設 計 自 動 化 67－2
（1993．6．25）

ビット幅を考慮した
大規模システム処理系の設計手法について

雨坪 孝尚* 上田 穣 吉田 裕 白井 克彦
早稲田大学理工学部
東京都新宿区大久保 3-4-1

あらまし
大規模化・複雑化しているVLSIの設計においては、設計コスト・期間の増大が大きな問題となっている。この問題を解決するために、アルゴリズム記述からRTレベルの機能・構造を合成するシステムを構築した。このシステムは、アルゴリズムのシミュレーションを繰り返すことによって、より良い設計にしていくという、試行錯誤的な設計方法を効果的に支援している。本論文では、適切なビット幅をシミュレーションを行うことで決定する機構に焦点を当てて、システムの概要と適用実験の結果について述べる。実験では、実規模の音声符号化アルゴリズムであるLD-CELPのVLSI化を試みた。

Towards a Design Method for VLSI Considering Bit Width

Takanao Amatsubo,* Yutaka Ueda, Yutaka Yoshida,
and Katsuhiko Shirai
School of Science and Engineering, Waseda University
3-4-1 Ohkubo, Shinjuku-ku, Tokyo 169, Japan

Abstract
With the increase in scale and complexity of VLSI designs, designing cost and time have become important problems. In order to solve these problems, we have constructed a system with the ability to synthesize RT-level functional and structual information from algorithm description. This system uses trial-and-error to iteratively improve a design by repeated simulation and evaluation. In this paper, an overview of the system and experimental results in selecting the optimal bit-width by simulation is presented. The function selected for synthesis was a VLSI executing LD-CELP for a speech coding algorithm.

*現在 （株）東芝 研究開発センター
*TOSHIBA CORPORATION R&D Center

1 はじめに

VLSI製造技術の急速な発展は、その応用分野を著しく拡大してきており、種々の特定用途向け集積回路(Application Specific Integrated Circuit, ASIC)が開発されている。これに伴い、VLSI化が期待される回路の動作は、年々複雑化・多様化しており、その設計コスト・期間の増大が大きな問題となってきている。

このような背景から我々は、アルゴリズム記述からRTレベルの機能と構造とを合成するシステムの、構築に取り組んできた[1]。これは、プログラミング言語によって開発されたアルゴリズムの記述を、そのままシステムの入力として用いることができるようにすることにより、短期間で設計が行えるような環境を提供することを目的としたシステムである。このシステムの主な特徴は、設計対象となるVLSIのアルゴリズムのシミュレーションをある設定の下に行い、その結果を参考にして設定の変更をするなどの操作を繰り返して、より良い設計にしていくという試行錯誤的な設計方法を、効果的に支援できるようにした点にある。これまでに、アルゴリズムを解析して並列化が可能な部分を抽出し、それを並列化した場合の効果をシミュレーションで確認するという方法で、高速なVLSIの設計を支援する環境を実現した[1]。

このシステムをさらに実用的なものにするために、適切なビット幅をシミュレーションを行うことにより決定する機構を、システムに組み込んだ。本論文では、その機構を中心に本システムの概要を説明し、実規模レベルの信号処理アルゴリズムの設計を例にとって、本システムによるビット幅を考慮した設計手法について論じる。

2 システムの概要

本システムの構成を図1に示す。本システムは、設計対象となるVLSIのアルゴリズム記述が与えられると、まず中間表現に変換する。中間表現は、変数の宣言や3番地文の列などを含んでいる。解析系は、シミュレータと解析部とからなっている。シミュレータは、サンプルデータを用いてシミュレーションを行う。解析部は、並列化の可能性を探索するなどの解析を行う。合成系は、RTレベルの機能・構造を合成し、ハードウェア情報ファイルとして出力する。なお、このシステムはCMU Common Lisp 16e上で実現されている。

2.1 アルゴリズム記述言語

本システムでは、アルゴリズム記述言語としてPascalを採用している。例として、自然対数の底を計算するアルゴリズム記述を図2に示す。ここで、関数`castfloat`はデータ型を`float`型に変換するためのものである。

Pascalにおいては、データ型として`integer`や`float`などがあり、それぞれビット幅が決められているが、VLSI化する際は、各データ型をそのビット幅で実現すればよいというわけではない。効率の良いVLSIを設計するためには、演算精度などの観点から、より適したビット幅を検討し、最適なものを採用する必要がある。

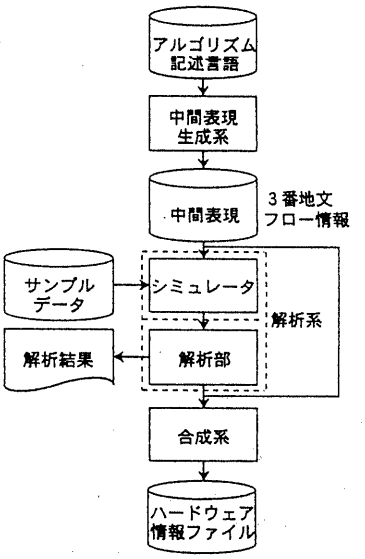


図1: システムの構成図

```
program base(oport);
  output oport: float;
  var n: integer;
      e,a,prev: float;
begin
  e:=0; a:=1; n:=1;
  repeat
    prev:=e;
    e:=e+a;
    a:=a/castfloat(n);
    n:=n+1
  until e=prev;
  oport:=e
end.
```

図2: アルゴリズム記述

最適なビット幅を探索するには、変数のビット幅の定義を様々に変えて、シミュレーションを行うという方法が考えられる。アルゴリズムの処理に必要なビット幅を、理論的に算出することは不可能ではないが、必要以上に大規模なものとなる可能性が大きい。しかし、音声処理の場合を例にとると、何を言っているのかが分かる程度でよいという条件下の通信に用いるCODECには、理論値ほどの高い精度は要求されない。この場合、どの程度の精度が適当かという判断は、人間の聴覚によるべきである。従って、シミュレーションによって最適なビット幅を探索することは、有効な手法であるといえる。

ビット幅を与える方法としては、アルゴリズム記述内の変数宣言においてビット幅を定義できるようにすることが考えられる。しかしこの方法では、異なるビット幅でシミュレーションするたびに、記述を変更しなければならない。本システムでは、アルゴリズム記述はPascalの形式の通りにしておき、

シミュレーション時に各データ型に対するビット幅を、別に指定するという方法をとることによって、柔軟にビット幅を変更してシミュレーションできるようにしている。ここで、特に指定されなければ、Pascal と同様のビット幅で演算を行う。例えば、integer 型の場合、指定しなければ固定小数点の 16bit で演算されるが、これを 12bit の固定小数点としたり、仮数部が 24bit で指数部が 8bit の浮動小数点としたりして演算させるような指定が可能である。

```
(PROGRAM BASE NIL (OPORT)
 (OUTPUT
  (OPORT FLOAT))
 (VAR
  (N INTEGER) (E FLOAT) (A FLOAT)
  (PREV FLOAT))
 (TEMPORARY
  (T0003 FLOAT))
 (PROCESS
  (B0001
   (1 (E := 0))
   (2 (A := 1))
   (3 (N := 1)))
  (B0002
   (4 (PREV := E))
   (5 (E := E + A))
                    :
```

図 3:　中間表現

2.2　中間表現生成系

中間表現生成系は、字句・構文解析部とフロー解析部からなっている。例として、図 2 を変換したものの一部を図 3 に示す。

字句・構文解析部は、アルゴリズム記述を、シミュレーションや並列化の可能性の探索などの解析に適した、単純な 3 番地文に展開する。

フロー解析部は、まず記述された制御構造に従って、3 番地文を基本ブロック単位に分割する。ここでの基本ブロックとは、連続した文の列のうち、先頭に与えられた制御が途中で分岐・停止することなく、末尾から離れるものを指す。次に、この基本ブロックを節とし、節間の制御の遷移を有向辺で表すことによって、フローグラフとして表現する。このフローグラフを用いて、次のような最適化処理を行う。

- ループ不変式の移動などによるループ最適化
- コンパイル時に一部の計算を行う定数の畳み込み
- 同じ計算を行う部分を 1 つにまとめる共通部分式の削除
- 不要な代入文の削除
- 無駄な演算を減らす算術恒等式の利用

最後に、基本ブロック間の大域的なデータフロー解析を行い、ライフタイム情報を得る。これは、基本ブロックにおける変数の生死、すなわち使用される可能性の有無を解析した情報である。

以上の処理により得られる中間表現は、次の情報をからなっている。

- 変数・入出力変数・手続き・関数の宣言
- 3 番地文の列からなる基本ブロックの集合
- 基本ブロックごとの変数のライフタイム情報

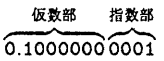
2.3　シミュレータ

シミュレータは、中間表現の 3 番地文の列と、各データ型へのビット幅の指定に基づき、サンプルデータを用いて演算を行う。

2.3.1　シミュレータ内部での数値の形式

浮動小数点型数値のシミュレータ内部での表現形式は、以下の通りである。仮数部は MSB とその 1 つ下位のビットとの間に小数点があるとし、指数部の底は 2 であるとして、いずれも 2 の補数で表すものとしている。これをシステム内部では、仮数部と指数部とを整数で表したリストとして扱う。

例えば、数値 1 を仮数部 8bit、指数部 4bit で表すとすると、



となる。これをシミュレータ内部では、(64 1) という 2 つの整数のリストとして扱っている。

一方、固定小数点型数値の表現形式は、小数点は LSB の下にあるとし、符号付き整数のみを表すものとしている。これを内部では通常の整数の形式として扱っている。

2.3.2　浮動小数点演算

浮動小数点演算は、前述の数値の形式を用いて整数演算で行われる。例えば、加減算の場合は、指数部を小さい方に合わせてから、仮数部について加減算を行い、正規化する。15+7 を、仮数部が 8bit 及び指数部が 4bit で演算する例を図 4 に示す。このようにして、各演算を整数演算によって行えば、VLSI 上で演算する場合と同程度の精度が得られる。

```
                          15
+                          7
                          ↓
        0.1111000  0100  (120 4)
+       0.1110000  0011  (112 3)
                          ↓
        1.1110000  0011  (240 3)
+       0.1110000  0011  (112 3)
                          ↓
       10.1100000  0011  (352 3)
                          ↓
        0.1011000  0101   (88 5)
                          ↓
                         22.0
```

図 4:　浮動小数点演算（加算）

2.3.3 頻度解析

シミュレーション時における、各命令及び各基本ブロックの実行頻度を、実行に要したコントロールステップ数を単位として解析し表示する。前者からは、与えられたアルゴリズムを実行するために必要なコントロールステップ数、VLSI内に用意すべき演算器の種類が分かる。後者からは、アルゴリズム内のどの部分が時間的に大きな割合を占めているかということが分かる。

2.4 解析部

解析部では、まず、各基本ブロックごとに命令をグラフ構造で表す。そして、与えられたアルゴリズムのより高速な実行を可能にするために、基本ブロック内における命令間のデータ依存性を求め、並列化の可能性を解析する。また、シミュレーションから得られた頻度解析の結果を基に、与えられたアルゴリズムにおいて頻繁に行われる命令群を解析し抽出する。

2.5 合成系

合成するVLSIの形式として、マイクロプログラム制御とハードワイヤードの2つを想定しており、どちらのハードウェア情報ファイルを出力させるかを選択できる。実際の製作は、その情報ファイルをハードウェア記述言語の形式に変換し、VLSIメーカーに委託する。

マイクロプログラム制御の場合は、まず、アルゴリズムの実行に必要な命令セットを生成する。次に、これに基づいて中間表現の3番地文の列からマイクロコードの列を生成する。そして、これらをハードウェア情報ファイルに格納する。

ハードワイヤードの場合は、命令のスケジューリングと、レジスタ・ポート・メモリ等のアロケーションを行い、結果をハードウェア情報ファイルに格納する。

3 適用実験

LD-CELP (Low-Delay Code Excited Linear Prediction) [2] という実規模レベルの信号処理アルゴリズムを持つASICの設計を例に、本システムを用いて行う、ビット幅を考慮した設計方法について述べる。

LD-CELPは、音声の高能率符号化アルゴリズムであり、移動電話などへの応用が期待されている。エンコーダは、8kHzでサンプリングされた16bitの音声データの5サンプルごとに、10bitのコードを1個ずつ生成することで16kbit/sの符号化をする。デコーダは、10bitのコードから乱数表を基にして音声波形を再生する。なお、本研究ではエンコーダの設計を取り扱ったが、デコーダは本質的にこの中の一部として含まれている。

エンコーダのアルゴリズムの記述には、固定小数点のデータ型として`integer`型、浮動小数点のデータ型として`float`型を用いた。アルゴリズム記述及び中間表現の記述量は次の通りとなった。

アルゴリズム記述	...	968行
中間表現の3番地文	...	2245行

3.1 シミュレーションの実行方法

アルゴリズム記述内の各データ型に対するビット幅を、様々に変更してシミュレーションを行い、適切なビット幅を試行錯誤的に探索する方法について述べる。

まず、浮動小数点型の仮数部のビット幅のみを与えて、シミュレーションを行う。ここでは、浮動小数点型の指数部及び固定小数点型には、無限のビット幅が与えられているものと解釈するモードで、シミュレーションを実行する。このため、データ演算の過程で、オーバーフローとアンダーフローは発生しない。設計者は、シミュレーションの結果を解析し、不満であれば仮数部のビット幅の指定を変更して再度のシミュレーションを行う。このような操作を繰り返すことにより、適切な浮動小数点型の仮数部のビット幅を見つけ出す。なお、シミュレーション終了時には、浮動小数点型の変数の指数部がとった最大値と最小値、及び固定小数点型の変数がとった最大値と最小値が表示される。この情報を用いてすべてのビット幅を指定し、再度シミュレーションを行い、オーバーフロー・アンダーフローが起こらないことを確認する。

本適用実験においては、`float`型の仮数部のビット幅が、8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32の各場合についてシミュレーションを行った。

3.2 シミュレーション結果

エンコーダのアルゴリズムのシミュレーションを行うと、符号化されたコードが出力される。そのコードが有する音声情報の質を設計者が評価できるようにするには、コードを音声に復号化する必要がある。エンコーダは、符号化したデータを復号化して、次のデータの符号化に用いる構造となっているので、内部にデコーダを含んでいる。この部分を用いてデータを音声に復号化し、評価することによってVLSIのビット幅の決定を試みた。

LD-CELPのように音声信号の処理を行うアルゴリズムの場合、評価の基準は人間の聴覚とすることが望ましい。様々なビット幅の設定によるシミュレーションの結果として示される、それぞれの音声を人間が聴き比べることによって、どの設定が適切かを判断することになる。LD-CELPは先述の通り、移動電話などへの応用が期待されるものである。従って、何を言っているのかが分かる程度の音質であれば、可能な限り小規模なハードウェア量である方がよいとする考え方で評価することとする。

本適用実験では「さ」と発声された0.21秒間の音声データを用いた。符号化・復号化を経た音声の質を、設計者が聴覚によって評価するには、さらに長いデータに対するシミュレーションが必要となるが、0.21秒間の音声データでも誤差に対する評価は可能であることと、今回の予備的な実験では時間的な制約から適当と判断した。なお、8bitの場合と10bitの場合は、途中で0による整数の除算を行う状態に陥り、シミュレーションが中断した。

また、LD-CELPをC言語の倍精度で記述し、同じ音声データについてC言語による計算も行った。C言語の倍精度浮動小数点演算は、仮数部53bit・

指数部 11bit であり、これによる LD-CELP の計算は、極めて高い精度であると考えられるため、この結果を音声の質及び演算精度を比較する基準とした。

シミュレーション結果の評価としては、C 言語による結果と 12bit から 32bit までの各ビット幅での結果とを聴き比べた。C 言語による結果に対する各ビット幅での結果の音声の劣化は、ほとんど認められなかった。このことは次に示すように、各結果と原音声のケプストラム距離の誤差の検討からも理解される。以上のことから、計算の中断しない最小のビット幅である 12bit の周辺が、LD-CELP エンコーダのビット幅としては適切と考えられる。

表 1:　ケプストラム距離

仮数部のビット幅	比較対象	
	原音声	C 倍精度
8	—	—
10	—	—
12	0.713	0.415
14	0.668	0.402
16	0.697	0.361
18	0.661	0.367
20	0.771	0.364
22	0.700	0.358
24	0.679	0.296
26	0.727	0.200
28	0.726	0.200
30	0.749	0.111
32	0.841	0.151
C 倍精度	0.798	—

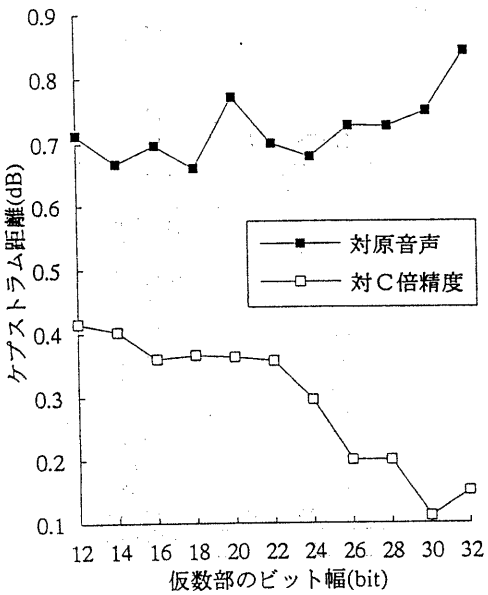


図 5:　ケプストラム距離

各結果間の違いは認識できなかったが、これについてはケプストラム距離を用いて、さらに詳しく解析する。ケプストラム距離は 2 つの音声波形の違いの度合を表すものであり、違いが大きいほど距離は大きくなる。単位は dB である。各結果と原音声及び C 言語の倍精度による結果とのケプストラム距離を表 1、図 5 に示す。

図 5 において、対原音声の距離はビット幅に伴う特徴的な変化が見られず、ほぼ横ばいの推移を示していることが分かる。これは、ビット幅を大きくしても距離が小さくならない、即ち得られる音声の質が向上しないということを意味する。このことより、聴覚による判断において、互いの違いが認められなかったことの正当性がある程度確認できる。ビット幅を大きくしても結果の音声の質が向上しない原因としては、LD-CELP が再現する音声の質の高さを必ずしも追求しているものではなく、質については限界があるアルゴリズムであるということも考えられる。

対 C 言語のグラフは、ビット幅を大きくするにつれて距離が縮まる傾向にある。従って、仮数部のビット幅を拡大すれば、LD-CELP のアルゴリズムに対する演算精度は、確かに向上していることが分かる。それにもかかわらず、原音声とのケプストラム距離が減少しないということは、アルゴリズムの詳細を解析しなくては正確にはいえないが、アルゴリズム自体の問題と言えよう。

以上の検討結果より、LD-CELP エンコーダの設計においては、仮数部のビット幅は 12bit 程度でよいということが判断できる。また、もし仮にアルゴリズムに対する演算精度という観点から仮数部のビット幅を決定するのであれば、グラフが平坦な推移を見せている部分においては、小さいビット幅を選択する方が効率が良いということがいえる。例えば、16bit から 22bit の辺りで選択するのであれば 16bit を、26bit と 28bit とでは 26bit を、それぞれ選択すべきであるといえる。

表 2:　ビット幅解析

float 型			integer 型	
仮数部	指数部			
bit 幅	変動幅	bit 幅	変動幅	bit 幅
12	-23～38	7	-8440～12064	15
14	-26～36	7	-9496～11360	15
16	-29～34	7	-9320～11280	15
18	-31～32	7	-8824～11352	15
20	-25～32	7	-8394～11392	15
22	-32～34	7	-8456～10272	15
24	-29～32	7	-8536～10448	15
26	-26～34	7	-8416～10928	15
28	-29～34	7	-8416～10928	15
30	-25～37	7	-8408～10920	15
32	-27～36	7	-8408～10920	15

各場合の指数部及び固定小数点型データのビット幅に関する解析の結果を表2に示す。これは、仮数部のビット幅の各設定における、指数部及びinteger型の変数の取り得る値の変動幅と、それに耐え得るビット幅を示したものである。これにより、仮数部のビット幅がいかなる場合であっても、その指数部には7bit、integer型には15bitを確保すれば十分であることが分かる。

以上、本システムを用いたLD-CELPエンコーダのビット幅の決定について述べ、システムの有効性を示した。本設計手法では、ビット幅の決定が終了すると、続いて実現方法の具体化を進めていく。この工程では、表3,4に示す頻度解析結果を用いる。表3の合計の欄はアルゴリズムの実行に要するステップ数を示しており、今回の0.21秒間のデータの処理に、これだけのステップ数が必要であることが分かる。このため、実時間で実現させるためには、マイクロプログラム制御の場合、約61MHzのクロック速度を要することになる。ただし、これは乗除算等も1サイクルで実行可能と仮定した時の動作周波数であり、複数サイクルを要する演算を含んでいれば、これを加味してステップ数に重み付けを行い、動作周波数を求めて実現可能性を検討し、可能であればマイクロプログラム制御の合成系を起動すればよい。また、もしマイクロプログラム制御での実現が困難であれば、演算間の並列性を解析し、ハードワイヤードでの実現（合成）を検討する。

4 おわりに

設計対象となるVLSIのアルゴリズム記述を入力に用いて、シミュレーションと評価の繰り返しによる試行錯誤を中心に設計を行う、設計支援システムを構築した。本システムではシミュレーションによって、VLSI内で使用するデータや演算器のビット幅を、決定するための情報を得ることができる。このシステムを用いて、実規模の信号処理アルゴリズムであるLD-CELPのアルゴリズムのVLSI化を試みた。

適用実験では、アルゴリズム記述内の変数のビット幅を柔軟に変化させたシミュレーションを行い、それぞれの場合の結果を得た。これらを基に、人間の聴覚による評価及びケプストラム距離を用いた定量的な評価を行い、最適なビット幅を検討することが可能であることとその有効性が確認できた。今後の課題としては、高速なシミュレータを開発し、より多くのデータを短期間で評価できるような環境を構築することが挙げられる。なお、今回はその設計例を示していないが、本設計システムではこの段階の後に、頻度解析結果に基づく高機能命令の導入[1]、及びパイプライン設計が可能[3]で、当然その場合は本結果より大幅に変化する。

参考文献

[1] H. Kitabatake, K. Shirai: "Functional Design of a Special Purpose Processor Based on High Level Specification Description", IEICE Trans. Fundamentals, Vol. E75-A, No. 10, October 1992, pp. 1182-1190.

[2] J. H. Chen, M. J. Melchner, R. V. Cox, D. O. Bowker: "Real-time implementation of a 16kb/s low-delay CELP speech coder", ICASSP-90, pp. 181-184 (1990).

[3] 吉田裕, 雨坪孝尚, 白井克彦: 「専用プロセッサ設計支援システム (SYARDS) におけるパイプライン処理の考慮」, 情報処理学会第46回全国大会, 9M-3 (1993).

表3: 頻度解析（命令種別）

命令型	ステップ
A = B	321631
A = B op C	5098439
A = op B	18683
A = B[C]	3217192
A[B] = C	970733
GOTO X	1527508
IF GOTO X	1714124
CALL A	10991
合計	12879301

表4: 頻度解析（演算種別）

演算子	ステップ
boolean型	90570
AND	(90570)
integer型	4037536
+	(1317549)
-	(1128219)
*	(56112)
DIV	(7125)
=	(1465726)
>	(420)
>=	(54096)
<=	(1680)
CASTFLOAT	(6609)
float型	2703140
+	(914241)
-	(171696)
-（単項）	(8378)
*	(1315659)
/	(6362)
INT	(1680)
LOG	(336)
POWER	(336)
>	(116619)
<	(74114)
<=	(91706)
<>	(333)
CASTINTEGER	(1680)
合計	6831246