設 計 自 動 化 85－24
（1997. 10. 29）

ハードウェアコンパイラ Bach

西田 浩一 † Andrew Kay †† 山田 晃久 † 神戸 尚志 † 野村 俊夫 ††

† シャープ株式会社 生産技術開発推進本部 精密技術開発センター
〒632 奈良県天理市櫟本町 2613-1
E-mail: {nishida, yamada, kambe}@edag.ptdg.sharp.co.jp

†† シャープヨーロッパ研究所 Systems Architecture
Edmund Halley Road, Oxford Science Park, Oxford OX4 4GA, United Kingdom
E-mail: {andrew.kay, tosh}@sharp.co.uk

あらまし
本稿では，システム LSI 設計環境 Bach を提案する．Bach は，C 言語をもとに，データのビット幅の指定，明示的な並列実行記述のための構文を独自に追加した Bach-C による記述を入力とする．Bach は，動作レベルシミュレーション，対話式シミュレーション，合成可能な VHDL の生成を行うことができ，システム LSI を短期間で効率良く設計することを可能とする．Bach を画像処理用 LSI の設計に適用した実験結果を報告し，システム LSI 設計における Bach の有効性を示す．

キーワード システム LSI 設計，動作合成，C 言語，VHDL 生成，シミュレーション，デバッグ

Hardware Compiler Bach

Koichi NISHIDA † Andrew Kay †† Akihisa YAMADA †
Takashi KAMBE † Toshio NOMURA ††

† Precision Technology Development Center, Production Technology Development Group, SHARP Corporation
Ichinomoto-Cho, Tenri, Nara, 632 Japan
E-mail: {nishida, yamada, kambe}@edag.ptdg.sharp.co.jp

†† Systems Architecture, Sharp Laboratories of Europe Limited
Edmund Halley Road, Oxford Science Park, Oxford OX4 4GA, United Kingdom
E-mail: {andrew.kay, tosh}@sharp.co.uk

Abstract
In this paper we describe design environment Bach for system LSI design. The input language is based on a modification of the standard C language which supports explicit parallelism and bit-width specification. The output language is synthesisable VHDL which makes it easy to design efficient system LSI. We show the effectiveness of Bach by presenting experimental results from image processing designs.

key words system LSI design, behavioral synthesis, C language, VHDL generation, simulation, debugging

1 まえがき

近年，ディジタルLSIの設計自動化において，ハードウェアの構造を含まない，動作のみを記述した動作記述から，RTレベルの記述を合成する動作合成の研究が盛んに行われてきた[1]．動作合成CADは実用段階に入っており，人手設計に比較して遜色のないLSIを短期間に設計することが可能になってきている．動作合成CADを使用する場合，LSIの本質的な動作を決定するアルゴリズム設計が人手設計の大部分を占めるため，設計者はアルゴリズムに集中することができ，回路の品質も向上させることができる．一方，近年のLSIのさらなる大規模化，微細化にともない，複雑なアルゴリズムを持つシステムを1チップに収めたシステムLSIの実現が可能になってきており，このようなシステムLSIを効率的に設計・検証できる環境が求められている．

本稿では，システムLSI設計環境Bachを提案する．Bachは，C言語をもとに，データのビット幅の指定，明示的な並列実行記述のための構文，並列実行記述間のデータ通信命令を独自に追加したBach-Cによる記述を入力とする．Bachは，シミュレータによる動作レベルシミュレーション，デバッガによる対話式シミュレーション，合成可能なVHDLの生成を行うことができ，システムLSIを短期間で効率良く設計することを可能とする．

Bachを画像処理用LSIの設計に適用した実験結果を報告し，システムLSI設計におけるBachの有効性を示す．

2 システムLSIの設計

2.1 システムLSIの設計過程

動作合成CADの使用を前提としたシステムLSIの一般的な設計過程を以下に示す．

システムレベル分割

システムのLSIの設計においては，まず設計の初期段階でシステムレベル分割[2]を行う．ここでシステムとは，目的とする全ての機能を実現したものであり，システムレベル分割とは，システムの要求仕様からあるまとまりを持った機能を実現する複数のサブシステムの仕様を作成することを言う．システムレベル分割を行う主な目的を以下に示す．

- 人間にとって把握し易い，あるまとまりを持った処理に分割することによって，設計を容易にし，設計段階での誤りの混入を防ぐ
- 複数の設計者による同時設計を可能にする
- システム全体の並列性を抽出し，並列に動作するサブシステムに分割することによって，合成，レイアウトの対象となる回路の規模を制限する

システムレベル分割は設計の初期段階で行われる．システムレベル分割の段階では，サブシステムの設計は完了していないため，最適な分割を自動で行うのは困難であり，システムレベル分割は人手で行われている．

データ通信方式の決定

システムレベル分割を行った後，サブシステム間のデータ通信方式を決定する．サブシステム間でデータ通信方式の一致が取れていなければ，システムを正しく動作させることができない．

複数の設計者によりサブシステムを同時設計する場合，設計段階では通信相手のサブシステムの設計が終了していないため，データ通信部分に誤りが混入し易い．サブシステム間の通信を確実に行うために，ハンドシェークによるデータ通信が多く用いられる．ハンドシェークのプロトコルを統一しておけば，通信相手とのデータ通信のタイミングを考慮することなく，サブシステムの設計を行うことができる．

アルゴリズム設計・動作レベル検証

アルゴリズム設計とは，システムの動作を実現するアルゴリズムを決定し，動作記述を作成することである．動作合成CADを使用する場合，アルゴリズム設計は動作合成CADの入力言語で行われる．

アルゴリズム設計の後，動作レベル検証が行われる．動作レベル検証では一般的に，動作レベルシミュレータを用いて動作記述のシミュレーションを行うことにより，アルゴリズムの正しさを検証する．

合成・シミュレーション

アルゴリズム設計，動作レベル検証の終了後，動作合成CADを用いてRTレベルの記述を合成し，論理合成CADによりゲートレベルの回路を合成する．RTレベル，およびゲートレベルのシミュレーションを行い，動作レベルシミュレーションの結果と一致するかどうか確認する．

2.2 システムLSI設計における問題点

従来は，システムLSIを構成する各サブシステムに対して，通信プロトコルを実現するためのハードウェアを人手で設計する必要があったため，サブシステム間の通信部分に誤りが混入することがあった．同じ通信プロトコルを実現するハードウェアが複数必要なため，複数の設計者によりシステムを同時設計する場合，同じ回路が重複して設計されることがあり，開発効率が悪くなっていた．

動作レベル検証を行う場合，サブシステム間の通信プロトコルを実現するハードウェアだけでなく，テストベンチにおいて，システムLSIとの通信プロトコルを実現するための記述も人手で作成する必要があった．

複数のサブシステム間でRAM等の記憶素子を共有する場合，アクセスの競合を防ぐための仲裁回路を人手で

設計する必要があった．

これらによりシステムの動作レベル検証にかかる時間が長くなり，開発効率の低下につながっていた．アルゴリズム設計では，このようなシステムの動作と直接関係のない煩雑な作業を極力減らし，アルゴリズムそのものに集中する必要がある．

3 ハードウェアコンパイラ Bach

ハードウェアコンパイラ Bach は，前記の問題点を解決することにより設計期間の短縮を可能とする，システム LSI 設計環境である．

Bach は，C 言語をもとに，ハードウェア設計に適するよう独自に拡張を行った Bach-C による動作記述を入力とする．Bach-C シミュレータ/デバッガにより，Bach-C 記述のシミュレーション，デバッグを行うことができる．

Bach-C には明示的な並列実行記述のための par 構文が独自に追加されており，アルゴリズム設計では par 構文を用いて，並列に動作する複数のサブシステムを持つシステムの全ての動作を記述することができる．

Bach-C は同期通信用の命令を持っており，Bach の VHDL ジェネレータはこの命令に対してハンドシェーク回路の記述を自動で生成するため，設計が容易になり，誤りの混入を防ぐことができる．複数の設計者によりシステムを同時設計する場合でも，ハンドシェーク回路が重複して設計されることがなくなる．

テストベンチの記述においても同期通信用の命令を用いてシステムと同期通信を行うことができるため，ハンドシェーク回路を人手で記述する必要がない．

Bach の VHDL ジェネレータは，配列アクセスの競合を防ぐための仲裁回路を含む配列インターフェースを自動生成する．

3.1 Bach によるシステム LSI の設計フロー

図 1に，Bach を用いたシステム LSI の設計フローを示す．

まず，人手によりシステムレベル分割を行い，Bach-C によりアルゴリズム設計を行う．次に，Bach-C シミュレータ/デバッガを用いて，システムの動作レベル検証を行い，VHDL ジェネレータによりシステムの VHDL 記述を生成する．

VHDL ジェネレータは，par 構文の各並列実行記述に対し，独立した回路を生成する．並列に動作するこれらの回路をスレッドと呼び，スレッドを記述した VHDL 記述をスレッドエンティティと呼ぶ．スレッドエンティティはビヘイビアレベルの VHDL となる．

par 構文の各並列実行記述間では，変数，配列を共有することができる．これらを共有資源と呼ぶ．共有資源を記述した共有資源エンティティ，およびトップレベルエンティティに対しては，RT レベルの VHDL 記述が生成される．

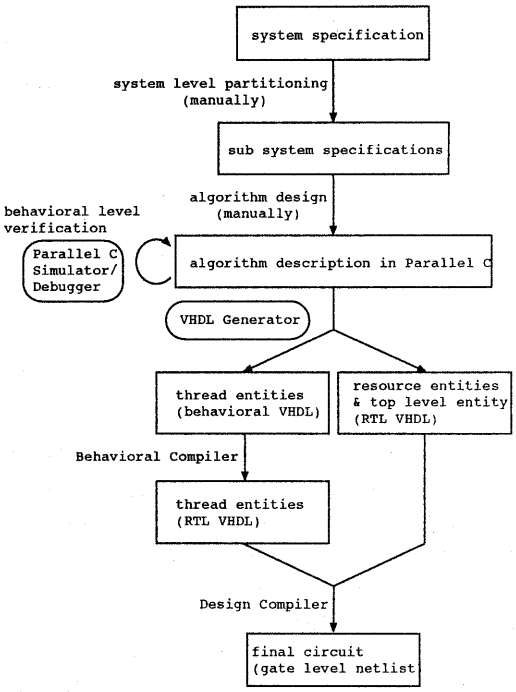

図 1 Bach によるディジタルシステムの設計フロー

3.2 Bach-C

Bach-C の特徴を以下に示す．

データのビット幅の指定

Bach-C では，int(signed) 型，unsigned 型において，ビット幅を指定することができる．設計者は，必要なビット幅を明示的に記述することにより，冗長なハードウェアの生成を防ぐことができる．ビット幅は型名の後に#をつけることにより指定する．図 2(a) の例では，8 ビットの unsigned 型の変数 a と 4 ビットの int 型の変数 b を宣言し，b を 8 ビットの unsigned 型にキャスト変換した結果に 2 を掛けたものを a に代入している．

ビット幅が指定されていない変数はデフォルトのビット幅となる．デフォルトのビット幅は，Bach の初期設定ファイルに記述しておくことができる．

par 構文

Bach-C では，par 構文により，並列動作を明示的に記述することができる．図 2(b) の例では，2 つのステートメント，a = b * c と d = e * f が並列に実行される．これらの二つのステートメントを並列実行記述と呼ぶ．VHDL ジェネレータが出力する回路においては，一つの par 構文における各並列実行記述の動作は異なるスレッドエンティティで並列に実行される．

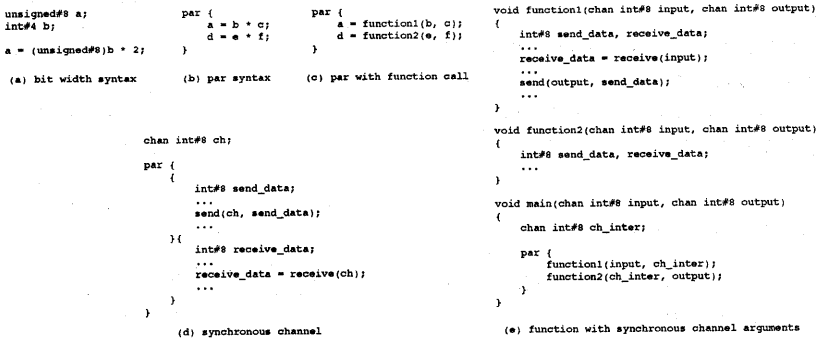

図 2　Bach C

スレッドエンティティの起動，終了のための速度的なオーバーヘッドが存在するため，実際には図 2(b) のような短い記述を par 構文により並列化するのは効率が悪く，通常は図 2(c) のように関数呼び出しと併せてある程度まとまった記述に対し使用する．

同期チャネルによる通信

同期チャネルとは，par 構文における並列実行記述間で同期を取りながらデータ通信を行うための通信路である（図 2(d)）．データの送信側は受信側の準備ができるまで待機し，準備ができたらデータを送信する．データの受信側は送信側の準備ができるまで待機し，準備ができたらデータを受信する．

同期チャネルによるデータ通信を用いるには，まず par 構文の外で同期チャネルを宣言し，データの送り側は send 関数を用いてその同期チャネルへデータを送り，データの受け側は receive 関数を用いてその同期チャネルからデータを受け取る．図 2(d) の例では，まず 8 ビットの int 型のチャネル ch を宣言し，データの送り側は変数 send_data の値を ch に送り，データの受け側は ch から受け取ったデータを変数 receive_data に代入している．

関数の引数に同期チャネルを指定することができる（図 2(e)）．main 以外の関数の引数を同期チャネルとし，その関数呼び出しを par 構文における並列実行記述として記述することにより，並列に動作する関数間でデータ通信を行うことができる．その場合，通信を行う二つの関数の引数には，同じチャネルを指定する．main 関数の引数を同期チャネルとすると，システム外部とのハンドシェークによる入出力を行うことができる．図 2(e) の例では，function1，function2 の呼び出しは par 構文における並列実行記述となっており，同期チャネル ch_inter は function1，function2 の共通の引数となっているため，function1，function2 は並列に動作し，同期チャネル ch_inter を通して通信が行われる（図 3）．main 関数の引数である input，output チャネルはシステム外部とのハンドシェークによる入出力となる（図 3）．

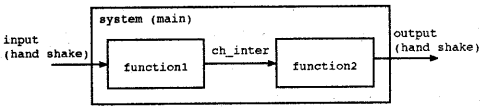

図 3　関数の引数への同期チャネルの指定

共有変数による通信

par 構文における並列実行記述間では，同期チャネルの他に共有変数を用いてデータ通信を行うこともできる．共有変数を用いた通信では同期を取らないため，書き込み，読み出しが行われる順序によって結果が変わり，Bach-C のシミュレーション結果と実際の回路の動作結果が一致しない場合がある．Bach-C では，配列も，共有変数として用いることができる．

3.3　VHDL ジェネレータ

VHDL ジェネレータは，まず Bach-C を字句解析，構文解析し，内部表現に変換し，ソフトウェア最適化，関数のインライン展開を行う．次に内部表現から，並列に動作するスレッドエンティティのビヘイビアレベル VHDL 記述，スレッド間の共有変数，配列インターフェース等を記述した共有資源エンティティおよびトップレベルエンティティの RT レベル VHDL 記述を生成する．

スレッドエンティティの生成においては，同期チャネルを実現するためのハンドシェーク回路の記述，共有変数および配列インターフェイスにアクセスする回路の記述も自動生成する．生成されたビヘイビアレベル VHDL は，市販のツール [3] を用いて動作合成可能である．同期チャネルや共有変数，配列インターフェイスへ正しくアクセスするために，スレッドエンティティの I/O にはスケジューリング制約を与える必要がある．VHDL 生成部では，これらのスケジューリング制約を記述したスクリ

プトファイルを自動生成する．

配列インターフェースの生成においては，アクセスの競合を防ぐための仲裁回路も自動生成される．VHDL ジェネレータは，特に指定しなければ，書き込みの行われない配列に対しては組合せ回路による ROM を，書き込みの行われる配列に対してはフリップフロップによる RAM を生成する．Bach-C 記述で pragma により指定することにより，配列を外部 ROM，外部 RAM にマッピングすることもできる．

図 4に例を示す．図 (a) の記述から図 (b) の各 VHDL エンティティが生成される．図 (a) の記述は，par 構文により 2 つの並列実行記述が記述されているため，これらを並列に動作させるために 2 つのスレッドエンティティが生成される．2 つのスレッドで共有変数 shared が共有されているため，共有変数 shared の RTL の VHDL が生成される．配列 array が宣言されているため，配列インターフェースの RTL VHDL が生成される．配列 array は RAM にマッピングされているため，生成される配列インターフェースには RAM のモデルが含まれる．

4 計算機実験

人手で VHDL 記述された画像処理用 LSI と同じ動作を Bach-C で記述し，Bach-C シミュレータ/デバッガで動作レベル検証を行った後，VHDL ジェネレータでビヘイビアレベル VHDL 記述を生成し，市販のツール [3] [4] を用いて合成を行った．合成結果を，人手で記述された VHDL 記述の合成結果と比較した．2 機種の LSI について本実験を行った．

実験結果において，area はゲート数を，throughput は 1 データの処理に必要なクロックサイクル数（スループット）を示す．Bach は Bach により生成された VHDL の合成結果を，original はオリジナルの VHDL 記述の合成結果を示す．sub n は回路に含まれるサブシステム n の合成結果を，total は回路全体の合成結果を示す．

4.1 画像処理用 LSI(1)

本実験では，人手によるビヘイビアレベル VHDL 記述の合成結果と，Bach-C 記述の合成結果の比較を行った．

本 LSI は 7 つのサブシステムがパイプライン構造により接続されており，オリジナル回路の各サブシステム間では，各サブシステムのスループット，データ転送時間を合わせることにより通信を行っている．Bach-C による記述では，各サブシステムを par 構文の並列実行記述として記述し，サブシステム間の通信には同期チャネルを用いた．

表 1，表 2に，Bach により生成された VHDL 記述の合成結果と，オリジナルのビヘイビアレベル VHDL 記述の合成結果の比較を示す．

表 1 画像処理用 LSI(1) の合成結果 （面積）

	area [#gate]	
	Bach	original
sub1	2,927	2,309
sub2	8,526	11,322
sub3	927	896
sub4	2,142	1,722
sub5	4,287	2,550
sub6	1,051	1,023
sub7	2,376	2,329
total	22,777	22,382

表 2 画像処理用 LSI(1) の合成結果（スループット）

	Bach	original
throughput [#clock]	4	2

表 3に，オリジナル回路の設計期間と Bach による設計期間の比較を示す．

表 3 画像処理用 LSI(1) の設計期間

	Bach	original
design time[#day]	2	7

Bach とオリジナルの面積はほぼ同じであるが，スループットを比較すると，Bach はオリジナルの 200%となっている．スループット増加の原因は，ハンドシェークによるオーバーヘッドである．使用した市販の動作合成ツール [3] は，全ての出力をラッチし，if 文により状態遷移が起こる場合，if 文の条件式を一度ラッチするという制約があるため，ハンドシェークのような高速な I/O の実現が難しい．しかしハンドシェークによる同期チャネルを用いることにより，通信相手との通信のタイミングを考慮することなく各コンポーネントの設計を行うことができたため，設計が容易になった．Bach による本回路の設計は 2 日で終了しており，Bach を使用することにより設計期間を 29%に短縮できた．

4.2 画像処理用 LSI(2) への適用

本実験では，人手による RT レベル VHDL 記述の合成結果と，Bach-C 記述の合成結果の比較を行った．

本 LSI は 4 つのサブシステムからなり，処理データを格納するための外部 RAM，およびテーブルを格納するための外部 ROM/組合せ回路による ROM/外部 RAM を含む．

Bach-C による記述では，各サブシステムを par 構文の並列実行記述として記述し，サブシステム間の通信には同期チャネルを用いた．処理データを格納する配列，テーブルを格納する配列を外部 RAM/組合せ回路による ROM にマッピングした．

表 4，表 5に，Bach により生成された VHDL 記述の合成結果と，オリジナルの RT レベル VHDL 記述の合成結果の比較を示す．本結果における面積に，外部 RAM，

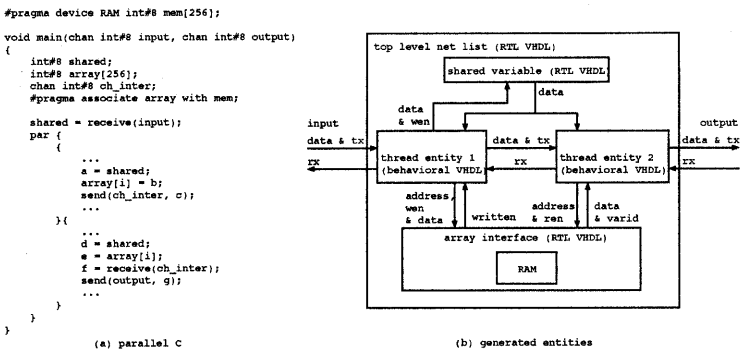

図4　VHDL エンティティ生成例

外部 ROM は含まれていない．

表4　画像処理用 LSI(2) の合成結果（面積）

	area [#gate]	
	Bach	original
sub1	13,337	10,876
sub2	1,810	997
sub3	5,521	1,759
sub4	1,948	800
total	26,595	15,173

表5　画像処理用 LSI(2) の合成結果（スループット）

	Bach	original
throughput [#clock]	1,486	1,280

表 6に，オリジナル回路の設計期間と Bach による設計期間の比較を示す．

表6　画像処理用 LSI(2) の設計期間

	Bach	original
design time[#week]	1	4

Bach とオリジナルの面積を比較すると，Bach はオリジナルの 175%となっている．面積増加の原因は，Bach の結果に含まれる組合せ回路による ROM の一部がオリジナルでは外部 ROM となっていること，市販のツール[3]がそれぞれのスレッドに対し独立したコントローラを生成することによるオーバーヘッドが挙げられる．Bach とオリジナルのスループットを比較すると，Bach はオリジナルの 116%となっている．スループット増加の原因は，ハンドシェーク，共有配列アクセスによるオーバーヘッドが挙げられる．しかし，Bach による本回路の設計は一週間で終了しており，Bach を使用することにより設計時間を 25%に短縮できた．

5　おわりに

本稿では，システム LSI 設計環境 Bach を提案した．Bach は，C 言語を拡張した Bach-C を入力とし，動作レベルシミュレーション，対話式シミュレーション，合成可能な VHDL の生成を行うことができる．Bach は，複雑なアルゴリズムを持つシステム LSI を短期間で効率良く設計することを可能とする．Bach を画像処理用 LSI の設計に適用した実験結果を報告し，システム LSI 設計における Bach の有効性を示した．VHDL による手設計と比較すると，記述量は大幅に削減され，システム全体の動作レベル検証を確実にしておくことにより設計の後戻りを防ぐことができるため，設計期間は大幅に短縮される．

Bach の現時点での問題点として，

- ハンドシェーク，共有資源アクセスを実現することによる速度的なオーバーヘッド
- それぞれのスレッドに対し独立したコントローラを生成することによる面積の増加

が挙げられる．今後の課題として，以上の問題点を解決するとともに，低消費電力システム LSI の設計への対応などが挙げられる．

参考文献

[1] D. Gajski, A. Wu, N. Dutt, S. Lin, *High-Level Synthesis: Introduction to Chip and System Design*, Kluwer Academic Publishers, 1992.

[2] P. Michel, U. Lauther, P. Duzy, *The Synthesis Approach to Digital System Design*, Kluwer Academic Publishers, 1992.

[3] *Behavioral Compiler* ユーザーガイド *version 1997.01*, Synopsys, 1997.

[4] *Design Compiler Reference Manual version 1997.01*, Synopsys, 1997.