論　文

PEG マシンの FPGA 実装について

マイ　マイクオン† 本多峻† 倉光君郎†

Mai MAICUONG†, Honda SHUN†, and Kuramitsu KIMIO†

あらまし　解析表現文法 (PEG) は、2004 年に Ford によって提案された形式文法であり、正規表現や文脈自由文法の代替として人気が高まっている。本稿では、より高い性能要求を目指すため、PEG の FPGA 実装、特に PEG 演算子の仮想マシン化によるバーチャルマシン方式について報告し、性能に関する初期レポートを行う予定である。

キーワード

1. まえがき

構文解析とは、定義された文法に従ってテキストの構造を解析することである。この技術はプログラミング言語処理系のみならず、現在 Web Page(HTML や XML) の読み込みや Twitter つぶやきの解析、通信パケットの不正検出など、あらゆる場面で活用されている。現在構文解析を実現するために、正規表現や文脈自由文法という形式文法が広く用いられている。

一方、2004 年に解析表現文法 (Parsing Expression Grammar) は Ford によって提案され、正規表現や文脈自由文法の代替として人気が高まっている (PEG 論文)。PEG は Packrat Parsing（PP 論文）により線形時間に解析することができる。しかし、Packrat は大きな入力に対して莫大なメモリ容量を使用するため、大きなデータの分析に向いていない。そこで、大きな入力を受理するため、Medeiros が PEG のための Virtual Parsing Machine を提案した（VM 論文）。PEG ファイルを一つのプログラムに変換して、Virtual　Machine で実行される。

一方、近年、IoT(Intenet of things) の発展につれ、様々な機器への組み込みやすさやより高度なパケット処理が必要になった。また、ビックデータ時代になった今は、より高速な構文解析技法が求められている。そのため、組み込みやすさと高速な処理の面から、FPGA が注目されている。構文解析分野でも、FPGA を用いる研究がいくつかある。例えば、文脈自由文法を使って FPGA 上の構文解析 [**?**] や FPGA 上で正規表現を用いたパターンマッチングマシン [**?**] などの研究がある。

正規表現論文 () では、正規表現のマッチングマシンを作るには、3 ステップが必要である。まず正規表現を木構造に変換し、次に非決定オートマトンに変換する。非決定オートマトンに変換する際、HDL に変換しやすいように、修正した McNaughton 構造を提案した。最後に非決定オートマトンから HDL に変換する。修正した McNaughton 構造を採用することより、コンパクトな構造が実現できた。一方、文脈自由文法論文（）では、文脈自由文法を解析するために、Cocke-Younger-Kasami アルゴリズムを用いた。

本稿では、より高い性能要求を目指すため、PEG の FPGA 実装、特に PEG 演算子の仮想マシン化によるバーチャルマシン方式について報告し、性能に関する初期レポートを行う予定である。

本稿の構成は次の通りである。第 2 節、第 3 節では、PEG 及び PEG バーチャルマシンについて述べる。第 4 節及び第 5 節は FPGA の設計と実装、性能評価となる。また、第 6 節は結論と今後の課題を述べる。

†

2. 解析表現文法

プログラミング言語やプロトコルのシンタクスを表すには、文脈自由文法 (Context-free grammars) や正規表現 (Regular Expressions) が広く使われている。

＋ＰＥＧは `A <= e` というルールの集合である。解析表現は表 2.1 にある値と演算子を組み合わせた式である。

＋例：
＋正規表現と文脈自由文法と比べる

3. 仮想マシン

本研究で用いるバーチャルマシンは Medeiros が提案したバーチャルマシンをベースにする。Medeios が提案したマシンは、次に実行する命令のアドレスを保存するプログラムカウンタ、文字列の現在位置を保存するレジスタ、そして return address と backtrack entry を持っているスタックがある。Return address はプリグラムカウンタのための値である一方、backtrack entry はアドレスと文字列のポジションの両方を持っている。

基本的な命令は以下となる。
Char x: 文字列の現在位置の文字と文字 x をマッチさせ、成功すれば、文字列の位置を 1 つ増やす
Any: 文字列の末端に到着しなければ、文字列の位置を一つ増やす。文字列の末端に到着すれば、失敗となる。
Choice l : backtrack entry をスタックにプッシュする。l は別の選択肢との相対位置である。
Jump l : 相対位置の l の命令にジャンプする
Call l: 直後の命令アドレスをスタックにプッシュし、相対位置の命令にジャンプする
Return : スタックから一つの命令アドレスをポップし、その命令にジャンプする
Commit l : スタックから一つ backtrack entry を消し、そして相対位置の l の命令にジャンプする
Fail: スタックから backtrack entry が出るまでポップし、その backtrack entry が新しい状態になる

本研究で採用したバーチャルマシンは上記のバーチャルマシンを拡張したものである。扱いやすいために、Return スタックと Fail スタックに分ける。Return スタックは、Non-Terminal を呼び出すとき、プログラムレジスタ PR の値を退避するためである。Fail スタックはバックトラックの戻り値を保存するためである。

Medeios の VM に以下の命令を追加した。
Set [x-y] : 文字列の現在位置の文字を ASCII の文字での x 以上 y 以下にマッチさせ、成功すれば、文字列の位置を一つ増やす。
OChar x, OSet[x-y] :文字リテラル、文字クラスのオプション（以下で述べる）
Alt addr: 指定されたアドレスを Fail stack にプッシュする
RChar x, RSet[x-y] : 文字リテラル、文字クラスの 0 個以上
First x addr: 最初の文字が x の場合、addr にジャンプする
NChar x, NSet[x-y], NAny: 先読み
Pos : 文字列の現在位置を専用レジスタに保存する
Back: 保存された位置を文字列の現在位置にする

3.1 オプション

命令数を減らすために、以下の命令を定義する。
OChar x: PEG の'x'?に対応する。文字列の現在位置の文字と文字 x をマッチさせ、成功すれば、文字列の位置を 1 つ増やす。失敗した場合、次の命令に進む。
OSet [x-y]: PEG の [x-y]?に対応する。Set と同様であるが、マッチが失敗した場合、次の命令に進む。

本来であれば、例えば'PEG の x'?は以下の命令列が生成される。
L1 Alt 3
L2 Char x
L3 Commit 1

OChar x を使うことによって命令数を減らすことができた。

3.2 0 個以上と先読み

同様に、以下の命令を定義する。
RChar x : PEG の'x'* に対応する。文字列の現在位置の文字と文字 x をマッチさせ、成功すれば、文字列の位置を 1 つ増やし、もう一回実行する。マッチされなくなれば、次の命令に進む。
RSet [x-y] : PEG の [x-y]*に対応する。文字列の現在位置の文字を ASCII の文字での x 以上 y 以下にマッ

チさせ、成功すれば、文字列の位置を一つ増やし、もう一回実行する。マッチされなくなれば、次の命令に進む。
NChar x: PEG の!'x' に対応する。Char x の逆であり、マッチされれば、失敗となる。マッチされなければ、成功となり、次の命令に進む。ただし、文字列の位置を変更しない。
NSet [x-y] , NAny: それぞれ PEG の![x-y] に対応する。NChar x と同様である。

4. 設　　計

4.1 プロセッサアーキテクチャ

4.1.1 ハードウェア仕様

NezProcessor のワードは 16 ビットである。メモリは FPGA に搭載しているブロックメモリで実装する。メモリアドレスを保持してプログラムの実行を制御するのはプログラムレジスタ (PR) である。PR は、次に実行すべき機械語が格納されたメモリアドレスを指定する。プログムの実行に従って順次にインクリメントされ、ただし、分岐回路が実行された場合は、分岐先のアドレスが PR に書き込まれる。NezProcessor では、Return スタックと Fail スタックがある。それぞれのスタックにスタックポインタがある。

4.1.2 実　行　例

4.1.3 命令の形式

第 15 ビットから第 11 ビットまではオペレーションフィールド（Op フィールド）であり、各命令に対応したコードが割り付けられる。第 10 ビットから第 0 ビットまでは命令の対象データとなる。命令によってこのデータの意味が異なる。Byte、Obyte、Rbyte の場合、文字である。Set、Oset、Rset の場合、Set テーブルのインデックスとなる。Set テーブルは 256 行 n 列の行列であり、8 ビットの文字を受け付けて、1 ビットを出力する。Jump, Call, Alt の場合、命令アドレスを意味する。

4.2 制　御　部

4.2.1 状態図と制御信号

NezProcessor の動作は、メモリからの命令の取り込み、解読、演算・データ転送といった一連の処理の繰り返しである。制御部の役割は、それを実現するための制御信号を適時生成して、演算回路やデータ転送回路に伝えることである。制御部はそのための制御信号生成回路と、命令を解読するためのデコーダから構成される。
命令フェッチは命令が格納されているメモリアドレスの設定、メモリデータレジスタへの読み込み、命令レジスタへの転送の 3 つの動作で構成される。命令デコードでは、命令レジスタに取り込まれた命令のビットパターンから、どのような演算・データ転送を行うかを判断する。各状態はそれぞれ 1 クロックサイクルで実行される。

F1: 命令、文字データ読み込み
Dec: 命令デコード
Ex：演算・データ転送の実行

4.2.2 命令フェッチ

NezProcessor では、FPGA に搭載した BlockRAM をメインメモリとして、使う。FPGA では、メモリを作るには、FFRAM、LUTRAM 及び BlockRAM という 3 つの方法がある。大容量のメモリが必要な場合、BlockRAM はよく使われている。論理合成する際、メモリへの書き込み・書き出しが同期・非同期かによって、どのメモリを使うか、合成ツールが推論する。BlockRAM に推論されるために、書き込み・読み出しが同期でなければならない。前の命令での Ex 状態で、次の命令アドレスの設定を行う。そして、F1 状態でメモリにアクセスし、IR に転送する。Dec 状態では、IR のデータをデコードする。この処理が繰り返される。

4.2.3 制御信号生成回路の構成

NezProcessor の制御信号生成回路は、配線論理制御方式を採用している。配線論理制御方式は、目的の状態遷移をもつ順序回路によって制御を行う。図 () に制御信号生成回路を示す。状態 F1,Dec、Ex に対して、3 つのフリップフロップが直列に接続されている。プロセッサが起動するとき、Reset 信号を一時的にハイレベルにする。スタート回路からハイレベルのフェッチ起動信号が出力される。Reset 信号をローレベルに戻すと、次のクロックの立ち上がりで、状態 F1 に対するフリップフロップにフェッチ起動信号のハイレベル値が取り込まれる。同時にフェッチ起動信号はローレベルへ変換する。そのクロックの間、メモリへの read 信号がハイレベルにする。その次のクロックの立ち上がりで、状態 Dec に対するフリップフロップにハイレベルが取り込まれ、状態 F1 に対するフリップフロップの出力値はローレベルになる。そのクロッ

クの間、IR が持っている命令データがデコードされ、PRlat, Sinc がハイレベルにする。同様にして、その次のクロックサイクルでは、命令実行のための信号がハイレベルにして、命令を実行する。そして、次のクロックから新たな命令フェッチを実行する。

4.2.4 基本命令

Char 命令の実行ステップは、デコードを行うクロックサイクル Dec と、文字比較を行うクロックサイクル Ex からなる。各クロックサイクルで生成された制御信号を以下に示す（図）。
Set 命令は Set テーブルを使う。Set テーブルは 256 行 q 列の配列である。Set 命令の実行では、文字 x とインデックス i を受け取り、Set テーブルを参照し、文字を 10 進数に変換した値を n とすると、n 行目 i 列目の値を出力値となる。その値は'1' であれば、マッチ成功、'0' であればマッチ失敗となる。

オプション命令（OChar, Oset) の実行は同様であるが、違いは文字消費信号を持っているところである。オプション命令はマッチ成功した場合、文字を消費し、マッチ失敗した場合、文字を消費しないが、Fail にならず、次の命令に進む。先読み命令（NChar, NSet) の実行も Char,Set 命令の実行と類似するが、文字を消費しない。

0 個以上命令（RChar, RSet) の実行ステップには、デコードを行うクロックサイクル Dec と、文字比較を行うクロックサイクル Ex があり、結果により、次の処理が分けられる。マッチ成功の場合、次の文字へのアクセスし、命令がもう一度実行される。マッチ失敗の場合、次の命令フェッチが始まる。

4.2.5 分岐命令

分岐命令には、Jump 命令がある。Jump 命令の実行は、デコードを行うクロックサイクル Dec と、次に実行すべき命令のアドレスをプログラムレジスタ (PR) に転送するクロックサイクルからなる。各クロックサイクルで生成された制御信号を以下に示す。

4.2.6 スタック操作命令

スタック操作命令には、Call、Alt、Return、Succ がある。Alt 命令は Fail スタックにデータをプッシュダウンし、Succ 命令は Fail スタックからデータをポップアップする。

Alt 命令の実行ステップは、デコードを行うクロックサイクル Dec と、メモリスタックにデータをプッシュダウンするクロックサイクル Ex からなる。スタックポインタのインクリメント及びスタックメモリアドレスの設定は、前回のスタックプッシュが終わり次第、すでに実行された。このため、Call 命令は 3 つのクロックサイクルで実行が可能になる。クロックサイクル Ex が終ったら、新たな命令フェッチが実行される同時に、スタックポインタのインクリメントとスタックメモリアドレスの設定も実行される。各クロックサイクルで生成された制御信号を以下に示す。Succ 命令も同様である。

Non-Terminal を呼び出す命令が Call 命令であり、それを呼び出したプログラムへ実行制御を返すのが Return 命令である。Call 命令は、プログラムレジスタ PR の値を Return スタックにプッシュダウンして退避させ、Non-Terminal の先頭番地であるアドレスを PR に転送する。その実行ステップはデコードを行うクロックサイクル Dec と、メモリスタックにデータをプッシュダウンするクロックサイクル Ex、及び PR にポップアップされた命令アドレスへの転送を行うクロックサイクルからなる。同様に、スタックポインタのインクリメント及びスタックメモリアドレスの設定は既に実行済みである。各クロックサイクルで生成される制御信号は以下に示す（図）。

一方、Return 命令は、Call 命令によってスタックに退避した Non-Terminal からの戻り番地をポップアップして PR へ転送する。これによって、Non-Terminal を呼び出したプログラムへ実行制御が返される。Return 命令は Succ 命令と類似しているが、両者の違いは、Succ 命令は Fail スタックからポップアップした値を捨てるのに対して、Return 命令はポップアップした値を PR へ転送する点である。

4.2.7 割り込み

PEG では、バックトラックがある。バックトラックは、Choice がある場合、ある選択肢でマッチが失敗した場合、前の状態に戻り、別の選択肢を評価する仕組みである。Choice がある場合、選択肢を評価する前に、バクトラックが起こるときの戻り先を Fail スタックにプッシュダウンされる。どこかでマッチが失敗したら、バックトラック処理が実行される。
バックトラック処理では、Fail スタックからポップアップするクロックサイクル、ポップアップされた値を PR へ転送するクロックサイクルからなる。バックトラック処理で生成される制御信号は以下に示す

（図）。

5. 性 能 評 価

6. ま と め

謝辞

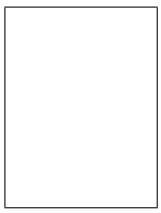
文 献

[1]

付 録

1.

（平成 xx 年 xx 月 xx 日受付）



Abstract

Key words