MP3 ハードウェアデコーダの IMDCT 処理における演算誤差の評価

田 中 大 輔[†] 神 山 智 章[††] 清 水 尚 彦[†††]

我々は MP3 デコーダの一部分である IMDCT 部のハードウェアを設計した。演算に積和演算器を用いることにより高速で低消費電力な回路を実現した。また、16 ビットのハードウェア用フォーマットに変換する際に発生した誤差を演算器内で定数の加算をすることにより補正することに成功した。

Evaluation of Operation Error for Hardware MP3 Decoder IMDCT Processing

Daisuke Tanaka,[†] Tomoaki Kouyama[††] and Naohiko Shimizu[†††]

We designed IMDCT circuit that is a part of MP3 decoder. We made fast and low power circuit using multiply-add ALU and we succeeded to correct error generated when changing into the 16-bit format for hardware by adding a constant within an ALU.

1. はじめに

我々は MP3 デコーダの IMDCT 部のハードウェア設計を行った。IMDCT はデコードのプロセスにおいて周波数域の成分から時間域の成分へと変換を行う主要な部分であり、Inverse Modified Discrete Cosine Transform の略である。ソフトウェアでデコードを行う場合、この IMDCT 部において多量の演算を行う必要があるため、ハードウェア化することにより消費電力の大幅な削減を計ることができる。特にポータブル MP3 プレーヤの場合には，これは大きな利点となる。我々は文献[1] において、積、和を個別の演算器とした場合の評価をしてきた。本稿では積和演算器を用いた IMDCT 回路について評価を行う。

2. IMDCT 回路

2.1 IMDCT の入出力

IMDCT の入出力はグラニュールという単位で行なわれる。1 グラニュールは 32 に分割されたサブバンドと、それぞれ 18 のサンプルの、576 サンプルから構成される。サブバンドとは周波数成分を帯域ごとに分割したものであり、サブバンド 0 は最も低い周波数帯域にあたる。各サンプルは振幅の値を持ち、サンプル 0 はそのグラニュール内で最も早くサンプリングされた値である。IMDCT はサブバンドごとのそれぞれのサンプルの値を周波数領域の振幅から時間領域の振幅へと変換を行なう。

2.2 浮動小数点型式

我々のハードウェアでは文献[2] で用いた、図 1 に示す 16 ビットの浮動小数点形式を用いる。浮動小数点形式は符号部、指数部、仮数部からなり、MSB 側からそれぞれ 1 ビット、5 ビット、10 ビットが割り当てられる。符号部は 0 であれば正の数、1 であれば負の数として扱う。指数部は $(00000 \sim 11111)_2$ であり、それぞれ $2^0 \sim 2^{-31}$ に対応する。仮数部は $(0000000000 \sim 1111111111)_2$ であり、MSB に暗黙の 1 を持つため $(1.0000000000 \sim 1.1111111111)_2$ となる。よって、この浮動小数点形式の範囲は $\pm 1.0000000000 \times 2^{-31} \sim 1.1111111111 \times 2^0$ となる。

3. 浮動小数積和演算器

IMDCT 処理において、非常に多くの積和演算 $(x \times y \pm z)$ が必要となる。この演算を乗算器と加算器を用いて別々に行なうと、クロック数の増大による消費電

† 東海大学工学部通信工学科
Department of Communications Engineering, School of Engineering, Tokai University
†† 東海大学大学院工学研究科
Graduate School of Engineering, Tokai University
††† 東海大学電子情報学部コミュニケーション工学科
Department of Communications Engineering, School of Information Technology and Electronics, Tokai University

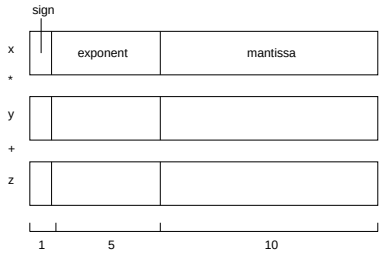

図 1　浮動小数点型式

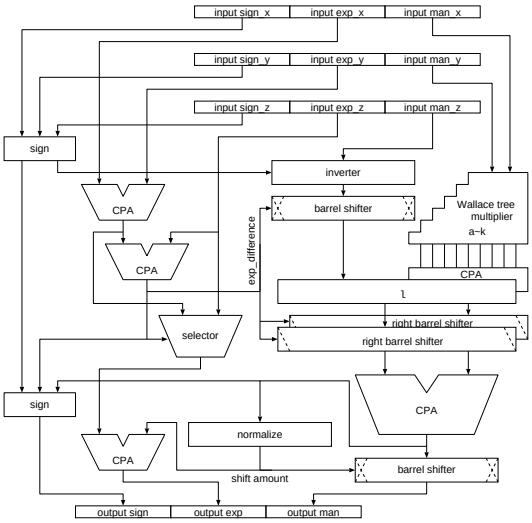

図 2　積和演算器ブロック図

力の増加や、乗算の後、一度浮動小数点フォーマットに丸めることによる誤差の増加がおこる。そこで、演算処理を 1 つの回路にまとめ、1 クロック内で演算することで無駄な処理を省き、消費電力を削減すると共に、精度および演算速度を向上させることができる積和演算器を設計した。本節ではこの演算器の詳細について述べる。全体の処理の流れを示すブロック図を図 2 に示す。

3.1　演算器の構成

積和演算器に入力された 3 つのデータは符号部 (sign)、指数部 (exponent)、仮数部 (mantissa) に分割される。回路は指数制御部と仮数演算部に大きく分かれ、指数制御部には符号部と指数部が、仮数演算部には仮数部が入力される。

3.2　仮数演算部

仮数演算部では仮数部の乗算および加算を行う。乗算は加算の繰り返しであり、加算器にキャリーパスアダー (CPA) を使用した場合、演算結果が決定するまでの遅延時間は大きなものとなる。そこで、キャリーセーブアダー (CSA) を用いて Wallace-Tree を形成

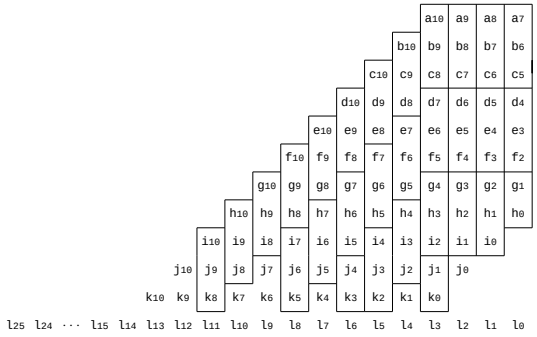

図 3　Wallace tree multiplier

し、遅延時間を大幅に削減した。最終段ではキャリーを伝搬させ、解を求めなくてはならないため CPA を 1 段用いる。乗算と加算を行うため、合計で CPA が 2 段必要となる。しかしながら、遅延時間の大きい CPA の使用を 2 段のみとすることで、全段 CPA を使用する乗算器と比べて大幅な遅延時間の削減を実現している。また、乗算と加算を別々に行う場合には指数部が $(11111)_2$ を越えた場合、すなわち 2^{-31} 以下になった場合、一度最小の値に丸める必要があるが、積和を統合することで積の結果を保存しておき、後の加算で指数部が 2^{-31} を上まわった場合でも演算精度を確保することができる。

図 3 は 1 段目の Wallace-Tree を示したものである。a∼k は乗算の部分積を筆算の形に並べたものである。図中に囲われているように同位のビットが 3 ビット集まった場合に CSA に入力し、これらを繰り返す。l は乗算の後に行う加算の加数であり、積と加数が異符号であれば補数型式にしておき、積と加数の指数部の差を求め、バレルシフトをして Wallace-Tree に加える。すなわち、加数の指数部が大きい場合には左シフトを行い、積の指数部が大きい場合には右シフトを行う。

乗数、被乗数の仮数部は必ず $(1.0000000000)_2$ 以上であるので、繰り上がりを考慮した積 22 ビットのうち上位 12 ビットを見ればよい。よって、有効桁への繰り上がりのための 3 ビットを含む上位 15 ビットのみをを計算し、それ以下のビットは精度にほとんど影響がないことを確認したため計算しておらず、回路規模や消費電力とのバランスを計っている。

左シフトを行った場合、最終段 CPA の入力から外れてしまうため、図 2 に示すように左シフトと同じ量を積、加数共に右シフトする必要がある。右シフトを行った場合、l_2 ∼l_0 の 3 ビットが機能し、Wallace-Tree の下位桁を削減したことによる演算精度の低下を防ぐ。

3.3 指数制御部

3.3.1 指数部の演算

指数制御部では指数部の演算、符号の処理、正規化および出力を行う。乗算の場合の指数部の演算は符号に関わらず加算をすることで求められる。加算の場合は積と加数の指数部を減算し、仮数部のバレルシフタのシフト量を決定する。なお、シフトの方向は減算結果の正負を調べる事により判定する。

加算において、通常指数部は大きい方を選択するが、左シフトの場合 12 ビット以上、右シフトの場合 14 ビット以上のシフトを要する場合には、シフトおよび加算の必要はなく、指数部と共に仮数部も大きいものをそのまま出力とする。

3.3.2 正 規 化

浮動小数点型式では、小数点は最上位の 1 とその 1 ビット下位との間になくてはならないが、乗算や加算による繰り上がりや繰り下がりなどにより小数点の位置がずれる場合があり、その場合は正規化を行い指数部や仮数部を補正する必要がある。

正規化前の仮数部は normalize 回路に送られ、正規化に必要なシフト量が返される。必要なシフト量と指数部の補正の量は均しいため、仮数部は求められたシフト量をバレルシフトし、同時に指数部は正規化前の指数部に求められたシフト量を加算する。

正規化後の指数部が最小値、または最大値を越えた場合には、この浮動小数点型式の最小値、最大値に丸めを行う。

3.3.3 符号の決定

乗算では乗数と被乗数が異符号の場合にその積は負となる。また、仮数演算部の項で述べた通り、加算では積と加数が異符号の場合に加数を補数型式にしておく。積が正で加数が負の場合、積が負で加数が正の場合のどちらにおいても符号の反転を行うのは加数のみであるため、後者の場合には最後に符号を反転させる必要がある。

4. 評価ツール

4.1 Compliance Test

MP3 デコードが正しく行われたかを評価する方法として Compliance Test がある。これはあらかじめ用意されているサンプルデータとの RMS をとり、一定の基準に達しているかを調べるものである。比較は以下の式により行われる。

表 1 Compliance Test

RMS Level	デコーダの性能
8.810e-06 以下	Fully Compliant
1.410e-04 以下	Limited Accuracy
1.410e-04 以上	Not Compliant

$$\sqrt{\frac{1}{N}\left(\sum_{i=1} N\left(t_i - r_i\right)^2\right)} \qquad (1)$$

ここで、N はサンプル数、t_i はサンプルデータ、r_i はデコードしたデータである。ただし、t_i、r_i は -1.0 から $+1.0$ の間で振幅を取る。

Compliance Test の基準は表 1 に示す 3 段階に分けられる。ここでは Limited Accuracy 以下の値であることをデコーダの要求仕様とする。

なおサンプルデータの波形は sin 波の周波数スイープである。

5. 評 価

5.1 ソフトウェアの誤差

まずソフトウェアのみを用いた場合の誤差の影響を調べる。

Compliance Test を行った結果、

$$1.781908 \times 10^{-5} \quad \text{(Limited Accuracy)} \qquad (2)$$

となった。なおソフトウェアデコーダのソースは C++ 記述であり、演算には double 型を用いている。

5.2 浮動小数点フォーマットによる誤差

ハードウェアを用いたデコードには前述の 16 ビット浮動小数点フォーマットを用いるが、ソフトウェアの double 型に比べて精度が落ちるため、このフォーマットに変換することによる誤差の影響を調べる必要がある。そこで、IMDCT のハードウェアで処理が行われる演算部分の前後において double 型のデータを一度 16 ビット浮動小数点フォーマットに変換し、再び double 型へと戻すことで誤差の影響を調べた。演算部分以外の環境は全てソフトウェアのみの場合と同様である。

その結果、Compliance Test は

$$4.468135 \times 10^{-4} \quad \text{(Not Compliant)} \qquad (3)$$

となった。これより、この浮動小数点フォーマットでは Limited Accuracy を満たすのに必要な精度を確保することができないことがわかる。

5.2.1 評 価

16 ビット浮動小数点フォーマットに変換することに

よる誤差の原因は変換の際に仮数部の有効桁以下が切り捨てられることによる。従って IMDCT の出力における誤差率はほぼ一定であると考えられ、誤差の大きさは指数部に依存することになる。

IMDCT の出力値を調べたところ、前半から中盤にかけて出力成分のほとんどがサブバンド 0 で占められており、周波数が上昇するにつれて次第に主要な出力のサブバンドは上昇していった。そこで、サブバンド 0 のみの出力を誤差と共にグラフにプロットした。これを図 4 に示す。誤差は 100 倍に拡大表示されている。ここではソフトウェアのみを用いた IMDCT の出力を比較対象とし、誤差は 2 値の差により求めた。

この結果より、16 ビット浮動小数点フォーマットに変換した場合の IMDCT の出力はソフトウェアにくらべてその絶対値が常に負の方向に生じていることがわかる。これはフォーマット変換の際に端数の切捨てを行なっており、IMDCT では演算の解を再帰的に使用する場合が多いため、誤差が一方向に累積してしまうことが原因であると考えられる。

誤差の原因が端数の切捨てによるものであるかを確認するために、フォーマットの変換を四捨五入を用いて行うものに変更し、同様の検証を行った。その結果、Compliance Test は

$$6.184840 \times 10^{-5} \quad \text{(Limited Accuracy)} \qquad (4)$$

となり、誤差の原因がフォーマット変換時の切捨てによるものであることが確認された。

文献[1] においてもこの 16 ビット浮動小数点フォーマットを用いて、Limited Accuracy を満たす精度を確保できるとあるが、この場合もフォーマット変換には四捨五入を行うものを使用しており、今回の検証の結果と一致する。

5.3 補正なしハードウェア積和演算器による誤差

誤差の原因としてフォーマット変換によるものの他に演算器によるものも考えられる。よって、前項ではソフトウェアにより行っていた演算をハードウェアで行うことにより誤差がどのように変化するかを調べる。なお、フォーマット変換は共に切捨てのものを用いる。

ハードウェアシミュレーションの処理の流れを図 6 に示す。ハードウェア積和演算器の記述は SFL 言語にて行い、シミュレーションには PARTHENON システムの SECONDS を用いた。また、C++から SECONDS を呼び出すために関数パッケージ runseconds を使用した。ソフトウェアの演算部分において 16 ビット浮動小数点フォーマットに変換し、runseconds 関数を用いて SECONDS を起動、ハードウェアシミュレーション

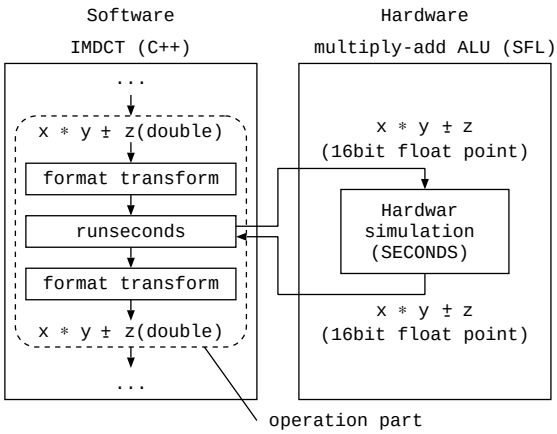

図 6 runseconds 関数を用いたハードウェア演算器のシミュレーション

による演算結果をソフトウェアへ戻し、double 型へ再び変換してデコードを行なった。

Compliance Test の結果は

$$3.208897 \times 10^{-4} \quad \text{(Not Compliant)} \qquad (5)$$

となり 16 ビット浮動小数点フォーマットを用いたソフトウェアデコードの場合よりもよい結果となった。

5.3.1 評　　価

内部精度の問題から、ハードウェアを用いたデコードの方が double 型で演算を行なうソフトウェアに比べて演算精度は悪化すると考えていたが、実際の結果は予想とは逆であった。この原因に挙げられるのがソフトウェアのみのデコードではハードウェアをシミュレートするために演算結果を全て一度フォーマット変換していることである。フォーマットを変換している関数は端数を切り捨てるためなんらかの誤差により少しでもこの値を下まわると再び次回の変換時に 1 ステップ下の値に切り捨てられてしまう。

ハードウェアデコードの場合にはフォーマット変換は入力前および出力後の二度のみであり、浮動小数点フォーマットのまま出力まで値は保存されるため、演算器内部精度による誤差を差し引いてもソフトウェアデコードの結果を上回ったと考えられる。

5.4 補正ありハードウェア積和演算器による誤差

図 4 より誤差は常に負の方向に生じているので演算器の結果に補正をかけることで誤差の改善を試みる。図 7 に演算器の結果の仮数部にそれぞれ $(000)_2 \sim (111)_2$ の補正値を加えた場合のそれぞれの Compliance Test の結果を示す。これよりハードウェア演算器内において、演算の結果の仮数部に $(110)_2$ を加えた場合、最もよい結果となることがわかる。

演算結果に $(110)_2$ を加える補正を行い、図 4 と同

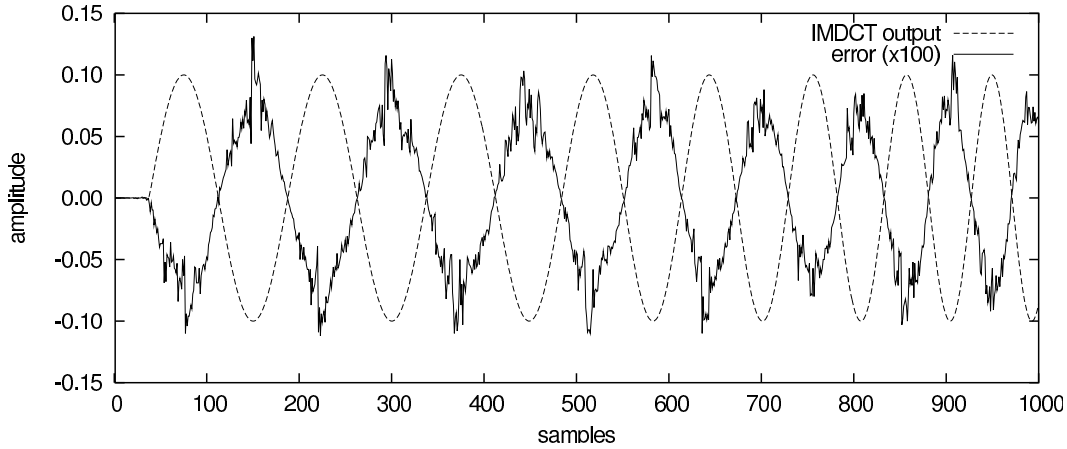

図 4 サブバンド 0 における IMDCT 出力および誤差。破線は IMDCT のソフトウェア出力、実線は 16 ビット浮動小数点フォーマットへ一度変換を行った場合の出力とソフトウェア出力の誤差である。誤差は 100 倍に拡大して表示している。実際の出力はソフトウェア出力の振幅よりその絶対値が下回っていることがわかる。

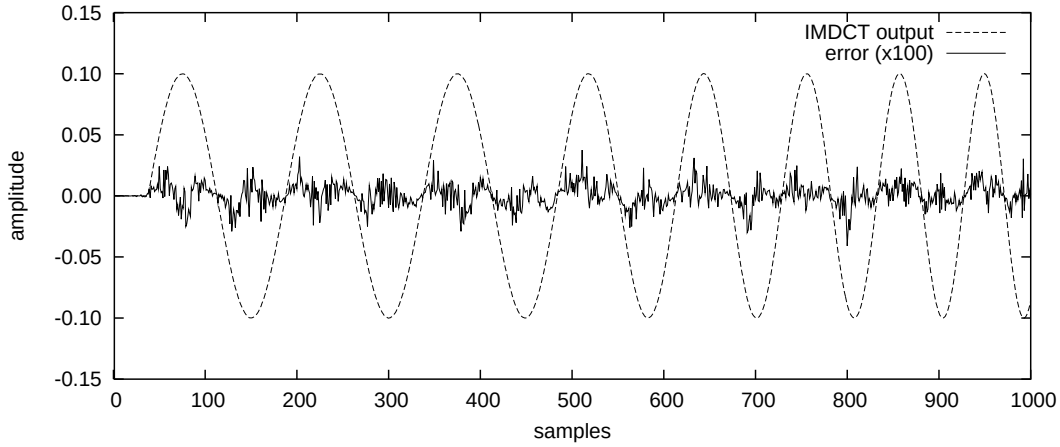

図 5 サブバンド 0 における補正後の IMDCT 出力および誤差。演算器の出力の仮数部に $(110)_2$ を加え、誤差を補正する。破線は IMDCT のソフトウェア出力、実線はハードウェア出力とソフトウェア出力との誤差である。誤差は 100 倍に拡大して表示している。補正により誤差が減少したことがわかる。

様にサブバンド 0 における IMDCT 出力、および誤差をグラフにプロットしたものを図 5 に示す。誤差は 100 倍に拡大して表示している。図 5 より、図 7 に示される通り、誤差を大幅に減少させることができたことが確認できる。

以上より、演算器における、仮数部に $(110)_2$ を加算する補正の結果、Compliance Test は

$$8.865371 \times 10^{-5} \quad \text{(Limited Accuracy)} \qquad (6)$$

となり、Limited Accuracy を満たす音質を確保することができた。これにより、四捨五入処理を行わなくとも定数の加算器を用いて同様の結果を得ることができ、デコーダ全体でのハードウェア実装を考慮した場合、このような単純な方法で補正可能であることは消費電力や処理速度の点から有利であるといえる。

5.5 IMDCT 回路全体での誤差

補正あり積和演算器を含むハードウェア IMDCT 回路全体を用いて誤差を検証する．Compliance Test の結果は

$$8.865371 \times 10^{-5} \quad \text{(Limited Accuracy)} \qquad (7)$$

となり，演算のみハードウェアでデコードを行なった場合と完全に一致した．このことから，IMDCT 回路の誤差の原因は全て 16 ビット浮動小数点フォーマットおよび演算器により生じたといえる．

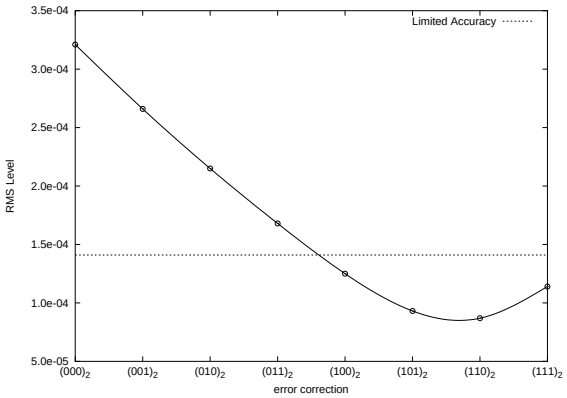

図 7 仮数部に加える補正の値と RMS Level の関係

表 2 Compliance Test のまとめ

環境	RMS Level	Result
SW のみ	1.781908×10^{-5}	Limited Accuracy
16 ビット浮動小数点 (切捨て)	4.468135×10^{-4}	Not Compliant
16 ビット浮動小数点 (四捨五入)	6.184840×10^{-5}	Limited Accuracy
補正なしハードウェア	3.208897×10^{-4}	Not Compliant
補正ありハードウェア (+001)	2.663428×10^{-4}	Not Complaint
補正ありハードウェア HW(+010)	2.148302×10^{-4}	Not Complaint
補正ありハードウェア (+011)	1.676896×10^{-4}	Not Complaint
補正ありハードウェア (+100)	1.249091×10^{-4}	Limited Accuracy
補正ありハードウェア (+101)	9.306782×10^{-5}	Limited Accuracy
補正ありハードウェア (+110)	8.685371×10^{-5}	Limited Accuracy
補正ありハードウェア (+111)	1.144959×10^{-5}	Limited Accuracy
補正ありハードウェア (+110) のみ	8.685371×10^{-5}	Limited Accuracy
文献[3] のハードウェア	1.010266×10^{-4}	Limited Accuracy

6. ま と め

表 2 に評価を行なった Compliance Test の結果をまとめる.

フォーマットの変換により不足していた精度を演算器で補正することで Limited Accuracy を満たすのに必要な精度を確保することができた。補正は単純な定数の加算のみで実現可能なため、これを用いることにより、デコードの精度を落とさずに高速、低消費電力で、かつ回路規模の小さなハードウェアを実現することができる。特に、組込み用途での実装を考慮した場合，このように簡潔な方法で誤差を補正できることは非常に有利であり、IMDCT 回路だけでなく、浮動小数点において一定の誤差率が現れる場合には有効な手法であると考えられる。

また、積と和を別々に行った文献[3] の方法に比べ、消費電力、処理性能が向上した上で、精度も向上させた。

7. 謝　　辞

MP3デコーダハードウェア/ソフトウェア協調シミュレーション環境を提供して頂いた、京都大学の泉知論先生に感謝致します。

参 考 文 献

1) 神山智章．清水尚彦：
MP3 デコーダ IMDCT 処理に必要な制度の評価
第 3 回システム評価研究会
2) 田中大輔，神山智章，金井友明，山内陽右：
IMDCT 回路 ROA の設計第 22 回パルテノン研究会資料集.
3) 神山智章：
IMDCT 回路 非 YO の設計
第 20 回パルテノン研究会資料集.
4) 福島徹也，李星日，境和久，泉知論，尾上孝雄，中村行宏：
PARTHENON を用いた MP3 デコーダのハードウェア/ソフトウェア設計
第 18 回パルテノン研究会資料集.
5) ASIC デザインコンテスト規定課題 「MP3 デコーダの IMDCT 処理のハードウェア化」