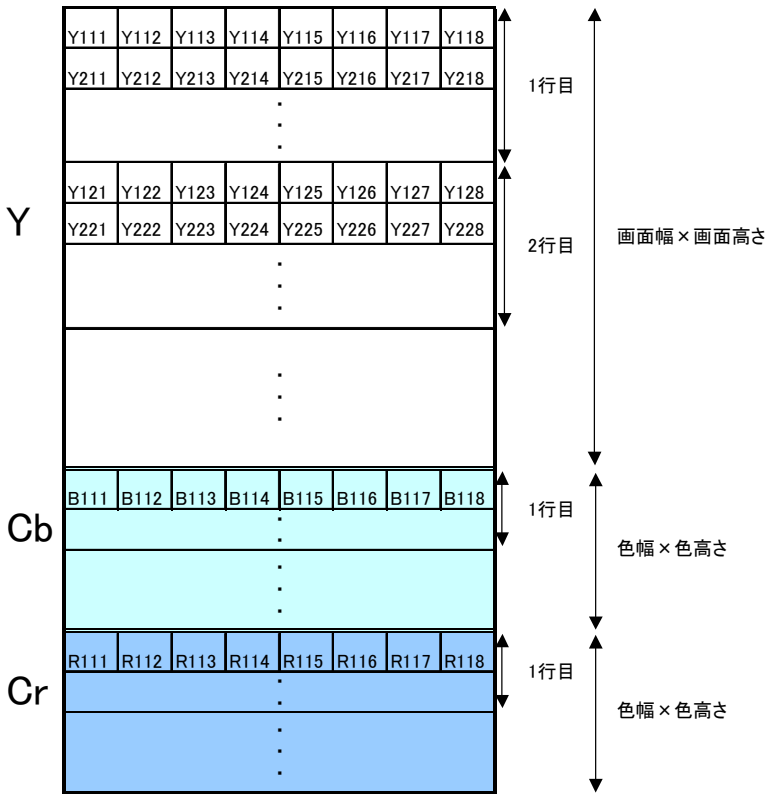
Cb

B111	B112	B113	B114	B115	B116	B117	B118
B121	B122	B123	B124	B125	B126	B127	B128
B131	B132	B133	B134	B135	B136	B137	B138
B141	B142	B143	B144	B145	B146	B147	B148
B151	B152	B153	B154	B155	B156	B157	B158
B161	B162	B163	B164	B165	B166	B167	B168
B171	B172	B173	B174	B175	B176	B177	B178
B181	B182	B183	B184	B185	B186	B187	B188

Cr

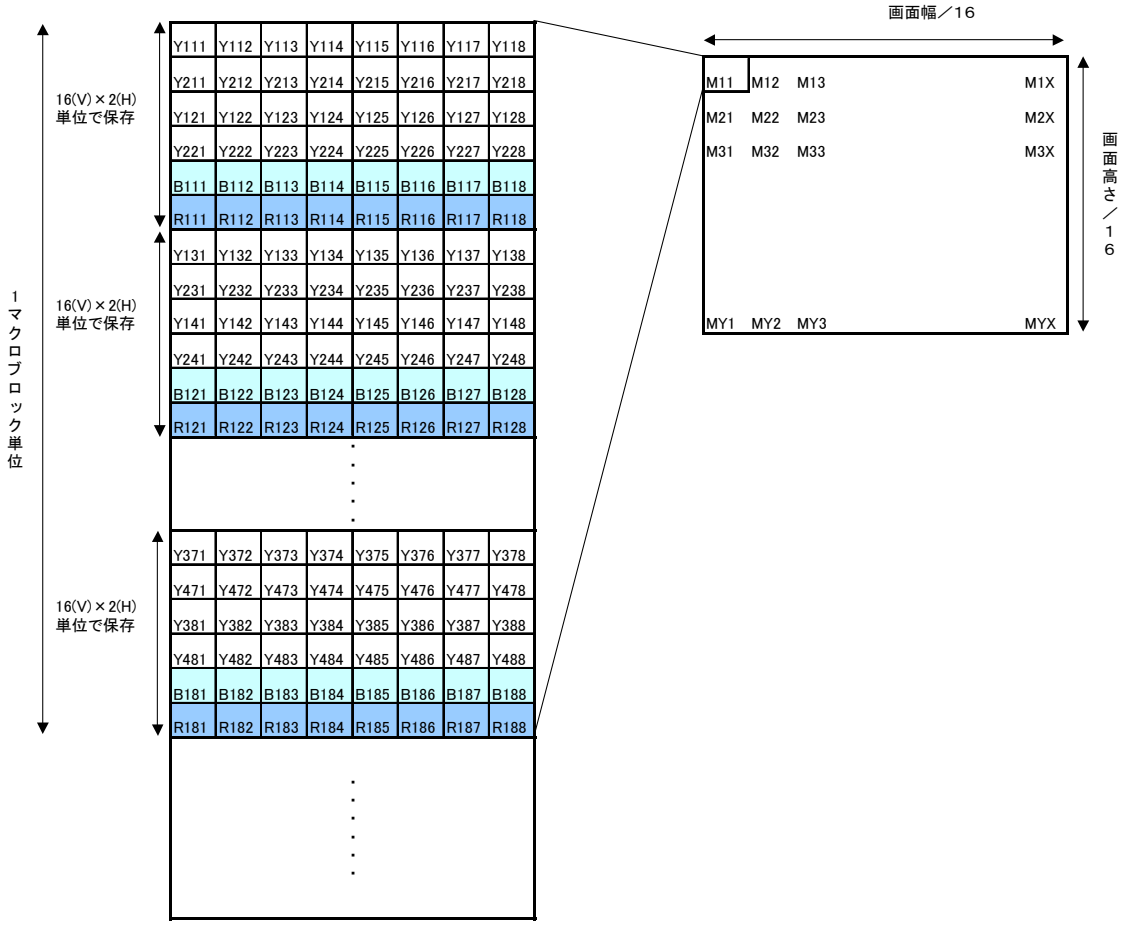
R111	R112	R113	R114	R115	R116	R117	R118
R121	R122	R123	R124	R125	R126	R127	R128
R131	R132	R133	R134	R135	R136	R137	R138
R141	R142	R143	R144	R145	R146	R147	R148
R151	R152	R153	R154	R155	R156	R157	R158
R161	R162	R163	R164	R165	R166	R167	R168
R171	R172	R173	R174	R175	R176	R177	R178
R181	R182	R183	R184	R185	R186	R187	R188

1つのマクロブロックを VLD 処理した結果が上記の場合、REF データのフォーマットは下図の構成となる。



REF データフォーマット変更案

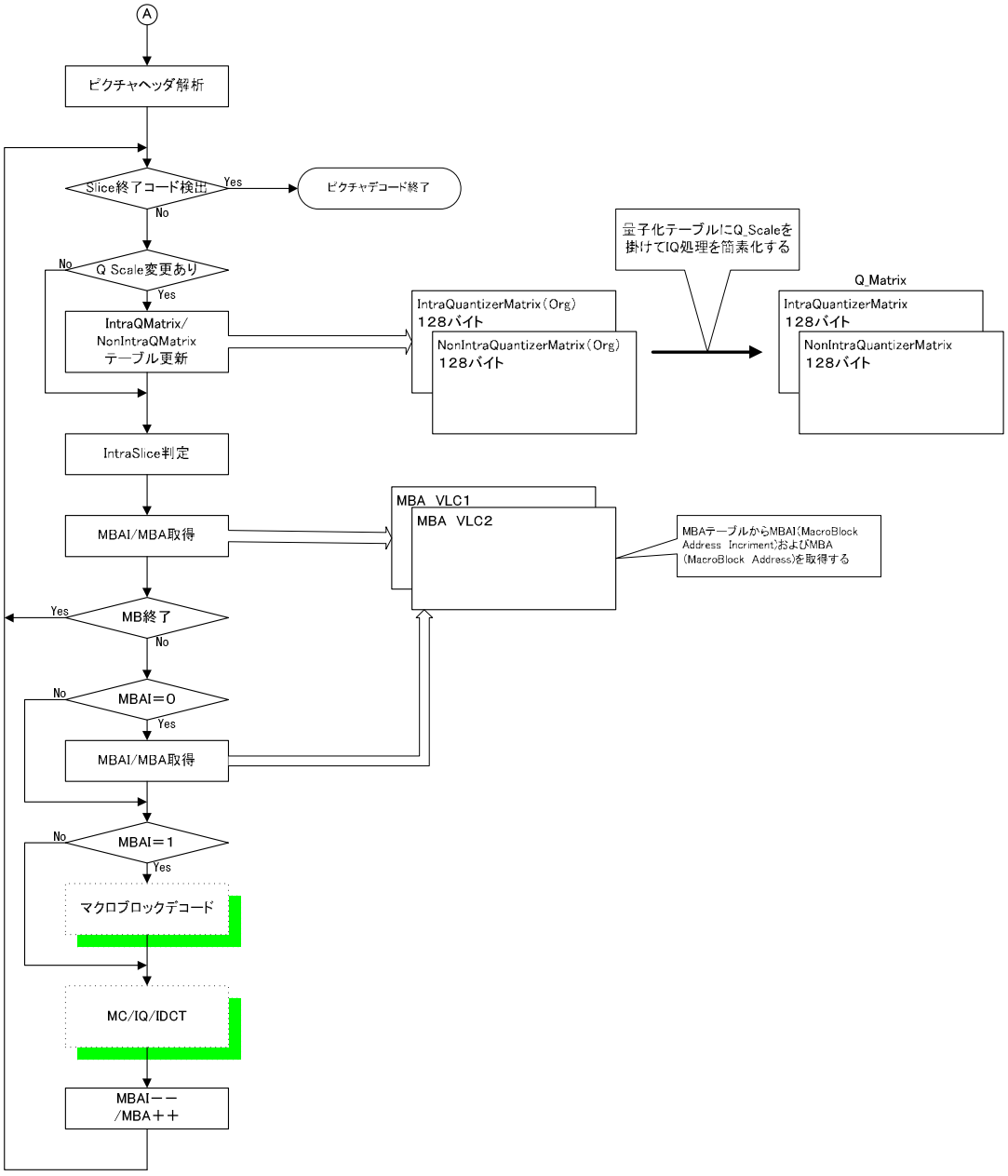
IDCT 処理で MMX 機能を使用して出力されるデータは8バイト単位である。このため8バイト単位のデータで取り扱いたい。また、C データは偶奇数ラインに1つなので2ライン単位で取り扱いたい。
よって、最小管理単位を16（Virtical）×2（Horizontal）とし、このブロック8個で1マクロブロックを管理することを考えた。



５-３-３-１-４　ピクチャデコード

ピクチャ層・スライス層のデコード・解析を行う。

現状のピクチャデコード



現状のピクチャデコード処理から以下の高速化案が考えられる。

- Q_Matrix（量子化マトリックステーブル）の作成は Q_Scale(量子化スケール)が変更される毎に行われている。この処理を削減するため Q_Matrix を多重化し、一度作成された Q_Matrix を作成しないようにする。
- MBA テーブル構成を再検討し、テーブルサイズを縮小することによりキャッシュ汚染を削減する。

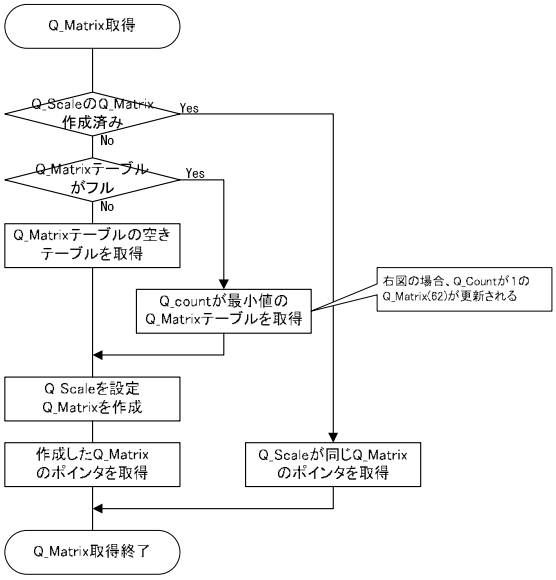
Q_Matrix 多重化案

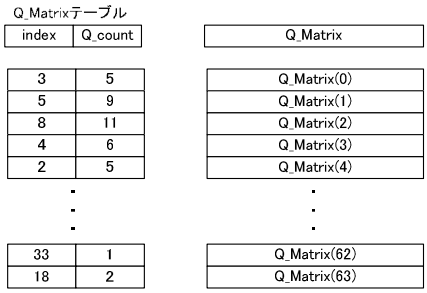
Q_Matrix とこれを管理する Q_Matrix テーブルを作成する。(以下はテーブル数64の場合)

```
// Q マトリクス管理テーブル。テーブル要素番号＝Q スケール値。
struct    q_mtrx_mng {
          int index;        //Q マトリクスの要素番号を格納
          int Q_Count;    //使用回数
}Q_Mtrx_Mng[128];
//Q マトリクス本体
struct Q_Mtrx_t    {
          WORD  Q_Matrix[64];     //Intra Quantizer Matrix
          WORD  NQ_Matrix[64];  //Non Intra Quantizer Matrix
} Q_Mtrx[64];
```

Q_Mtrx バッファ使用量:(4+4)*128 + (2*64*2)*64 =17408Byte

Q_Matrix テーブル使用方法

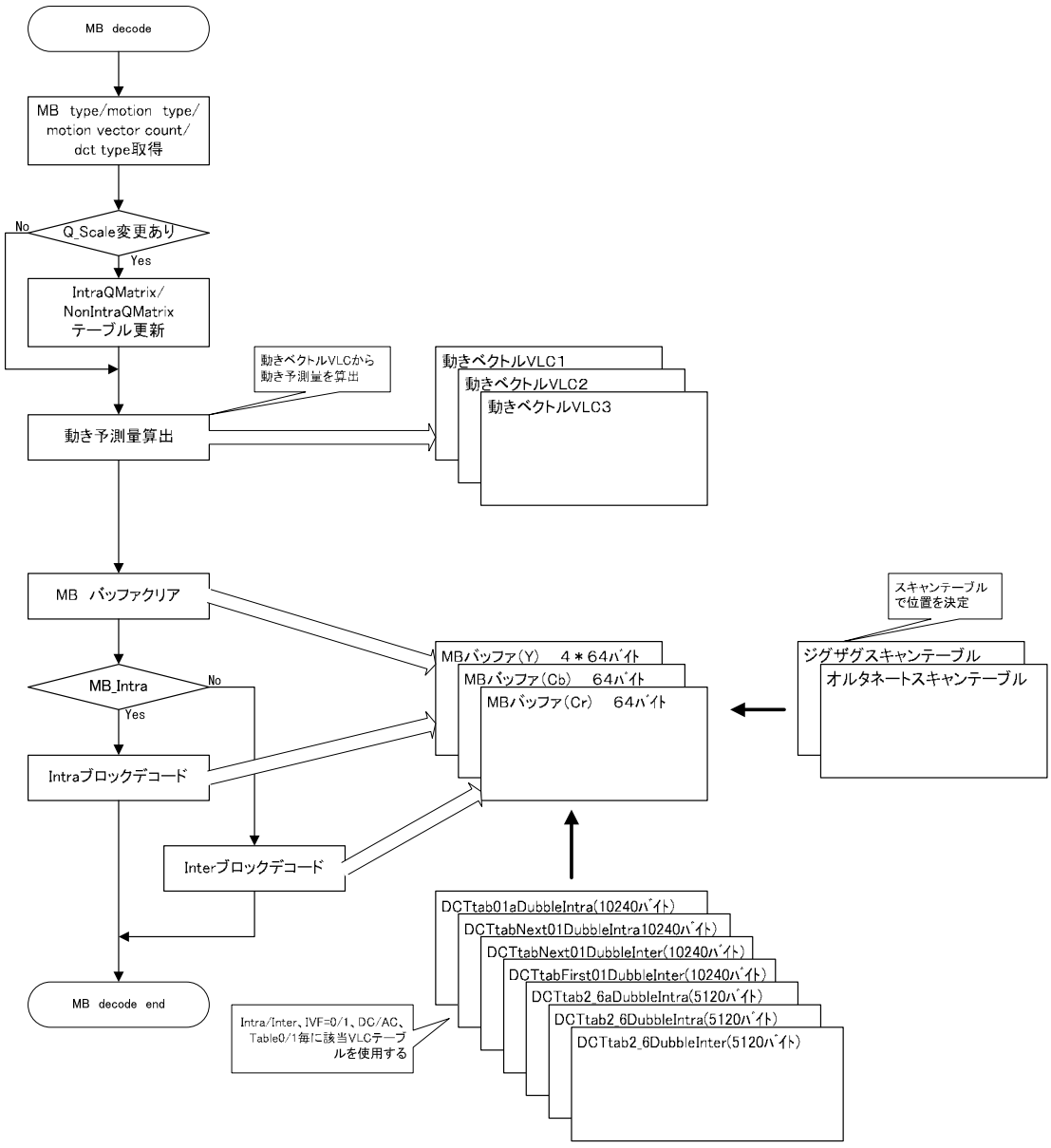




- Q スケールが同じ Q_Matrix は作成しない。
- Q_Matrix テーブルがフルの場合 Q_Matrix 使用回数が少ないテーブルから削除する。

５-３-３-１-５　マクロブロックデコード

マクロブロック層・ブロック層のデコード・解析を行う。
現状のマクロブロックデコード

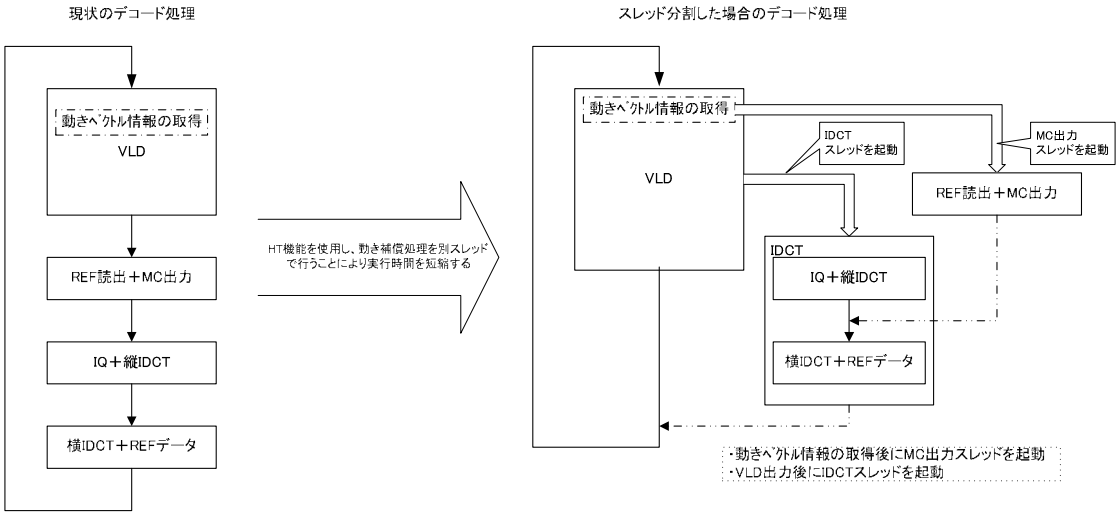


現状のマクロブロックデコード処理から以下の高速化案が考えられる。

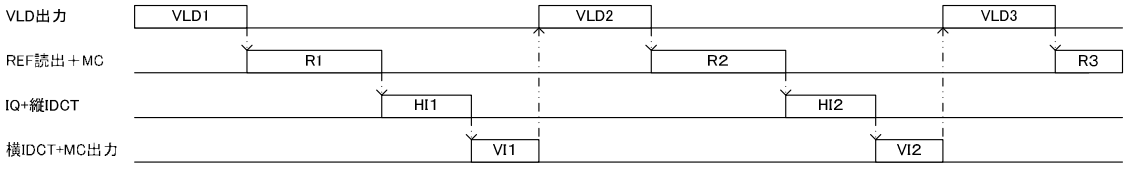
- Q_Matrix の作成を Q_Scale が変更される毎に変更されている。この処理を削減するため Q_Matrix を多重化し、一度作成された Q_Matrix を作成しないようにする。（前項と同様）
- MC 処理・IDCT 処理を別スレッドで行うことにより実行時間を短縮する。
- Intra/Inter ブロックデコード処理のインラインアセンブラ化により処理の効率化を行う。

スレッド化した場合を使用したマクロブロックデコード処理

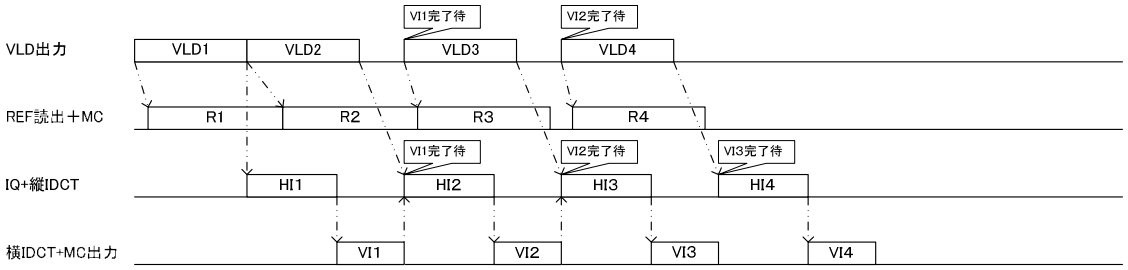
MC 出力処理および IDCT 処理をスレッド化することにより、VLD 処理と MC 出力処理・IDCT 処理の並列処理が可能である。ただし、このスレッド化に伴い VLD 出力用のバッファの多重化が必要となる。



現状の処理タイミングチャート



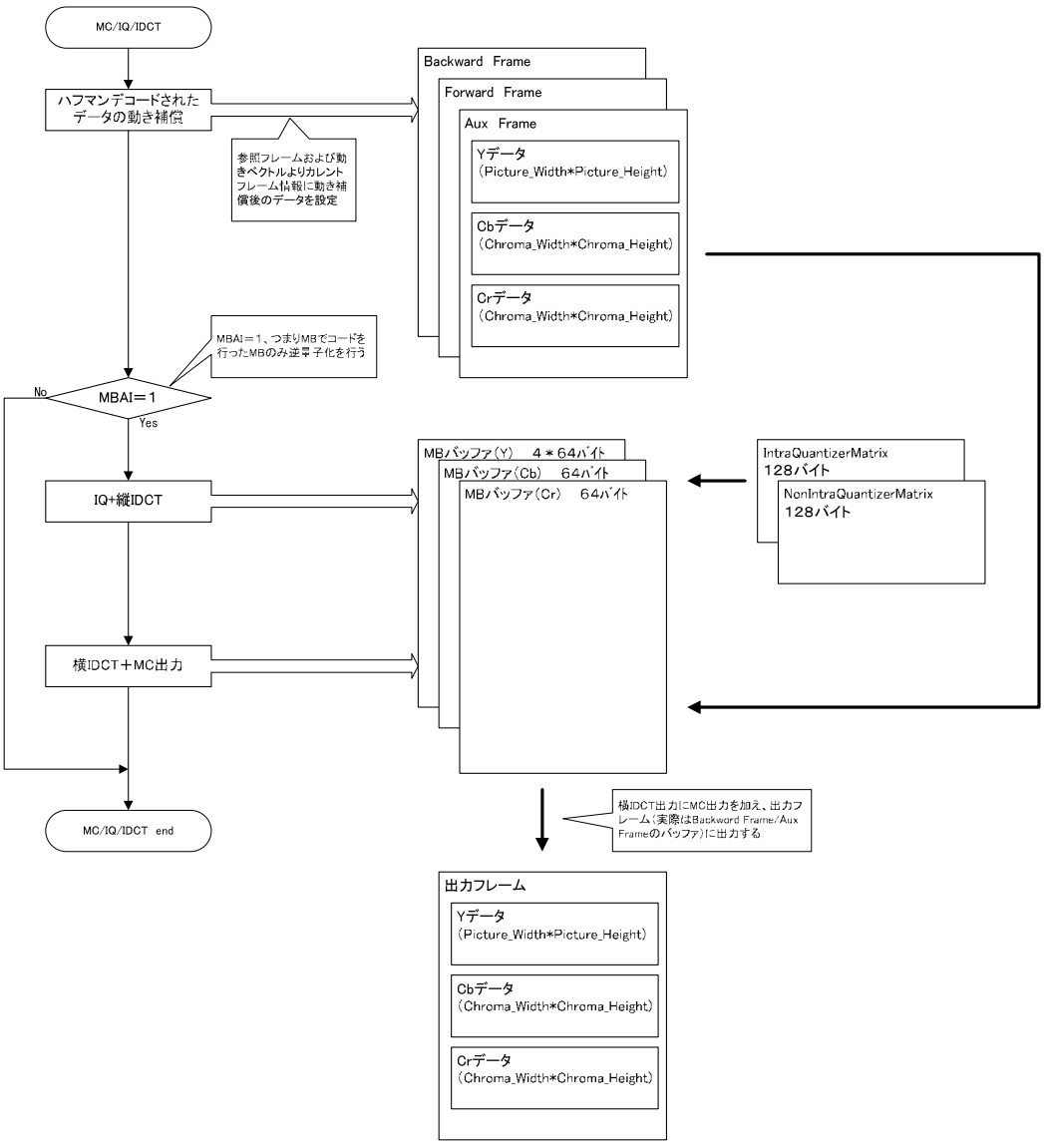
MC 出力処理・IDCT 処理をそれぞれスレッド化した場合の処理タイミングチャート



５-３-３-１-６　MC・IQ・IDCT

マクロブロック単位で MC・IQ・IDCT を行う。

現状の MC・IQ・IDCT 処理



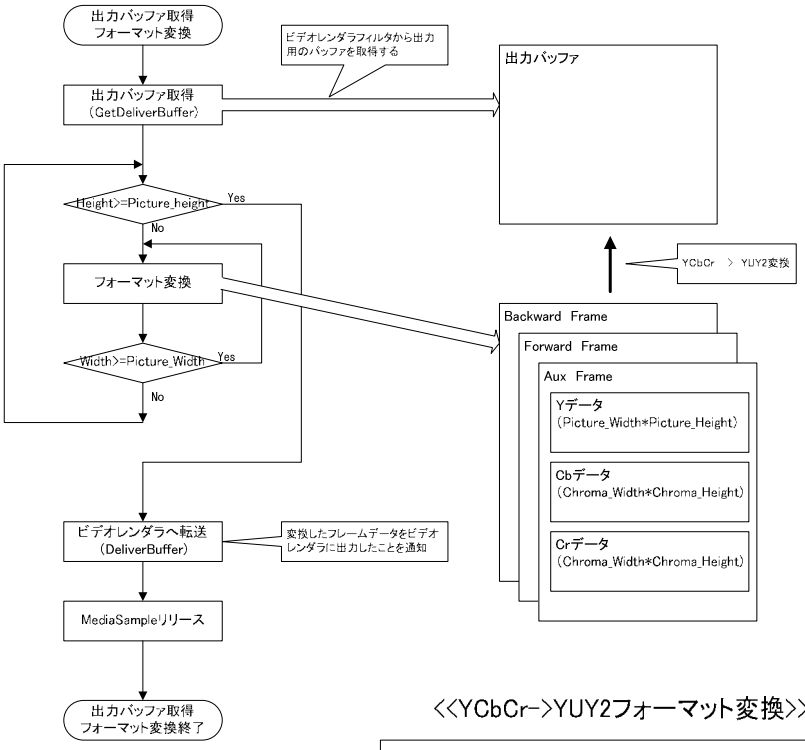
現状の動き補償・逆量子化処理から以下の高速化案が考えられる。

- MC 処理・IDCT 処理を別スレッドで行うことにより実行時間を短縮する。（前項と同様）
- IDCT アルゴリズムを高速化することにより実行時間を短縮する。

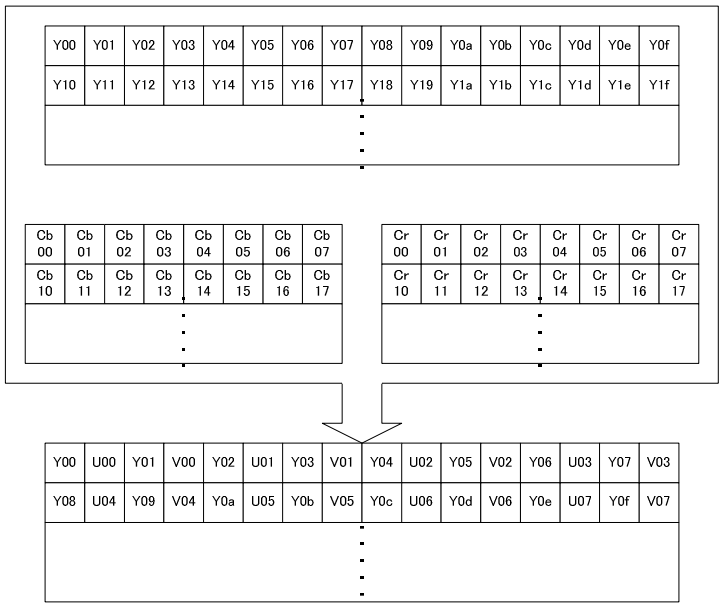
５-３-３-１-７　フォーマット変換

デコードされたフレームデータをビデオレンダラフィルタに出力するために、ビデオレンダラフィルタの許容するフォーマットに変換し、出力する。

現状のフォーマット変換処理



<<YCbCr->YUY2フォーマット変換>>

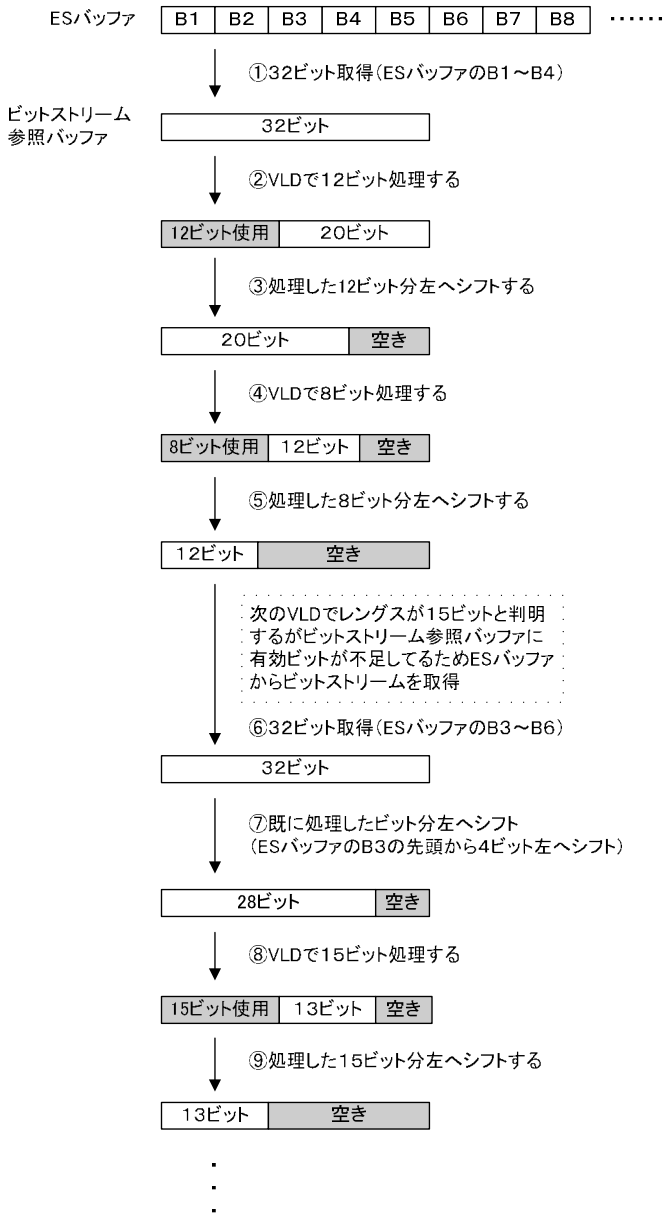


現状のフォーマット変更処理は高速化対象であるがインラインアセンブラ（MMX）による最適化がなされており、高速化の効果が小さいと考えられる。

５-３-３-１-８　ビットストリームの参照（VLD）

ES バッファからビットストリームを取得する処理である。

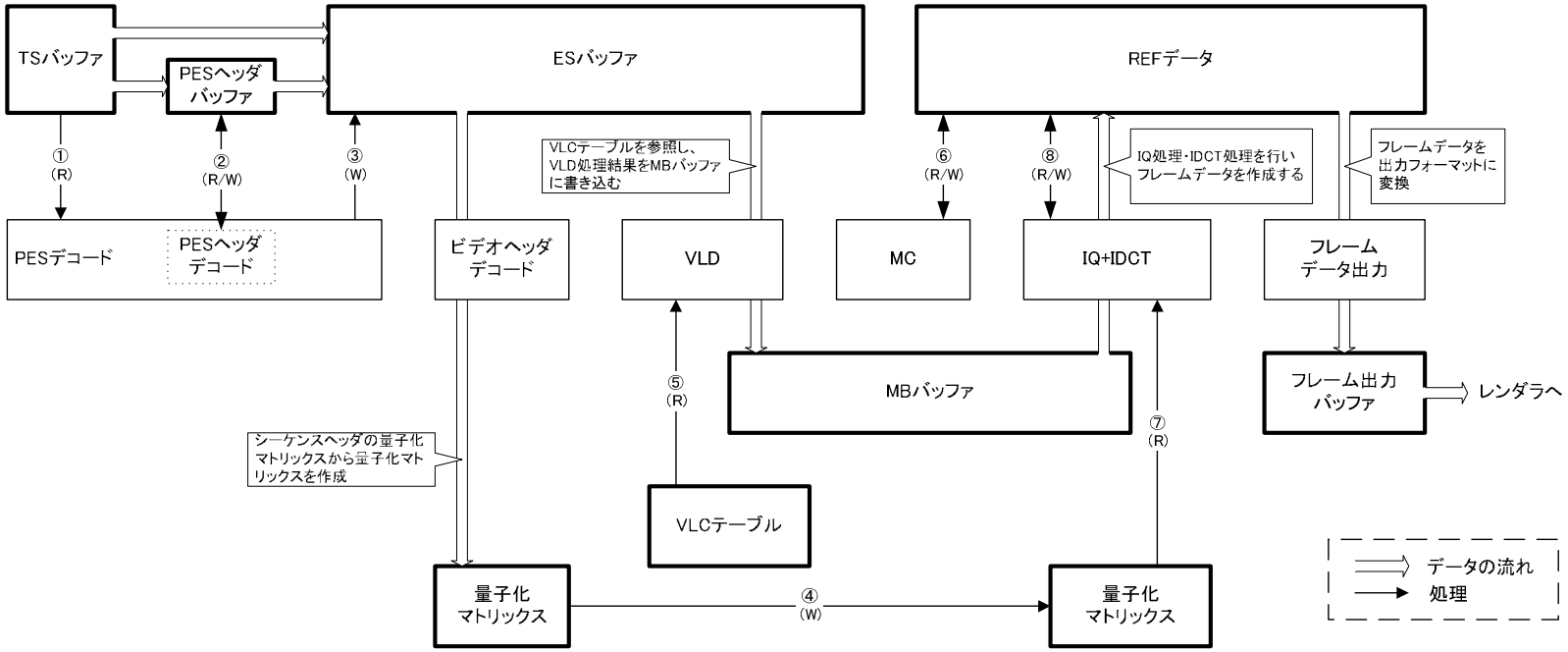
現在のビットストリーム参照処理（例）



上記のように取得したいビットストリームがビットストリーム参照バッファの有効ビットより大きい場合、ES バッファからビットストリームを取得しなければならない。このビットストリーム参照バッファを32ビットから64ビットに変更することにより、ES バッファからのビットストリーム取得回数を削減できる。

５-３-３-２　メモリ転送ポイント／メモリタイプ

現状のビデオデコーダ内メモリ構成



① TSパケットよりペイロードを取り出し、PESデコードを行う。
② ①処理でPESヘッダが存在した場合、PESヘッダが完成するまでPESヘッダバッファに保存する。
　PESヘッダが完成した場合、PESヘッダの解析を行う。
③ ①処理でPESペイロードが存在した場合、ESバッファにコピーする。
④ 量子化スケールが変化した場合、量子化マトリックスに量子化スケールを掛ける
⑤ 各VLCテーブルを参照し、マクロブロックデコードを行う。
⑥ VLD処理で取得した動き補償情報とREFデータをもとに動き補償処理を行う。
⑦ MBバッファと量子化マトリックスをもとにIQ処理を行う。
⑧ ⑦処理後にIDCT処理を行う。この際、⑥処理で出力されたREFデータをIDCT出力結果に加算し、フレームデータを作成する。

バッファ詳細

バッファ名称	構造体		メンバ	サイズ	使用メソッド	（R/W） 備考
TSバッファ						(R)スプリッタフィルタのTSバッファ
PESヘッダバッファ	char * m_pPESBuff			512	CPesParser::Process	(RW)
ESバッファ	char * m_Buffer			VBV_delayに依存		
					CPesParser::Process	(W)
						(R)各デコード処理でリードするためメソッド省略
量子化マトリックス	base	struct layer_data	WORD intra_quantizer_matrix[64]	128	Slice	(W)
					DecodeMPEG2Macroblock	(W)
					MotionCompensationMpeg2P3	
					MotionCompensationMpeg2	未コール
			WORD non_intra_quantizer_matrix[64]	128	Slice	(W)
					DecodeMPEG2Macroblock	(W)
					MotionCompensationMpeg2P3	
					MotionCompensationMpeg2	未コール
量子化マトリックス			WORD g_intra_quantizer_matrix[64]	128	sequence_header	(W)シーケンスヘッダ情報からコピーまたはデフォルト値を設定
（×量子化スケール）					Slice	(R)
					DecodeMPEG2Macroblock	(R)
					quant_matrix_extension	(W)
			WORD g_non_intra_quantizer_matrix[64]	128	sequence_header	(W)シーケンスヘッダ情報からコピーまたはデフォルト値を設定
					Slice	(R)
					DecodeMPEG2Macroblock	(R)
					quant_matrix_extension	(W)
MBバッファ			short block[6][64]	768	DecodeMPEG2Macroblock	(W)クリア
					Decode_MPEG2_Inter_Block_ALL	(RW)
					Decode_MPEG2_Intra_Block_ALL	(W)
					MotionCompensationMpeg2P3	(RW)
REFデータ	backward_reference_frame		short * pAllocData(pdata)	0:Coded_Picture_Width*Coded_Picture_Height 1,2:Chroma_Width*Chroma_Height		
	forward_reference_frame		short * pAllocData(pdata)	0:Coded_Picture_Width*Coded_Picture_Height 1,2:Chroma_Width*Chroma_Height		
	auxframe		short *pAllocData(pdata)	0:Coded_Picture_Width*Coded_Picture_Height 1,2:Chroma_Width*Chroma_Height		
					InitializeDecoder	アロケート
					MotionCompensationMpeg2P3	(R)
					MotionVector1Direct16	(RW)
					MotionVector1Direct8	(RW)
					MotionVector2Direct16	(RW)
					MotionVector2Direct8	(RW)
					MotionVector1Direct16P3	(RW)
					MotionVector1Direct8P3	(RW)
					MotionVector2Direct16P3	(RW)
					MotionVector2Direct8P3	(RW)
VLCテーブル	DCTtab01aDubbleIntra		struct DCTtabDubble	10240(sizeof(DCTtab01aDubble))		MakeMpegDCTTableメソッドにて初期化
	DCTtabNext01DubbleIntra			10240(sizeof(DCTtabNext01Dubble))		
	DCTtabNext01DubbleInter			10240(sizeof(DCTtabNext01Dubble))		
	DCTtabFirst01DubbleIntra			10240(sizeof(DCTtabFirst01Dubble))		
	DCTtabFirst01DubbleInter			10240(sizeof(DCTtabFirst01Dubble))		
	DCTtab2_6aDubbleIntra			5120(sizeof(DCTtab2_6aDubble))		
	DCTtab2_6DubbleIntra			5120(sizeof(DCTtab2_6Dubble))		
	DCTtab2_6DubbleInter			5120(sizeof(DCTtab2_6Dubble))		
					Decode_MPEG2_Inter_Block_ALL	(R)
					Decode_MPEG2_Intra_Block_ALL	(R)
フレーム出力バッファ	char * g_pOutBuffer			フォーマット依存		
					Format_YUY2P3	(W)

５-４　総括

これまで述べたように、平成１５年度は前年度までに開発したセキュア回路を中心として、マイクロプロセッサ、PCI インタフェース回路等の各種インタフェース回路を１チップ化したセキュア LSI の開発を行った。また、前年度までに開発したセキュア MPEG2 デコーダの高速化検討を行い、ハイビジョン画像である MP@HL ストリームの毎秒約３０フレームのリアルタイム処理を実現した。そして、開発した LSI と地上デジタル放送の受信に必要なチューナ、OFDM 復調 LSI 等を１ボード化した PC ボードを開発した。この PC ボード上で前年度までに開発したセキュリティ機能を実際に動作させ、Windows PC 上のセキュアな環境での地上デジタル放送の受信／視聴を行うことができた。

１）LSI 化研究開発

前年度までに開発したセキュア機能をセキュア LSI として実現した。セキュア LSI は前年度までに FPGA で実証した機能をセキュアコアとして、マイクロプロセッサ、インタフェース機能を１チップ化している。セキュア LSI に集積した機能は、セキュアコアを中心として、32bit RISC プロセッサである FR マイクロプロセッサコア、PCI インタフェース回路、公開鍵暗号化マクロ、ハッシュ関数マクロ、I2C インタフェース回路、PC カードインタフェース等である。LSI は FBGA 288pin のパッケージとしてまとめた。

このセキュア LSI は放送された地上デジタル波の放送用暗号解除、ローカル暗号化をリアルタイムに行うことができ、HDD に保存するコンテンツやバス上のコンテンツを保護することができる。また、開発した PC ソフトウェアの保護機能を搭載しており、PC 上のソフトウェアの解析、改ざんを困難にしている。こうして、このセキュア LSI を使用することにより、PC 上での地上デジタル放送のセキュアな視聴、コンテンツの保存を行うことができる。

２）ソフト・リアルタイム化（改良）研究開発

セキュア MPEG2 デコーダのリアルタイム化のために高速化の検討を行った。検討では、セキュア MPEG2 デコーダの実行解析を行い、デコードアルゴリズムの見直しを行った。また、MMX/SSE/MMX2/SSE2 マルチメディア命令や Hyper-Threading technology の効果、キャッシュミス率とキャッシュ汚染防用命令の効果の調査を行った。これらの調査・検討結果を基にして、デコード演算アルゴリズムの修正を行い、特に負荷の多い処理部分を中心に高速化の適用を行った。適用した手法は、インラインアセンブラ使用による MMX/SSE/MMX2/SSE2 マルチメディア命令の適用、メモリ転送におけるキャッシュ汚染防止、明示的なプリフェッチ、参照テーブルの最小化／最適化、

Hyper-Threading technology 適用、非効率な C 言語記述の最適化である。

このような高速化手法を適用することにより、セキュアな機能を組み込みながらも、ハイビジョン画像である MP@HL MPEG ストリームをハードウェアの機能に頼ることなく、リアルタイムで再生表示が可能となった。

3）PC カード化

セキュア LSI を含めて、地上デジタル放送の受信必要な機能を搭載した PC カードを試作した。試作ボード上は、セキュア LSI 、地上デジタルチューナ、OFDM 復調 LSI、IC カードコネクタ、マイクロプロセッサ用の Flash ROM、SDRAM を搭載しており、PCI インタフェースで PC に接続するフルサイズのボードである。セキュア LSI は I2C インタフェースでチューナ、OFDM 復調 LSI を制御し、地上デジタル放送の受信制御が可能となっている。

この試作ボードではセキュア MPEG2 デコーダに対するセキュリティ保護機能が実現されており、セキュア MPEG2 デコーダと組み合わせて実際の地上デジタル放送波を受信し、視聴することが可能である。

（添付資料）

1．研究発表、講演、文献等一覧

なし