システムＬＳＩ設計技術　101－3
（2001. 5. 17）

Plastic Cell Architecture (PCA) における通信仮想化のためのフレームワーク

奥山　祐市、黒田　研一
会津大学大学院コンピュータ理工学研究科
〒965-8580　会津若松市一箕町鶴賀
Tel: 0242-37-2620　Fax: 0242-37-2595
E-mail: {d8011202, kuroken}@u-aizu.ac.jp

あらまし　本論文では PCA 上で動作するオブジェクトを擬似的にソフトウェアで作成し、これらのオブジェクトを PCA オブジェクトと協調してシミュレーションできる環境を提案する。現在の PCA の設計は上位工程のシミュレーション方法がなく、下位工程を終えた後のシミュレーションに頼っているため全体の設計期間が長くなる傾向にある。これを解決するために上位レベルの設計でシミュレーションを可能とするソフトウェアインターフェイスを作成し、実際に乗算機の設計に適用した。この結果下位工程での試行回数を減らし、設計期間の短縮を図ることができた。

キーワード　PCA、非同期回路、シミュレーション環境

Simulation Framework for Circuits on Plastic Cell Architecture (PCA)

Yuichi OKUYAMA, Kenichi KURODA
Graduate School of Comp. Sci. and Eng., The Univ. of Aizu,
〒965-8580　Tsuruga, Ikkinachi, Aizu-Wakamatsu
Tel: 0242-37-2620　Fax: 0242-37-2595
E-mail: {d8011202, kuroken}@u-aizu.ac.jp

Abstract

This paper describes a methodology for prototyping PCA object. We propose interfaces that allow us to describe PCA object by software. It can be used for early prototyping, reduction for trial assembling low level PCA circuit and co-simulation between PCA object and prototyped object. Using this methodology, we can reduce errors of highly level design and period for design of PCA objects.

key words　PCA, asynchronous circuit, simulation framework

1．はじめに

近年PCAが提案され、RAMと演算部を均一にな構造にすることによって、布線論理を動的に再構成しながら汎用性の高い処理を高いスループットで行うことができるようになってきている。このデバイスを用いることによって、従来のマイクロプロセッサとメモリを用いた演算方式では十分に高速化が達成されなかった分野に対して、柔軟性のとんだ論理を組めるようになった。また、このデバイスに対して負荷に応じて演算部の構成を変化し、動的に処理を分散するような構成が提案されている[1]。

このデバイスの設計手法として、[2][3][4]のように論理やFSMを直接PCA可変部へマッピングするものや 、[5]のように非同期回路を高位レベルの言語で記述するものがある。また、PCAのデバイスを反映させたシミュレータとしてPCASIM[6]が発表されている。これらの手法は従来の論理回路設計をベースにしたものであり、論理を PCA 上へ実現するための手法や環境としては非常に有用である。

しかし、これらをベースに PCA に対して応用回路を設計しようとした場合、

（1） 問題を記述するレベルが低いため、設計の見通しが悪くなる
（2） シミュレーションを行う時期が開発の後期であるため、上流での設計の誤りを早期に発見しにくい

等の問題が生じる。

このような問題に対するアプローチとして，近年SpecCや、SystemCなどのC言語に拡張を加えたシステム記述言語が開発されている。これらの言語は回路のプロトタイプを抽象度の高い状態から記述することができる。さらにこれらの抽象度を下げるための構文が用意されており，抽象度の低い（実際の回路に近い）レベルと抽象度の高いレベルが相互に連携しながらシミュレーションできるようなインターフェイスを持っている。

これらの言語をPCAに適応する場合，2つの問題点がある。

（1） 一般的な回路設計を対象としているため，記述が煩雑である。
（2） C言語をベースにしているため，PCAのパラダイムにそぐわない

また、PCA通信の仕組みがメッシュベースであり，これらの通信の抽象化をする必要があった 。したがって、PCA パラダイムにあったシミュレーション機構を考慮する必要が出てきた。

一方で，PCAの動的再構成可能な特徴を生かそうとした場合，従来のOSにあるようなページング機構などを2次元化し，それぞれのモジュールの生成や消滅を管理する必要が出てくると思われる。この際，機能モジュールのライブラリを組み合わせて，問題を解決しようとする方法が有効であると考えられている。しかし，現在のライブラリ群[]は，あらかじめ通信対象となるモジュールへの経路を回路内で生成しなくてはならず，複雑な経路生成を行うことができない。このようなことから，経路情報はライブラリ自身が生成するのではなく，システムの実行時に動的に与える必要が出てくる。

これらの問題を解決するために，筆者らは，PCAの経路設定に対する手順を統一し、これらをユーザから隠蔽することによって，高いレベルからの容易なシミュレーションと，経路の動的な割り当て問題を解決することを目標に、その環境を構築してきた。

2．PCA

PCA はプログラマブルハードウェアの1種であり、論理を形成する Plastic part と論理を制御する Built-in part から構成されている。本節ではこのハードウェアを用いた論理の再構成の方法について述べる。

2．1 PCAの構成

PCAの論理はPlastic partの構成要素であるRAMテーブル内に数値を書き込むことによって決定することができる。このRAMテーブルは図1のように隣接する4方向にそれぞれ入出力を持っている。

このように、論理をメモリのみによる構成にすることによって，均質なRAM構造からなるプログラマブルハードウェアを得ることができる。

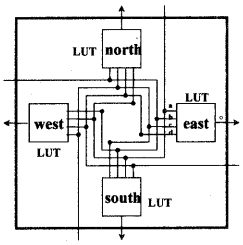

図1：PCAのLUT

PCA はその論理構成を変更するために Built-in part (BP)と呼ばれるコントローラを8×8=64個のPPセルに対して1つ持っている。BP は2次元メッシュ上に配置され，（1）メッセージを伝達するための経路の機能、（2）PP と通信をするための機能を持っている。これらの機能はBPの解釈する12のコマンドに集約されている。以下にそのコマンドを示す。

命令	内容
CLEAR	設定された経路を解除する
OPEN	オブジェクトをリセットし、PP と BP を接続する。命令を受け取った PCA セルはオブジェクトの通信ポートとして機能する。
CLOSE	PP と BP の接続を切る
CO	構成情報を読み出す。
COCI	構成情報を読み出す。隣接するオブジェクトをスキャンして、別の場所に複写
CI/M	PCA セルに後続のデータを書き込む。メモリオブジェクトの構成に用いる。
CI/F	PCA セルに後続のデータを回路の構成情報として書き込む。機能オブジェクトの構成に用いる。
WEST	経路を西方向に設定する。
NORTH	経路を北方向に設定する。
EAST	経路を東方向に設定する。
SOUTH	経路を南方向に設定する。
PP	プラスティックパートにデータが送られるよう経路を設定する。

表1：PCA のコマンド

2．2　PCA による処理の実際

ここでは、PCA に実際にどのように回路が書き込まれるのかを具体的な例を用いて説明する。

初期状態では、回路は PCA チップの外部から書き込まれる。PCA チップの外部ピンは内部の BP に接続されており，BP のコマンドを直接受け取る。外部からの値は，まず経路情報を形成するコマンド NORTH, SOUTH, EAST, WEST のうち必要なものを用いて，回路の書き込み先までの経路を形成する。その後，BP から PP に経路を設定するための PP 命令，回路構成情報を書き込むための CI/F 命令を入力する。ここまでの操作で、外部から PCA 内のセルに対しての回路入力のためのパスができたことになる。その後，1-PCA セルの回路構成情報 1024 ビットを入力して，1-PCA セルの回路設定が終了する。

このようにして作られた 2 つ以上の PCA 上のオブジェクトの通信の初期化について考える。
ここでは、図 2 のように A と B の 2 つのオブジェクトが通信の準備に入ろうとしていると仮定する。この図では，A が B にデータを転送しようとしている。オブジェクト A はまず，B への相対的な座標位置、および中間の経路が使用されていないことを知る必要がある。その上で，A はまず外部からの入力により，送信ポートに相当する PP セルと BP を接続してもらう。その後，送信ポートから B の受信ポートまでの経路を示すコマンドを（NORTH, SOUTH, EAST, WEST）から任意に選択し，B までの経路を生成する。そして，経路の終端を B の受信ポートに接続するために，PP, OPEN コマンドを生成し，経路の生成が終了する。ここまでの操作によって，A は B にデータを転送することができるようになる。

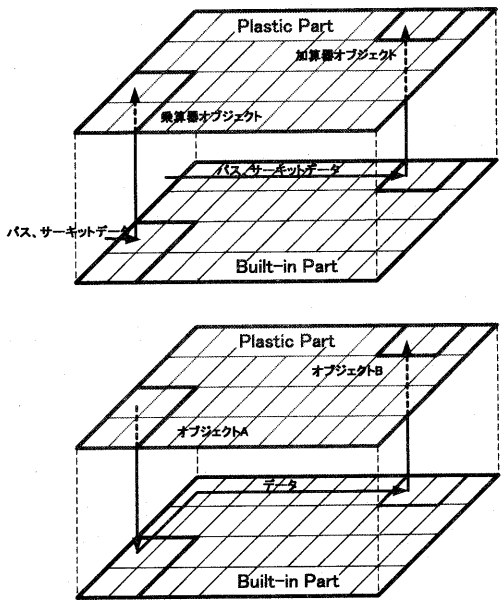

図2：BP と PP の関係

2．3　オブジェクト通信の経路設定の問題

前節では，2 つのオブジェクトが通信を行う際に必要な条件とコマンドについて述べた。PCA チップ内には複数のオブジェクトが任意に通信することができるため，この制約を正しく守る必要がある。

PCA チップのコマンド仕様から、中間の経路が使用中であった場合に対して，それを検知する方法が存在しない。このため，これらの経路設定を個々のモジュールで独自に行うと，システムが混乱する可能性がある。

また、PCA は動的再構成可能な LSI であり，設計時にオブジェクト群の相対位置を固定して考えることが，オブジェクトの空き領域の管理を導入した場合に不都合が生じると思われる。したがって，経路設定は他の専用モジュールに委託し、その経路を受け取り、個々のモジュールでは独自に経路を設定しないと仮定

する。

3. システムレベル記述

PCA 上の回路設計では下流の設計手法が確立しておらず、個々の設計者の技術に依存する。これは現在の論理回路設計における設計フローよりも、下流の設計期間が上流の設計期間よりもかなり長くなることを意味する。このため、上流での厳密な仕様定義と、設計の誤りのなさが従来のシステム設計よりもいっそう重要になる。

3．1 SpecC における設計レベル

SpecC の設計フローでは設計対象となるシステムを4つの段階、すなわち、仕様モデル、アーキテクチャモデル、コミュニケーションモデル、実装モデルに分割して記述する。

仕様モデルは、システムの振る舞いを記述するモデルである。それぞれのモジュールでの通信はグローバル変数で表現され、平行性や同期などの処理は記述しない。

アーキテクチャモデルは回路の分割に着目したモデルである。記述したシステムをハードウェアとソフトウェアに分割し、それぞれのモジュール間の平行性や同期に関する記述を行う。通信は依然としてグローバル変数で行う。

コミュニケーションモデルでは、モジュール間の通信に着目し、システムで行われるモジュール間の通信を決定する。これらの通信に対する記述を行うことで、それぞれのモジュールでの動作タイミングを確定することになる。またこのモデルでは、互いに違ったプロトコルを接続するためにトランスデューサと呼ばれるプロトコル変換モジュールを生成する。

実装モデルは最終的なシステム記述の形態であり、ハードウェア部分はこれに基づいて論理合成を行い、ソフトウェア部分はこの記述からデバイスドライバなどが生成される。

これらの設計レベルとレベルの詳細化を行うにあたって必要な作業を図3に示す。SpecC を用いた設計手法では、ある設計レベルが完了次第シミュレーションによる動作検証を行うことができるため、早期に設計の見落としを発見することができる。また、異なった設計レベルでの協調検証も行うことができるため、プロジェクト間で進行度合いにばらつきがあった場合でも問題なくシステム全体を最新の設計状況を反映させながらシミュレーションできることになる。

3．2 PCA コミュニケーションの抽象化

SpecC における設計レベルを適用するため、PCA 上で矛盾なく設計を詳細化できるようなインターフェイスが必要になる。本節ではこれらのインターフェイスを PCA のコミュニケーションに着目し、これらに各設計階層において異なる抽象度を設けることにより、システムレベル設計を実現する。

ところが、コミュニケーションの詳細に関する記述は抽象度の高い上位の設計では詳細に記述することができない。上位の設計では、これらの作業を意識することなく動作モジュールを構築できることが必要なる。特にアーキテクチャレベルでの記述は何度も試行錯誤が行われるため、設計の効率化には抽象化が必要である（図4）。

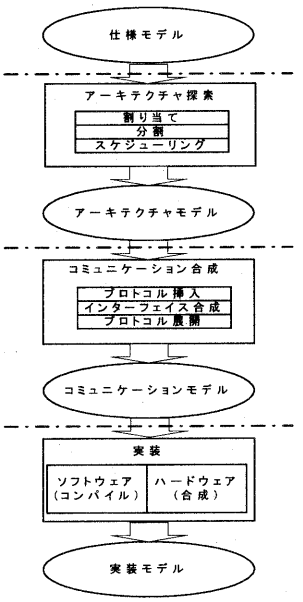

図3： SpecC における設計レベルと作業

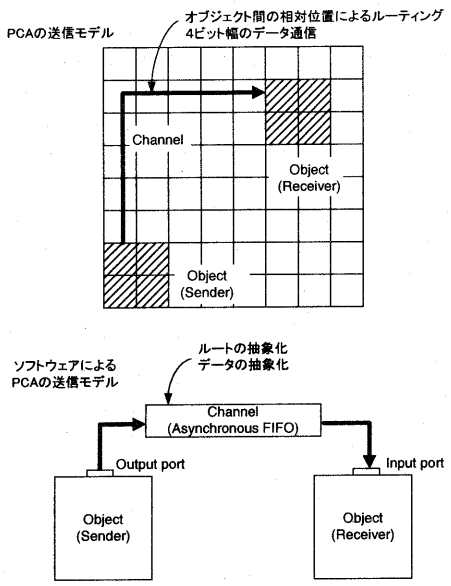

図4： PCAの通信モデルとそのソフトウェア表現

これらのPCAでの通信の抽象化には以下の2つが考えられる。

（1）ルーティングの抽象化

（2）データの抽象化

ルーティングの抽象化は、PCAの2次元構造を意識することなくアーキテクチャ設計を行うための抽象化である。再配置可能な回路を設計する場合、ルーティングの決定は管理機構が行うと考えるのが自然である。したがって、経路の長さを考慮した回路設計は一切できなくなる。上流の段階でこのような回路を設計しないようにするために、チャネル間の経路長は仮定できないような仕組みが必要である。また、ルーティングのパスも不確定であるため、経路の指定を東西南北のコマンド列で行うのではなく、オブジェクトのポート単位で指定できるようにする必要がある。

送信データの抽象化は、送信したいデータを直接送信できるような環境を提供することである。これは、上流の設計でデータを4ビットのデータ列として捉える場合に比べて、アーキテクチャの検討をスムーズに行うことができる。また、データの抽象化の中には、異なったフォーマット同士の変換も含まなければならない。このプロトコル変換機はSpecCではトランスデューサと呼ばれ、一度特定のフォーマットでデータを送信するように作成したモジュールを再利用する場合に必要である。次節以降では、これらの詳細化を行うために必要なインターフェイスを述べる。

4． PCAのためのクラスインターフェイス

本研究ではこのシステムの実現にJava言語を使用した。これらのインターフェイスを言語上に実装し、それぞれのモジュールをスレッドで表現することにより、並列に動作するPCA上のオブジェクトのインターフェイスに関するシミュレーションと機能シミュレーションを行う。

ソフトウェアでPCAオブジェクトを表現する場合には、それらの並列性を考慮しなければならない。並列性を表現するために筆者らはこれらのオブジェクトをスレッドで表す。オブジェクトは演算機能以外にポートを示すChannelオブジェクトを保持する（図4）。

PCAの経路情報はChannelオブジェクトを用いて表現する。Channelオブジェクトはデータの読み出し(read)、書き込み(write)、データが読み出し可能かを判定するメソッド(isReadable)が含まれている。これらのメソッドのうちデータの送信側にはwriteメソッドが公開されており、受信側にはreadとisReadableが公開されている。Channelクラスは無限容量のバッファを持っており、容量が特定できないようになっているが、何らかの理由で容量を特定しなければならない場合は、その際大容量を設定することができるようになっている。オブジェクトがChannelに対して読み込み、書き込みができない場合は、それらのメソッドを呼び出したオブジェクトが待機状態になる。この機能によって、オブジェクト間の通信がデッドロックになった場合でも、ソフトウェア上で早期に認識/デバッグすることが可能となる。

それぞれのPCAオブジェクトが保持するChannelオブジェクトは設計の階層によって、初期化時に通信可能か経路設定を独自で行うかを選択することができる。初期化時にポートを通信可能にする方法は主に上位の設計で用いられ、PCA上のルーティングに対する煩雑な設計を省きながらアーキテクチャレベルからコミュニケーションレベルの導出を行うために用いられる。経路設定をオブジェクトが独自に行う場合は、通信先のオブジェクトの位置情報がどのように通信元に伝送されるかを明確に定義し、実際のPCA上で正しく動作するように経路設定を行わなければならない。

これらのルーティングに関する情報は、ソフトウェア内のPCAオブジェクトに対するリファレンスとポート名の対で表現する。PCAオブジェクト内部に経路情報を保持する場合は、これらの情報があらかじめ通信もとのオブジェクトにストレージを用いて保持されることになる。ソフトウェアでこの機能を実現するために、イントロスペクションを用いた。

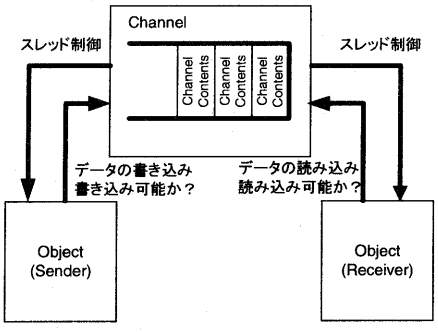

図5　PCA 通信モデルと機能

Channel 上に流れるデータ型は ChannelContents 型を設計時に継承することによって、容易に定義することができる。例えば 32 ビットの int 型をチャネルを用いて送信する場合には、ChannelContents 型を継承して独自に int 型のデータを持たせることにより、Channel を用いた通信を行うことができるようになる。ただし、実際の PCA シミュレータやチップへの移行を容易にするために、ユーザが新しく定義した型にはこれらの内容を 4 ビットのデータ列に変換するメソッドを用意しなければならない。

また、PCA のシミュレータである PCASIM との連携を行うために、ファイルを介したデータの入出力を行うトランスデューサを作成した。PCASIM は外部からのデータの入出力をファイルを介して行うことができるため、あらかじめ PCASIM に対する入出力ファイルを指定し、これらに対してシミュレーションに必要なデータを書き込む。これらの動作をソフトウェアによるオブジェクトと PCASIM 上にあるオブジェクトとでやり取りするために、このトランスデューサを用いた（図6）。

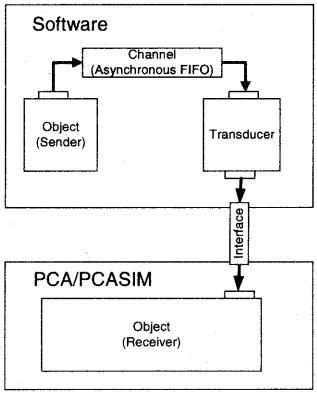

図 6　ファイルを介したソフトウェアと PCA の連携

5.　設計例

以上に述べられてフレームワークを利用して，乗算器の設計を行った。この乗算器は，PCA の再構成機能を用いることによって，任意のビット長の乗算がを行うことができる。PCA のコミュニケーションは 4 ビットで行われるため，4 ビットごとのビットスライスを効率良く演算できるような仕組みが必要であった。また、ビット数に制限を課さないようにするために，他の演算機に見られない工夫がされている。

5．1　PCA 乗算器

PCA のコミュニケーションは 4 ビットごとに行われる。コミュニケーションを制御するポートに比べて、演算を行う部分は小さいため，乗算や加算はほぼ同じ面積を占有すると考えてよい。このため、これらの回路を以下に効率よく組み合わせて，ビット長の制限のない乗算回路を効率よく生成するかが焦点となる。通常の論理回路における加算アルゴリズムは，同時に複数のビットが扱えることを前提としているため，このような目的には合致しない。

入力されるデータは図 8 のように LSB から 4 ビットづつ入力されるとする。したがって，4 ビットのごとの乗算は乗算器で行うこととする。図 7 に筆算形式で表した 16 ビットの乗算を示す。

この筆算で計算される部分積は，パイプライン内に A のビット列と B のビット列を反対方向に流すことによって，得ることができる。これを示したのが図 8 である。さらにこれらの部分和は図のように乗算結果の上位 4 ビットの部分和とパイプラインを 1 つ進めた状態の下位 4 ビットの部分和の和になる。(図 9)　計算が行われなかった部分は値 0 を加算器に流すことによって，正しい計算が行われる。

5．2　開始と終了の検知

スケーラブルに乗算を行うためには，それぞれのモジュールがカウンタ等の計数回路を持つことなく演算を進める必要がある。特に，計算開始前後のパイプラインの各段は値 0 を出力しなければならないため，この値がどのタイミングで出力されるかをそれぞれのモジュールが知る必要がある。この回路構成では，乗算モジュールを開始段と最終段とそれらの中間段に分けて考える。(図 10)

乗算はまず乗数 A を設定することから始まる。書く乗算モジュールは自分自身に乗数が入っていなければ，前段からきた乗数を受け取り，自分自身に乗数が入っていれば前段からきた乗数を次の段に送る。こうすることによって，乗数 A を段数にかかわらず設定することができる。

被乗数 B も同様に乗算モジュールに 4 ビットごとに

送られる。この際、（1）演算開始前には値 0 を出力する。（2）演算中は乗算器の結果を出力する（3）演算終了後は値0を出力する、の3つのモードを計数回路を導入することなく管理しなければならない。

（1）のモードは被乗数 B_0 を受け取ったモジュールが0出力を行うための制御信号を次の段に送ることによって，解決する。0 出力の制御を受け取った回路は，その制御を次の段に転送することによって，乗算に使われていないモジュールが値0を出力する。

（2）の状態は乗算が行われている状態であり，乗算に必要な A_n と B_m がそろった時点で乗算を実行し、結果を加算器に送出する。

（3）の状態を検知するためには，最集団が値 B_0 を受け取った時点で後ろ向きに制御信号を送ることによって，開始段が値0を出力し始めなければならないことを知らせる。開始段は最終段が0出力の制御信号を受け取るまで，その信号を送出する。最終段が0出力制御信号を受け取ると，全体の掛け算が終了したことになる。このことをモジュール全体に伝えるために，最終段は、開始段と加算器にこの制御を送出する。

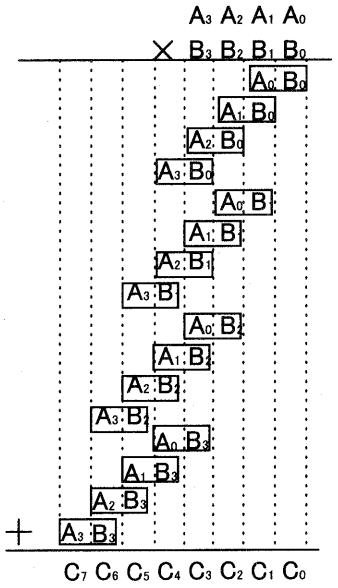

図7：筆算で表した乗算

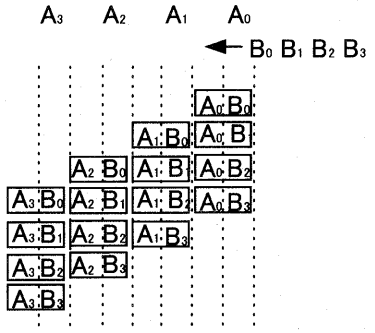

図8：部分積のパイプライン演算

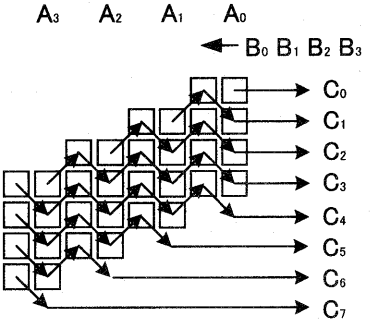

図9：部分和の計算方法

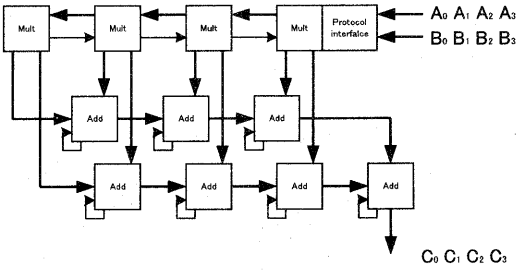

図10：16ビット時の乗算回路の構成

5．3　乗算回路のシミュレーション

これらプロトコルが正しいことを実験するために，それぞれ入出力する制御信号の違った3種類の乗算器と加算器のソフトウェアモジュールを作成し，プロトコルを確認した。この場合，中間の段にあたるモジュールの数と入力のビット数を変化させ，プロトコルと演算が正しい事を確認した。また、これらのモジュールは実装の複雑な乗算モジュールをソフトウェアでエミュレートし，比較的簡単な加算モジュールを PCA モジュールで実装した状態で強調シミュレーションを行うことによって，相互の誤りを訂正することができた。

6　まとめおよび今後の課題

本論文ではPCAモジュールの設計にSpecCにおける設計手法を取り入れた方法を提案した。この手法により、複雑であった上位の設計に対してシミュレーションが行えるようになり、フィードバックを設計の初期の段階で掛けることができるようになった。このようなことから、PCAオブジェクトの設計が容易になり、非同期回路の複雑さにとらわれることなくPCAの本質的な機能である再構成を十分に利用したアプリケーションが作成できると思われる。

今回の研究はPCAの設計アプローチに対する初の試みであり、不備な点も多い。これらのアプローチに対して実装すべき機能として、早期のタイミングの見積もりによるボトルネックの発見、上位仕様からのLUT使用面積の見積もりや消費電力の見積もり、非同期スレッド群に対するデバッグインターフェイス、下位記述からの非同期回路とPCAレイアウトの導出などが考えられる。これらの機能を追加していくことによって、PCAオブジェクトの設計を容易にかつ短期間で行えるようなシステムを構築できると思われる。

参考文献

[1] 塩澤恒道, Norbert Imlig, 小栗清, 佐野浩一,"PCAによる通信アプリケーションの一実現法,", 第13回パルテノン研究会資料集, pp 57-66 (1998)

[2] 管竜二, 泉知論, 中村行宏, "プラスティックセルアーキテクチャへのアレイ型論理マッピング手法", 第13回パルテノン研究会資料集, pp 67-76 (1998)

[3] 渡辺大洋, 檜田和宏, 中村行宏, "PCAにおけるLUTアレイへの回路レイアウトのための一手法", 第14回パルテノン研究会資料集, pp 67-76 (1999)

[4] 稲森稔, 中田広, 小西隆介, 小栗清,"PCA可変部への有限状態機械の配置配線法の提案", 第15回パルテノン研究会資料集, pp 57-66 (1999)

[5] 奥山祐市, 黒田研一, "SFLからの非同期論理抽出手法の提案", 第16回パルテノン研究会資料集, pp 21-30 (2000)

[6] 小西隆介, 伊藤秀之, 小栗清, 名古屋彰, "PCAの実現を支援するソフトウェア", 第16回パルテノン研究会資料集, pp 3-12 (2000)

[7] D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, S. Zhao, SpecC: Specification Language and Methodology, Kluwer Academic Publishers, Boston, MA, ISBN 0-7923-7822-9, March 2000

[8] D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, S. Zhao, "The SpecC Methodology" UC Irvine, Technical Report ICS-TR-99-56, December 1999.

[9] 塩澤恒道, Norbert Imlig, 小栗清「Communicating Logicによる汎用情報処理機構の実現」信学技報 CPSY99-91 (1999-11)