システムＬＳＩ設計技術　97－1
（２０００．９．７）

出力品質を考慮した変数ビット幅最適化手法

山下 源　エコー ファジャル ヌルプラセティヨー　安浦 寛人

九州大学 大学院システム情報科学研究科

〒816–8580 福岡県春日市春日公園 6-1
Tel. 092-583-7622
{hajime, eko, yasuura}@c.csce.kyushu-u.ac.jp

携帯器機および家電の LSI 部を設計する際は，設計者が音声，画像をはじめ器機出力の品質および仕様を考慮して，ハードウェアおよびソフトウェアの設計最適化を行っている．我々はワード幅が出力品質にもたらす影響に着目し，出力品質駆動設計 (output-quality-driven) 最適手法を提案している．本論文では，出力品質の最適化を考慮した動作記述の作成とハードウェア設計手法について提案する．

キーワード：出力品質，ハードウェア/ソフトウェア・コデザイン，システム最適化

Output Quality Based Variable Bit-width Optmization Method

Hajime Yamashita　Eko Fajar N.　Hiroto Yasuura

Graduate School of Information Science and Electrical Engineering,
Kyushu University

Kasuga-koen 6-1, Kasuga, Fukuoka 816–8580, Japan
Tel. 092-583-7622
{hajime, eko, yasuura}@c.csce.kyushu-u.ac.jp

Designers of mobile or household appliances often optimize both hardware and software in order to make products which complies to a designated output quality specification. Noticing that word-width can affect the output quality, we come into a novel optimization method that we call "output quality driven" optimization method. In this paper, we will overview our recent researches on system description and hardware design deploying the aforementioned optimization method.

key words : Output quality, Hardware-software codesign, System optmization

1 はじめに

システムLSIのような大規模な集積回路の設計では，設計の長期化が問題になっている．反面，製品サイクルの短期化により，用途に応じて最適なシステムを短期間で設計することがますます重要になってきている．同様の機能で演算精度が異なる複数のシステムを設計する場合，1つの動作記述を元に演算精度のみを変更してシステムを設計することで，システム設計者は用途に応じてコストを最適化した複数のシステムを効率よく設計できると考えられる．このような設計を行うシステムとして，動画処理や音声処理システムが考えられる．例えば動画デコーダのような，表示デバイスに処理の結果を出力するシステムを考える．性能の低い表示デバイスでは多少の画質の劣化は問題にならないため，携帯電話に搭載されるような低解像度ディスプレイ向けのデコーダシステムでは，高解像度ディスプレイ向けのデコーダシステムに比べて，デコード結果に要求される画質は低くなる．このような設計の場合，デコーダ内部のIDCTといった算術演算部の演算精度を落とすことでデコーダ内のレジスタや演算器に必要な幅を削減し，コストを設計目的に応じて最適化する手法が考えられる．演算精度を考慮してコストを最適化する設計では，シミュレーションなどによる出力データの確認とコストの見積もりを行い，設計目標を満さない場合には演算精度を再変更しなければならない．演算精度の変更には動作記述の変更が必要になる場合もあり，容易ではない．よって，手動で行う必要がある変更以外について自動化するなど，演算精度の変更の際にシステム設計者が行う作業を最小限にすることは重要である．

本論文の構成は以下の通りである．2章では，例題にMPEG2ビデオデコーダを用いて，演算精度の変更による出力データへの影響を示し，システムのコスト最適化について述べる．3章で，演算精度を静的に変更することを考慮した動作記述の作成手法を提案し，4で，演算精度を削減した動作記述を元にプロセッサの低消費電力化について考察する．最後に5章でまとめと今後の課題について述べる．

2 システムの演算精度

本章では，演算精度の変更がシステムの出力データに与える影響を示す例をあげる．また，演算精度を変更することによるシステムのコストの最適化について考察する．

表 1: *idct.c*

関数	変数の数	行数	処理
idct()	1	11	2次元 IDCT
idctrow()	9	54	row IDCT
idctcol()	9	54	column IDCT

2.1 アプリケーション

アプリケーションプログラムには，ソフトウェアMPEG2デコーダ"mpeg2decode"[1]を用い，IDCT部の演算精度を2ビット単位で変更したMPEG2デコーダを作成した．表1にmpeg2decode内のIDCT部のソースプログラム *idct.c* を示す．関数 *idct* は，16ビットの固定小数点数64個を入力とし，32ビットの局所変数を用いて処理を行い，16ビットの固定小数点数64個を出力する．ここでは，演算結果を格納するIDCT部の全ての変数に対して，図1の様に下位nビットを固定する記述を追加することで，固定小数点数のダイナミックレンジを変更しないで演算精度のみを変更した．

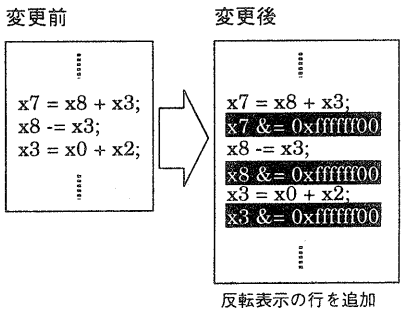

図 1: *idct.c* の変更

2.2 デコード結果の評価

演算精度を30ビットから16ビットまで2ビット毎に変更したデコーダの出力結果について，元の演算精度のデコーダの出力結果に対するPSNR(ピーク信号対雑音電力比)を下の式を用いて算出し，比較した．

$$PSNR = 10\log_{10}\frac{MAX^2}{MSE}[dB]$$

(MAX : 画素の最大値)
(MSE : 2つの画像間の平均2乗誤差)

図2は，IDCT部の演算精度とPSNRの関係を示したグラフである．このグラフによれば，IDCTの演算精度の低下にほぼ比例してPSNRが低下，つまり画質が低下する．

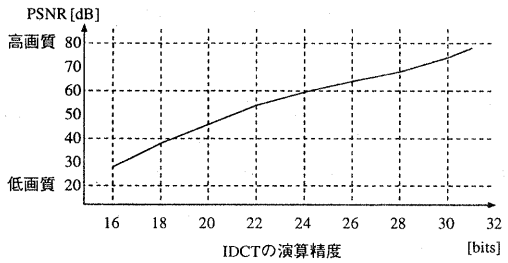

図 2: IDCT の演算精度と PSNR

図3は，IDCTの演算精度が(A)32ビット，(B)20ビット，(C)18ビットとしたデコーダの出力結果の中の1フレームである．(A)と(B)を人の目で比較すると，演算精度を20ビットまで低下させても画質には影響がないと判断できる．一方，演算精度を18ビットに落としてデコードした(C)では，画像にノイズが入っていることが分かる．

図 3: IDCT の演算精度と画質

図4は，IDCTの演算精度が32ビットの場合と18ビットの場合のデコード結果を，それぞれ2つの画像サイズで表示したものである．画像サイズが352×240の場合では，図3中の(A),(C)と同様に画質の違いが明確である．しかし，画像サイズが176×120の場合では，IDCTの演算精度が18ビットのデコーダを用いたデコード結果のノイズは352×240の場合よりも目立たなく見える．つまり，出力先の表示デバイスが携帯電話に搭載されるような低解像度・小画面であれば，多少の画質の低下は問題ないと思われる．

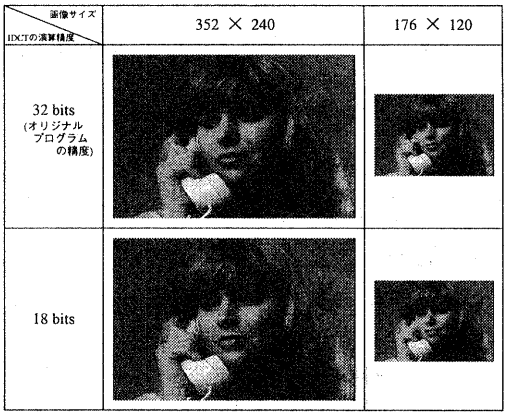

図 4: IDCT の演算精度と画像サイズ

2.3 演算精度と設計最適化

要求される演算精度が比較的低いシステムは，高い演算精度を要求されるシステムよりも低コストを図ることができる．例えば，演算精度を低く抑えたシステムでは，動作記述中の変数の幅を削減することができる．そして，最適化された変数の幅を考慮することで，レジスタや演算器などのデータパスの幅を小さくすることができる [2]．また，組込み用途の比較的小規模なプロセッサでは，その面積がデータパス幅にほぼ比例するという報告がある [3],[4]．このことから，図5に示すように，要求される出力データの品質を考慮してシステムの演算精度を変更することで，設計目的に合わせたシステムのコストを最適化する設計手法が考えられる．また，動画や音声用のデコーダのように，動作が共通で設計目的によって出力データの品質が違う複数のシステムを設計する場合は，システムの演算精度を変更できる設計環境を用意することで設計期間の短縮化が期待できる．

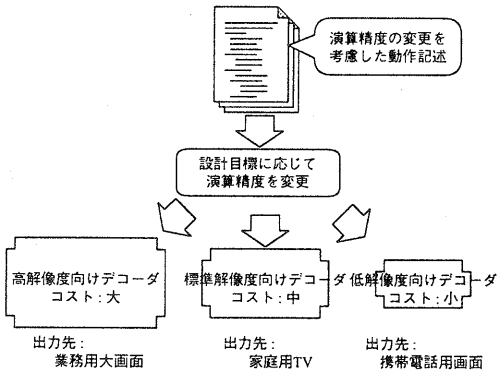

図 5: 演算精度を静的に変更する設計例

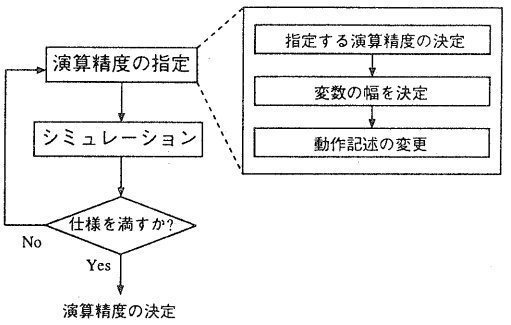

図 6: 演算精度の決定

3 演算精度の変更を考慮した動作記述の設計

動作記述中で演算精度が変更可能な演算は，実数の算術演算である．この演算の精度は，被演算数の精度と演算結果を格納する際の丸めに依存する [5],[6]．丸めは，演算結果を格納する変数の幅に合わせて行われる．よって，システム設計者は実数を格納する変数の幅を変更することで，システムの演算精度を動作記述レベルにおいて変更することができる．

演算精度を決定するフロー (図 6) では，シミュレーション結果などを用いて設計仕様を満すか否かの判断をし，必要であれば演算精度を再変更する．演算精度の再変更と出力データの確認は設計仕様を満すまで繰り返さなければならない．よって，演算精度を再変更する際に発生する動作記述の変更をできるだけ少ない手順で行える環境が必要となる．本章では，システムの演算精度を動作記述レベルで変更するプロセスについて，自動化を含めた設計手法の考察を行う．

3.1 動作記述での演算精度の変更

実数を格納する変数の幅を変更する作業は，データフローグラフが大きくなるにしたがって困難になる．演算精度の変更を考慮していない動作記述に対して，システム設計者が演算精度を変更する場合は，さらに大変な作業になると考えられる．そこで，演算精度を変更する際のシステム設計者の負担を軽減するために，変数の幅の変更をある程度自動化する設計手法を提案する．この手法では，システム設計者が動作記述中のいくつかの変数の幅を変更する．次に，この指定された幅と予め用意された規則にしたがって，他の変数の幅が自動的に変更される．下に例を示す．

$$x = a + b; \quad \text{——— 式 1}$$
$$y = a * b; \quad \text{——— 式 2}$$
$$z = 2 * x + y; \quad \text{——— 式 3}$$

この 3 つの式は，基本ブロック中に存在する式とする．式 1 の変数 x に必要な幅は，変数 a と変数 b の幅と加算結果を何桁に丸めるかという情報から決定することができる．例えば，加算の場合は被演算数の有効桁数と同じ桁数に丸めるという規則を設けると仮定する．この場合は，変数 a と変数 b の幅を指定し，変数 x の幅は変数 a と変数 b の大きい方の幅と自動的に決定することができる．式 2 については式 1 と同様にして，式 3 については式 1,2 の変数 x と変数 y と，加算と乗算の場合の規則から変数 z の幅を決定することができる．以上より，5 つの変数のうち，2 つの変数の幅を手動で指定することにより，残りの 3 つの変数の幅を自動で決定することができる．

丸めについての規則は，演算誤差を小さくするものである必要はない．演算によっては小さい桁数に丸める規則を用意しておくことで，演算結果を格納する変数の幅を小さくでき，2.3 節で述べたように，システムのコストを最適化することができる．

3.2 演算精度の変更を考慮した動作記述

3.1節で述べた設計手法を用いるために，動作記述を作成する際には，演算精度の変更の際に必要となる情報を合わせて用意する必要がある．必要となる情報は，下の3つが考えられる．

(i) システムの演算精度に関係する変数

(ii) システム設計者が設計時に手動で幅を変更する変数

(iii) 演算結果の丸めに関する規則

以下，この3つの情報について考察する．

(i) システムの演算精度に関係する変数

システムの演算精度に関係する変数とは，幅の大小がシステムの出力データの品質に関わる変数を指す．演算精度の変更はシステムの動作に変更を加えること無く行われる必要があるため，システムの演算精度に関係する変数の幅だけを扱う必要がある．例えばフラグの格納に用いられる変数には，整数型変数を用いられる．しかし，固定小数点数を用いて実数演算を行う動作記述の場合では，制御に用いられている変数と同じ整数型変数を用いて実数演算が行われるため，制御用の変数と実数演算用の変数の区別がつかないことがある．動作記述中の変数のうち，システムの演算精度に関係する変数の情報は動作記述の作成者が持っている．そこで，動作記述を作成する際には，動作記述の作成者がこの情報を用意しておく．システム設計者は，演算精度を変更する際にこの情報を用いることで，システムの動作を変更すること無くシステムの演算精度を変更することができる．

(ii) システム設計者が設計時に手動で幅を変更する変数

動作記述中の全ての変数の幅の変更をシステム設計者が行うと，データフローグラフが大きい場合は作業が困難になるだけでなく，変更の誤りが混入する恐れがある．そこで，システムの演算精度を変更する際にシステム設計者が指定するパラメータの数は，必要最小限である必要がある．(i)で述べたシステムの演算精度に関係する変数のうち，どの変数の幅をシステム設計者が手動でして指定する必要があるかといった情報も動作記述の作成者が持っている．よって，この情報も動作記述の作成時に合わせて用意される必要がある．

(iii) 演算結果の丸めに関する規則

システム設計者が手動でいくつかの変数の幅を指定した後は，残りの変数の幅は自動で指定される．自動で指定する変数の幅は，システム設計者が手動で指定した変数の幅を元に決られることになる．3.1節で述べたように，演算や代入文で変数に値が代入される際に何桁への丸めが行われるかによって，値が代入される変数の幅が決まる．自動で変数の幅を指定する際には，この何桁への丸めを行うかといった規則を予め用意する必要がある．例えば，演算別に丸めに関する規則を用意する手法が考えられる．表2は，演算別に被演算数の精度と演算結果の精度の関係を示す式の一例である．この表の式を用いた場合は，演算結果の精度は落とされず最大限に保たれる．演算精度を落とす設計を行う場合は，表2中の式よりも精度が低くなる式を用いる手法が考えられる．また，変数間の

表 2: 演算別の丸めに関する規則の例

演算	演算結果の精度 [bits]
加算, 減算	$\max(n_1, n_2) + 1$
乗算	$n_1 + n_2$
除算	$\max(n_1, n_2)$

n_1, n_2 : 被演算数の精度 [bits]

精度の関係を示す式を動作記述の作成者が用意する手段も考えられる．システムの演算の性質を考慮して作成した，変数間の精度の関係に関する情報が動作記述とともに用意されることで，システム設計者はシステムに適した演算精度の変更ができる．

3.3 システム設計では演算精度の変更

3.2節で述べた3つの情報を用いて，システム設計者は図7手順でシステムの演算精度を指定する．まず，システム設計者は (ii) の変数について，手動で幅を指定する．次に，(iii) の情報とツールを用いて，(i) の変数から (ii) の変数を引いた残りの変数の幅を自動的に指定する．その後，指定された変数の幅を考慮してシステムを合成し，テスト

データを用いてシミュレーションを行う．システムのコストと出力データの品質を確認し，設計目標を満していれば変数の幅を決定する．

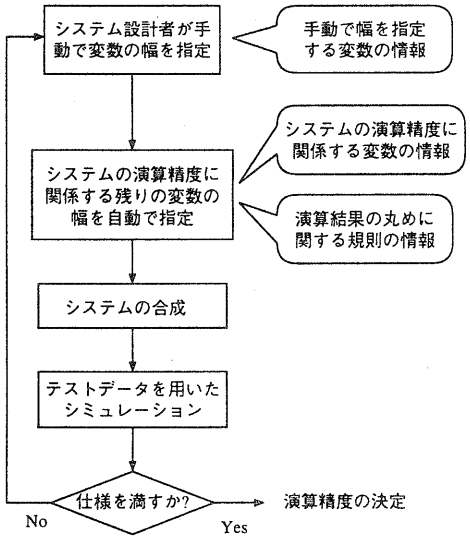

図 7: システム設計者による演算精度の決定

出力データの品質を 2.2 節で用いた PSNR のように客観的に確認する場合は，品質確認を自動で行う手法も考えられる．

4 演算精度を考慮したプロセッサ

4.1 基本概念

ハードウェア側はビット幅解析手法による動的下位ビット列マスキングをサポート可能であると仮定する．この場合，最低の条件としてハードウェアに「演算マスキング」可能な命令が必要である．一般に全データワードをマスキング可能にする必要はなく，例えば下位 16 ビットをマスキング可能なフィールドとする手法が考えられる．

演算精度の削減をハードウェアでサポートすること，下記の利点が考えられる．

- 実行効率の向上
  プロセッサがビットマスキングを直接サポートする場合，コンパイラがマスキング命令を変数がアクセスされる毎に発行する必要がなく，命令メモリの節約および実行速度の向上が期待できる．
- 詳細な消費電力制御
  発行マスク命令を通じて，プロセッサが不要なビットの情報を正確に関知できるため，回路レベルの電力制御が可能になる [7].

演算マスキングを行うための形態は，下の 2 種類考えられる．

- 命令マスキングタグ
- 命令列マスキング

それぞれの実現方法を次節以降で述べる．

4.2 命令マスキングタグ

命令マスキングタグとは通常の命令に演算精度の情報を示すタグである (図 8). *add2 r1,r2,r3* では下位の 2 ブロックを無視し演算を行う．同様に *add2 r1,r2,r3* で下位の 3 ブロックを除いて加算演算が行われる．ここでマスキング範囲が 0, 1, 2 および 4 ブロックを選択可能であり，命令エンコーディングにマスキングタグを保持するため 2 ビットが必要となる．

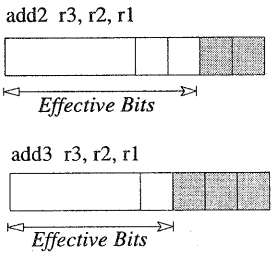

図 8: マスキングタグを持つ命令の例

4.3 命令列マスキング

命令マスキングタグを採用した場合，マスキング可能なブロック数が増えるとともにマスキングタグを保持するためのビット数が長くなり，命令長が長くなる恐れがある．一つの解決策としては，マスキング専用命令が考えられる．

マスキング命令発行後に続く演算に対して，プロセッサがマスキング解除命令が発行されるまで暗黙に演算マスキングを行う (図 9)．図 9 では，1 行目にプロセッサのデフォルト・マスキングを 2 ブロックに設定する．このマスキングのスコーピングは 5 行のマスク変更まで有効である．6 行目に

```
1:  mask 2          ;set default mask 2
2:  add  r3,r4,r5   ;implicit masking
3:  sub  r6,r3,r6
4:
5:  mask 3          ;set default mask 3
6:  add  r3,r2,r1
7:  unmask          ;release mask
```

図 9: マスキング命令の例

前マスクを解除し, プロセッサがマスクなしの通常の演算モードに戻る.

マスキング命令がプロセッサの状態を陰に変えることに注意して頂きたい. したがって, 例外が発生したときプロセッサの現在のマスク情報を退避しなければならない. マスク退避を行わない場合, プロセッサがもとの状態に復帰できない可能性がある[1].

4.4 ハードウェア実装について

ハードウェア・マスキングを行う回路を記憶素子以外のデータパスモジュールに接続することができる. この実装方法ではマスキング情報を利用することによってマスキング情報を利用した詳細な電力制御が可能となる.

5 おわりに

本論文では, システムの演算精度を変更することを考慮した設計手法について述べた. 演算精度を変更する際のシステム設計者の負担は, 必要最小限である必要がある. そのために演算精度の変更をある程度自動化するために, 予め用意しておくべき情報について考察した. また, 演算精度を削減することで下位ビットの処理を省略できることを利用した, プロセッサでの低消費電力化手法について考察した.

動作記述中の実数演算に固定小数点数を用いた場合は, 実数は整数型変数に格納される. そこで, 実数のダイナミックレンジを保持する整数型変数の幅, つまり有効ビット幅 [2] の解析結果と演算精度を削減する手法の組み合わせを用いた, データパス幅の最適化手法が考えられる. また, ハードウェアについては, 演算精度に応じた ASIC のデータパス幅最適化も考えられる. 今後の研究では, 上記の最適化手法も含めての, 出力品質を考慮したシステム設計手法について取り組む予定である.

[1] しかし, 有効ビット解析では「局所演算精度」を落とすことによって消費電力の改善を目的とするため. 復帰後のマスキング情報を失っても, データフローに支障を与えない.

謝辞

本研究は, 一部 STARC プロジェクト番号 : PJ-No.987「コアプロセッサベースシステム LSI の最適化設計技術に関する研究」の支援による.

参考文献

[1] *ftp://ftp.mpeg.org/pub/mpeg/mssg/.*

[2] H. Yamashita, H. Tomiyama, A. Inoue, F. N. Eko, T. Okuma, and H. Yasuura. "Variable Size Analysis for Datapath Width Optimization". In *Proc. of Fifth Asian Pacific Conf. on Hardware Description Languages*, pages 69–74, 1998.

[3] B.Shackleford, M.Yasuda, E. Okushi, H. Koizumi, H. Tomiyama, A. Inoue, and H. Yasuura. "Embedded System Cost Optimization via Data Path Width Adjustment". *IEICE Trans. Information and Systems*, E80–D(10):974–981, October 1997.

[4] エコー ファジャル, 井上 昭彦, 冨山 宏之, and 安浦 寛人. "ハードウェア/ソフトウェア・コデザインのためのソフトコア・プロセッサの検討". **電子情報通信学会研究報告**, *VLD96–13*, pages 85–92, 1996 年 6 月.

[5] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers, Inc., 2nd edition, 1996.

[6] 森 正武. "**数値計算**". 岩波書店, 1985.

[7] 石原 亨, 安浦 寛人, 岩井原 瑞穂. "低電力化のための Gated Clock によるデータパス幅可変アーキテクチャ". In *Proc. of* **第 *10* 回 回路とシステム軽井沢ワークショップ**, pages 247–252, 1997 年.