A-08

オブジェクト指向によるシステム LSI 設計の考察

An Object Oriented Design Methodology for System LSI

小原 貴文†　　枡岡 正悟††　　神戸 尚志‡
Takafumi KOHARA　Seigo MASUOKA　Takashi KAMBE

1　はじめに

現在、システム LSI の高機能、高集積化により、設計対象の大規模化、複雑化が進み、抽象度を上げた言語による設計が注目されつつある。またソフトウェアも含めたシステム LSI 全体の設計が効率よく行える手法への関心が高まっている[1]-[3]。また、組込みシステムの分野で Java 言語が世界的に普及しつつある。

ハードウェア／ソフトウェア協調設計を行う際、ハードウェアとソフトウェアの記述を同じ言語を用いて設計することで、ハードウェア／ソフトウェア協調検証が容易になり、システム LSI 全体の開発期間の短縮につながる。これらの点から Java 言語をベースとするオブジェクト指向を用いたシステム LSI 設計手法が考えられる。

本研究では Java 言語をベースとするオブジェクト指向を用いたシステム LSI 設計手法の確立を目的とし、設計言語として筆者らがハードウェア向けに拡張した Java 言語（“Jackal”という）言語を用いる。Jackal 記述を Bach C 記述に変換し、Bach システム[1]による動作合成を行なう。また、Jackal 記述を Java 言語に変換し、ハード／ソフト協調検証を行う。これにより、Jackal 言語からのハードウェア設計を実現している。

本文ではこの Jackal 言語による設計環境を用いたオブジェクト指向の回路設計を行い、その評価を行う。

2　Jackal 言語によるシステム LSI 設計

Jackal 言語によるシステム LSI 設計の利点と設計フローについて述べる。

2.1 Jackal 言語によるシステム LSI 設計の利点

Jackal 言語を用いたシステム LSI 設計において、以下の利点が挙げられる。

(1) Java 言語によるハードウェア／ソフトウェア協調設計環境の実現

Java 言語でソフトウェアを記述する場合、ハードウェアも Java 言語と類字の言語である Jackal 言語で設計することで、ハードウェア／ソフトウェア協調検証が容易になる。

(2) オブジェクト指向による回路設計の効率化

クラスの継承による再利用性や拡張性、カプセル化による堅牢性、クラスを単位とした設計モジュールの分割など、オブジェクト指向の特長を回路設計に生かすことができる。

(3) ソフトウェア資源の有効活用

Java 言語で記述された既存のアルゴリズムを Jackal 言語に書き換えることにより、迅速な回路化が可能となる。

(4) Java 環境下でのシミュレーション

JavaVM 上でのシミュレーションにより、メモリリークなどが発生せず、回路の検証に集中することが出来る。

Jackal 言語を用いたシステム LSI 設計では、これらの利点により設計の効率化と生産性の向上が見込める。

2.2 システム LSI 設計フロー

Jackal 言語によるシステム LSI 設計フローを図 1 に示す。

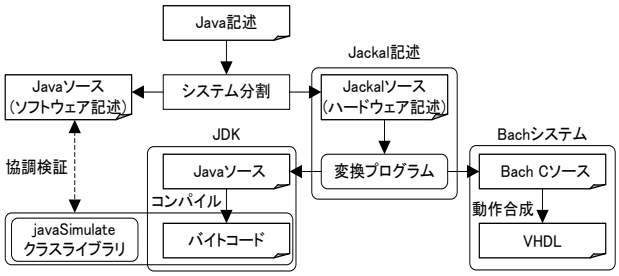


図 1　Jackal 言語によるシステム LSI 設計フロー

設計の手順として、まず Java 言語で記述されたソースコードをハードウェア部分とソフトウェア部分に人手により分割を行い、ハードウェア部分の Java 記述を Jackal 記述に書き換え、最適化する。書き換えた Jackal 記述を Jackal-Java 変換により、ソフトウェア部分との協調検証を行う。

また、Jackal 記述から Jackal-Bach 変換により Bach C 記述を生成し、Bach システムを用いて VHDL 記述を自動合成する。Jackal-Bach 変換の詳細については付録に掲載する。

Bach システムとは、シャープ株式会社で開発された C 言語ベースの動作合成システムであり、ANSI C を拡張した Bach C 言語でハードウェア記述を行い、シミュレーションにより動作を確認した後、VHDL 記述を自動合成するシステムである。

3　Jackal 言語によるハードウェア記述

Jackal 言語は、Java 言語をハードウェア設計に用いるため、以下の拡張を行っている。

3.1 符号とビット幅指定

ハードウェア設計では、ソフトウェアと違い、変数など使用する資源の量が回路規模や処理速度に影響する。Jackal 言語では Java 言語の構文を拡張し、整数型変数の符号の有無、ビット幅の指定を可能にしている。

† 近畿大学大学院 総合理工学研究科
エレクトロニクス系工学専攻
‡ 近畿大学　理工学部電気電子工学科
†† 株式会社メガチップス

3.2 連結演算子、ビット選択演算子

Jackal 言語では 2 つの整数を 1 つの整数に連結する連結演算子“@”、ある整数から指定したビットを取り出して新しい整数を作るビット選択演算子“[..]”を扱えるように機能拡張を行い、ハードウェア記述を容易にしている。

3.3 並列動作と同期通信

Jackal 言語では Java 言語のスレッドによる並列動作の記述が可能である。これにより設計者は通常の Java 言語と同様にスレッドを用いて並列動作を記述できる。並列動作を行う際、独立して動作するプロセス間での通信が必要となる。図 2 に並列動作と通信の例を示す。

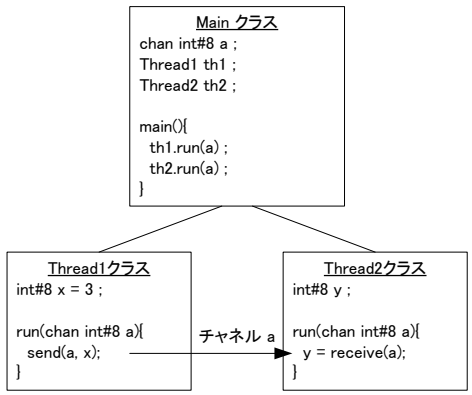


図 2　並列動作と同期通信イメージ

通信に使用する変数はチャネル変数として他の変数と区別する必要があり、キーワード chan を用いて変数を宣言する。send メソッドと receive メソッドの 2 つを用意しており、チャネル変数と併せて使用することで同期通信を実現している。図 2 では Thread1 クラスは変数 x の値をチャネル変数 a を通じて送信する。受信待ちしていた Thread2 クラスは a に送られてきた値を受信し、変数 y に代入する。

3.4 動作合成オプションプラグマ

Jackal 言語では Bach システムで合成する際のプラグマオプションを記述することができる。プラグマの種類としては、RAM／ROM 指定用、同期方法指定用、通信タイミング指定用、専用回路指定用のプラグマなどがある。これらを適切に用い、目的とする回路が生成できる。

4　Jackal 言語によるオブジェクト指向設計

オブジェクト指向言語にはデータと処理をまとめる言語構文であるクラスという概念がある。クラスを単位として再利用でき、また整理されているため、プログラムの可読性も向上する。さらに容易に修正･改造を行え、開発期間などの短縮にもつながり、VHDL や手続き型で設計するより、効率的に設計することが期待できる。

Jackal 言語ではこのオブジェクト指向の利点を生かした回路設計が可能である。以下に基本的なオブジェクト指向設計の種類と特徴、および回路への応用について述べる。

4.1 継承

Jackal 言語では継承を用いて記述することができる。継承の利点としてプログラムの保守性、再利用性の向上が挙げられる。継承を用いていないプログラムでは、いくつかのメンバ(フィールドやメソッド)が重複し、同じメンバを何度も記述するという無駄がある。継承によって、共通のメンバをスーパークラス、独自のメンバをサブクラスとすることにより、記述量を削減することができる。また、1 つのスーパークラスの内容を修正または改造すれば、それを継承しているクラスへも反映でき、保守効率も向上する。

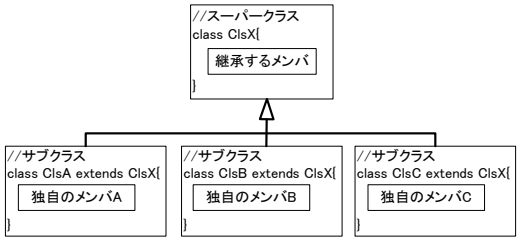


図 3　継承によるプログラムの再利用

4.2 継承の回路への応用

継承による回路合成の最適化について述べる。従来の Jackal システムでは、継承したクラス(以下スーパークラス)が同一スレッドにある場合、他クラスの処理とも演算器を共有しつつ回路を生成していた(図 8)。この演算器の共有化によって演算に待ち時間が発生し、処理時間が低速になる場合がある。また、スーパークラスの再利用回数に応じて回路がインスタンス化され、回路規模の増大に繋がる危険があった。

本文では、スーパークラスを専用回路として合成する方法を考える。専用回路化とは、指定した処理全体(ここではクラス)を組み合わせ回路の演算器として実現することである。これにより、回路規模は増大せず、かつ他のブロックとの回路共有もないため、待ち時間の発生が軽減され、処理時間の高速化が見込まれる。

4.3 カプセル化

Jackal 言語ではカプセル化を用いて設計できる。アクセスする変数やメソッドは公開し、それ以外は隠蔽することで入出力関係を明確にできる。また、Jackal 言語では図 4 のような間接的なアクセス(アクセサメソッド)を容易に設計することができる。これによりアクセスしてはいけない変数に誤ってアクセスする事を防ぐ。カプセル化によって入出力関係が明確になっているので、流用設計も容易になる。

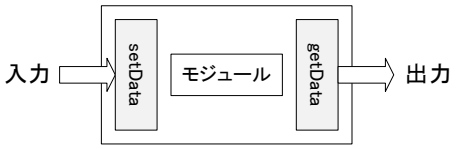


図 4　カプセル化による入出力の明確化

4.4 オーバーロード

Jackal 言語ではオーバーロードを用いて設計できる。オーバーロードとはオブジェクト指向言語の利点であるポリモーフィズム(多態性)の 1 つであり、この手法を用いて設計することでプログラマはそれぞれのメソッド独自の機能などを覚えることが少なくて済む。また、図 5 のようにメソッ

ド名を統一することでコードの可読性および保守性を向上させ、バグを減らすことができる。

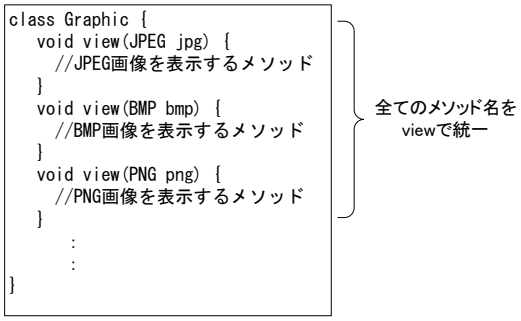


図 5　オーバーロードを適用した設計

5　設計結果とその評価

本章では、継承とカプセル化について実際の回路を用いた設計とその評価について述べる。

5.1 継承の適用結果

JPEG 符号化に用いる離散コサイン変換（DCT）回路を用いて継承の比較実験を行った。入力データには 320×240 画素のフルカラー画像を用いた。Jackal 記述の構成は処理に用いる基本的な変数の初期化を行う DCT クラス、パイプライン化された演算を制御する Chen クラス、Chen アルゴリズムの高速演算を行う Step クラスで構成されており、Chen クラスは Step クラスを継承しており、DCT クラスで Chen クラスをインスタンス化して処理を実行する。

表 1 に継承を適用していない記述と適用した記述における記述量の比較を示す。動作合成は動作周波数を 100MHz とし、ライブラリは日立 0.18μ ライブラリを用いた。

表 1　継承適用による記述量の比較

	記述量[行]
継承なし	264
継承あり	234

重複する記述が 1 つのクラスとしてまとめられ、再利用されることで記述量が削減された。

次に、従来の回路とスーパークラスを専用回路とした合成結果を表 2 に示す。

表 2　継承の検証結果

	回路規模 [ゲート]	処理時間 [μs]
通常の継承	394,020	6,537
専用回路化した継承	319,385	4,521

専用回路化によって、約 19%の回路規模が削減され、また約 2ms の高速化となった。従来の実現法では、160 個の演算器(加算器、乗算器など)が生成され、さらにこれら多数の演算器を使用して処理することで 13 サイクルかかっていた。これが専用回路化されることで演算器が 11 個の専用回路としてまとめられ、処理時間も 7 サイクルに高速化された。これはインスタンス化された回路の重複が削除され、回路共有による待ち時間が軽減されたためであると考えられる。

5.2 カプセル化の適用結果

JPEG 符号化回路を用いてカプセル化の適用を行った。入力データは継承の検証と同じ画像を用いた。検証に用いた JPEG 符号化回路は色空間変換、離散コサイン変換(DCT)、量子化、ハフマン符号化を 6 並列 4 段パイプラインで行う。これらの中で、DCT、量子化、ハフマン符号化の各処理についてカプセル化を適用した。その回路の合成結果を表 3 に示す。

表 3　カプセル化の検証結果

	回路規模 [ゲート]	処理時間 [μs]
カプセル化なし	1,841,330	4,387
カプセル化適用	1,841,330	4,387

カプセル化を適用することで回路規模・処理時間共に変化は見られなかった。これは Bach システムにより間接アクセスを行うためのアクセサメソッドを回路化しないように最適化したためであると考えられ、回路規模の増大や処理のオーバーヘッドは発生しなかった。

6　まとめ

Java 言語をハードウェア向けに拡張した Jackal 言語を用い、オブジェクト指向設計として継承とカプセル化を適用した回路設計とその評価を行った。

今後はさらにポリモーフィズムなどを含む Jackal 言語によるオブジェクト指向設計の実現と大規模回路による評価を行う。

謝辞

Java ベースシステム LSI 設計言語“Jackal”を開発するに当たり、多大なるご指導を頂いたシャープ株式会社電子デバイス開発本部先端技術開発研究所 Bach 開発グループの皆様に心から御礼申し上げます。

文献

[1] K.Okada，A.Yamada，T.Kambe，“Hardware Algorithm Optimization Using Bach C”，IEICE Trans. Fundamentals vol.E85-A，No.4，pp.835-841 (2002).

[2] T.Kambe，A.Yamada et al.，“A C-based Synthesis System, Bach and Its Application”，ASP-DAC2001，pp.151-155 (2001).

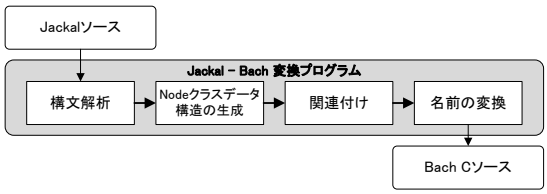
[3] 黒坂 均、竹村 和祥、橘 昌良、“システムレベル設計フローと設計言語”、情報処理、Vol.45，No.5，pp.456-463 (2004).

[4] 小原 貴文、寺井 裕幸、枡岡 正悟、山田 晃久、神戸 尚志、“オブジェクト指向を用いたシステム LSI 設計手法とその評価”、2007 年度 電子情報通信学会技術研究報告　信学技報 Vol.107 No.508、pp.59-64(2008).

[5] 五月女 健治、“JavaCC－コンパイラ・コンパイラ for Java”、テクノプレス、(2003).

付録　Jackal 記述から Bach C 記述への変換

Jackal 記述から Bach C 記述への変換にはオブジェクト指向言語から手続き型言語への変換という問題を解決しなければならない。それは手続き型言語にはデータと処理を関連付けてひとまとめにする「クラス」という概念がないためであり、変換には関連を壊さずにデータと処理を分離する必要がある。我々はこれらの問題を解決し、一連の変換を自動的に行う Jackal-Bach 変換を実現した。Jackal-Bach 変換の処理手順を付 1 に示す。

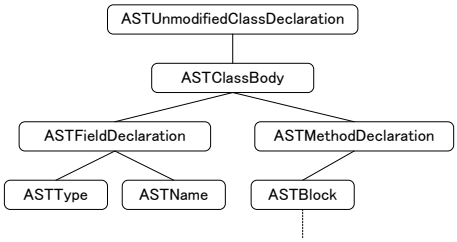


付 1　Jackal-Bach 変換の処理手順

Jackal-Bach 変換プログラムは全部で 145 クラスから成り、ステップ数は 19,769 である。また、継承の検証に用いた DCT 回路の Jackal 記述を Bach C 記述に変換する際に要した時間は約 2 秒（動作周波数 2.26GHz）で、出力された Bach C 記述が Bach システムによる動作合成により期待する回路が生成されることを確認した。

● 構文解析

構文解析には JavaCC[5]を用いて生成した構文解析プログラム(パーザ)を用いた。JavaCC はトークンや生成規則と呼ばれる構文上の要素を指定することで、構文解析を行う Java プログラムを自動生成する。これにより Java 言語で記述された Jackal-Bach 変換の各処理と構文解析プログラムを連携させることができる。JavaCC による構文解析の結果は、付 2 に示す抽象構文木を表す AST(Abstract Syntax Tree)の形で得られる。

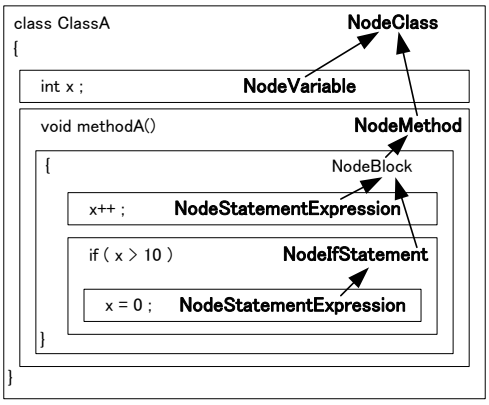


付 2　生成される AST の例

この AST の各ノードは生成規則固有のクラスで構成され、「AST＋生成規則名」の名前で定義される。

● Node クラスデータ構造

構文解析によって生成された AST は解析した記述の文法としての構造を表す。Bach C 言語へ変換する際、クラスの継承関係など構文の意味的な関係を表す必要があるため、AST を解析して一旦 Node クラスと呼ぶ独自のデータ構造を生成する。これにより文法的なデータだけを持った構文解析結果から、構文同士の関係性を持った構文データの表現が得られる。Node クラスの構造を付 3 に示す。

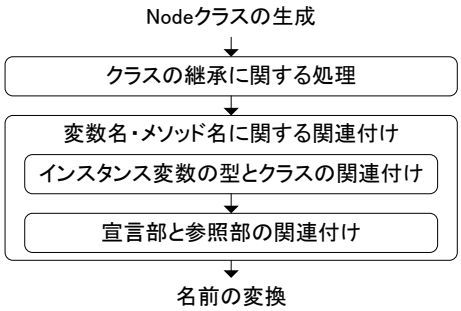


付 3　Node クラスデータ構造

Jackal 言語の構文において、クラス内にメソッドやフィールドが属し、メソッドがブロックを、ブロックがステートメントを、というようにそれぞれが包含されている。全てのメソッドはクラス内で定義されるため、どのような記述であっても必ず最後にはクラスに属する。よって最終的にクラスを単位とするデータ構造が形成される。そのクラスのデータを Node クラス内に格納し、記述中に定義されているクラスの数だけその内容を Node クラスに格納していく。

● 関連付け

Node クラスの生成によって構文データの関係性の表現を行ったが、この時点での Node クラスは構文上の包含関係の表現に過ぎず、Node クラス構造同士の繋がりが無いため、Node クラス間においてクラスの継承、メソッドの呼び出し、変数の参照などの関連付けを段階的に行う必要がある。付 4 に示す手順で関連付け行い、Node クラス構造同士に関係を持たせる。



付 4　関連付け処理の手順

● 名前の変換

Jackal 言語や Java 言語ではメソッド及び静的変数はクラスごとに管理されるため、異なるクラスで同じ名前を用いた宣言が可能であるが、これらの名前は Jackal-Bach 変換によってグローバルな変数及び関数に展開されるため、名前の重複が生じないように書き換える必要がある。Jackal-Bach 変換では、メソッドとフィールド変数についてそれぞれの名前の先頭に「クラス名」と下線「_」を追加することで名前の変換を行い、最後に Node クラスを順に取得しながら Bach C ソースファイルへ書き出しを行う。