システムＬＳＩ設計技術　97－5
（２０００．９．７）

Cベース設計システム Bach の現状と今後の構想

山田 晃久

シャープ株式会社 IC 開発本部 設計技術開発研究所
〒 632-8567 奈良県天理市櫟本町 2613-1
E-mail: yamada@icg.tnr.sharp.co.jp

年々大規模化、複雑化するシステム LSI の設計において、システムの分割、動作記述、動作レベル検証、ハードウェア生成を効率的に行うことができる設計環境が望まれている。本稿では、我々が構築したシステム LSI 設計環境である Bach システムを紹介し、本システムを用いて設計した MPEG-4 ビデオコーデックを例に設計フロー、本システムの効果、現状の課題を示す。また現在構築中の、LSI 化するハードウェアと組み込み用ソフトウェアの同時検証、及び自動生成を行なう協調設計システムについても述べる。

キーワード　システム LSI 設計，高位合成，C 言語ベース設計，HW/SW 協調設計

A C-Based Design System, Bach: Its Present and Future

Akihisa YAMADA

Design Technology Development Laboratories, Integrated Circuits Development Group
SHARP Corporation
Ichinomoto-Cho, Tenri, Nara, 632-8567 Japan
E-mail: yamada@icg.tnr.sharp.co.jp

In system LSI design, a design environment where a designer can describes, partitions, and verifies a system, and generates circuits efficiently is desired. In this paper, we describes a system LSI design system called Bach which we have developed. Using an example of MEPE-4 video codec design, we show its design flow, effects, and current issues. We also show our concept on a hardware/software co-design environment using Bach.

key words　system LSI design, high-level synthesis, C based design, HW/SW co-design

1 はじめに

近年のLSIの微細化により大規模なシステムLSIが実現可能となっており、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。90年代に入り実用され始めた論理合成ツールに続き、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ツール[1]が実用段階に入り、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[2]。しかしながら、VHDLやVerilog-HDLを入力とする動作合成ツールは広く定着しているとはいえない。

一方で、LSI設計の初期に行なわれるアルゴリズム設計では、ANSI C/C++がよく使用されていること、また通信、マルチメディア関係など数多くのアルゴリズムがANSI C/C++を用いて既に実現されていることから、これらのアルゴリズムを短期間でLSI化できるようにANSI C/C++をベースとした記述言語を用いた設計手法も数多く提案されている[3-10]。本来のANSI C/C++では、ハードウェアの並列動作やビットアキュレートな演算を扱えないため、これらの手法では、文法を拡張したり、C++のハードウェア記述用のクラスライブラリを用意することで対応しているが、ツール間の互換性を持たせるために、言語の標準化作業も進められている[11, 12]。

当社でも、複雑なアルゴリズムを短期間でLSI化するために、C言語を用いたLSI設計フローを構築している。その中心となるのが、Bachシステムである。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言語による記述を入力とする．Bachは，アルゴリズム検証を行なうためのシミュレータ・デバッガ，アルゴリズム記述からRTL回路を合成し、論理合成可能なVHDLを生成するコンパイラにより構成され，大規模な回路を短期間で効率良く設計する環境を提供する．

本文では，Bach-C言語、Bachシステムの概要を説明し、本システムを用いて設計したMPEG-4ビデオコーデックを例に設計フロー、本システムの効果、現状の課題を紹介する。最後に、これらの課題を元に現在構築中であるハードウェア／ソフトウェア協調設計環境について述べる。

2 Bachシステムの概要

2.1 Bach C言語

我々は、アルゴリズムから回路を自動合成するBachシステムの入力言語としてANSI C言語を選択した。これは、社内のアルゴリズム開発者の多くが利用しており、設計資産も活用できると考えたからである。ただし、C言語の枠内だけでハードウェアのアルゴリズムを記述しようとすると記述が繁雑になってしまい、設計者にとっては記述しづらくなってしまうため、直観的に記述できるように文法を拡張した。

拡張した点は、

- 任意のビット幅演算を実現するためのデータ型
- ハードウェアの並列動作を明示するためのpar構文
- 並列に動作するスレッド間の通信

である[10]。図1にBach C記述の例を示す。

```
 1 void main(void){
 2 {
 3   chan int #8 to_ckt, from_ckt;
 4
 5   par {
 6     circuit(to_ckt, from_ckt);
 7     tbench(to_ckt, from_ckt);
 8   }
 9 }
10
11 void circuit(chan int#8 to_ckt,
12              chan int#8 from_ckt)
13 {
14   unsigned #4 i;
15   int #8 x;
16
17   for(i = 0; i < 10; i++){
18     x = receive(to_ckt);
19     send(from_ckt, i*x);
20   }
21 }
22
23 void tbench(chan int#8 to_ckt,
24             chan int#8 from_ckt)
25 {
26   unsigned #4 i;
27
28   for(i = 0; i < 10; i++){
29     send(to_ckt, i);
30     putint(stdout,10,0,receive(to_ckt));
31   }
32 }
```

図1: Bach C記述の例

[i] 任意ビット幅のデータ型

Bach C言語では、int(signed)型、unsigned型において、ビット幅を指定することができる。設計者は演算精度を確保するのに必要なビット幅を明示す

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。例えば、図1中14行目では変数iが4ビット幅のunsigned型として宣言され、15行目では変数xが8ビット幅のint型として宣言されている。ビット幅を明示していない場合は、デフォルトのビット幅とみなされる。

[ii] par構文

並列に動作するハードウェアを直観的に記述できるように、Bach C言語にはpar構文が追加されている。図1の5--8行目では、関数circuit()とtbench()が並列に動作することが指定されている。

[iii] 通信

Bach C言語では、並列に動作するスレッド間でデータをやり取りするための通信を記述することができる。図1の18行目と29行目ではtbench()のスレッドからcircuit()のスレッドへのデータ転送が記述されており、19行目と30行目ではその逆方向のデータ転送が記述されている。

データ転送に用いられる通信路には同期通信路と非同期通信路がある。同期通信路では、送信側と受信側の双方が準備できるのを待ってデータ転送が行なわれ、データ転送が完了するまで次の処理は行なわれない。一方、非同期通信路では、通信相手が準備できていなくても送信、または、受信が実行され、ただちに次の処理が行なわれる。図1の3行目では、同期通信路(chan型) to_cktと from_cktが宣言される。

2.2 ツール

Bachはツールとして、コンパイラ、会話型デバッガ、コンパイル型シミュレータ、プロファイラを持つ。

コンパイラ コンパイラはシステムの動作を記述したBach C記述を入力とし、動作合成を行なって論理合成可能なRTL記述を出力する[13]。このコンパイラではpar構文の各分岐は1つの階層となるように合成する[14]。動作合成では、記述中の並列動作を自動で抽出できるが、回路が大規模になった場合に、合成した回路が階層的な構造になっていた方が、FPGAへの回路の分割等が行ない易いため、設計者が意図的に全体の回路を分割できるようになっている。

デバッガ、シミュレータ デバッガはBach C記述上でブレークポイントを設定したり、ステップ実行することが可能で、並列動作を含む回路動作を会話的にデバッグすることができる。コンパイル型シミュレータは、Bach CをANSI Cに変換し、Cコンパイラで実行モジュールを生成するため、並列動作を含む回路動作を高速に検証することが可能である。

プロファイラ プロファイラはシミュレーション中に記述中の各構文が実行される回数をカウントする。実行回数が多い箇所を改善するなど、システムの性能向上に役立つ情報を得ることができる。

3 設計例

Bachシステムを用いてMPEG-4ビデオコーデック(以後単にMPEG-4と呼ぶ)のコアを開発したので、これを例にBachシステムを用いた設計フローを説明し、この設計を通して明らかになった現状の課題について述べる。

3.1 設計フロー

今回の設計では、すでにC言語で記述されたプログラムを元に設計を行なった。

1. アーキテクチャ設計(HW/SW分割)

   MPEG-4の処理フローを図2に示す。MPEG-4では低ビットレートでの画像圧縮を実現するためにフレームレートの調整や量子化の調整を細かく行なうなどの、ソフトウェアに適した処理がある一方で、動き予測やDCT/IDCTなどの計算量の必要なハードウェアに適した処理も含まれる。このため、一部をCPUで、残りを布線論理として実現することにした。この分割は人手で行なった。分割後のアーキテクチャは図3のような構成となった。

2. Bach-C記述の作成

   Bach-Cは、ANSI Cの主要なデータ構造、制御構文を扱うことができるが、ポインタおよび関数の再帰呼び出しに制限があるため、まずこれらの構文を書き換えてシミュレーション可能なBach-C記述を作成する。

   次に、回路に適したBach-C記述を作成する。主な作業としては、

   - par構文により異なる部分回路として実現したい処理については明示的に記述する

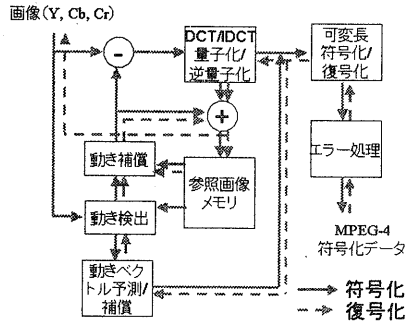

図 2: MPEG-4 の画像圧縮・伸長処理フロー

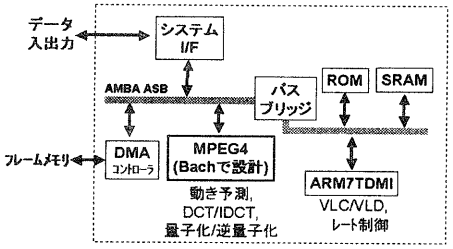

図 3: MPEG-4 回路のアーキテクチャ

- 各変数のビット幅を指定する

である。

3. Bach コンパイラによる合成
   動作周波数とセルライブラリを指定してコンパイラを起動する。コンパイラは、セルライブラリから遅延、面積情報を算出するので、所望の動作周波数で動作する RT レベルの回路を生成するが、最終的な遅延の確認は、論理合成後に行なう必要がある。

4. RTL 検証
   合成された回路の機能はすでに Bach-C 記述を用いて検証されているが、パフォーマンスや外部回路とのデータ転送に問題がないかは生成された VHDL 記述を用いて確認する。これには VHDL シミュレータを用いる。

5. 論理合成
   Bach コンパイラが生成する VHDL 記述は、市販の論理合成ツールで合成可能な記述である。必要な制約を付けて、論理合成を行なう。

6. テスト容易化設計
   合成された回路については、フルスキャン回路を挿入することにより対応する。これは、論理合成ツールの機能を利用する。MPEG-4 では複数のメモリが存在するが、メモリテスト用の回路をコンパイラで自動生成することもできる。

7. 組み込みソフトとの検証
   コンパイラにより生成された VHDL 記述を用いて、シミュレータおよびボード上で検証を行なう。

Bach-C の検証は、VHDL の検証と比べて 100 倍程度高速であり、短時間でビットアキュレートな検証を行なえる。今回の設計の場合、QCIF サイズの画像を 100 フレーム分 VHDL で検証しようとすると約 1 日かかってしまうが、Bach-C で検証すると数分もあれば完了してしまう。したがって、Bach-C のレベルで確実に機能を検証した後に合成をすることにより、手戻りのない回路設計が行なえる。

3.2 現状の課題

Bach はこれまでアルゴリズムをハードウェア化することを主目的として開発してきた。しかしながら、MPEG-4 画像コーデックのようにハードウェアと組込みソフトウェアから構成される組み込みシステムを設計するには以下のような課題がある。

- Bach-C 記述をハードウェアと組込みソフトウェアに分割するしくみの実現

  現在の Bach では、ユーザは予めシステム分割しておき、ハードウェアの動作と外部環境の動作（テストベンチ）を Bach C 言語を用いて記述する。ユーザがハードウェア化する動作を記述した関数名をコンパイラに与えることにより、コンパイラは関数に含まれる動作をハードウェア化できる。しかし、一部を組み込みソフトウェアとして実現するといった分割をすることができない。

- Bach-C 記述レベルでハードウェアと組込みソフトウェアを統合したシステム全体の性能評価

  ハードウェア化される部分については、スケジューリング結果やアロケーション結果、生成したコントローラの状態数などから、回路動作時のステップ数、面積、消費電力等の性能を見積もることができるが、ソフトウェアと同時に動作するときの性能を評価することができない。

- インタフェース回路の設計自動化

  現在のバージョンでは、Bach で合成したターゲット回路と ARM プロセッサとを AMBA バス接続を行なうためのバスインタフェース回路を自動生成する機能を有している [15]。しかし、他のバスプロトコルや通信プロトコルについては、サポートできていない。

- IP 利用のしくみの実現

  大規模化、複雑化するシステムを効率良く設計するために、IP(Intellectual Property) を利用することが考えられる。今回の設計でもすでに VHDL 設計により最適化された DCT/IDCT コアについては、再利用した。しかし、このような IP を設計中の回路に自動的に取り込むしくみを持たないため、生成された VHDL 記述を手修正することにより行なった。

4 今後の構想：Bach を用いたハードウェア/ソフトウェア協調設計

前節で述べた課題を解決し、Bach を用いたハードウェア/ソフトウェア協調設計環境を構築するために、Bach の設計環境に以下の機能を付加する。

- システム分割支援機能
- システム性能評価機能
- インタフェース生成機能
- IP ライブラリリンク機能

これらの機能を実現することにより (図 4) のようなフローを実現できる。

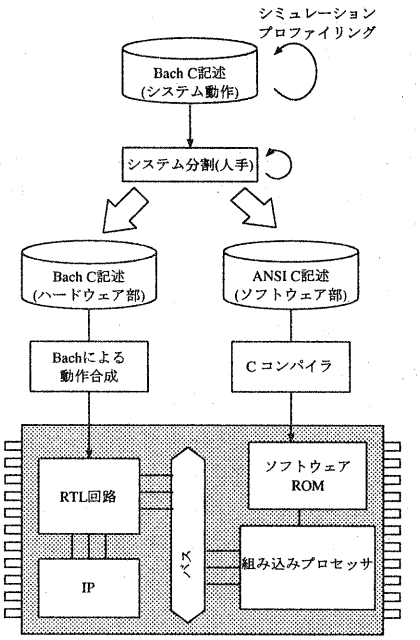

図 4: Bach を用いた協調設計フロー

4.1 システム分割支援機能

システム分割支援は、Bach C 言語で記述されたシステム全体の動作記述のうち、どの部分をハードウェアとして実現し、どの部分を組込みソフトウェアとして実現するかを決めるシステム分割を支援する。

設計者は、Bach C 記述中の構文単位でハードウェアにするか組込みソフトウェアにするかを GUI を用いて会話的に指定する。ハードウェアとして指定された部分は動作合成系により動作合成され、RTL 記述が出力される。組込みソフトウェア部分は組込みプロセッサのソフトウェア開発環境でコンパイルされ、組み込みプロセッサ上で実行されるプログラムに変換される。

4.2 システム性能評価機能

システム性能評価は、ハードウェア部分の動作合成や組込みソフトウェアをコンパイルする前に、分割されたハードウェアと組込みソフトウェアのアーキテクチャでシステムが動作するときのクロックサイクル数、回路面積、消費電力を見積もる。

動作合成後には、サイクルアキュレートなシミュレーションが行なえるため、ハードウェアだけでなく、インタフェース部や組み込みプロセッサ上でソ

フトウェアが動作するときのクロックサイクルや消費電力を評価することができ、高い評価精度を実現する。

4.3 インタフェース生成機能

Bach-Cでは、通信路を用いてsend()、あるいはreceive()という抽象的な関数（手段）で通信が実現されている。これらの通信用の関数をさまざまなCPUバスや通信プロトコルに自動でマッピングすることにより、アルゴリズム設計とインタフェース設計とを切り離して行なうことができるようになる。

4.4 IPライブラリリンク機能

大規模なシステムを短期間に設計するために、既設計資産の再利用を進める必要がある。Bachでソフトマクロやハードマクロとして存在するIPを再利用するために、IPの動作モデルをBach Cのライブラリ関数として登録し、シミュレーションや動作合成時に設計中の回路にリンクする。

5 おわりに

本稿では，Bachシステムについて説明し、現状の課題について示した。また、今後の構想として現在構築中のHW/SW協調設計環境についても示した。

現在、さまざまなCベースの設計言語が提案されており、その中でいくつかの団体で標準化作業が進められている。標準化される言語が、我々が構築している設計フローに適合する言語となれば、その言語に対応することも検討していく。

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] 若林, 古林, 斉藤, 加納, 篠原, 田中, 中越, 北村, "伝送用LSIを動作合成で開発, 機能設計の期間が1/10に短縮", 日経エレクトロニクス, no. 655, pp. 147–169, 日経BP社, 1996.

[3] A. Ghosh, J. Kunkel, and S. Liao, "Hardware Synthesis from C/C++," Din *Proc. of DATE'99*, pp.387–389, March 1999.

[4] G. Arnout, "C for System Level Design," in *Proc. of DATE'99*, pp. 384–386, March 1999.

[5] K. Wakabayashi, "C-based Synthesis Experiences with a Behavior Sysnthesize, "Cyber"," in *Proc. of DATE'99*, pp. 390–393, March 1999.

[6] D. Gajski, J. Zhu, R. Domer, A. Gerstlauer, S. Zhao, *Spec C: Specification Language and Design Methodology*, Kluwer Academic Publishers.

[7] http://www.cynapps.com/

[8] http://www.cleveldesign.com/

[9] http://www.frontierd.com/

[10] A. Yamada, K. Nishida, R. Sakurai, A. Kay, T. Nomura, T. Kambe, "Hardware synthesis with the Bach system," in *Proc. of IEEE ISCAS'99*, Vol. VI, pp.366–369, 1999.

[11] http://www.specc.org/

[12] http://www.systemc.org/

[13] 西田, 岡田, 大西, A. Kay, P. Boca, 山田, 神戸, "ハードウェアコンパイラBachの動作合成系," DAシンポジウム'99, pp.95–100, 1999.

[14] 高橋, 石浦, 山田, 神戸, "ハードウェアコンパイラBachにおけるスレッド分割手法", 第12回回路とシステム（軽井沢）ワークショップ, pp. 103–108, 1999.

[15] 大西, 西田, 岡田, 山田, 神戸, "ハードウェアコンパイラBachを用いたハードウェア/ソフトウェア協調設計環境について," 情報処理学会DAシンポジウム, 2000.