プログラミング 18－4
(1998. 3. 23)

オブジェクト指向HDLに向けた自律再構成可能アーキテクチャ

小西 隆介　小栗 清　永見 康一　塩澤 恒道　伊藤 秀之

NTT光ネットワークシステム研究所

〒239 神奈川県横須賀市光の丘1-1

E-mail: {ryusuke,oguri,nagami,shiozawa,hi}@exa.onlab.ntt.co.jp

あらまし

動的なセマンティクスを導入したハードウェア記述言語と再構成可能なFPGAを結び付け、プログラミングレベルの並列性を直接実行できる汎用計算パラダイムの実現を目指している。本目標の達成に不可欠な自律的な再構成が可能なデバイスを、FPGAとセルラオートマトンを組み合わせた構造として実現する考えを示す。提案するデバイスアーキテクチャをプラスティックセルアーキテクチャ(PCA)と名付け、その設計方法を述べる。PCAはユーザの望む粒度での能動的な計算要素の構成を可能とすることから、プロセッシングエレメントを意識しない汎用の並列計算が可能になると期待している。

キーワード 再構成可能計算, FPGA, オブジェクト指向, ハードウェア記述言語, セルラオートマトン

Autonomously Reconfigurable Architecture Towards Object-Oriented HDL

Ryusuke Konishi　Kiyoshi Oguri　Kouichi Nagami　Tsunemichi Shiozawa　Hideyuki Ito

NTT Optical Network Systems Laboratories

1-1 Hikarinooka Yokosuka-shi Kanagawa 239 Japan

E-mail: {ryusuke,oguri,nagami,shiozawa,hi}@exa.onlab.ntt.co.jp

Abstract

We are aiming at a new paradigm of general-purpose computing that enables directly executing parallel semantics at programming level. Our approach couples a hardware description language enhanced dynamic semantics, and a reconfigurable FPGA. To achieve this goal, we propose a new device architecture that possesses autonomous reconfiguring facilities by fusing FPGA and cellular automata. We call this architecture Plastic Cell Architecture (PCA). We also describe the design policy of PCA. Because PCA is able to configure active and variable-grained processing elements, it will support to realize general-purpose parallel computation in which users do not need to recognize the granularity of processing elements.

key words Reconfigurable Computing, FPGA, Object-Oriented, Hardware Description Language, Cellular Automata

1. はじめに

ソフトウェアの実行を高速化するために、並列実行のセマンティクスを導入する努力が長く続けられているにも関わらず、その成果は一般には十分生かされていない。これは、本質的な並列性を表現する労力に対して、汎用の計算機構が採用するノイマン型アーキテクチャで得られる性能向上の見返りが十分でないことが大きい。

並列性の記述を実際の計算に結び付ける試みに関しては、むしろ論理設計の分野で普及しているハードウェア記述言語(HDL: Hardware Description Language)とその論理合成システムが成功している。勿論、これらは専用の処理を行うハードウェアを効率的に実現するための手段であり、それ自体は汎用計算を行うためのものではなかった。

しかし最近になって、SRAM ベースの FPGA (Field Programmable Gate Array)などの書き換えが可能なデバイスを応用して、実行時に構成を柔軟に変化させるハードウェアシステムを実現するという新しい可能性が見えてきた。

そこで我々は、記述性の優れた HDL とこういった柔軟なデバイスを結び付けることで、プログラミングレベルでの並列性を最大限に生かせるような新たな汎用計算のパラダイムを創出することができないかと研究を進めている。

我々は、1)抽象性と並列処理の記述能力を備えた HDL を用意し、そこに計算要素の動的な生成・消去のセマンティクスを導入すること、2)このセマンティクスを前述のデバイスの再構成能力に結び付けること、によってこの目標の達成を目指している。

既に PARTHENON[5]において、物理的な要素を排除し、かつ既存 HDL のような構造記述的なスタイルではなく人間にとってより理解が容易な動作記述に基づくプログラミングが可能となっている。PARTHENON のハードウェア記述言語である SFL[1]は、そもそもオブジェクト指向を強く意識して作られた言語であり、制御とデータを明確に分離した上で、計算要素間の制御の記述にメソッド起動のスタイルを採用している。1)の動的なセマンティクスの導入は、この言語思想の延長で自然に達成できると考えている。

一方、現在の再構成可能な FPGA は、動的な再構成能力と部分的な再構成能力を実現しているものの、デバイスの書き換えはあくまで外部の制御で行うことを前提としている。2)の動的なセマンティクスの実行には、デバイスに自律的な再構成能力を与える必要がある。ここで言う自律的な再構成能力とはデバイス上にプログラムされた論理回路(布線論理)が、その動作によってデバイス上に別の論理回路をプログラムできる能力を指す(図 1)。

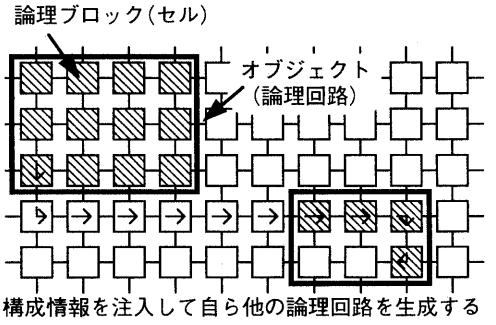

図1: 自律的な再構成能力をもつ FPGA

本稿では以下、2節でオブジェクト指向の HDL に自律的な再構成のセマンティクスを導入する方針を述べ、3節で自律再構成可能なデバイスのアーキテクチャモデルとしてプラスティックセルアーキテクチャ(PCA: Plastic Cell Architecture)を提案する。そして、4節で PCA の設計方法を示すとともに、5節でシミュレータについて述べる。6節はまとめである。

2. オブジェクト指向のハードウェア記述

2.1. SFL のオブジェクト指向言語としての特徴

オブジェクト指向のハードウェア記述がどのようなものであるかを想起するために図 2の例を見てほしい。これは信頼のある通信を行うためのプロトコルである ABP(Alternating Bit Protocol)の送信オブジェクトを SFL で記述したものである。この例は SFL のオブジェクト指向モデルの記述能力を示唆している。

メソッド起動スタイル

モジュールの端子は制御系端子とデータ系端子に概念的に区別されており(図 2, 4~9行目)、モジュールの制御は、プロシージャコールのスタイルで記述される(図 2, 30行目)。逆に制御端子のアサ

ート事象は instruct 文(図 2, 24～25行目)によって動作に置き換えられる。

これらの構文はオブジェクト指向のメソッド起動と直接対応づけることができる。

```
 1 /* a prototypical interface */
 2 /* declaration of receiver module */
 3 declare RECEIVER {
 4   input din<4>;
 5   input bi;
 6   instrin trans;/* assumed lossy */
 7   instrout ack; /* assumed lossy */
 8   output bo;
 9   instr_arg trans(din, bi);
10 /* SENDER transmits data with an */
11 /* alternating bit. And RECEIVER sends*/
12 /* an ack with a same protocol bit of */
13 /* the message received. */
14 }
15
16 module SENDER {
17   instrin send, tick;
18   input din<4>;
19   reg_wr buf<4>, pb;
20   RECEIVER R; /* declared statically */
21   stage_name abps {
22     task t(buf);
23   }
24   instruct send
25     if(^abps.t) generate abps.t(din);
26   stage abps {
27     state_name send_s, wait_s;
28     first_state send_s;
29     state send_s par {
30       R.trans(buf, pb);
31       goto wait_s;
32     }
33     state wait_s alt {
34       R.ack & ^(R.bo @ pb) : par {
35         pb := ^pb;
36         goto send_s;
37         finish;
38       }
39       tick : goto send_s;
40     }
41   }
42 }
```

図2: SFL による Alternating Bit Protocol の送信オブジェクトの記述例

タスクレジスタによる実行の制御

今一つの特徴は、スレッドに相当する概念が回路動作を制御するレジスタ(タスクレジスタ)により実現されていることである(図 2, 21～25行目)。ステートマシンとして記述される ABP の送信処理ステージ(図 2, 26～41行目)は、タスク(abps.t)が活性化されないと機能しない。タスクは、回路が存在することと動作することを概念的に分離している。

非同期的な計算主体を記述する抽象化能力

SFL により生成される論理回路は同期回路である。しかし上で述べた 2 つの特徴は、同期回路としての記述をモジュール内に隠蔽し、モジュール間では非同期的な用法を採ることを支援している。

この延長で、オブジェクト指向のコンカレントな計算モデルを自然に導入できると考えられる。

2.2. HDL への動的セマンティクスの導入

オブジェクトの動的生成と消去

HDL では従来静的に宣言されていたモジュールインスタンス(図 2, 20行目)を動的に生成・消去させる動作を、例えば以下の構文で導入する。

<インスタンス名> :=

new <モジュールクラス名>(<初期化パラメータ>);

delete <インスタンス名>;

この例では、new 構文は<初期化パラメータ>で指定された値を初期値としてレジスタに代入したインスタンスを生成し、その参照を返すことを表す。参照はメソッド起動や、delete 構文によるインスタンスの削除で利用される。

オブジェクト間の通信

オブジェクトの配置が実行時に決定されることを前提とすると、オブジェクト間の配線方法とその遅延時間は静的に解決できない。したがってオブジェクト間の通信は、従来の FPGA に見られるような固定的な配線ではなく、非同期のメッセージ伝達機構を前提とする方が考えやすい。

一方、生成されたオブジェクトを既存のオブジェクトと動的にリンクして組み立てるような機能は、計算資源の量を動的に調整するような用途で有望と考えられる。ただしその実現には領域管理と密に結びついた解法が要求され、汎用的に行うのは難しい。

以降ではメッセージ伝達に基づくモデルを前提として話を進める。

3. 自律再構成可能アーキテクチャ

3.1. デバイスの具備条件

動作時の書き換えが必須であることから、SRAM ベースの FPGA を前提とする。つまり、従来デバイスと同様、メモリの書き換えによって論理を実現する部分(論理層)の機能を決定する。デバイスは以下のような性質と機能を備えたものである必要がある。

均質性

実行時に論理回路の配置を行うことから、論理層は単純でかつメモリのように均質であることが望まれる。既存の FPGA は微妙に機能の異なる要素やチップ内を縦横に張り巡らされた配線資源が存在して、配置の問題を実行時に解決することが難しい。

そこで論理層は均質な論理ブロック(セル)がメッシュ状に接続されたものとして考える。クロックやリセット以外のデバイス内のグローバルな配線資源は考えない。そのかわり、セルの論理的な機能として配線の要素を用意することで、セルをまたがる接続を実現する。

メッセージの伝達機能

オブジェクト間のメッセージパッシングや、構成情報の運搬のためには、論理層の所定の領域に対して情報を伝達する機能が備わっている必要がある。並列性を損わないために、情報伝達は複数独立して行えるものとする。

セルの位置指定をアドレスで行うことはセルの均質性と拡張性を損なうと考えられるので、ルーチングは相対的な手段で実現する。

構成情報の部分的な注入機能

機能の書き換えは、構成情報を SRAM に注入することで実現される。論理層の部分的な領域に対して、このような注入が行える必要がある。

オブジェクトの起動・消去機能

構成情報を注入する作業は一斉に実施することが難しい。従ってこれとは分けられた過程として、オブジェクトの起動を一斉に行う機能がなければならない。同様に、オブジェクトの消去も単に構成を抹消するのではなく、まずオブジェクトとしての機能を一斉に停止させる必要がある。

これには先で述べたタスクの概念が適用できる。つまり、オブジェクトが必ずタスクレジスタをもつと仮定して、それを点火・抹消する操作として実現するのである。

3.2. セルラオートマトンによる組み込み機能の実現

このような機能は、論理層だけでは実現できない。明らかにデバイスの本来的な機能として組み込まれている必要がある。しかし、求められる機能を論理層の均質なセル構造と分離して実現すれば、セルのアレイと組み込み機能の間の繋ぎ目がアーキテクチャ上のボトルネックになるだろう。

そこで我々は、組み込み機能は論理層と同じようなセルの一様な構造によって実現されるべきであろうと考え、これをセルラオートマトン(CA: Cellular Automaton)によって実現するという方法を採る。すなわち、図 3のように概念的に論理層と重なるような CA 層を設け、CA 層と論理層の相互作用と、CA 層を構成するセルの連鎖的な働きとして組み込み機能を実現するのである。

例えば、前述のメッセージの伝達機能に関しては、1)論理層のオブジェクトを構成するセルのあるものが CA 層に対して、メッセージの伝達を指示する。2)CA 層は指示を受けメッセージを目的のオブジェクトの受信口となるセルまでルーチングする。3)メッセージを受け取ったセルは対象のオブジェクトにその中身を渡す。という機能の連結として行う(図 4)。

オブジェクトを構成するセルで、メッセージの送受信を担うものを以降では連絡口と呼ぶ。

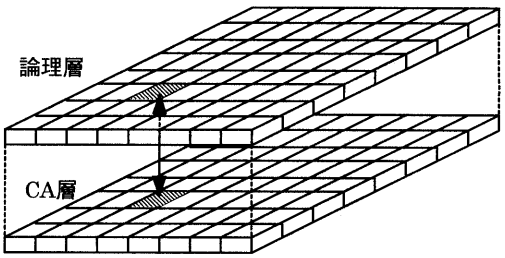

図3: 再構成能力の付与するための 2 重構造

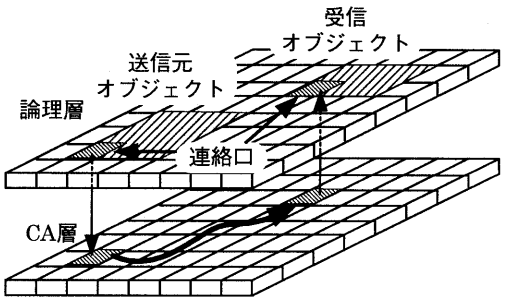

図4: CA によるメッセージの伝達機構

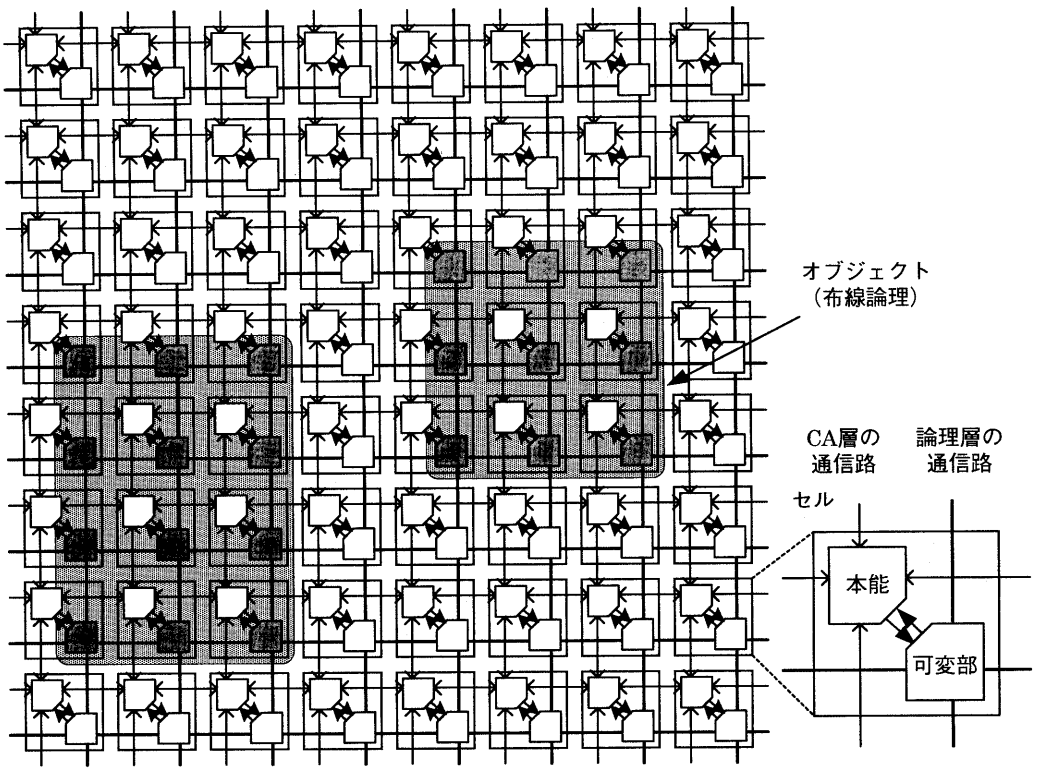

図5:プラスティックセルアーキテクチャ

3.3. プラスティックセルアーキテクチャ

ここで、提案されたデバイスのアーキテクチャを省みると、1)従来の FPGA と同様、SRAM によりその機能を変えることができ、布線論理を構成する論理層をもつ。2)セルラオートマトンとして実装される組み込み機能の層をもつ。3)論理層と CA 層は相互に作用を及ぼすことができる。という性質として説明される。

このようなアーキテクチャを、我々はプラスティックセルアーキテクチャ(PCA: Plastic Cell Architecuture)と名付ける(図 5)。ここでプラスティックとは、「柔軟な」とか「形成力のある」という意味で用いている。プラスティックセルアーキテクチャのセルにおいて、論理層を成す部分を可変部(Plastic Part)と呼ぶ。そして、組み込み機能を実現する部分を本能もしくは組み込み部分(Built-in Part)と呼ぶ。

このアーキテクチャは見方を変えると、ノイマン型計算機におけるプロセッサとメモリが、一つのチップ上で満遍なく展開されたもののようである。しかしながら、メモリに相当する部分が論理回路のような別の計算主体を形成するところが本質的に違う。

既存の SRAM ベースの FPGA には、構成情報の注入をシフト的に行うデバイスも存在する。しかし、このようなデバイスと比較してみても、論理層から CA 層に対して能動的に作用を及ぼすことで、汎用計算を可能にしている点に飛躍がある。

プラスティックセルアーキテクチャの性質

複数のプロセッサエレメント(PE)からなるような並列計算機上でプログラムを考えるときの困難の一つは、プログラマが PE を意識しなければならないことである。PE の粒度は固定的であるから、問題を PE の粒度に分解し、それをどのように協調させて動作させるかを考えなければならない。これは労力のかかる作業であり、自動化するにしても汎用的に行うのは困難である。

一方 PCA は、FPGA の特徴を継承して、計算

主体は可変粒度である。プログラマが解きたい問題の規模や構成に合わせて、自由な粒度で計算主体を作ることが可能になるので、プログラマは問題の本質を記述することに専念できる。このような柔軟性の獲得が並列性を犠牲にすることなく達成されている。

プラスティックセルアーキテクチャはまた、非常に均質な構造をもつアーキテクチャとなっている。プロセッサのように均質でないハードウェアでは、歩留りが絶えず問題となるが、このような均質な構造はデバイスの集積度を上げるのに都合がよい。また複数のチップを連結して、計算可能な領域を広げることも考えやすく、拡張性にも富んでいる。

4. プラスティックセルアーキテクチャの設計

4.1. ストリームプロセッサとしての本能の実現

CA 層に複数の組み込み機能を持たせることを考えると、情報の伝達方法について同じスキームが繰り返し用いられることに気づく。これを分離して実装することには無駄があるので、情報の伝達を統一して行う方法を考える。

図 6はその原理を示すもので、渡すべきデータの頭に経路を設定する命令(経路設定命令)を付け、また末尾に経路を解除する命令(0 命令)を付けて、各セルの本能にルートの設定解除を行わせることでメッセージの伝達を行う例である。

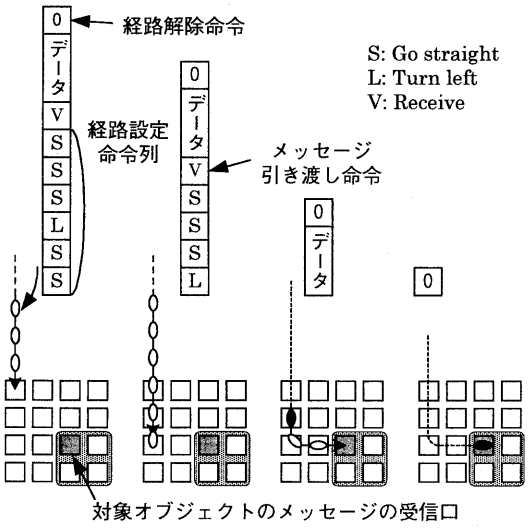

図6: メッセージ伝達のストリームの処理

すなわち、1)データとそれを処理するための適切な命令を同じスキームで伝達させる。2)本能をこのようなストリームを解釈し実行するようなプロセッサとして設計する。というアプローチを採る。

このように CA 層での伝達手段と指示手段を統一することで、本能の設計に柔軟さが与えられるだけでなく、可変部と本能のインタフェースも単純になる。すなわち、論理層のオブジェクト(布線論理)は高々数ビットの信号線を制御するだけで種々の組み込み機能を起動できるようになる。

4.2. ストリームの通路

ストリームの通路は、1)情報の伝達が複数独立に行われうることと、2)ルーチングで損失が生じないことを前提とすると、次の条件を満たさなければならない。

1. 停止可能であること
2. デッドロックフリーであること

1 は例えば、複数のメッセージがぶつかりあった時に、一方が待つ必要があるためである。一方 2 はメッセージが相互に進路をふさいでしまい、デバイスが機能不全に陥ってしまわないための条件である。プログラミングレベルでのデッドロックは含まない。

条件 1 を満たすために、図 7に示すような 2τ パイプラインを基本構造として採用する。

2τ パイプラインは、データビット以外にそれが有効であるかどうかを示すビットを持つ 2τ レジスタのカスケード接続であり、前が空いたことを確認してからビットをシフトさせるので、データの流れを制御するのが容易である(図 8)。

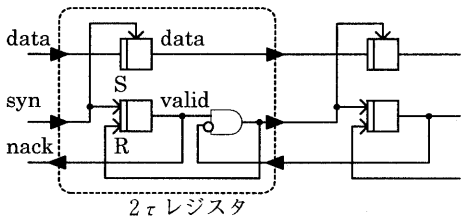

図7: 2τ パイプラインの基本構造

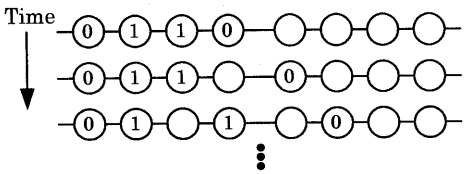

図8: 2τパイプラインの原理

一方条件 2 が満たされているかどうかは、様々なストリームの動きを加味して判断する必要があり、現時点で議論するのは難しい。

我々は一つのアプローチとして、図 9のように、隣接セルに対する入力パスと出力パスを独立に設け、各方向の出力パスは入力パスと独立に選んで接続できるようにすることで、本能の通信路を出力方向が重ならない限り交差可能にする方法を考えている。

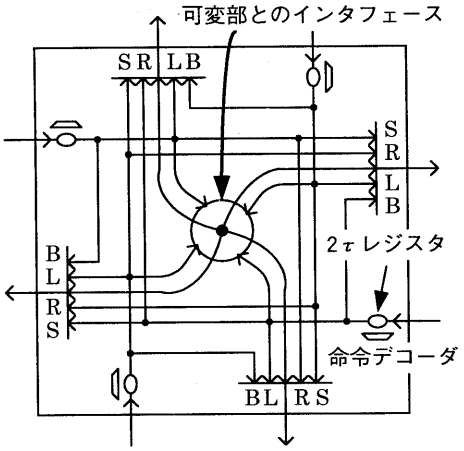

図9: 交差可能なストリーム通信路の原理

4.3. オブジェクトの動的再構成と起動

本能をストリームプロセッサと考える時、オブジェクトの生成は以下のようなフォーマットのストリームとして実現される(左側を先頭方向として表記する)。

経路設定命令列	X_1	X_2	...	点火命令	0 命令

ただし、X_1,X_2,...は各々以下のストリームとする。

構成開始命令	構成データ	経路設定命令

ストリームは 4 つの部分からなる。最初の部分は構成情報を目的のオブジェクトまでルーチングするための経路設定命令の系列である。後続のストリーム X_1, X_2, ...は、それぞれがオブジェクトを構成するセルに構成情報を注入し、次のセルまで経路を延長する意味の系列である。3 つ目はオブジェクトを起動する部分である。これはレジスタ構成のセルをタスクレジスタとみなしてセットする命令により行うことを想定している。最後は経路を解除する 0 命令である。

オブジェクトの構成と起動の前後関係は、ストリームの順序によって保証される。

4.4. オブジェクトの消去

オブジェクトの消去は以下のストリームで実現される。

経路設定命令列	Y_1	Y_2	...	0 命令

ここで Y_1, Y_2, ...は各々以下の命令列である。

構成クリア命令	経路設定命令

すなわち可変部の機能を停止させる命令を、オブジェクトを構成する全てのセルに届けることにより行う。消去中のオブジェクトが誤動作するのを防止するため、タスクレジスタが存在するセルを最初にクリアさせる。

4.5. メモリオブジェクト

生成するオブジェクトの構成データは生成元のオブジェクトが提供する必要がある。しかし、1 つのセルの構成には 10～100 ビット程度要するので、布線論理として実現されるオブジェクトでこれを行うことは実際上難しい。仮に論理回路の全てのセルがレジスタだと考えても、親オブジェクトは生成するオブジェクトの 10～100 倍の規模を要することになってしまう。

この問題を解決するためには、そもそも構成情報を保持するために用意されている SRAM を、メモリとして使う方法を考えれば良い。

一つの方法は、雛形として用意された不活性な布線論理を複写する機能を持たせることである。我々は、より一般的な方法として、図 10のようにメモリとして機能するセルを数珠つなぎにしたオブジェクトを考えている。このようなオブジェクトをメモリオブジェクトと呼ぶ。

メモリオブジェクトは受動的なオブジェクトで、連絡口に命令を与えることで読み書きを行う。アドレスの概念はなく、データはオブジェクト単位

でアクセルしストリームとして取り出す。

オブジェクトの雛形をメモリオブジェクトを用いてチップ上に置くことで、構成データの記憶保持の問題は解決できると考えている。

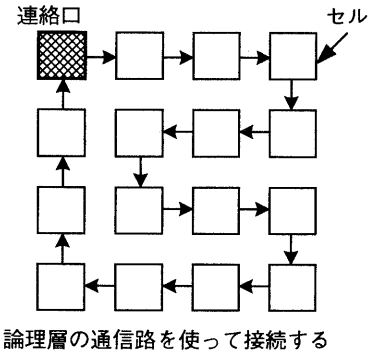

図10: メモリオブジェクト

4.6. CA 層と論理層のバランスと機能分担

CA 層は少なくとも以上で述べたような機能を実装している必要がある。一方、論理層のセルはなるべくシンプルにして粒度を細かくしたい。この差を埋めるために CA 層のセルと論理層のセルの対応関係は必ずしも 1 対 1 に固定して考えない。実際には可変部を複数の論理層のセルで構成し、両者の回路規模を見てバランスをとる必要があるだろう。

また、例えばオブジェクトを生成するための空き領域の割り当てのように、組み込み機能として実現するアプローチと、論理層の布線論理の動作として実現するアプローチと、両方考えられる場合がある。これについては実現性と設計のバランスを考慮して機能分担を決める必要があると考えている。

5. シミュレータ

現在我々は、PCA のアーキテクチャそのものを実行するようなシミュレータの開発を進めている。シミュレータの論理層部分は従来のような同期回路のシミュレーションを行うが、論理回路の自律的な再構成を実行できる点が新しい。

CA 層部分は各セルの本能の機能を模擬するが、C 言語で抽象化して記述しているため HDL で記述したものより高速に動作する。また、アーキテクチャの特性を生かしてクロック内の実行を各セルが独立に行える設計となっており、並列化が容易である。

このようなシミュレータの開発から着手しているのは、本デバイスを実現する開発工程において、従来の工程のさらに上流に、(下流工程からのフィードバックも含めて)アーキテクチャそのものを精査する工程が必要であろうという認識からである。また、動的生成を導入した HDL のツール開発や、アプリケーションの検討にあたってプラットフォームとなる環境が欠かせないのためでもある。

6. まとめ

自律的な再構成能力をもつデバイス・アーキテクチャとして PCA を提案した。PCA は相互に作用する FPGA と CA を融合することで、自律的な再構成能力を獲得していることを述べた。提案したアーキテクチャは汎用計算のハードウェアによる直接実行を目指しているという点で、既存の再構成可能なデバイスアーキテクチャとは方向性が異なるものである。

現在シミュレータの開発以外に具体的なアプリケーションの検討が進められている。今後はデバイスの実現作業と並行して、動的セマンティクスを導入した新しい HDL と論理合成ツールの具体化を予定している。

参考文献

[1] 小栗 清. ハードウェア記述言語と高位論理合成システムに関する研究. 学位論文, 九州大学, 1997.

[2] 永見 康一, 塩澤 恒道, 小栗 清. 自律的再構成可能アーキテクチャ. 設計自動化研究会. 情報処理学会, 1998 年 1 月

[3] 永見 康一, 塩澤 恒道, 小栗 清. 自律再構成可能アーキテクチャとオブジェクト指向 HDL. 第 11 回パルテノン研究会. 1997 年 12 月

[4] 小栗 清, 伊藤 秀之, 塩澤 恒道, 永見 康一, 小西 隆介, 中村 行宏. 布線論理による汎用計算機構. 第 11 回 回路とシステム軽井沢ワークショップ. 電子情報通信学会. 1998 年 4 月

[5] PARTHENON home page. http://www.kecl.ntt.co.jp/car/parthe/.

[6] Jr. Ray A. Bittner and Peter M. Athanas. Computing Kernels implemented with wormhole RTR CCM. In *Proc. of the Fifth IEEE Symposium on FPGAs for Custom Computing Machines*, pages 98-105. IEEE Computer Society Press, April 1997.