社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2004－ＳＬＤＭ－115　（６）
2004／5／27

〔チュートリアル講演〕

システムレベル言語による設計

小林　和淑†

† 京都大学大学院情報学研究科通信情報システム専攻
E-mail: †kobayasi@kuee.kyoto-u.ac.jp

あらまし　本稿では, LSI 設計者向けに, システムレベル設計の概説を行なう. まず計算機の開発の歴史からなぜ C 言語がアプリケーションの開発に用いられているかを説明する. 次に, LSI や DA の分野におけるシステムレベル設計の定義とその必要性を議論する. システムレベル設計の大きな目的のひとつは, 動作レベルからの合成である. これまで用いられてきた RT レベルの記述と動作レベルの相違を説明し, C 言語によるマージソータを RTL に変換する例を示す. さらに, 動作レベル合成による効果を述べ, 最後にシステム LSI 設計言語の開発動向を説明する.

キーワード　システムレベル言語, 動作合成, C 言語, LSI 設計, RTL, SystemC, SpecC

LSI Design Using System-level Languages

Kazutoshi KOBAYASHI†

† Dept. of Comm. and Comp. Eng., Graduate School of Informatics, Kyoto University
E-mail: †kobayasi@kuee.kyoto-u.ac.jp

Abstract　This paper introduces system-level design methodologies for LSI designers. First, we explain the history of computer development and the reason why the C language is widely used to describe applications and OSs on computers. Then we discuss the definition and necessity of the system-level design in the area of LSIs and design automation. One of the most emphasized features of the system-level design is to synthesize circuits from behavioral levels. We explain the differences between RTL and behavioral descriptions and the effectiveness of the behavioral synthesis. Finally, several existing system-level languages are introduced briefly.

Key words　System-level Languages, Behavioral Synthesis, C Language, LSI Design, RTL, SystemC, SpecC

1. はじめに

近年の LSI の微細化, 高集積化により, 一つの LSI に大規模なシステムが搭載できるようになってきた. これまで単なる部品として扱われてきた LSI が, システムそのものを構成することになる. 日本の電機業界は, DRAM ショック, アジア各国の台頭により, 各社とも赤字を計上するなど, 非常に苦しい状態にあった. しかし, 自社で開発した自社のためのシステム LSI を搭載したディジタル家電により, 他社との差別化を図ることができ, 業績を回復させる大きな原動力となっている.

LSI の大規模化につれて, 設計手法も高度化を図ってきたが, そのギャップは以前として大きく, 一つの LSI を設計するのに, 数百人月を要することもある. 関係する設計者が増えれば増えるほど, システム全体に大きなバグが潜む確率は高くなる. 現実に, 携帯電話の出荷後の回収という事態が頻繁に起こっている. 設計者の数を減らし, 効率的に SoC を開発するためには, 上位のレベルでの設計が不可欠であり, その要求の高まりとともに, システムレベル設計手法の提案がいくつか行なわれている.

本稿では, システムレベル言語による LSI 設計について, 歴史から現状までを包括的に概説する.

2. 計算機と開発言語の歴史

LSI 設計の分野でのシステムレベル設計は, C 言語ベースのものがほとんどである. まず, 簡単に計算機と開発言語の歴史について簡単に触れる.

INTEL 社の最初の CPU である 4004 が電卓用に開発されたように, 電子計算機 (以下コンピュータ) は, まさに計算 (Compute) するためのものであった. 半導体メモリが一般化するまでのコンピュータでは, 計算するためのプログラムの記憶容量をできるだけ減らすために, CISC(Complex Instruction Set Computer) 型のマイクロプログラム実行方式がとられた. プログラム開発には, Basic 等のインタプリタ言語を用いていた. 実行時にまずプログラムをインタプリタにより解析し, さらに解析したマイクロプログラムをプロセッサ内部で複雑に解釈して実行する. 1 命令は 1 クロックで行なわれるとは限らず, 複数のクロックにまたがって実行される場合もある.

初期のコンピュータと現在のコンピュータとの大きな違いは，乗算器にたとえることができる．乗算結果である積は積項にくらべて大きな数になる．乗算結果を蓄えるよりは積項を乗算器を使ってその都度演算したほうがメモリの容量は少なくすむ．ただし，乗算器は遅延が大きくその分演算時間がかかる (図 1)．なおここでは，大きな数ほどより多くのメモリがいると仮定している．図 2 に初期のコンピュータ，図 3 に現在のコンピュータの概念図を示す．現在のコンピュータは OS が走り，計算以外に様々なアプリケーションを走らせることができる．プロセッサは，大きなレジスタファイルを前提としたパイプライン方式の RISC(Reduced-Set Instruction Computer) 型が用いられている．RISC 型の CPU では，命令は命令フェッチ，命令デコード，演算実行などからなるパイプライン段で順に実行される．なお，Intel 社の x86 系のプロセッサ (Pentium 等) は，命令レベルでは CISC 型のプロセッサであるが，内部では CISC 命令を RISC 命令に置き換えており，ハードウエア的には RISC 型のプロセッサである．Transmeta 社の Cruesoe や，Effinion はソフトウエアで x86 の CISC 型命令を VLIW コードに動的に置き換えている．

コンピュータ上の「アプリケーション」を書くために開発されたのが C 言語である．C 言語は元々UNIX を記述するために作った B 言語を発展させて作成された．OS を書くために，メモリや IO 空間といった計算機の資源に簡単にアクセスできることが大きな特徴である．

現在，C 言語は OS からアプリケーション開発まで幅広く使われており，C 言語のコンパイラを持たないプロセッサは事実上ないといって良いほどである．一昔前は，数値計算は Fortran，事務処理系は Cobol といった棲み分けがされていた．現在は，大学の授業/演習のレベルでも，C 言語が事実上標準的なプログラム言語として教えられている．

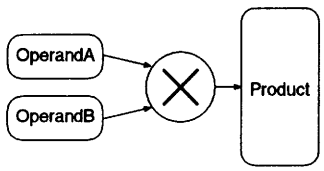

図 1　乗算器: 乗算すると数は大きくなる．乗算結果を格納するよりメモリは少なくて済むが乗算に時間がかかる．

3. システムレベル設計

システムとは何であろうか? 広辞苑によると，「複数の要素が有機的に関係し合い，全体としてまとまった機能を発揮している要素の集合体．組織．系統．」[1] と定義してある．また，英和辞典によると，“ギリシャ語「結合したもの」の意から” とある．LSI の世界での「複数の要素」とは，ソフトウエアとハードウエアを指すことが多い．つまり，「ハードウエアとソフトウエアが有機的に関係し合い，全体としてまとまった機能を発揮しているもの」ということになる．

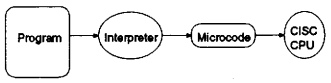

図 2　初期のコンピュータ: できるだけメモリを使わないように実装されていた．

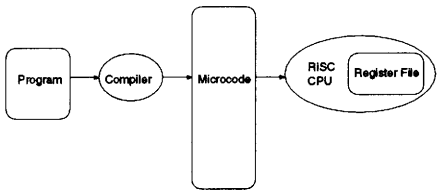

図 3　現在のコンピュータ: メモリをふんだんに使える．CPU 内部にもキャッシュや大きなレジスタファイルといった記憶素子を持つ．

表 1　旧三種の神器と新三種の神器の比較 (2011 年の普及率は予想)

旧三種の神器

		アナログテレビ	冷蔵庫	洗濯機
トランジスタ数		<10	~0	~0
普及率	1959 年	7.8%	20.2%	2.8%
	1965 年	95.0%	78.1%	68.7%

新三種の神器

		ディジタルテレビ	DVD レコーダ	デジカメ
トランジスタ数		>100M	>100M	>10M
普及率	2004 年	?%	10%	32%
	2011 年	99.0%	90%	90%

3.1 システムレベル設計の必要性

ご存じの通り，1958 年のキルビーによるトランジスタの発明より，システムに集積されるトランジスタの数は，ムーアの法則により飛躍的に伸びている．日本が得意な，家電製品に搭載されるトランジスタの数もその例外ではない．現在の日本では，バブル崩壊から IT バブル，IT バブル崩壊による IT 不況を経て，ようやく新三種の神器に代表される「ディジタル家電」の普及により電機業界が息をふきかえしつつある．

高度成長期の三種の神器と現在の新三種の神器の比較を表 1 に示す．初期のテレビは，真空管を用いていたがトランジスタテレビでもチューナー部に使用しているトランジスタ数はたかが数個である．旧三種の神器では，家電製品の中核は機械部品であり，故障のほとんどは，機械部品に集中していた．しかし新三種の神器は，すべてディジタル家電である．機械的な部分はあるものの，旧来のアナログ家電の流用であり，故障や不具合はほとんど電気的/システム的な部分に集中する．システムの複雑化と製品寿命の短期化に伴って，製品出荷時にすべての機能を検査することは不可能になってきている．携帯電話の回収騒ぎに代表されるように，製品出荷後に不具合が顕在化する例が頻発している．筆者の経験からも，この数年で購入したディジタル機器のいくつかを，初期不良による交換や修理に出している．

このような，システムの不具合を無くすために，LSI 設計者が行なわなければならないことは，システムの複雑化に対応して設計の抽象度をあげることに他ならない．LSI で実現するシステムの一部を，ソフトウエアとハードウエアに最適に分割し，短

期間にシステムを設計することが重要である.

3.2 システムレベルの定義

「システムレベル」という言葉は, LSI 屋 (ハードウエア) さんと, システム屋 (ソフトウエア) さんとで大きく意味が異なる. LSI を単なる部品として扱い, 組み込みシステムを作るシステム屋さんに言わせると

> **[システム屋] C 言語によるモデルは最終の成果物である.**

という立場で, モデリングは UML(Unified Modeling Language) 等のもっと上位のレベルで行なうのが普通である. これに対して, LSI 屋さんは,

> **[LSI 屋] C 言語によるモデルはシステムレベル設計の第一歩である.**

という立場をとる. LSI 屋にとっては, 「動く」ことが命であり, 計算機上で「動く」C 言語による実装を出発点と考えるのである.

4. RT レベルと動作レベル

4.1 RT レベル

RT レベル (RTL(注1)) とは, Register-Transfer Level の略である. 論理合成ツールで正しく取り扱えるのは, 単相クロックの同期回路である場合がほとんどである. 単相クロックの同期回路の RTL 記述は, Register を記憶素子として, その振る舞いを 1 クロックで動作が完了する無限ループ内に記述する.

RTL 記述では, 一つのループ内に 1 クロックで完了する動作しか記述できない. 状態遷移図を RTL に変換するさい, 状態内での動作が 1 クロックですべて完了する場合には RTL の記述は非常に簡単なものになる. しかし現実はそうともいかない. クロック毎に状態を分けることもできるが, 設計者が把握できないほど状態数を多くすることは不可能である. 各状態内にさらに状態遷移機械を作るか, 状態内のクロック数が決まっている場合は, カウンタを導入するなどしなければならない. 処理に手続き的なものが多い場合は, RTL で記述するのは非常に大変である.

RTL で書きやすい記述として, パイプライン構造のハードウエアがある. パイプラインの各段では, 1 クロックによる処理が前提となっており, それらが並列に動作する. 現在の計算機がとっているパイプライン型 RISC CPU は, RTL での記述が非常に容易なハードウエアの一つといえる. 実際, ARM や SH といった合成可能な RTL 記述の組み込み型 CPU が存在する. RTL で記述しやすい CPU の上で, プログラマーが記述しやすい手続き型言語で書かれたアプリケーションを走らせるのは実は, 非常に理のかなったことである.

4.2 動作レベル

動作レベル (Behavioral Level) は, 文字通り, 動作を記述する. RTL もある意味動作記述であるが, 1 クロック毎の記述となるために, 通常の人間の思考パターンと掛け離れたものとなる. LSI の世界で言う動作レベル記述とは, ハードウエアを設計者のわかりやすいように手続き的に書き下すことである. ただし, 完全に手続き的に書いてしまうと, ハードウエア化による大きなメリットである「並列化」が十分に行なえない. 動作レベル記述には必ず明示的に並列性を記述するための機能が備わっている.

4.3 動作レベルから RTL への変換例

ここで, C 言語による動作レベル記述を HDL による RTL に変換する例を示す. 例としてはソートを取り上げる. ソフトウエアアルゴリズムとしては, Quick Sort アルゴリズムが非常に有名であるが, これは, プロセッサによる逐次処理を前提としており, そのままハードウエア化しても, 計算オーダーは変わらない. ここでは, ハードウエア化に適したアルゴリズムとして, マージソートを取り上げる [2]. マージソートは, $2n$ 個の数を, まず 1 つずつに区切り, 2 組のデータのソートから始める. ソートされた長さ 2 のデータの組を先頭からソートして (マージ作業), 長さ 4 のソートされたデータを作る. これを繰り返して $2n$ 個のデータになるまで繰り返す. C 言語の記述としては, [3] の記述を用いた. これは, 再帰的呼び出しを用いてマージソートを実現している. 再帰的呼び出しは, RTL や動作レベルでの LSI 設計に用いることは通常できない.

逐次的に行なう処理をそのままハードウエアにしても効果が少ないので, マージソートの各段をパイプライン的に行なうパイプラインマージソータを RTL で実現する. 図 4 に, 入力ポートから毎クロックごとにひとつずつやってくる 8 個の連続したデータをソートするパイプラインマージソータのブロック図の一例を示す. 左の Step0 では, 長さ 1 の二つのデータのソートを行なう. Step1 では, Step0 でソートされた長さ 2 のデータのソートを行ない長さ 4 のデータを作成する. 図中に (4, 1, 5, 2) という 4 つのデータが順に入力されたときに, 各 Step でどのようにソートが行なわれるかを示している. Verilog-HDL で記述したパイプラインマージソータの設計例を図 5, 6 に示す. ここでは, parameter を使うことにより, 各 Step の記述を一つにまとめている. 各 Step あたり比較器は一つで済み, データ 1 個あたり 1 クロックで処理を行なうことができる. 各 Step のデータの右端には ∞ を置いている. 現在比較しているデータ列の値が ∞ となると, かならずもう一方のデータ列の値が選ばれる. こうすることにより, RTL で記述しにくい条件判断による分岐を減らしている.

この RTL 記述には, 状態を使わずにパイプライン的に処理を行なっている. 状態を導入すると状態間の遷移のために, 記述がかなり複雑になる. なお, 図 5 の設計期間は約 0.5 人日であった.

4.4 動作レベル合成の効果

RTL からの合成では, 1 行あたり約 4.2 ゲートの回路を生成することができ, Cyber [4] で用いられている BDL で記述した動作レベルから合成では, 1 行あたり約 31.3 ゲートの回路を生成することができる. 一人の人間が管理できるコード量は約 10 万行と言われているので, RTL では一人当たり 42 万ゲートま

(注1) : よく, 「RTL レベル」という記述を見掛けるが, 冗長である.

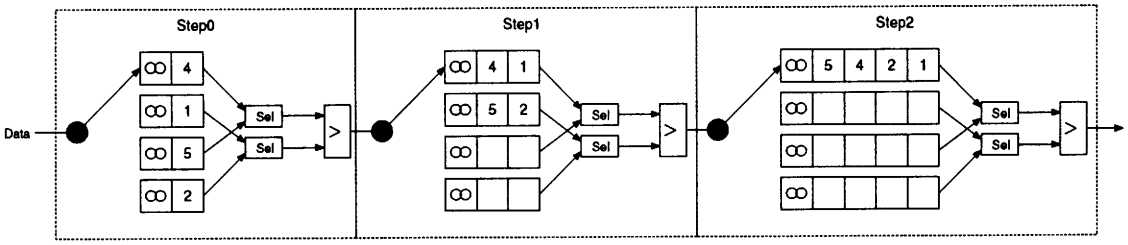

図 4　パイプラインマージソータのブロック図

での回路しか作成することができない．これに対し，動作合成では最大 313 万ゲートの回路まで作成することができる．今後ますます進む回路の大規模化において，動作合成を採用しなければ，いずれ設計者の数が膨大になり開発が不可能な状況に追い込まれる．

1 行あたりに生成されるゲート数が多いということは，記述の質により生成される回路が大きく変わるということを意味する．記述 A と記述 B から生成される回路の差が，RTL では最大 4 倍だったものが，動作レベルでは最大 28 倍にまで広がることになる．その差が誤差になるほどトランジスタをふんだんに使えるようになるまでは，RTL 以上に動作レベルでは記述に注意を払う必要がある．

また，4.3 節で設計を行なったソータは，あらかじめハードウエア化に適したアルゴリズムを選び，それを RTL 化した．RTL で設計した場合，その構造がそのままハード上になるために，論理合成により得られる回路の構造を大きく変えることはできない．動作合成では，合成時の制約や，ちょっとした記述の変更で回路構造 (アーキテクチャ) を大きく変えることができる．図 7 と図 8 に論理合成と動作合成で得られる回路の面積/遅延時間のトレードオフを表したグラフを示す．動作合成では，同じ記述で制約を変えることで様々な構造の回路を生成することができる．図 10 に，16 タップの FIR フィルタ回路を Verilog-HDL の RTL と，SystemC の動作レベル (図 9) で記述し，合成した場合の面積と遅延時間 (レイテンシ) のトレードオフを示す．動作合成では，同じ記述からレイテンシが 1 から 48 程度までの幅広い回路が合成できているのがわかる．一方，RTL からの論理合成では，ある程度のトレードオフは取れるが，そのたびに記述を変更しなければならず，設計者の負担は大きい．

5. LSI 設計用システム/動作レベル言語の開発動向

各種の C 言語ベースのシステムレベルおよび動作レベルの設計言語について概説を行なう．

5.1　C 言語のシステム記述/動作合成用言語への拡張

初期の計算機は，与えられたマイクロコードを順に実行するだけで，並列に演算するという機能を持たなかったために，C 言語には，「並列実行」を陽に記述する文法を持たない．現在のプロセッサは，並列実行機能を当たり前のように備えるために，コンパイラで並列性を抽出する VLIW 型のプロセッサや，実行時に並列性を抽出するスーパスカラ型のプロセッサにより並列実行を可能にしている．もちろん C 言語でもスレッドレベルのプログラミングを行なうことにより，マルチプロセッサ環境下での並列実行が可能であるが，あまり一般的とはなっていない．したがって，C 言語をシステム/動作レベルの言語に拡張するためには，「陽に並列性を記述する」ことが必要となる．

プロセッサはバイトと呼ばれる 8 ビットを単位としてデータを管理するために，8 , 16, 32, 64 ビットの変数しか定義できない．専用ハードウエアを作る場合，変数に応じて必要なビット幅を最適化するのが一般的である．したがって変数のビット幅を陽に記述するための拡張が必要である．

その他，ハードウエア間で同期的にデータをやり取りするための拡張等もハードウエアによる処理の高速化にには必須である．

5.2　各種システム記述/動作合成用言語の概要

LSI 向けのシステム記述/動作合成用言語の代表例を表 2 にあげる．

Cyber はプロセッサとその上のプログラム，BDL による動作レベル記述によるハードウエア等からなるシステムを ClassMate シミュレータにより上位のレベルで高速にシミュレーションを行なうことができる．動作レベルの高位の記述から合成条件や記述そのものを変更することで，様々なアーキテクチャの RTL を得ることができるために，これまでの RTL 設計では考えもしなかったようなアーキテクチャを探索することができる．これにより RTL より優れた回路設計が可能である．また，手設計による RTL では人間が把握できないほどに状態の数を多くすることは不可能であるが，動作合成からは状態数に制限のない回路を作成できることも有利な点である [4].

Bach C [5] と Handel-C [6] はともにヨーロッパの作曲家の名前を冠しているが，その出発点を英国の大学で行なわれていた Occam [7] という並列記述用のプログラミング言語に由来する．Bach C は，par 構文による粒度の粗い並列性の記述を陽に記述することができるのが大きな特徴である．ANSI C 言語への自動変換により高速にシミュレーションが行なえる他，変数をメモリやレジスタのどちらにマッピングするかなどを指定することができる．Cyber とは異なり，現状ではハードウエアのモデル化のみをサポートしている．ただし Mentor Graphics 社が同社の Seamless CVE に Bach C を組み込んでおり，これによりプロセッサやその他の要素を含んだシステムレベルでのシミュレーションが可能になる．

SystemC は，C++のクラスライブラリとして実装されてお

```
module merge(in,out,CLK,RST);
   parameter          l=0;  Step
   parameter     w=1;  2**l
   parameter     limit=(w+1)*4;parameter     maskadrwidth=l+3;
   parameter          adrb=3;
   input [7:0] in;
   input       CLK,RST;
   output [7:0] out;
   reg [7:0]          buffer[0:limit-1], outbuf;
   reg [2:0]          mc;reg [1:0]          sc,lc,rc;
   wire [15:0]          lefti,righti;
   wire [maskadrwidth-1:0] maskadr,tmp;
   integer          i,j;
   initial
     begin
        for(i=0;i<4*(w+1);i=i+1)
          buffer[i]=0;
        buffer[(w+1)*1-1]='hff;buffer[(w+1)*2-1]='hff;
        buffer[(w+1)*3-1]='hff;buffer[(w+1)*4-1]='hff;
     end
   always @(posedge CLK or negedge RST)
     begin
        if(!RST)
          begin
             mc<=0;sc<=0;
          end
        else
          begin
             buffer[maskadr]<=in;
             if(sc==(w-1))
               begin
                  mc<=mc+1;sc<=0;
               end
             else
                  sc<=sc+1;
          end
     end
   always @(posedge CLK or negedge RST)
     begin
        if(!RST)
          begin
             lc<=0;rc<=0;outbuf<=0;
          end
        else
          begin
             if(buffer[lefti]<buffer[righti])
               begin
                  outbuf<=buffer[lefti];lc<=lc+1;
               end
             else
               begin
                  outbuf<=buffer[righti];rc<=rc+1;
               end
             if((lc+rc)==(w*2-1))
               begin
                  lc<=0;rc<=0;
               end
          end
     end
   assign          out=outbuf;
   assign          lefti=(((mc>>1)+1)&'b1)*(w+1)*2+lc;
   assign          righti=(((mc>>1)+1)&'b1)*(w+1)*2+w+1+rc;
   assign          tmp=(mc*(w+1)+sc);
   assign        maskadr=(tmp<limit)?tmp:tmp-limit;
endmodule
```

図 5　パイプラインマージソータの Verilog HDL による RTL 設計例 (各 Step の共通記述, ただしこのままでは合成できない).

り, Synopsys 社と Coware 社が中心となって仕様策定を行なった言語である. 現在は OSCI(注2) に標準化の権利は移され, web サイトより SystemC の標準シミュレータ用の C++クラスライブラリがダウンロード出来るようになっている. SystemC でモデル化してその動作を確かめるだけなら, 無償で行なうことができる. 標準シミュレータはオーバヘッドが大きく, 並列度の高い (C 言語レベルでは, スレッドが多い) ハードウエアをモデル化すると極端に動作が遅くなってしまう. SystemC をサポートした動作合成ツールは Synopsys 社をはじめとして, 各社から出ている. ただし今のところ SystemC の一部の構文しか合

(注2) : Open SystemC Initiative: http://www.systemc.org

```
module mergemain(in,out,CLK,RST);
   input [7:0] in;
   input       CLK,RST;
   output [7:0] out;
   wire [7:0]          l011,l112;
   defparam          I0.l=0;defparam          I0.w=1;
   defparam          I1.l=1;defparam          I1.w=2;
   defparam          I2.l=2;defparam          I2.w=4;
   merge I0(in,l011,CLK,RST);
   merge I1(l011,l112,CLK,RST);
   merge I2(l112,out,CLK,RST);
endmodule
```

図 6　パイプラインマージソータの Verilog HDL による設計例 (3 段分の記述).

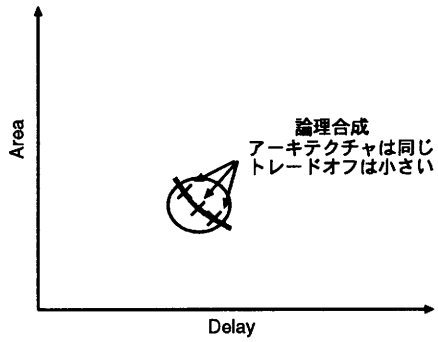

図 7　単一の RTL 記述から生成できる回路の面積, 遅延のトレードオフ

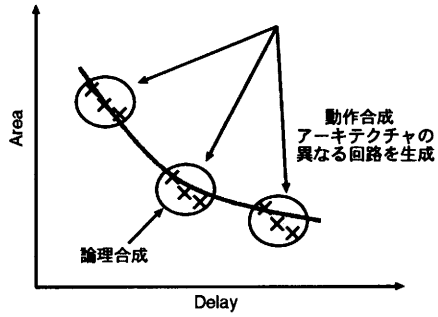

図 8　単一の動作レベル記述から生成できる回路の面積, 遅延のトレードオフ

成することはできない. OSCI では, SystemC の合成用のサブセットの標準化を行なっているところである.

SpecC は, UC Irvine の Gajski 教授らのグループが中心となって提唱している仕様記述言語である. 標準のコンパイラは UCI より無償でダウンロードすることが可能である. ただしモデル化言語としては優れているが, 動作合成をサポートしている商用ツールがほとんどないのが現状である.

5.3　LSI 向けシステムレベル設計の課題

先に紹介した通り, LSI 向けのシステムレベル設計は, Cyber や Bach C に代表される日本で先行的に行なわれている. ただし, RTL 設計が一般化し始めた 10 年前と比べてその普及の速度は非常に遅い. その理由としては, 「RTL 合成における Design Compiler のような標準かつ絶対的な商用の合成ツールが登場していない.」ことがあげられる. Synopsys 社が Design Compiler を引っ提げて創業したように, Cyber や Bach を別会

```
#include <systemc.h>
#include "fir.h"
void fir::entry() {
  sc_int<8>  sample_tmp;  sc_int<8> shift[16];
  sc_int<17> pro; sc_int<19> acc;
  for (int i=0; i<=15; i++)
      shift[i] = 0;
  result.write(0);
  output_data_ready.write(false);
  wait();
  while(1) {
    output_data_ready.write(false);
    wait_until(input_valid.delayed() == true);
    sample_tmp = sample.read();
    acc = sample_tmp*coefs[0];
    for(int i=14; i>=0; i--) {
      pro = shift[i]*coefs[i+1];
      acc += pro;
    };
    for(int i=14; i>=0; i--) {
      shift[i+1] = shift[i];
    };
    shift[0] = sample_tmp;
    result.write((int)acc);
    output_data_ready.write(true);
    wait();
  };
}
```

図 9 FIR 回路の SystemC での動作レベル記述例

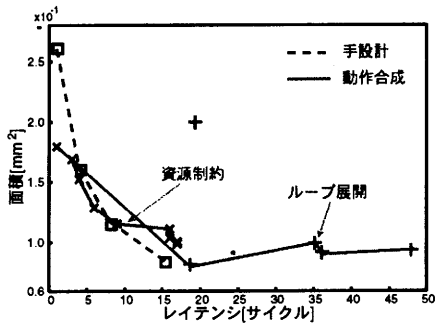

図 10 16 タップの FIR 回路の面積, 遅延のトレードオフ

社から商用ツールとして広く売れるようになれば, 普及の速度が加速するはずである.

また, 現在市販しているシステムレベル設計ツールも, モデル化だけを行なえるものが多く, 動作合成等による自動的なハードウエア化ができない場合が多い. プロセッサと周辺ハードウエア, メモリからなる SoC のアーキテクチャを決定するために使え, その検討結果からシームレスに LSI のハードウエアおよびその上のソフトウエアまで落とせるためのツールの登場が期待される.

現在提案されているシステムレベル言語のほとんどすべてが,

表 2 システムレベル/動作レベル記述言語の概要

言語名	開発元	概要
Cyber(BDL) [4]	NEC	システムレベルシミュレータ, 動作合成, デバッガ
Bach C [5]	Sharp	動作レベルシミュレータ, C 変換ツール, 動作合成
Handel-C [6]	Celoxica	動作レベルシミュレータ, 動作合成, FPGA 合成
SystemC	OSCI	C++クラスライブラリによるシステムレベル記述
SpecC [8]	UC Irvine	仕様記述, シミュレータ

C 言語をベースとしている. もともと, プロセッサ上の逐次的なソフトウエアを記述するために開発された C 言語を, ハードウエアを含めた並列的に動くシステムを記述するために用いることは, 無理があるのは事実である. ただし, プロセッサ上のアプリケーションがほとんどすべて C 言語で書かれている現状では, SoC 上のシステムを開発するための言語として C 言語をベースとせざるを得ない. C 言語は学生や新入社員のほとんどが知っており, 書店に行けばその解説書が容易に見つけられる. 新しい言語の初期教育にかかるコストを考えると, C 言語ベースというのは利点のひとつとなる. 今後, ユビキタスコンピューティングの進歩によりプロセッサをベースとしたシステムが根本的に変わるようなことがあれば, その新しいシステムのための新しい設計パラダイムが生まれる可能性もあるが, しばらくは C 言語ベースの設計が一般的に使われるであろう.

6. ま と め

本稿では, RTL と動作レベルの概説からはじめ, 動作合成の利点, 様々なシステムレベル設計/動作合成ツールの紹介等を論じた. C 言語をある程度知っている LSI 設計者が簡単に使えるツールはまだ出てきていないのが現状ではあるが, システムレベルでの設計の最適化は今後の nm プロセスにおける SoC の大規模化には不可欠の技術である.

文 献

[1] 新村出 (編). 広辞苑. 岩波書店.

[2] 田中譲. サーチ、ソート・ハードウエ アと記号処理への応用. 情報処理,vol. 23, no.8, pp. .742–747, 1982.

[3] TECHSCORE. http://www.techscore.com/tech/c/index.html.

[4] 黒川秀文, 池上裕之, 大坪基秀, 浅尾清, 桐ヶ谷和久, 勝哉三栖, 高橋聡, 川津哲仁, 新田功士, 笠浩史, 若林一敏, 友部実, 高橋渡, 向山輝, 竹中崇. C 言語ベースの動作合成を利用したシステム LSI 設計手法の効果分析と考察. 第 16 回回路とシステム (軽井沢) ワークショッ プ予稿集, 2002/04.

[5] T. Kambe, A. Yamada, K. Nishida, K. Okada, M. Ohnishi, A. Kay, P. Boca, V. Zammit, and T. Nomura. A C-based synthesis system, Bach, and its application. *Proceedings of the ASP-DAC 2001. Asia and South Pacific Design,no.78*, pp. 151–5, 2001.

[6] C. Sullivan, A. Wilson, and S. Chappell. Using C Based Logic Synthesis to Bridge the Productivity Gap. *Proceedings of 2004 ASP-DAC*, pp. 349–354, 2004.

[7] Susan Stepney. Pictorial representation of parallel programs. In Alistair Kilgour and Rae A. Earnshaw, editors, *Graphical Tools for Software Engineering*, BCS conference proceedings. CUP, 1989.

[8] 木下常雄, 冨山宏之. SpecC 使用記述と方法論. CQ 出版, 2000.