システムＬＳＩ設計技術　94－18
(2000. 1. 12)

動的な部分再構成デバイスを用いた仮想ハードウェアシステム

宇野 正樹† 柴田 裕一郎† 天野 英晴†
古田 浩一朗‡ 藤井 太郎‡ 本村 真人‡

†慶應義塾大学 情報工学科
〒223-8522 横浜市港北区日吉 3-14-1
E-mail: {uno, shibata, hunga}@am.ics.keio.ac.jp
‡NEC シリコンシステム研究所
〒229-1198 神奈川県相模原市下九沢 1120
E-mail: {furu, fujii, motomura}@mel.cl.nec.co.jp

あらまし

仮想ハードウェアシステム WASMII は、マルチコンテキスト化された書換え可能なデバイスを利用し、データ駆動的に制御しハードウェアの仮想化を実現するシステムである。これまでは、シミュレーションやエミュレータによってシステムの有効性が評価されてきたが、このようなデバイスが利用可能でなかったことから WASMII の実装を行うことは不可能だった。しかし、このほど NEC によって実用的なマルチコンテキスト化された書き換え可能デバイス DRL が開発され、利用可能となった。そこで、その柔軟な再構成機能を利用して DRL チップ上に WASMII システムの実装を行った。本稿では、この上での簡単なアプリケーションの実行とその評価結果について報告する。

キーワード 仮想ハードウェア, 動的再構成, 可変構造システム, データ駆動

A Virtual Hardware System on a Dynamically Reconfigurable Logic Device

M. UNO† Y. SHIBATA† H. AMANO†
K. FURUTA‡ T. FUJII‡ M. MOTOMURA‡

†Dept. of Information and Computer Science, Keio University
3-14-1 Hiyoshi, Kohoku-ku, Yokohama 223-8522, Japan
E-mail: {uno, shibata, hunga}@am.ics.keio.ac.jp
‡Silicon Systems Laboratories, NEC Corporation
1120 Shimokuzawa, Sagamihara 229-1198, Japan
E-mail: {furu, fujii, motomura}@mel.cl.nec.co.jp

Abstract

WASMII is a virtual hardware system that executes dataflow algorithms using a dynamically reconfigurable multi-context device with a data driven control mechanism. Although the effectiveness of the system has been evaluated through simulations and using an emulator, implementation of WASMII was infeasible due to the unavailability of such a device. However, the first practical dynamically reconfigurable multi-context device called DRL was developed by NEC. Using the best use of its flexible reconfigurability WASMII has been implemented on the DRL chip. Here, the implementation and evaluation of some simple applcations with WASMII on DRL is presented.

key words virtual hardware, dynamic reconfiguration, reconfigurable system, data driven

1 はじめに

FPGA (Field Programmable Gate Array) や CPLD (Complex Programmable Logic Device) などのプログラム可能なデバイスの技術的発展は目覚しく、計算機アーキテクチャの分野にも大きなインパクトを与え続けている。中でもアルゴリズムを直接 FPGA 上のハードウェアとして実現する**可変構造システム**は、従来の計算機とは全く異なった原理に基づく計算システムとして、最近ますます広く研究されている [1]–[4]。

これらのシステムは、アプリケーション毎に最適なハードウェア構成及びビット幅をとることや、対象とするアルゴリズムをそのままハードウェア化することにより、従来のシステムに比べ高速に演算を行える可能性がある。このため、アプリケーション専用ハードウェアの持つ高い性能と汎用計算機の持つ柔軟性を兼ね備えたシステムとして期待されている。しかしながら、アルゴリズムをハードウェア化する標準的手段がないことに加え、対象の問題の規模が大きくなり、必要なハードウェア量がシステムのサイズを超えてしまうと、全く計算不可能になってしまうという問題点があった。そこで、我々は仮想記憶の概念を可変構造システムに応用したデータ駆動型仮想ハードウェア WASMII [5][6] を提案している。WASMII はマルチコンテキスト化された書き換え可能なデバイスを、データ駆動的に制御することでハードウェアの仮想化を実現するシステムである。

このようなデバイスはこれまで利用可能でなかったことから、我々は主に通常の FPGA を複数用いたエミュレータ [7] を使用することによってシステムの有効性を検討してきた。しかし、通常の FPGA では動的な回路の書き換えを行うためには、その都度多大な時間をかけてチップ外部から構成情報を書き込む必要がある。このため、最終的に WASMII チップを実装した際に、コンテストの切り替えに要する時間がどの程度性能に影響するかを具体的に示すには至っていなかった。

一方、昨年になって、実用的なマルチコンテキスト化された書き換え可能デバイス DRL [8] が NEC によって実装された。そこで、我々は DRL の持つ動的な部分再構成機能を用いて仮想ハードウェア WASMII のメカニズムをチップ上に実現し、その上で簡単なアプリケーションを実行させることにより、WASMII の実現可能性を示すとともにコンテスト切り替え時間の影響などを評価した。

以下、本稿はまず 2 章で仮想ハードウェア WASMII について概観したのち、3 章で DRL チップの構成について述べる。続く 4 章では、どのように DRL 上に WASMII のメカニズムを実現したかについて詳しく述べ、5 章では実際にこの上でアプリケーションを実行させた結果について考察する。

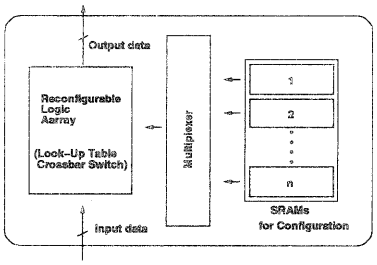

図 1: マルチコンテキスト化された書き換え可能デバイス

2 仮想ハードウェア WASMII

2.1 マルチコンテキスト化された書き換え可能デバイス

書換え可能なデバイスの利用は、ハードウェアの構造の動的な変更を可能にする。しかし、通常の FPGA ではハードウェア構成を変えるためには外部から結線情報を送り直す必要があり、このために大きな時間を要する。そこで、書き換え可能なデバイスをマルチコンテキスト化する方法が考えられている。すなわち、図 1 に示すように、デバイス内部の結線情報用の SRAM を複数のセットに拡張した構成を持たせる。これをマルチプレクサで切り替えることにより、デバイス上の回路を高速に入れ換えることが可能となる。

このようなデバイスの考え方自体は比較的古くからあり、例えば富士通によって提案された MPLD (Multi-functional Programmable Logic Device) [9] などがある。また、近年では Xilinx 社によって設計検討がなされた時分割 FPGA [10] などがある。この時分割 FPGA は同社の XC4000E を拡張したものであり、約 10,000 ゲート (400 CLB) に相当する回路の結線情報をチップ内に 8 セット保持することができ、チップ内部での回路の交換は約 30 ns で可能と報告されている。

2.2 WASMII の構成

マルチコンテキスト化された書き換え可能なデバイスを用い、動的に回路を交換することで、FPGA の見かけの回路規模を大きくすることができる。しかし、任意のアルゴリズムをハードウェア化する手段がまだ確立されていないうえ、一般的な論理回路では結線情報ページの切り替えのタイミングの制御は困難である。

この問題を解決するために、WASMII ではページ切替えの制御を含む処理全体にデータ駆動の考え方を導入した。WASMII では、まず対象とする問題をデータフローグラ

フに変換し、このグラフを結線情報ページひとつで実現可能なサイズのサブグラフに分割する。各サブグラフに対応する一連のページは、その全ての入力アークにトークンが到着し、実行可能になった時点でデバイス上に実現される(以下アクティベイトと呼ぶ)。サブグラフ内の各演算ノードもデータ駆動的に動作する。すなわち、それぞれのノードでは全ての入力アークにデータが揃うとその演算処理を行い、結果を次のノードの入力に転送する。

もし、静的なデータフローグラフをそのままハードウェア化したならば、各演算ノード内の記憶素子は演算中の一時的なデータを記憶するだけでよく、演算終了後はその内容を保持する必要がない。これは静的なデータフローグラフが副作用を持たないことに対応する。したがって、コンテキストの切替え時に順序回路の状態が失われてしまうようなデバイスにも、システムの制御メカニズムを変更することなく対応可能である。また、切り替え時にコンテキストの状態をバックアップするような機能がデバイスに備わっていれば、データフローグラフ内で局所的なループを構成するトークンをノード内に状態として保持することにより、演算効率を高めることができる。さらにWASMIIでは、より大規模なアプリケーションにも対応するために、仮想記憶の方式にならって結線情報用のバックアップRAMを外部に付加し、利用していないページに対するチップ内外での結線情報の入れ換えを行うことも可能となる。

仮想ハードウェアWASMIIは、上記の操作を実現するために、図2に示す構造を持つ。実行中のページから出力されたトークンは、トークンルータによって対応する入力トークンレジスタに送られる。ページコントローラは、この入力トークンレジスタのフラグをチェックすることにより、ページが実行可能かどうかを知り、現在実行中のページから全てのトークンが出力された後に、新しいページをアクティベイトする。同時に複数のページが実行可能になった場合は、あらかじめ決められた優先順位に従って、その中から1ページを選びアクティベイトする。

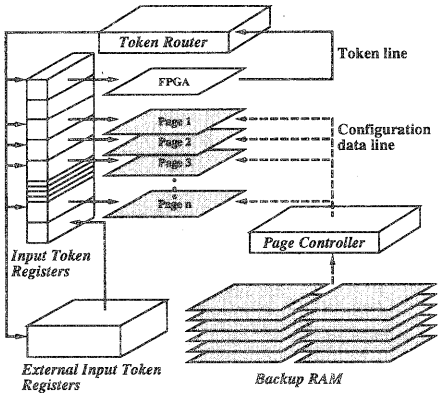

図2: 仮想ハードウェアWASMII

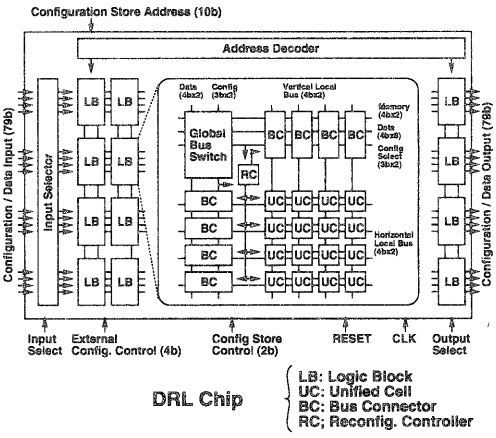

図3: DRLチップのブロック図

3 動的再構成ロジック DRL

3.1 チップの構成

DRLはNECによって開発された、マルチコンテキスト構成を持つ部分再構成可能な論理デバイスである。DRLチップは図3に示すように3つの階層からなり、チップ全体は4×12の論理ブロック(LB: Logic Block)および外部入出力用の制御回路から構成される。それぞれのLBの内部は、4×4の統合型セル(UC: Unified Cell)、チップ長の配線資源であるグローバルバスとの接続を行うバスコネクタ(BC: Bus Connector)、および動的再構成制御回路(RC: Reconfiguration Controller)から構成される。UCには次の3つのモードが用意されている。

- ルックアップテーブル(LUT)モード

  UCを4入力1出力あるいは3入力2出力の任意の論理関数が実現可能なLUTとして使用するモード。

- 配線モード

  縦横各4本の配線(ローカルバス)を任意の交点で接続可能なクロスバスイッチとして使用するモード。

- メモリモード

  アドレス入力 4bit、データ入出力 1bit のメモリ(ROM/RAM)として使用するモード。

このように、UCは従来のFPGAにおける論理セルと配線セルを統合した形となっており、論理と配線資源の配分比率を可変とすることで、回路タイプ(例えばデータパ

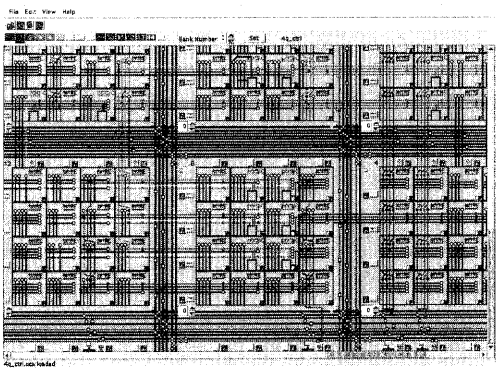

図 4: DRL エディタツール

スとランダムロジック) に応じた効率よい回路実装を可能としている。また、各 UC には 1 個のフリップフロップが設けられており LUT モードの場合、出力の値を保持することができる。また、このフリップフロップはバックアップモードに設定することで、再構成の際にデータをバックアップメモリへ書き戻すことが可能である。このことにより、他のコンテキストの回路が動作した後に再度アクティベイトされた場合にも、前の演算の状態を回復して計算を続けることができる。

実現される回路の動的な再構成は LB の単位で行うことができ、各 LB は 8 通りの構成情報を持っている。再構成の制御は、チップ外部からの制御信号あるいはチップ内部に構成した任意の場所の回路の出力信号のいずれからも行うことができ、これらも LB 毎に指定することが可能である。

DRL チップは 0.25μm の CMOS プロセスで実装されており、510 万トランジスタを 10.1mm 角のチップサイズに集積している。動的な再構成は 1 クロック (最短 4.6ns) で行うことができる。

3.2 開発環境

現在、DRL の開発環境として NEC で開発されたエディタツールがある。図 4 に示すように、このエディタツールを用いることによって、論理機能の配置、UC のモード設定、信号線のルーティングなどチップ上に回路を実現する作業を、GUI を介して人手で行うことが可能である。また、エディタツールには入力された回路を DRL 上に実現するために必要なコンフィギュレーション情報を生成する機能があり、さらにこの情報は簡単な処理によりそのまま DRL の Verilog モデルと合わせてシミュレーションすることも可能である。

動的に切り替わる複数のコンテキストからなる回路を設計する場合も、それぞれのコンテキストに対応する論理回路を同様にエディタツールで入力する。動的な再構成の処理も Verilog モデルによってあらかじめシミュレートすることができる。

4 WASMII システムの実装

4.1 制御部と演算部の配置

DRL の部分的再構成の機能は、WASMII システムを実装するために有用である。WASMII システムは制御部と演算部に分けて考えられる。制御部はトークンレジスタやページの切り替え制御回路を含み、アプリケーションの実行中は構成が変化することがない。演算部は実際に演算処理を行う部分であり、制御部からの指令によって動的に構成を変化させながら演算を進めていく。DRL の部分的再構成機能を用いると、演算部と制御部が単一チップ上で実現できる。図 5 に演算部と制御部の概念図を示す。

図 5 において、斜線部が今回のマッピングでコンテキストを固定とした部分である。DRL チップ上で、左の 32 個 (4×8) の論理ブロックが演算部で、右の 16 個 (4×4) の論理ブロックが制御部となる。演算部からは、トークンレジスタへの書き込みに用いる write_enable 信号、トークンの目的ページを指定するための addr、ページでの演算が終了したことを知らせる page_done 信号を制御部に対して入力する。制御部からは、入力トークンレジスタに格納されたデータと、演算部での演算の準備が整ったことを知らせる strobe 信号を送る。また、制御部はアクティベイトすべきページのコンテキスト番号を演算部に配る。

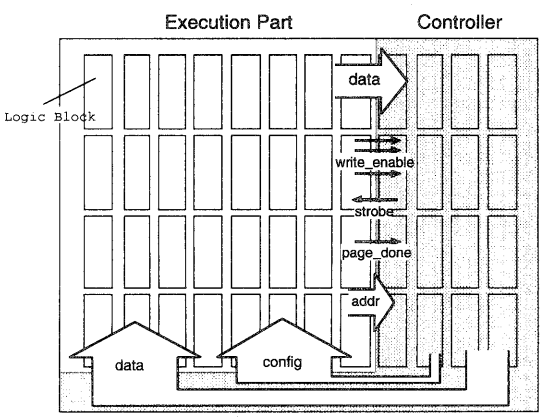

図 5: 構成面中の固定部分

4.2 制御部の構成

制御部を構成する主なコンポーネントは以下の通りである。

- 入力トークンレジスタ
  各演算ページがアクティベイトされるまで当該ページへの入力トークンを保持する。
- トークン到着フラグ
  入力トークン1つにつき1bitのエントリを持ち、トークンの持ち合わせ情報を管理する。
- レジスタアドレス生成器 (カウンタ)
  現在アクティベイトされているページの全演算終了後、制御 部では次にアクティベイトすべきページを決定するために、レジスタアドレスカウントアップしながら生成する。ページ制御回路はこのアドレスに従って順にトークン到着フラグをチェックし、初めて見つかった実行可能なページ (全てのトークンのそろったページ) をアクティベイトする。
- カレントページレジスタ
  現在アクティブなページの番号 (3bit) を保持する。

また、制御部は、次に示す簡単な3状態のステートマシンによって制御される。

- トークン書き込み状態
  アクティベイトされたページ内で演算が行われている状態であり、制御部では演算ページから出力されたトークンを入力トークンレジスタに書き込み、到着フラグをセットする。
- ページ探索状態
  次にアクティベイトすべきページを探索する状態であり、レジスタアドレス生成器のカウンタをインクリメントしながら到着フラグをチェックする。実行可能なページが見つかった時点で、次状態へ遷移する。この際カレントページレジスタを更新し演算部の再構成を行う。このページ探索処理はトークンの到着状況によって左右されるため、実行可能なページを見つけるのにかかる時間は場合によって変化する。できるだけ短い時間でこの処理を行うために、探索は演算部が最後にトークンを書き込んだページから行われる。
- トークン発行状態
  新たにアクティベイトされたページへ入力トークンを供給する状態である。当該ページの入力トークンレジスタの値を同時に読み出し、到着フラグをリセットする。また、同時にページがアクティベイトされたことを示すstrobe信号 (1クロック分) も発行する。次のクロックでトークン書き込み状態に遷移する。

実装した制御部では演算部から同時に1個のトークンしか受け取ることができない。従って、演算部はトークンの書き込みの際に、write_enableなどのタイミングを制御し時分割で書き込みを行う必要がある。

4.3 演算部の構成

演算部は、制御部からトークンの入力を受け演算処理を行う部分と、write_enable、page_done、目的ページなどを制御部に指定する部分から構成される。

演算処理は制御部からstrobe信号が送られると同時に開始される。演算を行った次のクロックからは、その演算結果を保持し、必要であればトークンの目的のページやレジスタに対してトークンの書き込みを行う。この際、write_enableや対象のページを時分割で制御する必要がある場合は、簡単なカウンタでタイミングを制御する。このカウンタは、strobe信号の入力でリセットされ、その後カウントアップを始める。

全てのトークンの書き込みが終了すると、演算部での動作が終了したことを示すpage_done信号を発行する。これ以後、演算部の動作は停止し、次のページがアクティベイトされるまで待機する。

4.4 初期化のメカニズム

今回の実装では、制御部のstrobe信号の発行により演算部の動作を開始する。しかし、制御部は動作の開始時には、コンテキスト番号が0の演算面をアクティベイトするとともに、トークン書き込み状態に遷移するため、最初に構成される0番のコンテキストにはstrobe信号は発行されない。

そこで、コンテキスト0ではアプリケーションの動作開始時には、次にアクティベイトされるべきページの入力トークンレジスタが満たされるように、トークンの書き込みを行う初期化の作業が必要である。また、この機構を利用して、コンテキスト0のみの結線情報を書き換えるだけで、容易にアプリケーションで用いられる初期値などを変更することができる。

5 評価

以上に述べたように、DRL上にWASMIIシステムを実現し、実チップ上でアプリケーションを実行させてシステムを評価した。今回、実装したアプリケーションは、連続系シミュレーションの分野で用いられるVan der Polの常微分方程式をオイラー法で解くプログラムと、N-Queen問題 (N=4) をニューラルネットワークを用いて解くプログラムである。

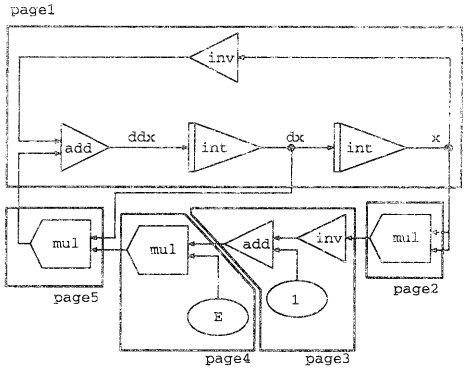

図 6: Van der Pol の方程式の結合グラフ

5.1 評価手順

今回のアプリケーションの評価手順は以下の通りである。

1. アプリケーションをデータフローグラフで表現し、演算部で実装可能な規模に分割する。
2. 演算部の各構成面及び制御部をエディタツールを用いて DRL 上に実現する。そして、エディタにチップのコンフィギュレーション情報を生成させる。
3. エディタツールが生成したコンフィギュレーション情報を用い、Verilog モデルでシミュレーションを行う。ここで、ルールチェックを行う。
4. ロジックテスタを用い、実チップ上でアプリケーションの動作を確認する。チップのコンフィギュレーション情報と入力パターンと出力の期待値パターンは、Verilog モデルを用いて出力させた。ロジックテスタ上では動作検証とともに動作周波数の測定が可能である。

5.2 Van der Pol の常微分方程式

連続系シミュレーションの分野で用いられる Van der Pol の常微分方程式は、初期値に関わらず常に一定の解曲線に収束することが知られ、次の式で表わされる。

$$\frac{d^2x}{dt^2} - E(1-x^2)\frac{dx}{dt} + x = 0 \qquad (E > 0)$$

この式の求解アルゴリズムは以下のようなる。今回は、データフローグラフを図 6 のように分割した。

$$\begin{aligned} ddx &= E(1-x^2)dx - x \\ dx &= \int ddx \\ x &= \int dx \end{aligned}$$

図 6 において、int は積分器、add は全加算器、inv はインバータ、mul は乗算器である。乗算器はアレイ型の乗算器で、Baugh-Wooley の 2 の補数乗算器と呼ばれるものである。また、積分器では前のサイクルの値を保持しておく必要があるため、構成面が切り換わってもその内容が消えないように DRL のバックアップ機能を利用した。図 6 に示したページの他に、この回路の初期値を入れるための回路をマッピングしたページ 0 という構成面も用意した。よって、演算部は全体でページ 0 からページ 5 の 6 面で構成される。

電源電圧 3.1V の場合の各回路の動作周波数を表 1 にまとめた。ページ 2、4、5 が最も遅くなったのは、複雑な乗算器のためと考えられる。また、今回の Van der Pol の求解アルゴリズムが 1 ステップ分の計算 (1 組の x と dx の値を求める計算) に要するサイクル数は 17 クロックであった。最も遅い回路の動作周波数 8.6MHz でアプリケーション全体を実行すると、約 1.97μ 秒ごとに 1 ステップの計算がなされることとなる。

表 1: 各回路における動作周波数

回路	動作周波数 (MHz)
ページ 0	50.0
ページ 1	11.2
ページ 2	8.6
ページ 3	14.3
ページ 4	8.6
ページ 5	8.6
制御部	11.4

次に 1 ステップ (17 クロック) あたりの処理の内訳について表 2 に示す。DRL がページを切り替えるのにかかる時間は 1 回あたり 1 クロックのみであり、今回は極端に分割されたアプリケーションだったにもかかわらず、ページ切り替えそのもののオーバヘッドは 29%程度であった。なお、表中に「メモリ処理」と示した処理は、DRL のメモリ構成が同期型の構造を持ちアドレス投入の次のクロックでデータが出力されることと関連したオーバヘッドである。また、このアプリケーションでは演算部が実行を終えてから、制御部が次に実行可能なページを探索するのに時間を要さなかった。これは、演算部が最後にトークンを書き込んだページからトークンフラグをチェックする手法の有効性を示している。

一方、全体の処理に占める制御のオーバヘッドの割合が約 59%近くになった。これは、単一構成面上で利用可能なハードウェアリソースがそれほど多くなかったことから、ページ内で演算の並列性を抽出することが出来なかったうえ、ページ切り替えの頻度も相対的に増えてしまったことによると考えられる。

表 2: 1 ステップあたりの処理の内訳

処理内容		クロック数	割合 (%)
演算処理		7	41.2
(内訳)	ページ 1	3	17.6
	ページ 2	1	5.9
	ページ 3	1	5.9
	ページ 4	1	5.9
	ページ 5	1	5.9
オーバヘッド		10	58.8
(内訳)	ページ探索	0	0.0
	ページ切り替え	5	29.4
	メモリ処理	5	29.4

5.3 N-Queen 問題

次に、Hopfield 型のニューラルネットワークを用いて、N-Queen 問題を解くプログラムを実装した。[11]。N-Queen 問題は $N \times N$ のチェス盤に、N 個のクイーンの駒を互いに取り合わないように配置する問題であるが、これをニューラルネットワークを用いて解く場合は、チェス盤の各升目に対応した N^2 個のニューロンを用意する。ニューロンはそれぞれ独立したエネルギーを持ち、そのエネルギーに従ってそこへクイーンを置くかどうかを決定する。i 行 j 列に位置するニューロンのエネルギーを U_{ij} で表すとすれば、エネルギーの更新は次の式に従って行われる。

$$\frac{dU_{ij}}{dt} = -\sum_{k=1}^{N} V_{ik} - \sum_{k=1}^{N} V_{kj} - \sum_{\substack{1 \le i-k,\ j-k \le N \\ k \neq 0}} V_{i-k,j-k} - \sum_{\substack{1 \le i-k,\ j+k \le N \\ k \neq 0}} V_{i-k,j+k} - 2$$

ここで、V_{ij} は 0 または 1 のどちらかの値をとるが、1 のときは i 行 j 列の升目にクイーンが置かれた状態を、0 のときはそこへ駒が置かれていないことを示す。なお、この Hopfield 型のニューラルネットワークは、VLSI の最適配線問題など他の多くの組合わせ最適化問題に応用可能である。今回は DRL チップの利用可能なハードウェアリソースの制限から、クイーン数を 4 とした。アルゴリズムをデータフローグラフで表わし、これをページ単位に分割すると 8 ページとなった。8 ページの処理内容は大きく 2 種類に分けることができ、同じ種類のページの構成はほぼ同じになった。

- ページ 0,1,2,3
  それぞれ 4 個分のニューロンのエネルギーの計算、保持、及び、Queen の配置決定を行う。減算器、カウンタ、インバータ等から構成される。
- ページ 4,5,6,7
  チェス盤上の縦横斜めの各成分に含まれる Queen の数を計算する。加算器、セレクタ、カウンタ等から構成される。

ニューロンのエネルギーはページが切り換わっても保持しなければならないので、フリップフロップのバックアップ機能を利用した。また、8 面の構成面を全て使用したため、Van der Pol の方程式の場合のように初期値を与えるページを確保できなかった。そこで、ページ 0 の未使用の部分を利用して、外部から初期化ビットを立てると初期化の動作を行なうような回路を付け加えた。

電源電圧 3.1V の場合の各回路の動作周波数を表 3 にまとめた。ページ 0 は、ページ 1、2、3 とほぼ同じ構成を持つが、初期化をするための回路を付け加えたためにやや遅くなった。また、今回の 4-Queen の求解アルゴリズムが 1 ステップ分の計算 (全てのニューロンのエネルギーを更新する計算) に要するサイクル数は 74 クロックであった。最も遅い回路の動作周波数 9.1MHz で実行すれば、約 8.1μ 秒ごとに 1 ステップの計算がなされることとなる。

表 3: 各回路における動作周波数

回路	動作周波数 (MHz)
ページ 0	9.1
ページ 1	16.7
ページ 2	16.7
ページ 3	16.7
ページ 4	13.9
ページ 5	13.9
ページ 6	13.9
ページ 7	13.9
制御部	9.9

次に 1 ステップ (74 クロック) あたりの処理の内訳について表 2 に示す。このアプリケーションでは、ページの切り替え時に次にアクティベイト可能なページを探索するのに平均 2.25 クロックかかっており、この処理にかかる時間が制御オーバヘッドの約半分を占めている。このオーバヘッドは制御部に利用可能なハードウェア規模が大きくなれば、例えばトークンの到着フラグの複数のエントリを一度に比較したりすることによりある程度改善できると考えられる。例えば、全ページのフラグを一度に比較することができると、ページ探索にかかるクロックは無くす事がきる。また、各演算ページでの実行時間が 5 クロックと Van der Pol 方程式よりも大きかったため、全体の実行時間に占めるページ切り替え時間の割合は約 10%程度と小さくなり、純粋に演算に費やされた時間は 50%をわずかに超えた。

5.4 演算性能に関する考察

以上に述べたように、DRL が動的かつ部分的にチップを再構成できる機能を用いて、仮想ハードウェア WASMII のメカニズムを実現し、実際にアプリケーションを動作さ

表 4: 1 ステップあたりの処理の内訳

処理内容		クロック数	割合 (%)
演算処理		40	54.4
(内訳)	ページ 0	5	6.8
	ページ 1	5	6.8
	ページ 2	5	6.8
	ページ 3	5	6.8
	ページ 4	5	6.8
	ページ 5	5	6.8
	ページ 6	5	6.8
	ページ 7	5	6.8
オーバヘッド		34	45.6
(内訳)	ページ探索	18	24.2
	ページ切り替え	8	10.7
	メモリ処理	8	10.7

せることができた。しかし、現在の DRL は、実用的なマルチコンテキスト構造を持つ書き換え可能デバイスとしては初めて実際に実装されたチップであり、プロトタイプ的な側面もあるためそれほど大規模な回路を構成できない。このため、演算部ではうまく演算の並列性を活かすことが出来なかったうえ、制御部も性能向上のために複雑な構成をとれなかった。

このため、今回のシステムでは可変構造システムの特徴を活かした性能を示すことはできなかった。例えば、高性能 PC(Celeron 450MHz、FreeBSD) 上で、前述の C 言語によって開発された 4-Queen 問題を解いた場合、1 ステップあたりの実行時間は 2.3μ 秒であり、DRL 上の WASMII と比較して 3.5 倍高速だった。そこで、本システムの性能を今後どの程度まで改善できるか考えてみる。さらなる性能の改善には、大きく分けて 2 通りの可能性が考えられる。一方は、アーキテクチャ的な改善であり、もう一方はデバイス側の改善である。

アーキテクチャの側面からは、例えばチップを複数用いて並列システムを構成することが考えられる。WASMII は制御部の簡単な変更で用意に並列化可能であり、N-Queen 問題では実際にエミュレータを用いた評価で 3 並列で 1.9 倍の性能向上を得られることが分かっている [12]。また、今回の評価からも明らかなように、制御に関するオーバヘッドが全実行時間の約半分を占めているが、制御部を高機能化することによって、このかなりの部分が削減できると考えられる。

また、デバイスの観点からは、例えば 0.13μm の CMOS プロセスを用いてデバイスを 10mm 角のチップで作ったとすれば、現在よりも 1 面あたり 4 倍以上の回路を構成可能と見積もられる。また、設計検討の段階ではあるが、セルや配線の構成などを変更することにより、実アプリケーションの動作速度を 100MHz 程度までに改善できると考えられている。

したがって、アーキテクチャおよびデバイスの両面からの改善を加えることにより、最終的には高性能汎用 PC の 5 から 6 倍の性能を持つ、仮想ハードウェアシステムが実現可能であると考えている。

6 おわりに

本稿では、動的かつ部分的にチップを再構成可能なデバイス DRL を用いて、仮想ハードウェア WASMII のメカニズムを実装し、実際にアプリケーションを実行することにより評価を行った。今後は、DRL を搭載したボードの開発などを行い、ソフトウェア環境を含んだ WASMII システム全体の評価を行っていきたいと考えている。

参考文献

[1] 沼昌宏: FPGA を利用したアーキテクチャとシステム設計, 情報処理, Vol. 35, No. 6, pp. 511–518 (1994).

[2] 末吉敏則: Reconfigurable Computing System の現状と課題—Computer Evolution へ向けて—, 信学技報, Vol. 96, No. 426, pp. 111–118 (1996).

[3] Miyazaki, T.: Reconfigurable Systems: A Survey, *Proc. Asia and South Pacific Design Automation Conference*, pp. 447–452 (1998).

[4] Amano, H. and Shibata, Y.: Reconfigurable Systems: Activities in Asia and South Pacific, *Proc. Asia and South Pacific Design Automation Conference*, pp. 453–457 (1998).

[5] Ling, X. and Amano, H.: WASMII: a Data Driven Computer on a Virtual Hardware, *Proc. Intl. Conf. on FPGAs for Custom Computing Machines*, pp. 33–42 (1993).

[6] Ling, X. and Amano, H.: WASMII: An MPLD with Data-Driven Control on a Virtual Hardware, *J. Supercomputing*, Vol. 9, No. 3, pp. 255–276 (1995).

[7] 柴田裕一郎, 宮崎英倫, 日暮浩一, 凌暁萍, 天野英晴: 仮想ハードウェア WASMII のエミュレーション環境の構築, FPGA/PLD Design Conference & Exhibit 予稿集, pp. 233–239 (1997).

[8] Fujii, T., Furuta, K., Motomura, M., Nomura, M., Mizuno, M., Anjo, K., Wakabayashi, K., Hirota, Y., Nakazawa, Y., Ito, H. and Yamashina, M.: A Dynamically Reconfigurable Logic Engine with a Multi-Context/Multi-Mode Unified-Cell Architecture, *Proc. Intl. Solid-State Circuits Conf.*, pp. 360–361 (1999).

[9] 吉見昌久: マルチファンクションプログラマブルロジックデバイス, 公開特許公報 (a), 平 2-130023 (1990).

[10] Trimberger, S., Carberry, D., Johnson, A. and Wong, J.: A Time-Multiplexed FPGA, *Proc. Intl. Conf. on FPGAs for Custom Computing Machines*, pp. 22–28 (1997).

[11] Takefuji, Y.: *Neural Network Parallel Computing*, Kluwer Academic Publishers (1992).

[12] Miyazak, H., Shibata, Y., Takayama, A., Ling, X. and Amano, H.: Emulation of Multichip WASMII on Reconfigurable System Testbed FLEMING, *Proc. PACT Workshop on Reconfigurable Computing*, pp. 47–52 (1998).