

動作記述型シリコン・コンパイラにおける制御構造の実現方式†

平 山 正 治††

プログラミング言語等による動作レベルの高位記述から VLSI チップの製造情報を直接生成する動作記述型シリコン・コンパイラは、VLSI チップの開発効率を飛躍的に改善する新しい設計手法として期待されている。しかし、手続き的な動作記述から従来方式のハードウェア構造をソフトウェア的に自動生成することは容易ではなく、シリコン・コンパイラを実現するために有効な VLSI チップ向きのアーキテクチャを開発する必要がある。我々は、内部に制御回路を含んで独立に動作する複数の要素で構成され、これらが局所的に通信し合うことで所定の動作を実現する分散制御型の VLSI アーキテクチャを提案し、並列動作も記述できる手続き型のハードウェア仕様記述言語による動作記述から、このアーキテクチャに基づく VLSI チップを自動生成するシリコン・コンパイラの試作研究を行っている。このアーキテクチャでは、入力言語の各制御構文に対応して、それらの制御動作を専用に実行する制御要素を導入しているが、これまでの実験結果から、手続きの呼出しや並列に動作するプロセスの制御等の記述をこれらの制御要素の動作に容易に展開でき、かつ、各制御要素とも比較的簡潔な構造で実現できることが示された。これにより、動作記述型シリコン・コンパイラの実現方式として、分散制御型アーキテクチャの有効性を明らかにすることができた。

1. ま え が き

近年の半導体デバイス技術の進歩によって、百万個以上のトランジスタを含む VLSI チップが現実のものとなり、これに伴って広範な応用分野でのシステムの VLSI 化が急速に進展していくと考えられている。しかし、少量多品種という性格をもつ大規模な専用チップを最小限の労力によって短期間で開発するためには、チップの設計効率を飛躍的に改善する必要がある¹⁾。この問題を解決するための有望な手法として、VLSI チップの高位記述を入力してチップの製造情報を生成する“シリコン・コンパイラ”と呼ばれる VLSI 設計自動化ソフトウェアが提案されている²⁾。

現在の技術レベルでは、動作レベル等の高位記述から、人手によって設計されたものと同程度に最適化されたチップを自動的に生成することはほとんど困難であり、シリコン・コンパイラの入力形式の違いによって以下に示す2つのアプローチからの研究開発が進められている³⁾。

① 構造記述型シリコン・コンパイラ

VLSI チップの論理的な構成要素と接続関係の記述を受け入れ、これらから物理的な構成要素への展開と配置・配線を自動化するシリコン・コンパイラであり、生成されるチップの性能面（面積、速度）の最適化を重視している。

② 動作記述型シリコン・コンパイラ

プログラミング言語等による動作レベルの高位記述を受け入れ、その記述によって示された機能をもつ VLSI チップを自動生成するシリコン・コンパイラであり、開発期間の大幅な短縮を狙っている。

前者のシリコン・コンパイラは、これまでの VLSI 開発手法の延長線上にあって、これを効率的に支援するツールとして、実用化されつつある。一方、後者のシリコン・コンパイラは、まだ研究段階にあるのが現状であるが、VLSI チップの設計効率を飛躍的に向上させる新しい設計手法として、今後の発展が期待されている⁴⁾。

これまでに発表されている動作記述型シリコン・コンパイラでは、プログラミング言語等による動作記述からデータの依存関係や演算の並列性を解析し、VLSI チップとしてのサイズや動作速度の観点から最適なデータパスを抽出することに研究の主眼がおかれ、複数の演算・記憶要素からなるデータパスを1個の制御回路で管理する集中制御方式のチップを前提としているものが多い^{5), 6)}。しかし、手続き的に記述された大規模なプログラムから、チップ全体としての最適なデータパスの構成を決定し、高度の並列動作を実現する複雑な演算・制御回路を自動的に生成することは多大な計算時間を要する困難な処理である。また、チップ全体が単一のクロック信号に同期して動作する集中制御方式のチップでは、制御信号のスキューが発生しないようなチップ上での配置や配線長に対する十分な配慮が必要となる。

このような VLSI チップ内部の動作における時間依存性を回避し、将来の大規模チップを実現するための

† An Implementation Scheme of Control Structure on Behavioral Silicon Compiler by MASAHARU HIRAYAMA (Central Research Laboratory, Mitsubishi Electric Corporation).

†† 三菱電機(株)中央研究所

ジュールに送られて何らかの演算処理され、演算結果のデータは記憶モジュールに返送されて保持される。各演算モジュールとこれに関連するすべての記憶モジュールとは個別の専用配線で結ばれ、演算モジュールの前段に置かれた入力セクタによって必要なデータが選択される。一方、演算モジュールで生成される演算結果はデータバスと呼ぶ1本の共通バスを経由して記憶モジュールに返送される。

(3) 制御の流れ

各機能モジュール（演算モジュールと制御モジュール）は、起動時に指示される起動条件に応じて何らかの演算・制御動作を実行し、この動作が完了後、特定の起動条件を指示して次の機能モジュールを起動する。このように各機能モジュールが他の機能モジュールを次々に起動していくことにより、チップ全体としての一連の動作が実現される。このときに用いられるモジュール番号と起動条件の組を“起動情報”(MNFC)と呼び、この起動情報も制御バスと呼ぶ1本の共通バスを経由して転送される。

(4) チップ外部との通信方式

本アーキテクチャでは、チップ外部との通信のために2本の共通バス（データバスと制御バス）を外部に引き出す方式を採っている。したがって、このアーキテクチャに基づく複数のチップ間でも、データの送受や機能モジュールの起動をチップ内部と同様の方法で行える。この機能を利用して、チップの実行に必要なデータを設定したり、実行結果をチップ外部に取り出すことができる。また、チップの外部から最初の機能モジュールを起動することによって、このチップの動作が開始され、機能モジュールの起動状況を監視することによって、チップの動作終了を検知することができる。

ここで提案した分散制御アーキテクチャの論理的な構造を図2に示す。

2.2 記憶モジュールと演算モジュール

ISPS では、外部から見える構造と内部動作とが規定される構成要素のネットワークとしてデジタル・システムを記述する。このうち、動作が規定されず、構造だけが定義され

る構成要素は、データを保持するための“記憶要素”を表している。本アーキテクチャにおける記憶モジュールは、この記憶要素を実現するものであり、ユーザが記述した1個の記憶要素は、本アーキテクチャに基づくVLSIチップの中で1個の記憶モジュールとして実現される。本アーキテクチャでは実現上の容易さから、記憶要素の構造をレジスタやフリップフロップのような1次元の構造に限定している。

ISPS における演算式の記述に関しては、1個の演算子の記述が1個の演算モジュールの動作として実現される。通常、各演算モジュールには複数の動作が割り当てられるため、これが起動されたときに指示される起動条件ごとに、そのときの詳細動作を規定する情報（動作パラメータ）が演算モジュール内部に格納されている。代表的な演算モジュールの動作パラメータとしては、入力データを選択するためのセクタのポート指定、演算種類の指定、結果を書き込む記憶モジュールの指定、および、1組の起動情報がある。演算モジュールの種類と ISPS の演算子との対応を表1に示す。

2.3 制御モジュール

本アーキテクチャでは、ISPS における制御構文を実現するために、以下に示す5個の制御モジュールを用意している。これらの制御モジュールの機能と動作パラメータを表2に示す。

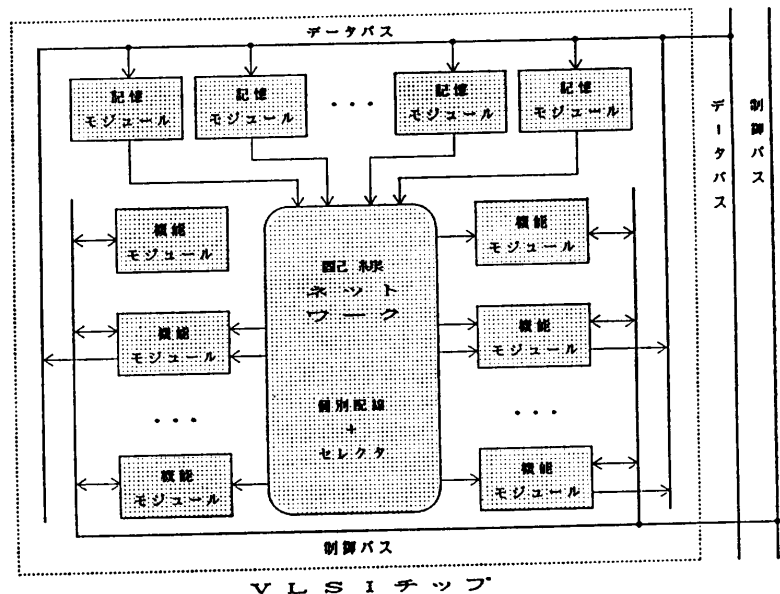


図2 分散制御アーキテクチャの論理構造

Fig. 2 Logical structure of distributed control architecture.

(1) 条件分岐

ISPS には “if” と “decode” という 2 種類の条件分岐構文がある。本アーキテクチャでは、これらの構文は各々 IF モジュールと DECODE モジュールの動作によって実現される。すなわち、IF モジュールは 1 ビットのデータを入力し、この値に応じて次に起動する機能モジュールを選択する。一方、DECODE モジュールは n ビット長のデータに対して 2^n 通りの分岐先が指定できる。したがって、IF モジュールの動作パラメータには 2 組の起動情報を含み、DECODE モジュールの動作パラメータには最大 2^n 組の起動情報を含んでいる。

(2) 並列動作の制御

ISPS においては、複数の動作を “;” で結合することにより、これらの並列動作を指定することができる。本アーキテクチャでは、このような並列動作の記述は FORK モジュールと SYNC モジュールの動作によって実現される。すなわち、FORK モジュールは 1 回の動作で複数の機能モジュールを起動し、SYNC モジュールは、指定された個数のモジュールから起動されるまで、次の機能モジュールの起動を待ち合わせ

表 1 演算モジュール
Table 1 Operational modules.

モジュール名	機 能	対応する ISPS の演算子
LTRANS	データ転送	—, =
ATRANS	符号拡張データの転送	<=
LOGIC	論理演算	or, xor, and, eqv, not
COMP	比較演算	eq, neq, lss, leq, gtr, geq
SHIFT	シフト演算	sl0, sl1, slr, sld, srl, srr, srd
ADDER	加減算	+, -
MULTI	乗算	*
CNT	カウンタ動作	+1, -1
ALU	算術論理演算	+, -, +1, -1, or, xor, and, eqv, not
ROM	ROM	(ユーザがモジュールの内容を定義する)
RAM	RAM	(ユーザがモジュールの内容を定義する)
PLA	PLA	(ユーザがモジュールの内容を定義する)

る。したがって、FORK モジュールの動作パラメータには n 組の起動情報を含み、SYNC モジュールの動作パラメータには同期する並列動作の個数 n と 1 組の起動情報を含んでいる。

(3) 動作要素の制御

ISPS 記述において構造と動作の両方が定義される構成要素（これを“動作要素”と呼ぶ）は、通常のプログラミング言語における“手続き”に対応するものである。しかし、これに“process”という修飾子を付与した場合には、並列動作が可能な“プロセス”を意

表 2 制御モジュール
Table 2 Control modules.

モジュール名	機 能	動 作 パ ラ メ ー タ
IF	条件分岐 (2 方向)	入力セレクトのポート番号*, 2 組の起動情報
DECODE	デコード (2^n 方向)	入力セレクトのポート番号*, 最大 2^n 組の起動情報 (n : データのビット幅)
FORK	並列動作の起動	n 組の起動情報 (n : 並列動作数)
SYNC	並列動作の同期	並列動作数 n , 1 組の起動情報
PRO-CONT	以下に示す動作要素の制御動作	すべてに共通: 動作要素番号**, 動作要素の型**
① ACTIVATE	手続きの呼出し	2 組の起動情報 (飛び先, 戻り先)
② LEAVE	プロセスの起動	2 組の起動情報 (飛び先, 次)
	手続きからの復帰	(なし)
	プロセスの終了	(なし)
③ RESTART	ラベル付き動作の終了	1 組の起動情報
	動作要素の再スタート	1 組の起動情報
④ TESTSET	動作要素の動作状況チェック	記憶モジュール番号**, 1 組の起動情報

起動情報: モジュール番号と起動条件の組 (MNFC)

*1: 入力データを選択するためのセレクトが組み込まれている

*2: すべての動作要素は一意的な番号が付与され、動作状況が管理される

*3: 手続き, プロセス, ラベル付き動作の区別

*4: 動作状況 (0/1 の値) が記憶モジュールに格納される

味し, “critical” という修飾子を付与した場合には, 複数のプロセスから同時に起動されることが禁止される “排他的動作要素” を意味する. また, 複数の動作をまとめてラベル付けしたもの (“ラベル付き動作”) も動作要素と同様に取り扱われる. 本アーキテクチャでは, これらに対する制御動作を実現するために **PRO-CONT** モジュールが用意されている. このモジュールは, 物理的には1個のモジュールとして実現さ

れているが, 論理的には以下に示す4個の制御モジュールの集合体と考えられる.

- ① **ACTIVATE** モジュール: 動作要素を起動する
- ② **LEAVE** モジュール: 動作要素を終了する
- ③ **RESTART** モジュール: 動作要素を再スタートする
- ④ **TESTSET** モジュール: 動作要素の動作状況をチェックする

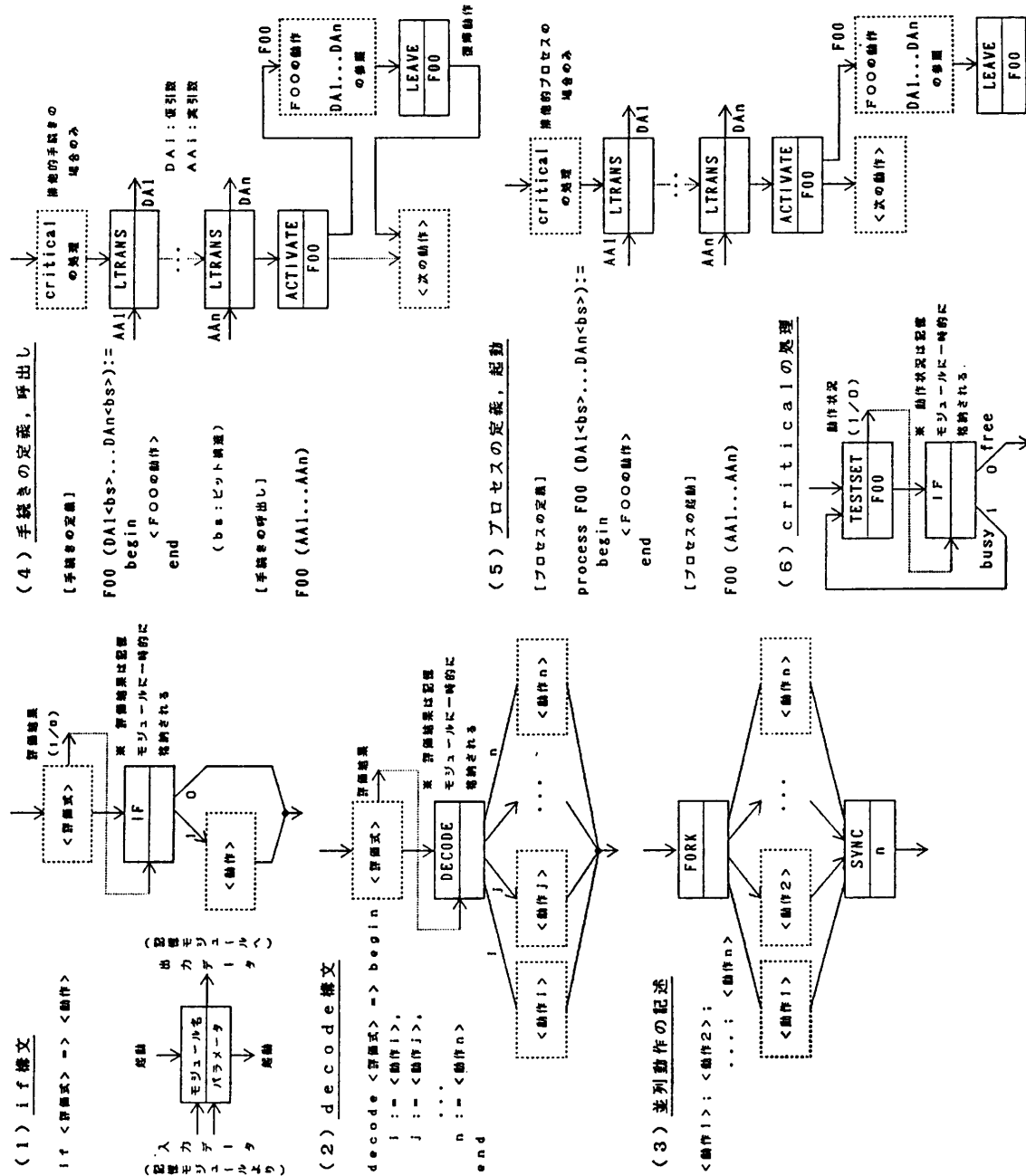


図 3 ISPS 記述から分散制御アーキテクチャへの変換
Fig. 3 Translation from ISPS description to distributed control architecture.

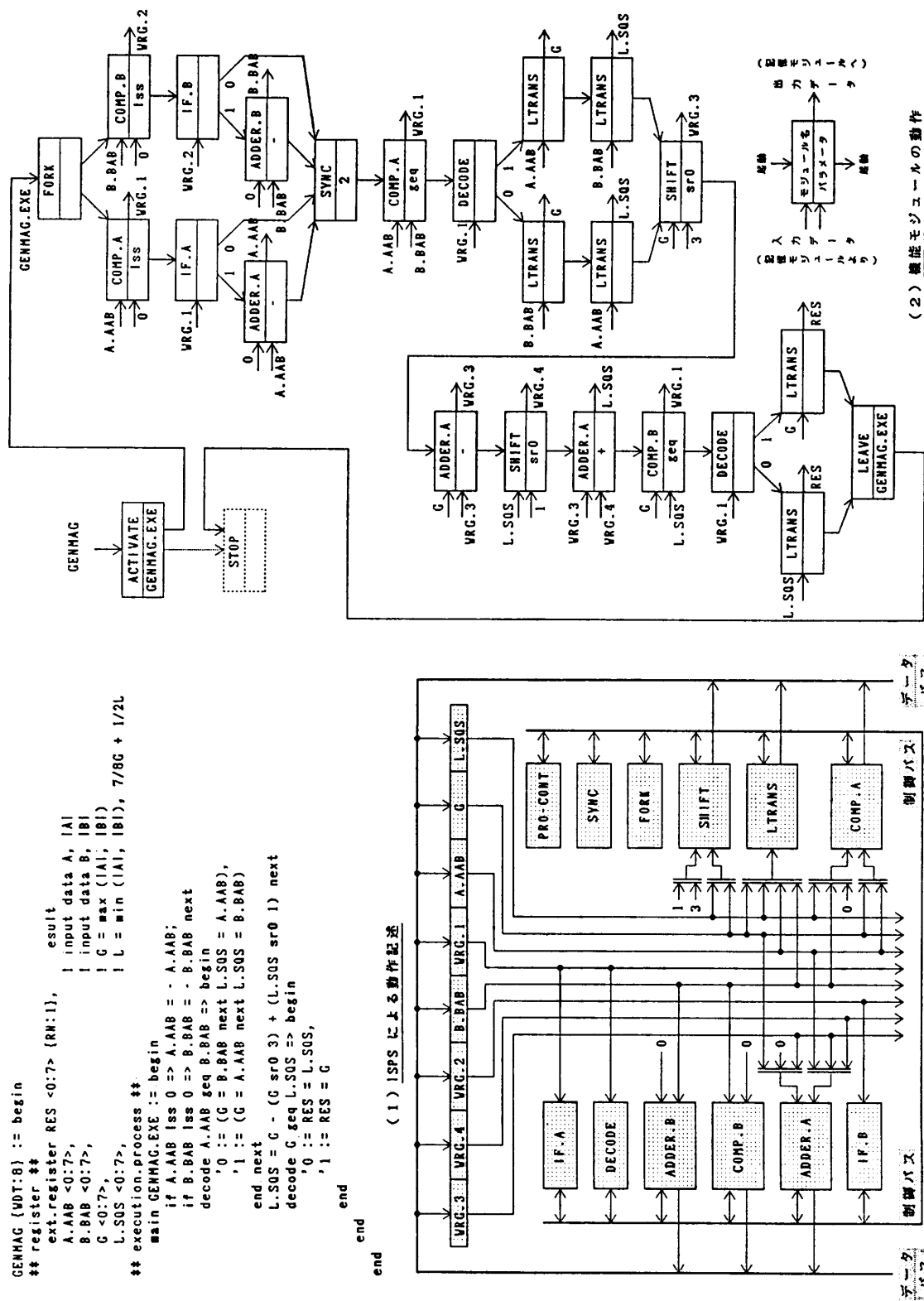


図 4 設計例—近似計算チップ⁽¹⁾—
 Fig. 4 Design example—GENMAG chip—

(3) チップの構成

3. 動作記述からの変換方法

本章では、シリコン・コンパイラの入力形式である ISPS 記述から、前章で示した機能モジュールによる動作への変換方法について述べる。ただし、演算式の変換は各演算子に対応する演算モジュールの動作に変換するだけであり、ここでは ISPS における各制御構文の変換方法について説明する。この変換方法の概要を図 3 に示す。

if, decode 構文では〈評価式〉として任意の算術・論理・比較演算を許すため、この式の計算を演算モジュールの動作によって実現し、評価結果を記憶モジュールに一時的に格納する。IF, DECODE モジュールでは、この記憶モジュールの内容を取り込み、その値に応じて対応する〈動作〉の最初の機能モジュールを選択的に起動する。

“;” で結合される並列動作に対しては、最初に FORK モジュールの動作を挿入し、このモジュールが各〈動作〉の最初の機能モジュールを連続的に起動する。各〈動作〉の最後では、並列動作の同期をとるために SYNC モジュールを起動する。

手続きとプロセスの定義に対しては、仮引数に対応する記憶モジュールが生成され、これらを参照した動作に展開される。また、この記述の最後や終了構文に対して、復帰動作（手続きの場合）や終了動作（プロセスの場合）を実現する LEAVE モジュールの動作が挿入される。ISPS においては leave, terminate, restart, resume という 4 種類の終了構文が用意されているが、本アーキテクチャでは、通常の終了を意味する leave と再スタートを指示する restart だけに限定し、他のプロセスを強制終了するための terminate やコルーチンのための resume は使用できない。

手続きの呼出しとプロセスの起動の記述は、実引数の記憶モジュールの内容を仮引数の記憶モジュールへデータ転送する動作と、対象とする動作要素に制御を移すための ACTIVATE モジュールの動作に変換される。ただし、排他的動作要素を起動する場合は、TESTSET モジュールと IF モジュールの動作が挿入され、動作要素の終了を待ち合わせる。この TESTSET モジュールの動作は、動作要素の動作状況を 0/1 で応答すると同時に、動作要素の動作状況を示すフラグを動作中にセットする Test & Set の機能をもっている。

分散制御アーキテクチャに基づく VLSI チップの具

体的な例として、以下に示す近似値計算アルゴリズムを実行する GENMAG チップ¹¹⁾の ISPS による動作記述、前述の変換規則によって展開された機能モジュールの動作、および、実際のチップの構成を図 4 に示す。図 4 (3) に示した記憶モジュールと機能モジュールの構成や構成要素間の接続情報は、機能モジュールの動作に展開された結果を解析することにより容易に抽出できる。

$$\sqrt{a^2+b^2} \approx \max\left(g, \frac{7}{8}g + \frac{1}{2}l\right)$$

$$g = \max(|a|, |b|), l = \min(|a|, |b|)$$

以上のように、ISPS の制御構文から機能モジュールによる動作への変換規則は単純なものであり、かつ、いずれも局所的な構文だけを対象にすればよく、簡単な変換プログラムによって容易に実現できる。実際のコンパイラ・プログラムは図 5 に示すように、ユーザからの構成要素に関する最適化の指示を受けながら ISPS のソース・プログラムを機能モジュールの動作に変換するソース・プログラム変換部 (ISPS 構造エディタを含む)、ISPS 記述を構文解析して中間データ構造 (GDB 形式⁹⁾) に展開する ISPS 構文解析部、チップとしての構成要素と制御情報や接続情報

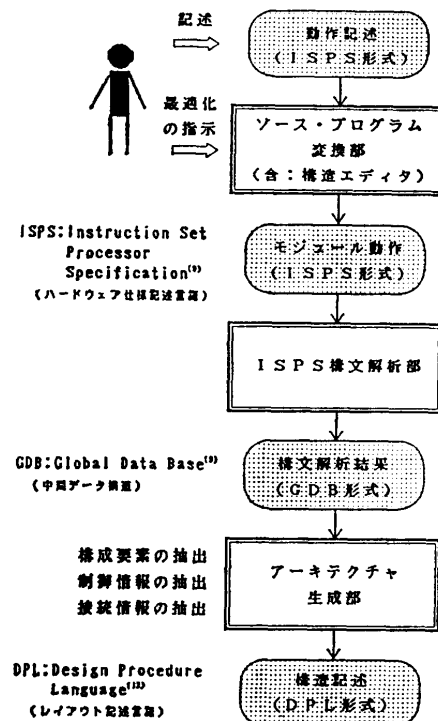


図 5 シリコン・コンパイラの構成
Fig. 5 Configuration of silicon compiler.

を抽出し、DPL 形式¹²⁾の構造記述を生成するアーキテクチャ生成部の3個の部分から構成されている。本章で示した変換方法は主としてソース・プログラム変換部に組み込まれているが、現在のプログラムでは、記憶モジュールや演算モジュールの統合、ALU やカウンタの利用、並列動作のための演算モジュールの併置等の最適化の可能性が残されており、現在は利用者の指示に従って最適化している。ISPS構文解析部はISPSシステムのものを利用しているが、他の2つの部分は今回試作したものであり、各々1,900行、2,400行のLisp プログラムで実現されている。

4. 制御モジュールの実現方式

本章では各制御モジュールの内部構造(図6参照)とその動作について述べる。各制御モジュールの実現方式は多数考えられるが、ここではシリコン・コンパイラとしてソフトウェア的に実現しやすいように、ROM、RAM 等のメモリ類やレジスタ、セレクト等、パラメータ化しやすい構成要素を利用することを前提とした実現方式を示している。

IF モジュールでは、選択された1ビットの入力データの値に応じて、制御テーブルに格納されている2組の起動情報のうちのどちらか一方を選択し、この値によって次の機能モジュールを起動する。DECODE モジュールでは、選択された入力データの値と起動情報が格納されている起動情報テーブルのベース番地を加算して、多数の起動情報の中から1個の起動情報を選択し、この値によって次の機能モジュールを起動する。

FORK モジュールでは、制御テーブルに格納されている起動情報テーブルのベース番地によって、これに格納されている起動情報を次々に読み出し、終了コードが検知されるまで、複数の機能モジュールを連続的に起動する。一方、SYNC モジュールにおける同期メカニズムは、完了した並列動作数を保持するテーブルの内容(初期値:ゼロ)と待ち合わせる並列動作数とが比較され、これらが一致したときに次の機能モジュールを起動する方式である。

PRO-CONT モジュールには、動作要素に関する制御動作を実現するために、起動条件ごとの動作パラメータを保持した制御テーブルに加えて、動作要素ごとに動作中かどうかを示すフラグと復帰時の起動情報を格納するためのレジスタ(手続きの場合のみ有効)とからなる動作要素テーブルが組み込まれている。手

続きの呼出しの場合は、復帰時の起動情報を動作要素テーブルに格納するとともに、呼び出された手続きの最初の機能モジュールを起動する。一方、プロセスの起動に関しては、起動されたプロセスの最初の機能モジュールと起動側の次の機能モジュールの両方を起動する。手続きからの復帰の場合は、動作要素テーブルに格納されている復帰時の起動情報を取り出し、この値によって次の機能モジュールを起動する。一方、プロセスの終了の場合は単に動作中フラグをリセットするだけである。また、ラベル付き動作の終了の場合は、この動作の直後の機能モジュールを起動する。RESTART モジュールの動作は各動作要素の最初の機能モジュールを再起動する。TESTSET モジュールの動作は、動作要素テーブルの動作中フラグの値を記憶モジュールに転送するとともに、この時点で動作中フラグの値を“動作中”にセットする。

以上のように、いずれの制御モジュールとも、レジスタ、ROM、RAM 等の記憶回路、加算器、比較器、インクリメント、セレクト等の演算回路、および、簡単な制御回路から構成されている。これらの構成要素は、ビット幅、語数等のいくつかのパラメータをもつ汎用的なモジュールとしてライブラリ化しておくことができ、シリコン・コンパイラのアーキテクチャ生成部から出力される制御情報に従ってパラメータの値を設定することにより、実際のチップごとに調整された制御モジュールを容易に生成することができる。

5. む す び

本稿では、動作記述型シリコン・コンパイラのターゲット・アーキテクチャとして、内部に制御回路を含んで独立に動作する要素で構成され、これらが局所的な通信を行いながら所定の動作を実現する分散制御アーキテクチャについて述べた。一般的には、並列プロセスの実行環境下で、手続きのリンケージや排他制御等をハードウェア的に効率良く実現することは容易でないが、ここで提案した分散制御アーキテクチャを適用することにより並列動作記述言語における各種の制御構文を制御モジュールの動作に容易に変換でき、かつ、これらの制御モジュールがシリコン・コンパイラを実現するのに適したパラメータ化された構成要素の組合せによって実現できることを示した。これにより、動作記述型のシリコン・コンパイラを実現する手法として、手続き的な動作記述をVLSIチップの構造に直接反映できる分散制御アーキテクチャの有効性を

明らかにすることができた。

ここで述べた分散制御アーキテクチャに基づくシリコン・コンパイラは、各応用分野のシステム設計者やプログラマ等が対象システムの動作を簡潔に記述するだけで、その機能を実現する VLSI チップを容易に実現でき、変換の容易性や構成要素の実現性等が優れている。しかし、分散制御アーキテクチャに基づくチップでは、制御機能を各機能モジュールに分散したことや、比較的多数のデータ配線を必要とすることなどから、手書きで設計されたチップよりも大きな面積を必要とする¹³⁾。また、分散制御アーキテクチャでは、相手の機能モジュールが動作可能であること確認し合うことなどのモジュール間通信のオーバーヘッドにより、集中型の制御方式に比べて動作速度が低下してしまうという問題もある。したがって、様々な記述から導かれるチップの構成要素を常にコンパクトにする物理的な実現方式や配置・配線アルゴリズム、および、機能モジュール間の通信オーバーヘッドを低減し、モジュール動作の並列性を高める等の高速化手法について、今後とも検討を重ねていきたいと考えている。

謝辞 本研究を行うにあたって、日頃より御指導、御討論頂いている三菱電機株式会社情報電子研究所知識処理開発部房岡次長に深謝いたします。

参 考 文 献

- 1) Sequin, C. H.: Managing VLSI Complexity: An Outlook, *Proc. IEEE*, Vol. 71, No. 1, pp. 149-166 (1983).
- 2) Werner, J.: The Silicon Compiler: Panacea, Wishful Thinking, or Old Hat?, *VLSI Design*, Vol. 3, No. 5, pp. 46-52 (1982).
- 3) Goldberg, A. V. et al.: Approaches toward Silicon Compilation, *IEEE Circuits and Devices Magazine*, Vol. 1, No. 3, pp. 29-39 (1985).
- 4) Parker, A. C. and Hayati, S.: Automating the VLSI Design Process Using Expert Systems and Silicon Compilation, *Proc. IEEE*, Vol. 75, No. 6, pp. 777-785 (1987).

- 5) Hafer, L. J. and Parker, A. C.: Automated Synthesis of Digital Hardware, *IEEE Trans. Comput.*, Vol. C-31, No. 2, pp. 93-109 (1982).
- 6) Trickey, H.: Flamel: A High-Level Hardware Compiler, *IEEE Trans. CAD*, Vol. CAD-6, No. 2, pp. 259-269 (1987).
- 7) Seitz, C. L.: System Timing, in *Introduction to VLSI Systems*, Mead, C. and Conway, L. (eds.), pp. 218-262, Addison-Wesley, Reading, Massachusetts (1980).
- 8) Yakovlev, A.: Designing Self-Timed Systems, *VLSI Systems Design*, Vol. 6, No. 9, pp. 70-90 (1985).
- 9) Barbacci, M. R. et al.: The ISPS Computer Description Language, Carnegie-Mellon Univ. Tech. Report No. CMU-CS-79-137 (1977).
- 10) Hirayama, M.: A Silicon Compiler System Based on Asynchronous Architecture, *IEEE Trans. CAD*, Vol. CAD-6, No. 3, pp. 297-304 (1987).
- 11) Southard, J. R.: MacPitts: An Approach to Silicon Compilation, *Computer*, Vol. 16, No. 12, pp. 74-82 (1983).
- 12) Batali, J. and Hartheimar, A.: The Design Procedure Language Manual, MIT A. I. Memo. No. 598 (1980).
- 13) Hirayama, M.: VLSI Oriented Asynchronous Architecture, *Proc. of 13th Annual Int. Symposium on Computer Architecture*, pp. 290-296 (1986).

(平成元年 5 月 15 日受付)

(平成元年 11 月 14 日採録)

平山 正治 (正会員)



昭和 25 年生。昭和 48 年北海道大学工学部電気工学科卒業。昭和 50 年同大学院工学研究科情報工学専攻修士課程修了。同年三菱電機(株)入社。以来、中央研究所にて、VLSI 方式技術、VLSI 設計支援技術に関する研究開発に従事し、現在に至る。