システムＬＳＩ設計技術　107－21
(2002. 11. 28)

PCAを用いた汎用アクセラレータの検討

堤　聡[†]　石川健一郎[†]　天野　英晴[†]

† 慶應義塾大学理工学研究科　〒223-8522 神奈川県横浜市港北区日吉 3-14-1
E-mail: †{tutu,ishikawa,hunga}@am.ics.keio.ac.jp

あらまし　PCA(Plastic Cell Architecture) は、動的再構成可能論理アーキテクチャである。本論文では、PCA をソフトウェアの汎用アクセラレータとして利用するアプリケーションの設計モデルを提案する。PCA の制御に対して API(Application Programming Interface) を提供することで PCA の動作の詳細を隠蔽し、ソフトウェアと PCA の協調設計を行う。これにより PCA に対してソフトウェアと同等のプログラマビリティを与え、多機能かつ柔軟なシステムの構築を可能する。本提案モデルのアプリケーションに必要な実行環境の試作を行い、その実現可能性について検討する。

キーワード　PCA、アクセラレータ、リコンフィギュラブルコンピューティング、協調設計

A General Purpose Accelerator Environment using PCA

Satoshi TSUTSUMI[†], Kenichiro ISHIKAWA[†], and Hideharu AMANO[†]

† Faculty of Science and Technology, Keio University　3-14-1, Hiyoshi, Kohoku-ku, Yokohama, 223-8522, Japan
E-mail: †{tutu,ishikawa,hunga}@am.ics.keio.ac.jp

Abstract　PCA(Plastic Cell Architecture) is a reconfigurable device architecture, which supports flexible runtime reconfiguration. For a system which integrates embedded CPU and PCA, we propose a design model of application using PCA as a general purpose accelerator of the software. Through the API(Application Programming Interface), detail controls of PCA are encapsulated, and Co-Design environment with PCA and the software on the embedded CPU is supported. A prototype runtime system for the environment of our model is developed, and an example application is implemented and evaluated on it.

Key words　PCA, accelerator, reconfigurable computing, Co-Design

1. はじめに

PCA(Plastic Cell Architecture) は、新しい概念に基づく動的再構成可能な論理アーキテクチャである [1]。FPGA(Field Programmable Gate Array) など、従来のユーザプログラム可能なデバイスと同様、プロトタイピングや Reconfigurable System に応用できる一方、非同期に基づく通信機構を持ち、動作中に自律的に回路構成を変更できる大きな特徴を持つ。この特徴を利用し、処理の負荷に応じて回路を複製するというシステムも提案されている [3]。現在、設計環境としてスケマティックエディタ、シミュレータが提供されており、記述言語から PCA で実行できるデータに合成する回路設計環境なども研究されている [4]。

PCA の高い動的再構成能力は、将来、プロセッサと混載されたチップ上でソフトウェアと協調処理を行なう場合にさらに有効に発揮されると考えられる。そこで、本稿では PCA とプロセッサが混載された環境を想定し、PCA をソフトウェアの汎用アクセラレータとして利用するアプリケーションの設計モデルを提案する。ソフトウェアと PCA の協調設計を行うことで、PCA に対してソフトウェアと同等のプログラマビリティが与えられ、各タスクに合せたハードウェアエンジンを作成することができ、多機能かつ柔軟なシステムの構築が可能になる。また、ハードウェアの構成情報をソフトウェアの一部とすることで、ネットワーク上を通して、システム全体のアップデートを容易に行うことが可能になる。

例えば、携帯電話などにプロセッサコアと PCA を混載させ、このシステムに適用することで、規格の違うさまざまな通信プロトコロルに柔軟に対応したり、処理するデータによってハードウェアエンジンを変更することで、省資源、省電力を図ることも可能になると考えられる。

PCA の制御は、API(Application Programming Interface) を通して行うことにより、動作の詳細が隠蔽される。ホストプ

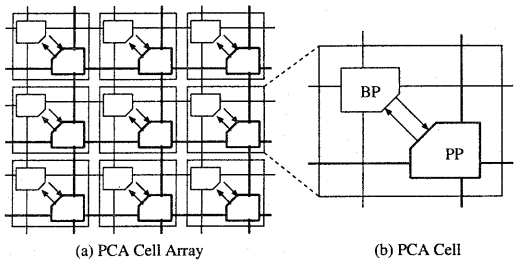

図 1　PCA の構成
Fig. 1　Structure of PCA

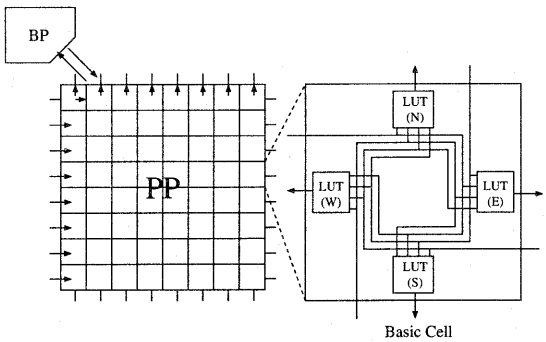

図 2　PP の構成
Fig. 2　Structure of PP

ロセッサ上で動作するプログラムは API を通して PCA を制御し、その一部の処理を PCA で実行させ高速化を図る。これにより、PCA-ホストプロセッサ間の構成が変更になっても、ホストプロセッサ上の OS がデバイスドライバを通して PCA を制御可能な構成になっていれば、同一のソースコードをコンパイルし、その環境の API とリンクすることで、実行可能である。

本研究では、アプリケーションに必要な API、実行環境の試作を行い、簡単なアプリケーションにより検証を行った。

以降、第 2 章において PCA の説明する。第 3 章において提案する設計モデルの開発手順の説明を行い、第 4 章においてアプリケーションの実行に必要なランタイムシステムについて説明する。第 5 章で API を利用したアプリケーションにより動作の検証を行い、第 6 章において本論文のまとめと今後の課題について述べる。

2. PCA(Plastic Cell Architecture)

2.1 構　　成

PCA(Plastic Cell Architecture) は、NTT により提案された、非同期動作を基本とする動的再構成可能なデバイスアーキテクチャであり [1]、図 1(a) に示すように PCA セルを 2 次元メッシュ状に接続した構成を持つ。PCA セルは論理を構成することのできる可変部 PP(Plastic Part) と、可変部の構成を制御するとともに、オブジェクト間の通信を行う組込部 BP(Built-in Part) から成る (図 1(b))。

PP は、4 入力 1 出力の LUT(Look-Up Table) が結合網により接続される構成を持つ単位セル (図 2) から成り、回路の構成はこの LUT に構成情報を書き込むことで行う。PP は機能オブジェクトを構成するのみでなく、メモリオブジェクトとしても用いることができる。回路は 1 つ、または隣接する複数の PCA セルを用いて 1 つのオブジェクト単位で構成する。

各 BP は隣接 4 方向 (N, E, W, S) の BP と隣接し、それら各隣接 BP との間、さらには該当する BP 自身が管理する PP に構成された機能オブジェクトとの間でハンドシェイクを行いながらメッセージ通信を行う。BP は隣接 BP および機能オブジェクトからのメッセージを解釈し、メッセージの経路設定、転送、メモリ操作、および BP とのインタフェース制御を行う有限状態機械となっている。

2.2 動　　作

PP に構成する回路は、束データ方式を用いた非同期回路として作成する。束データ方式において論理回路を実現する構成要素は、基本的には同期式論理回路と同様レジスタと組合せ回路からなり、レジスタ間の信号の授受の間に組合せ論理演算が施される。同期回路ではレジスタにデータを取り込むタイミングを決めるのはクロックであるが、非同期方式では代わりにそのレジスタに取り込むべきすべての有効なデータがレジスタに到達したことを保証する、完了信号を用いる。

具体的には、送信レジスタからはデータの送出と同時に 1bit の要求信号 (req) を送出する。要求信号のパスは、組合せ回路を経由するすべてのデータ信号の内で最長パスの遅延値以上に相当する遅延値を挿入することになる。これにより、有効なデータ信号が到達した後に受信側レジスタが信号を取り込むことを保証できる。受信側レジスタは信号を受信した後に、送信側レジスタに対して応答信号 (ack) を返す。

PCA の基本的な動作は、PP を用いて構成したオブジェクト間のパイプライン方式であり、この際、オブジェクト間の通信は BP を使用して行う。

オブジェクトは、動作中でも動的に再構成することができ、外部からの構成信号のみでなく、内部のオブジェクトからの制御信号によってもオブジェクトを構築、制御できる。

オブジェクトの動的構成に対応するため、PCA はクロックを用いない非同期方式で動作するように設計されている。個々のオブジェクトは自律的に動作し、BP によりオブジェクト間の通信を行うことで処理を行う。自律的に動作することで、PCA は高い並列性を持つ。

3. 設計モデル

今回提案する設計モデルは、プロセッサと PCA が同一チップ（ボード）上に搭載され、簡単な OS が動作するシステムを想定する。PCA は、プロセッサの周辺デバイスとして接続し、プロセッサで行う処理の一部を実行することにより、汎用アクセラレータとしての役割を果たす。

本モデルでは、PCA の制御は API を通して行うことにより動作の詳細を隠蔽し、PCA のリソースは次章で述べるランタ

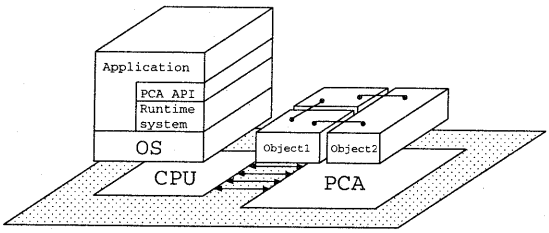

図 3　システム モデル
Fig. 3　System model

イムシステムによって管理する。ホストプロセッサ上で動作するプログラムの一部を PCA 上で実行するモジュールとして設計し、PCA に処理を分担させることでことで高速化を図る。図 3 がシステムのモデルである。設計者は、各タスクに合せたハードウェアエンジンを作成することができ、多機能かつ柔軟なシステムの構築が可能になる。API による制御を行うことで、PCA-ホストプロセッサ間の構成が変更になっても、ホストプロセッサ上のデバイスドライバを通して PCA が制御可能な構成になっていれば、同一のソースコードをコンパイルし、その環境の API とリンクすることで、実行可能である。ハードウェアの構成情報をソフトウェアの一部とすることで、ネットワーク上を通して、システム全体のアップデートを行うという方法も考えられる。

PCA は自律的に回路の再構成を行うことが可能なデバイスであるが、PCA のリソース管理はランタイムシステムが行うことを想定しているので、この機能を利用したアプリケーションに対応することは難しく、今回は考慮していない。

この設計モデルにおいて、FPGA をアクセラレータ用のデバイスとして利用することも考えられるが、次の点において現時点で、FPGA を利用することは難しい。

PCA では、BP を通してオブジェク間、ホストプロセッサとの通信を行うため、オブジェクトの配置、配線を動的に変更することができる。それに対して、FPGA では基本的にデバイスへのマッピングを合成後に静的に決定するため、ロジック間の配線を動的に変更することが難しい。これはオブジェクトの位置、デバイスのピンへのアサインを実行時に変更することができないことを意味し、ランタイムシステムによってリソースを管理する本モデルを FPGA に適用することはできない。

また、PCA を採用する長所として、PCA は非同期方式の回路構成を採用しているため、クロックに依存したシステム全体の設計を行う必要がないという点がある。これにより動作速度の異なるモジュールを混載させることが可能になる。設計者はオブジェクトのインタフェースを定義し、オブジェクト単体が動作することのみを保証すれば良いため、設計が FPGA と比較して容易である。インタフェースを定義することで、その内部の実装を任意に行うことができるため、より高機能のオブジェクトを用いてシステムのアップグレードを行うことも可能になる。

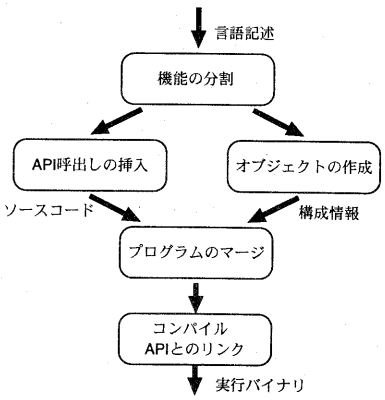

図 4　設計フロー
Fig. 4　Design flow

3.1　設計フロー

図 4 が提案するモデルの設計フローである。初めに入力となる言語記述を、自動もしくはユーザの指定によりホストプロセッサ上で実行する処理と PCA で分担する処理に分ける。PCA に処理させる部分は、PCA で実行できるモジュール (オブジェクト群) として作成し、モジュールを制御する API とインタフェース定義を元のソースコードに挿入する。これをコンパイルし、API とリンクすることで、ホストプロセッサ上で実行できるバイナリを作成する。

3.2　PCA API

今回は、ランタイムシステムに簡単な領域管理の機能のみを実装している。そこで、モジュールは一つのオブジェクトから構成されるものとし、入出力のポートはそれぞれ 4bit のポートを一つずつとしている。また、複数のプロセス、スレッドから実行を行うことは考慮していない。以降、実装した API について説明する。

3.2.1　オブジェクトの構築

PCA 上で処理を行うには、まずオブジェクトの構築が必要になる。オブジェクトの構成情報、入出力ポートの位置を指定し、API を呼び出すことで、PCA 上にオブジェクトが構成される。

PCA_CreateObject

```
PCA_ID PCA_CreateObject(
    PCAObject* Object
);
```

引数:

- Object 構成情報、インタフェース情報

戻り値:

オブジェクトを識別する正数、失敗すると負数が返る。

3.2.2　オブジェクトの実行

PCA での処理は、あらかじめ構築したオブジェクトの識別子、入力するデータが格納されたバッファへのポインタ、結果

が格納されるバッファへのポインタを指定して実行する。仮想記憶を採用しているため、実際にオブジェクトがPCA上に構成されていない場合があり、この場合はPCA上にオブジェクトが再構成されてから実行が開始される。実行は同期実行、非同期実行を指定することができる。非同期実行の場合、制御がプログラムの方にすぐに返ってくるが、実行が完了しているとは限らない。

PCA_Execute

```
BOOL PCA_Execute(
    PCA_ID nID,
    char* pInput,
    char* pOutput,
    int nInputSize,
    int* pOutputSize,
    BOOL bOverlapped
);
```

引数:

- nID オブジェクト識別子
- pInput 入力バッファへのポインタ
- pOutput 出力バッファへのポインタ
- nInputSize 入力バッファサイズ
- pOutputSize 出力バッファサイズへのポインタ
- bOverlapped 非同期実行を行うかの指定

戻り値:

関数が成功するとTRUEが返る。

3.2.3 オブジェクトとの同期

オブジェクトを非同期実行した場合は、その処理との同期を行うことが必要になってくる。これを非同期実行したオブジェクトの識別子を指定することで行う。

PCA_Wait

```
BOOL PCA_Wait(
    PCA_ID nID
);
```

引数:

- nID オブジェクト識別子

戻り値:

関数が成功するとTRUEが返る。

3.2.4 オブジェクトの強制終了

オブジェクトの処理は強制終了させることができる。これによりイベントドリブン型のプログラムやエラー処理による中断にも対応することが可能となる。

PCA_Terminate

```
BOOL PCA_Terminate(
    PCA_ID nID
);
```

引数:

- nID オブジェクト識別子

戻り値:

関数が成功するとTRUEが返る。

3.2.5 オブジェクトの解放

不要になったオブジェクトは解放する必要がある。これによりPCA上の資源が解放され、仮想記憶による不要なスワップを抑えることができる。

PCA_DeleteObject

```
BOOL PCA_DeleteObject(
    PCA_ID nID
);
```

引数:

- nID オブジェクト識別子

戻り値:

関数が成功するとTRUEが返る。

4. ランタイムシステム

本モデルでは、PCAの制御の詳細を隠蔽するため、ホストシステムのOSと同様に、ハードウェアの抽象化や、資源の適切な管理などを行なうランタイムシステムが必要である。

ランタイムシステムは、複数のプロセスからの要求を同時に処理するようにも実装できるが、本モデルは主にPCA-プロセッサ混載システムを想定し、多機能ではあるが、同時に動作する機能は限られているアプリケーションを対象とする。そこで、今回の実装では実験ボードとのインタフェースとの制限から、複数の実行要求が同時に存在することは考慮していない。

ランタイムシステムとPCAとの通信は、オペレーティングシステムのファイルシステムに対するシステムコールで行うことができるため、ランタイムシステムの移植性は高い。以降ランタイムシステムの機能について説明をする。

4.1 領域管理

複数のオブジェクトの構成が動的に実行されるため、ランタイムシステムは、領域管理の管理を適切に行う必要がある。2次元空間の領域管理は一般的には難しい問題であり、本稿ではこの問題は次の制限を設けて管理を行っている。

- 領域管理は1チップを1ページとし、ページ単位で行う
- 1ページに1つのオブジェクトのみ構成する
- 自律的な再構成は行わない

オブジェクトの配置アルゴリズムは、単純にホストプロセッサとの接続ポートに近いところから割り当てていき、オブジェクトの解放などによって空き領域が発生しても、再配置などは行っていない。

4.2 仮想化

PCAは動的に再構成可能なデバイスであるため、デバイス上の領域に空きが無くなっても、オブジェクトを一時的にホストプロセッサが使用するメモリ上に退避させることで、より多く

のオブジェクトを構成することができる。仮想化の際には、実行しているオブジェクトの状態を保存する必要があるが、PCAではそのような機能が無いため、実行中にオブジェクトが切り替わらないようにしている。また、ページの構成情報はメモリ上にすべて保存しておき、ページの切り替えはPCA上のページを上書きすることで実現している。置換するページの選択はLRU(Least Recently Used)を使用している。

4.3 ルーティング

領域管理はランタイムシステムが行うため、ユーザはどの位置にオブジェクトが構成されるか知ることはできない。このため、入出力、オブジェクト間のメッセージをルーティングする制御コマンドは、ランタイムシステムが適切に生成する必要がある。ルーティングメッセージは、PCA上にオブジェクトが構成されるたびに生成されキャッシュされる。これにより、オブジェクトの位置がスワップなどで変更にならない限り、再生成する必要は無い。オブジェクトとの経路は、オブジェクトが構成された後、実際に実行が開始される前に構成され、実行が終わるとクリアされる。

5. 評　　価

ホストPCとPCA-1の実験ボードから構成されるシステム上で簡単なアプリケーションを実行して、設計モデルの検証を行なった。アプリケーションには、白黒画像を読み込み2値化して表示するプログラムを用いた。ここでは、入力された値を閾値と比較して2値化する処理をPCAに分担させ、その他画像データの読み込み、表示などはホストPC上で動作するプログラムとして設計した。

5.1 PCA-1

実装の対象として用いるPCA-1の主な仕様を表1に示す。PCA-1はPCAという概念そのものの検証を第一の目的として試作された。このためデバイステクノロジとしては成熟したものが採用されており、チップとしての高集積化、高速化については必要最小限の検討にとどめられている。この結果として10ミリ角のチップ上に6×6の計36個のPCAセルを搭載している。

このチップを4×4の計16個搭載している実験ボードにおいて検討を行った。実験ボードは、USB1.1インタフェース経由でホストPCと接続されている。USBインタフェースは複数枚接続が可能になっており、今回はこれを2つ使用し、それぞれを、入力、出力用に割り当てた。ホストPCは表2のものを使用した。

5.2 設計手順

アプリケーションの評価手順は次の通りである。自動設計環境はまだ構築してないため、PCAアプリケーションの作成、APIの挿入は、人手で行った。

（1） C++言語を使用して2値化のプログラムを作成する
（2） 2値化を処理するPCAオブジェクトを作成する
（3） C++言語で記述されたソースにAPIを挿入する
（4） コンパイル、リンクを行う

表1　PCA-1のハードウェア仕様
Table 1　PCA-1

項目	内容
プロセス	0.35 μm CMOS スタンダードセル
チップサイズ	10.0 × 10.0 mm
ピン数	240(QFP) (ユーザ解放信号ピン 168 ピン)
PCA セル数	6 × 6 個
LUT 数	9216
メモリ容量	144 kbit (4 kbit/PCA セル)

表2　ホストPC
Table 2　Host PC

項目	内容
CPU	Celeron 400MHz
Memory	256MB
OS	Windows2000

図5　実行結果の例
Fig. 5　An example of execution

5.3 実行結果

作成したのは2値化するオブジェクトのみだが、複数のオブジェクトの実行をテストするために、オブジェクトの中で固定している閾値を変化させたものを16種類用意した。オブジェクトの面積は3 PCAセルとなった。

計測はホストPC側のプロセッサのチックカウントにより行った。参考にソフトウェアのみで処理した時間も計測している。

画像は16階調の白黒画像(384 × 512 ピクセル)に対して2値化を行った。16階調の画像なので、閾値は暗い順に並んだ0–15のパレット番号で指定した。図5の実行例は、パレット9(明度82)を閾値とした結果である。

実行の結果、1つのオブジェクトを構成するのに必要な時間は、129 ms、2値化に要した時間は12.4 sとなった。ソフトウェアのみで処理を行った場合、0.008 sという結果が得られた。仮想領域管理をテストするために、最大ページ数16を超える18個のオブジェクトを構成し、スワップ機能が働いているか確認した。この際、ページの位置による処理、構築の時間に差異を認めることはできなかった。また、非同期実行、同期処理、強制終了の機能の確認を行った。

5.4 考　　察

実行結果から、実装したAPIが正しく機能していることが確認された。これまでに述べてきたように、設計者はPCAをAPIを通して使用することで、オブジェクトの動的な領域管理、配置、経路設定などの詳細な制御を考慮する必要がなくなり、設計の負担が軽減される。

現時点では、オブジェクトは人手により作成しているなど、多くの課題が残されているが、今後、言語記述からのオブジェクト合成、ランタイムシステムの高機能化などによって、設計モデルの適用範囲が広がり、ソフトウェアと同等のプログラマビリティで設計を行うことができると考えている。

実行結果から、ソフトウェアのみの処理と比較して、3桁以上遅い結果になっている。シミュレーションの結果では、約0.022 sと見積もられている。これは、ホストPC-PCA間の接続がUSB1.1で接続されており、PCA本来の処理速度をホストPC側で計測するようにはできていないため、データ転送速度のみで処理速度が決っているからである。今後、ホストプロセッサとPCAとの接続形態も検討していく必要があると考えている。

6. ま と め

本稿では、ソフトウェアで行う処理の一部をPCAに実行させることにより、PCAを汎用アクセラレータとして利用するアプリケーションの設計モデルを提案した。ソフトウェアとPCAの協調設計を行うことで、PCAに対してソフトウェアと同等のプログラマビリティが与えられ、多機能かつ柔軟なシステムの構築が可能になる。

アプリケーションは言語記述され、自動もしくはユーザの指定によりPCAで分担する機能に分けられる。PCAで実行されるコードは、PCAで実行可能なPCAのアプリケーションとして合成され、ホストプロセッサで実行されるコードには、PCAを制御するためのAPIが埋めこまれる。これをコンパイルすることで、ホストプロセッサ上の実行可能バイナリを得ることができる。

PCAの制御はAPIを通して行うことにより、動作の詳細が隠蔽される。ランタイムシステムは、領域管理、PCAへの入出力、ルーティングなどの機能を提供する。PCAで分担されるプログラムはAPIを通してPCAで処理が行われることにより、PCA-ホストプロセッサ間の構成が変更になっても、ホストプロセッサ上のOSがデバイスドライバを通してPCAを制御可能な構成になっていれば、同一のソースコードをコンパイルしその環境のAPIとリンクすることで、実行可能である。

本研究では、アプリケーションに必要なAPI及び実行環境の試作を行った。今後、機能記述からPCAアプリケーションへの機能分割および合成、API呼び出しの自動挿入などの自動設計環境の構築を行い、オブジェクトの領域管理、動的スケジューリングなどの機能をランタイムシステムに追加して行く予定である。

謝　　辞

PCA実験ボード及びPCAの設計環境を提供して頂き、本研究に対する多くの御助言を頂きました名古屋 彰氏、小西 隆介氏をはじめとするNTT未来ねっと研究所の方々に深く感謝の意を表します。

文　　献

[1] 中田 広, 伊藤 秀之, 小西 隆介, "完全非同期回路によるPCAハードウェアの設計・評価," NTT R&D, vol.49, No.9, pp. 518–526, 2000.

[2] M. Uno, Y. Shibata, H. Amano, K. Furuta, T. Fujii and M. Motomura: "Implementation of Virtual Hardware on Dynamically Reconfigurable Logic," Proceedings of the Joint Conference of the World Multiconference on Systemics , Cybernetics and Informatics & the International Conference on Information Systems, Analysis and Synthesis, pp.124-129, Jul. 2000.

[3] 小西 隆介, 伊藤 秀之, 中田 広, 塩澤 恒道, 稲森 稔, 名古屋 彰, "非同期式動的再構成可能LSIによる自己複製回路," 電子情報通信学会技術研究報告, VLD2000-79, ICD2000-136, FTS2000-44,pp. 59-64, Nov. 2000.

[4] 岡本 卓也, 冨田 明彦, 杉本 成範, 境 和久, 泉 知論, 尾上 考雄, 中村 行宏, "プラスティックセルアーキテクチャのための回路設計環境の構築," 第18回パルテノン研究会資料集,p43–50(2001)

[5] Rryusuke Konishi, Hideyuki Ito, Hiroshi Nakada, Akira Nagoya, Kiyoshi Oguri, Norbert Imlig, Tsunemichi Shiozawa, Minoru Inamori and Kouichi Nagami: "PCA-1: A Fully Asynchronous Self-Reconfigurable LSI," Proc. of 7th International Symposium on Asynchronous Circuits and Systems (ASYNC 2001), pp. 54-61, Mar. 2001.

[6] 日本電信電話株式会社 NTT 未来ねっと研究所, "PCA ボード実験マニュアル (第1.01版)," Aug. 2001.