仮想ハードウェアWASMIIシステム用コンパイラの実装と評価

高山 篤史* 岩井 啓輔* 柴田 裕一郎* 宮崎 英倫*
日暮 浩一† 凌 暁萍‡ 天野 英晴*

* 慶應義塾大学 計算機科学専攻
〒223-8522 横浜市港北区日吉 3-14-1
† 日立製作所 ソフトウェア開発本部
〒244-8555 横浜市戸塚区戸塚町 5030
‡ 神奈川工科大学情報工学科
〒243-0292 厚木市下荻野 1030

e-mail: wasmii@am.ics.keio.ac.jp

概要

仮想ハードウェアWASMIIは、データ駆動型制御に基づく可変構造システムであり、データフローグラフの形で表現された問題の解を求める計算機アーキテクチャである。WASMIIは、書き換え可能なFPGAを拡張したMPLDに基づく仮想ハードウェアの概念により、従来の可変構造システムで不可能だったFPGAの許容を超える問題でもいくつかの問題に分割し、解を求めることが可能である。ここで重要な課題は、データフローグラフをいくつかのサブグラフに分割し、さらにWASMIIで実行可能な形に変換することである。本論文では、ユーザの記述したアプリケーションをデータフローグラフの形に変換し、いくつかのサブグラフに分割した後、それぞれに対応するハードウェア記述言語を生成するコンパイラの実装について述べる。また、その際に用いるデッドロック回避を基軸としたグラフ分割の手法としてIt/t法を提案する。最後に、本研究で開発したコンパイラを用いて、中間コードで記述したアプリケーションをコンパイルし、FLEMING上で実行させて得られた評価について述べる。

キーワード 仮想ハードウェア, データ駆動型制御, 可変構造システム, FPGA, ハードウェア記述言語, コンパイラ

Making and Evaluation of the Compiler for WASMII, a Virtual hardware system

A. Takayama* K. Iwai* Y. Shibata* H. Miyazaki*
K. Higure† X.-P. Ling‡ H. Amano*

* Dept. of Computer Science, Keio University
† Hitachi, Ltd. Software Development Center
‡ Dept. of Information and Computer Sciences, Kanagawa

e-mail: wasmii@am.ics.keio.ac.jp

Abstract

abstract

WASMII is a reconfigurable system with the data driven control which executes programs written in dataflow graphs. In WASMII, a target dataflow graph is divided into some subgraphs and executed on a programmable device called MPLD which is an extended FPGA. By replacing the configuration data on the MPLD, large scale programs which exceed the limit of hardware resources can be efficiently executed. As a software environment of WASMII, a compiler which translates a program written by a user in a high-level language into a corresponding dataflow graph and its HDL description is required. In this paper, we show the design and implementation of the compiler for WASMII which generates the VHDL description from an input program, and introduce It/t method, which is a deadlock free graph-partitioning algorithm. Compilation and execution results of a test program on a reconfigurable testbed called FLEMING are also shown.

key words Virtual hardware, Data driven control, Reconfigurable system, Hardware Description Language, FPGA, Compiler

1 はじめに

近年、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]–[2]。

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うことが可能である。しかしながら、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る。

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**仮想ハードウェア WASMII**[3][4] を提案している。WASMII は、チップ内に複数セットの結線情報用 RAM を持った FPGA の外部にバックアップ RAM を付加し、これ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。また、この WASMII チップを複数接続して並列に動作させるシステムをマルチチップ WASMII と呼んでいる。この WASMII システムは、本研究室で実装された可変構造システムテストベッド FLEMING(Flexilble Logic EMulatIon eNGine)[5] 上でのエミュレーションにより、その実用性・有効性が明らかにされている [7]。

しかし、WASMII システムのソフトウェア環境の整備は非常に遅れており、特に問題記述の際には尋常ではない手間と時間がかかる。現在、コンパイラとしては既存の WASMII データフローコンパイラ [6] が利用可能であるが、いくつかの問題点を抱えているため、実用的ではない。そこで、本研究ではこのデータフローコンパイラに変わる新しいコンパイラの開発および実装について述べる。

また、実装したコンパイラを用いて実際に WASMII アプリケーションを記述し、FLEMING 上で動作させて評価を行った。

2 仮想ハードウェア WASMII

2.1 SRAM 型マルチコンテキスト FPGA

書換え可能な FPGA を利用することにより、ハードウェアの構造を動的に変更することが可能となる。しかし、FPGA 上の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、富士通では MPLD (Multifunction Programmable Logic Device)[8] と呼ばれる拡張された FPGA を提案している。

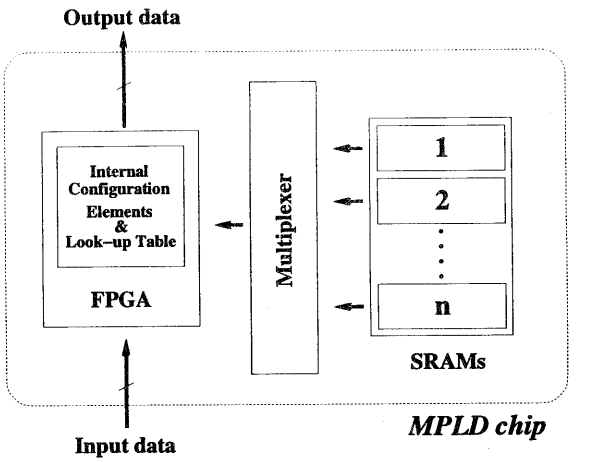

図 1: MPLD の構造

MPLD は図 1 に示すように、FPGA 内部の結線情報用の SRAM を複数セットに拡張した構成をとっている。各 SRAM のセット（以降ページと呼ぶ）は単一の FPGA に必要な結線情報を保持しており、これをマルチプレクサで切り替えることによって、FPGA 上の回路を数十 ns で高速に入れ換えることが可能となる。本論文では、このようなチップを SRAM 型マルチコンテキスト FPGA と呼ぶことにする。

2.2 WASMII の構成

SRAM 型マルチコンテキスト FPGA 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、各結線情報ページは純粋に FPGA 上の回路構成のみを決定するため、実現された回路上のレジスタの内容や順序回路の状態が、ページの切り替えによって消失してしまう。このため、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、FPGA 上に実現する（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完全に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。このとき、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しており、これによりページ切替え時の状態喪失の問題を解決することができる。

さらに、システムサイズに関する制限を取り除くため、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可能にしている。

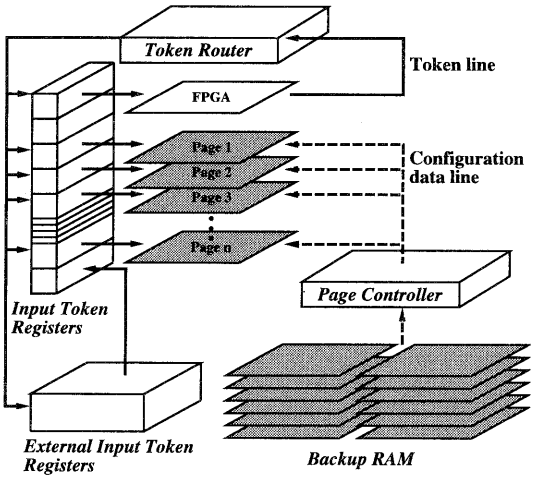

図 2: 仮想ハードウェア WASMII

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り、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し、演算を進めていく。

3 WASMII システム用コンパイラ

WASMII システムにおける問題記述は以下のような流れで行われる。

- 対象とするアプリケーションをデータフローグラフの形で表現する
- データフローグラフを適当な大きさに分割する
- サブグラフ間を移動するトークンに入力トークンレジスタを割り当てる
- サブグラフ間の静的スケジューリングを行う
- サブグラフをハードウェア記述言語で記述する

WASMII 用コンパイラはエントリの言語で記述されたアプリケーションモデルを受け取り、これらを全て自動で行うことを目的とする。

3.1 データフローコンパイラ

WASMII システムのソフトウェア環境として、データフローコンパイラが利用可能である。このコンパイラは、C 言語で記述されたアプリケーションを WASMII で実行可能な VHDL 記述に変換する。フロントエンドには既存のトランスレータである DFC[10] を用いている。DFC は、電子技術総合研究所で開発されたデータフローマシン SIGMA-1[9] 上で大規模な科学技術計算を行うために開発されたトランスレータであり、C 言語を入力として、SAS と呼ばれるデータフロー言語を生成する。しかし、データフローコンパイラは以下のような問題点を抱えている。

- マルチチップ WASMII に対応していない
- グラフの自動分割をサポートしていない
- 入力トークンのレジスタ割り当てをサポートしていない
- 演算器のビット幅をユーザが指定することができない
- フロントエンドが WASMII システム用に開発されたものではないため、中間コードに問題がある

上記の理由から、データフローコンパイラを拡張して必要な機能を全て持たせることは困難であると判断し、WASMII システム用コンパイラをフロントエンドから新たに実装することにした。今回は、フロントエンドを睨みつつ、需要の高いバックエンドの設計・実装を行った。

3.2 バックエンドの設計・実装

中間コードは、フロントエンドとバックエンドの双方の実装のしやすさも考慮し、以下のようにデータフローグラフの各ノード間の接続をポートマップとして記述した、ノード毎の記述とした。

```
1 mult (arg0,arg1,n8.in_r,n9.in)
2 add  (arg2,arg3,n3.in_l,n8.in_l)
3 sub  (n2.out_l,arg4,n10.in)
4 add  (arg5,arg6,n5.in_r,n6_in_l)
5 exp  (arg7,n4.out_l,ret0)
6 cmp  (n4.out_r,arg8,n7.in_l)
...
```

バックエンドは、中間コードに以下の処理を施し、最終的に、論理合成後そのまま WASMII システム上で実行可能な VHDL コードを出力する。なお、論理合成には商用の論理合成ツールを用い、FPGA 上へのマッピングを行なう。バックエンドでは以下の処理を行なわれる。

- 各ノードの接続関係の解析 (ループ解析含む)
  各ノードの入出力関係を解析し、ノードとポート名の対を signal として生成する。この signal は、VHDL 記述における信号線として用いられる。また、各アークのビット幅解析を行い、回路の最適化を行う。
- データフローグラフの分割
  ユーザが分割を行わない場合、グラフの自動分割を行う。
- 入力トークンレジスタの割り当て
  各トークンのルーティングを考慮し、退避レジスタのアドレスを割り当てる。アドレスの上位ビットが WASMII ユニットの番号をあらわす。
- ページ制御関数の作成
  入力トークンレジスタのアドレスから、ページ制御のための関数を作成する。
- 各ノードの VHDL 記述変換
  用意されたライブラリからノードに合った演算器をピックアップする。ここでユーザが定義したノードを用いることもできる。

なお、バックエンドの実装言語には、高度な文書処理関数を豊富に持つ perl5 を用いた。

3.2.1 データフローグラフ分割アルゴリズム It/t 法

全体のデータフローグラフを、FPGA 上に入り切る大きさの幾つかのサブグラフに分割する。このときに考慮しなければならない問題は実に多く、相互に複雑に影響し合っている。

1. デッドロック回避
   ページの相互依存によるデッドロックを回避する
2. 使用ゲート数
   1 ページになるべく多くのノードを詰めてページ数を抑えることにより、ページ切替え時のオーバヘッドを減少させる
3. 並列性
   グラフを横長に切ることにより、ノードの待ち時間を抑える
4. 時間効率
   入力から出力までの間の各ルートの演算にかかる時間の差を最小限に抑える
5. ページ入出力トークン数
   ページの切れ目のアーク数を減少させることにより、ページ間を移動するトークンの数を抑える

そこで、大前提であるデッドロック回避を基本としたグラフ分割アルゴリズム It/t 法 を提案する。

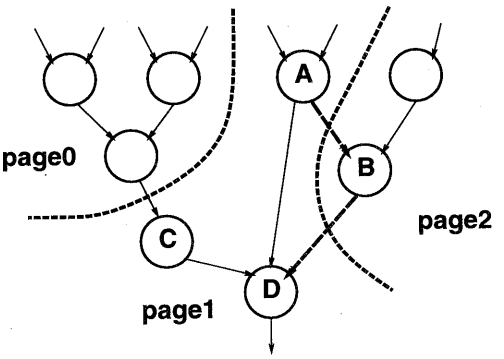

図 3: グラフ分割におけるデッドロック

デッドロックは、図 3の点線で表された太矢印のように、ページ間のトークンの入出力関係が相互に依存している場合に起きる。これを避けるためには、ページ分割時において、あるノードのすべての先行ノードがグルーピング済であればそのノードをグルーピング可能であるとすればよい。グルーピングとは、ページにそのノードを割り当てることを言い、グルーピングが可能なノードを Ready ノードと呼ぶ。例えば、図 3において、ノード D の入力はノード A,B,C からの出力であるため、これらのノードが全てグルーピングを終えられていなければノード D をグルーピングすることはできない。図 3

では、ノードBのグルーピングより前にノードDをグルーピングしてしまったことがデッドロックの直接の原因である。

It/t 法では上記のデッドロック回避アルゴリズムに従ってノードのグルーピングを行っていく。このとき、使用ゲート数はライブラリのノード情報から得ることとする。さらに、並列性を最大限に引き出すため、Readyノードから実際にピックアップするノードの検索は幅優先探索で行う。同時に、時間効率を引き出すため、ページの先頭からノードまでの合計実行時間を比較し、これが少ないノードから順番にピックアップしていくこととする。フローチャートを図4に示す。各関数の説明は以下の通りである。

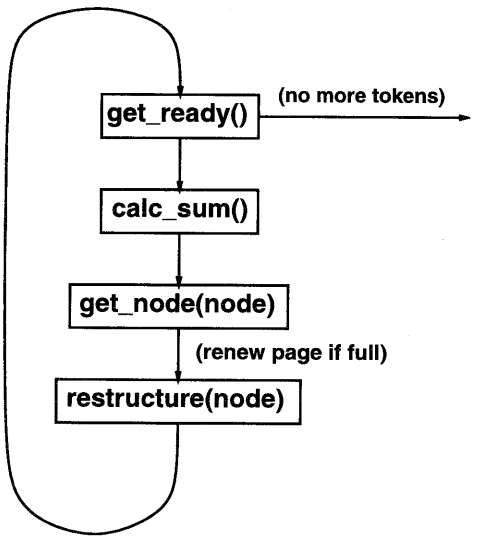

図 4: It/t 法 フローチャート

- get_ready()
  その時点でのグルーピング可能ノード (Ready node) を調べる。
- calc_sum()
  各 Ready node について、ページの先頭からそのノードまでを含めた最大実行時間を計算する。
- get_node(node)
  引数に get_ready(),calc_sum() の結果からピックアップされたノードを受け取り、現在グルーピング中のページの要員とする。
- restracture(node)
  引数にグルーピングしたノードを受け取り、Ready node のリストから外す。また、そのノードの出力先にあたるノードを Ready node 候補リストに入れる。

この It/t 法により、上記1〜4までの問題には効率よく対処できる。現在、5番目のページ入出力トークン数の問題についてはまだ対処しきれていないが、これを解決することにより、制御部の入力トークンレジスタに消費されるメモリ量も減少させることができるため、非常に有力である。この問題は従来の FPGA 上への回路分割問題とほぼ同一であるため、解決はそう難しくないと思われるが、It/t 法との組合せ方を考慮してゆかなければならない。また、マルチチップ WASMII に対応した自動分割の手法、同じ形の回路の抽出/再利用のためのアルゴリズムなども合わせて今後考えてゆく必要がある。

3.3 フロントエンド

エントリに用いる言語は C 言語のような高水準言語を用い、アルゴリズムを記述させた方がユーザには使用しやすい。しかし、システムの性質上、実行時の効率が見込まれる分野は、信号処理などの並列性が高く、繰り返し処理の多い分野であると考えられるため、このようなアプリケーションを記述しやすく、かつ中間コードを生成しやすい言語体系を考案したい。

また、フロントエンドでは、並列性の抽出処理の他に、バックエッジ (ループ部を含む) の検出を行わなければならない。WASMII システムの処理の流れは一方向であるため、バックエッジが出現すると、該当部分に関連するノードを特別に扱わなければアプリケーションを処理することができない。具体的には、ループ部が1ページに入り切らない場合は他のノードと同一のページには入れないことと、ループ部のページのアクティベイト条件を特別にページ制御部に書き込むことによって対処する。バックエッジの処理は、ページの分割における効率に大きく影響する。

4 可変構造システムテストベッド FLEMING

現時点で MPLD 機能を持つ FPGA は存在しないため、WASMII の実現性や有効性を明らかにすることは難しかった。そこで、WASMII を含む様々なアーキテクチャをエミュレートするためのプラットホームとして、可変構造システムベッド FLEMING(Fexible Logic EMulatIon eNGine) の実装を行った。

図5に示すように、FLEMING は FPGA とメモリから成る可変構造ユニット (RU) が6個と、内部に CPU

を持つインタフェースユニット (IU) が相互に結合された構成をとっている。

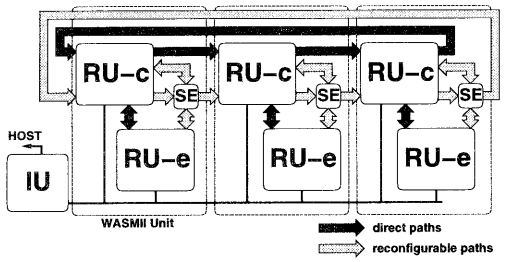

図 5: FLEMING の構成

4.1 可変構造ユニット (RU)

RU 内の FPGA には、15,000 ゲート相当の回路が実現可能な Xilinx 社の XC5215[12] が使用されている。当初、WASMII のエミュレーション用には Atmel 社の AT6000 シリーズ [14] などの動的な部分的書換えが可能な FPGA を使用する予定であったが [11]、当時では動的な部分的書換えを支援する CAD 環境の充実を待つ必要があると判断し、通常の FPGA の使用を決定した。

RU にはトークンルータ、入力トークンレジスタおよびページコントローラを実現するための RU-c と、実行されるデータフローグラフを実現する RU-e の二種類がある。これらの RU は同一構造であるが、接続の方法が異なり、RU-c, RU-e の一組により、WASMII チップ 1 個のエミュレーションを行なう。これは、WASMII システムを演算部と制御部とに分けて考えたとき、アプリケーションの実行中に制御部は決して書き換わることがないのに対し、演算部では実行に伴って次々とページを書き換えていくためである。すなわち、RU-e は RU-c の制御のもとに動的な構成の変更を行なう。これにより、FLEMING は全体として WASMII チップ 3 個から成る並列システムのエミュレーションが可能である。

RU 内には、結線情報用に 128Kbyte、FPGA 上に実現された回路が使用するデータメモリ用に 64Kbyte の SRAM がそれぞれ設けられている。これらのメモリは IU 内に設けられた制御用 CPU にも共有されており、このアービトレーションやコンフィギュレーションの管理を行う簡単な制御回路も RU 内に設けられている。RU のコンフィギュレーション用メモリのサイズは、FPGA の結線情報の 4 ページ分に相当し、これらを自由に選択してコンフィギュレーションすることが可能である。また、FPGA 上に実現された回路が、システムを止めることなく自己コンフィギュレーションすることによって、回路を動的に変更することも可能となっている。

4.2 インタフェースユニット (IU)

IU は制御用の 16bit CPU(モトローラ社 MC68000) と GPIB およびシリアルリンクで構成されており、システム全体の管理とホストワークステーションとの通信を行う。各 FPGA の結線情報や、実現される回路に必要な初期値データなどは、この IU によってホストから各 RU 内のメモリに転送される。また、FLEMING 上の全てのオペレーションは、IU 内のモニタプログラムによって管理されており、ユーザはホスト端末からモニタコマンドを用いて FLEMING の制御を行う。

4.3 結合方式

可変構造システムのテストベッドとして、様々な回路を自由に効率良くエミュレートするためには、FPGA 間の柔軟な相互結合が不可欠となる。このため、一般には多数のクロスバや FPID(Field Programmable Interconnect Device) などが用いられる。しかし、この方法は FPGA 間に多大なレイテンシを生じやすく、しばしばシステムパフォーマンスの低下を招く原因となる。そこで、FLEMING では、図 5 のように 2 種類の相互結合パスを設け、単純で高速な相互結合を実現している。

結合方法は、FLEMING の主用途である WASMII のエミュレーションを効率良く行なうことを狙っている。WASMII チップ 1 個は、RU-e と RU-c のペアでエミュレーションされるため、ペアとなる RU 間は密に結合される。トークンは RU-c により管理されるため、ペア間の接続は、それぞれの RU-c を循環構造により結合することで実現する。WASMII は基本的に静的なデータフローマシンであるので、単方向のトークンの流れに適合した循環構造は、最も自然な結合トポロジである。

さらに、FLEMNING は、ペア内、ペア間結合にそれぞれ、静的で変更不能な直結パス (Direct Path) と、スイッチングエレメント (SE) を介した可変パス (Reconfigurable Path) を設けている。信号線は基本的には直結パスが用いられるが、ペア内の接続、ペア間の接続の疎密はアプリケーションにより異なってくる。このため、可変パスを変えることにより、RU のペアを結合するパスと循環構造を提供するパスの転送容量のバランスを、実装するアプリケーションに合わせて任意に変更することが可能となる。可変パスは、アプリケーションが定まれば、変更の頻度は低いと考えられるため、SE には Lattice 社の EEPROM タイプの PLD である ispLSI2032[13] を用いた。このチップは、ボード上

に実装したまま、その回路内容を外部からプログラムすることができる。また、EEPROM 型のデバイスであるため、一度プログラムすると、次に再プログラムするまで、電源を供給し続けなくてもその内容は保持される。

FLEMING はユニバーサルボード上にワイヤラッピングによって実装され、システムクロック最大 10MHz で動作可能である。

図 6 に、FLMING ボードの外観を示す.

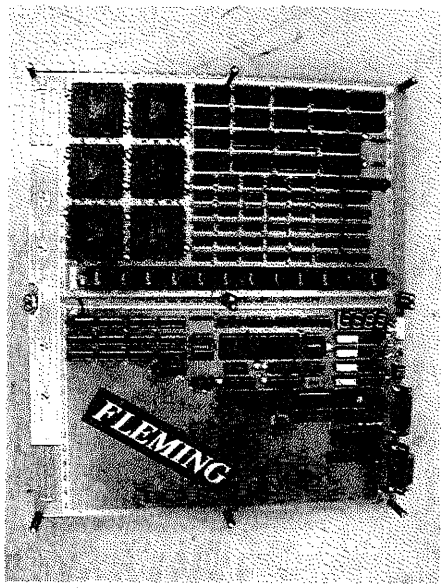

図 6: FLEMING ボードの外観

5 評価とまとめ

5.1 アプリケーション

フロントエンドが未実装であるため、加算器と乗算器のアレイが交互に並んだグラフの中間コードをテスト用のアプリケーションとして用いた。トークンのデータ幅は一様に 8 ビットである。総ノード数は、
(1 列のノード数) × (段数) = 7 × 14 = 98 とした。

5.2 結果

このアプリケーションを、実装した WASMII コンパイラバックエンドを用いて VHDL リストに変換し、WASMII エミュレータの各演算ページとして実行させた。VHDL 記述に変換されたページの一例を以下に示す。

```
--
-- Page_graph.vhd
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.wasmii.all;
use work.pack_p.all;

entity page2_u0 is
    port(
        arg0    : in    token;
        arg1    : in    token;
        arg2    : in    token;
        arg3    : in    token;
        arg4    : in    token;
        arg5    : in    token;
        arg6    : in    token;
        arg7    : in    token;
        clear   : in    std_logic;
        clock   : in    std_logic;
       ~
        n23out_r_n30in_l    : out    token;
        n24out_l_n30in_r    : out    token;
        n24out_r_n31in_l    : out    token;
        n25out_l_n31in_r    : out    token;
        n25out_r_n32in_l    : out    token;
        reset    : in    std_logic
    );
end page2_u0;

architecture rtl of page2_u0 is

    signal n0out_l_n7in_r   : token;
    signal n0out_r_n8in_l   : token;
    signal n10out_l_n16in_r : token;
    signal n10out_r_n17in_l : token;
    signal n11out_l_n17in_r : token;
   ~

begin
    u0: mult
        port map(clock,reset,clear,arg0,arg1,n0out_l_n7in_r,
         n0out_r_n8in_l);

    u1: add
        port map(clock,reset,clear,arg2,arg3,n1out_l_n8in_r,
         n1out_r_n9in_l);
   ~

    u27: adder
        port map(clock,reset,clear,n19out_r_n27in_l,
         n20out_l_n27in_r,n27out_l_n33in_r,n27out_r_n34in_l);

    u28: mult
        port map(clock,reset,clear,n21out_r_n28in_l,
         n22out_l_n28in_r,n28out_l_n35in_r,n28out_r_n36in_l);

end rtl;
```

グラフ分割はコンパイラによる自動分割を行なった結果、ページ数は 4 となった。以上の処理によって得ら

れたVHDLリストを、商用の論理合成ツールを用いてFPGA上に論理合成・配置配線を行なった結果を表1に示す。

表 1: 論理合成及び配置配線の結果

演算頁	CLB 占有率	FF 使用率	動作周波数
ページ 0	96%	37%	10.4MHz
ページ 1	98%	38%	8.5MHz
ページ 2	97%	40%	11.5MHz
ページ 3	77%	21%	10.9MHz

表1より、グラフの自動分割の結果、ゲート数については効率的な分割ができたと考えられる。

次に、実際にシステムを稼働させたときの項目別の処理時間とその比率を表2に示す。なお測定は、FLEMINGのシステムクロックを8MHzにして行った。

表 2: テストプログラムの実行結果

処理内容	処理時間 (μsec)	割合 (%)
演算 (Page0)	3.8	0.005
演算 (Page1)	3.6	0.005
演算 (Page2)	4.1	0.006
演算 (Page3)	3.4	0.005
トークン転送	8.4	0.012
ページ切替え	7.2×10^4	99.967

表2からも明らかなように、XC5215ではコンフィギュレーションに約$18ms$もかかるため、処理時間のほぼすべてをFPGAの書き換えに費やしている。しかし、MPLDをWASMIIに使用する本来の方式ではページ切替えが数十nsで実現できるため、コンフィギュレーションのボトルネックを効果的に解消できると考えられる。

5.3 まとめ

新たなWASMIIコンパイラのバックエンドを実装し、中間コードからVHDLリストを生成した。ノード数約100のグラフを処理するのに要した時間は6秒程度であった。また、It/t法を用いることにより、デッドロックを起こさずに効率的にグラフ分割を行うことができた。

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] R.Hartenstein, J.Becker and R.Kress: "Custum Computing Machines vs. Hardware/Software Co-Design: From a Clobalized Point of View" Proc. of FPL'96, (LNCS 1142), pp.65–76(1996).

[3] X.Ling and H.Amano: "WASMII: a Data Driven Computer on a Virtual Hardware" Proc. of FCCM, pp.33–42(1993).

[4] X.Ling and H.Amano: "WASMII: An MPLD with Data-Driven Control on a Virtual Hardware" Journal of Supercomputing, Vol.9, No.3, pp.253–276(1995).

[5] 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" 第5回 FPGA/PLD Design Conference & Exhibit 予稿集, pp.233–239(1997)

[6] 日暮 浩一, 宮崎 英倫, 柴田 祐一郎, 天野 英晴: "仮想ハードウェアWASMIIのためのデータフローコンパイラの研究" 電子情報通信学会技術研究報告, vol.97, No.27, pp.65–72(1997).

[7] H.Miyazaki, Y.Shibata, A.Takayama, X.Ling and H.Amano: "Emulation of Multichip WASMII on Reconfigurable System Testbed FLEMING" PACT'98 Workshop on Reconfigurable Computing, pp.47–52(1998)

[8] 吉見 昌久 (富士通株式会社): "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).

[9] 島田 俊夫: "細粒度ダイナミックデータフロー計算機の研究" 電子技術総合研究所報告 第946号,ISSN 0366-9106(1992).

[10] 島田 俊夫, 関口 智嗣, 平木 敬: "データフロー言語DFCの設計と実現" 電子情報通信学会,J71-D,No.3,pp.501–508(1988).

[11] Y. Shibata, X.-P. Ling, H. Amano: "An Emulation System of the WASMII: Data Driven Computer on a Virtual Hardware" Proceedings of FPL '96 (LNCS 1142),pp.55–64(1996).

[12] Xilinx Corp.: "The Programmable Logic Data Book" 1997.

[13] Lattice Semiconductor Corp.: "Lattice Data Book" 1997.

[14] Atmel Corp.: "Configurable Logic Design and Application Book" 1995.