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

近年、クラウドなどのデータセンターで使うコンピューティングデバイスとして性能向上や電力削減の期待から、FPGA が注目されている。例えば、Microsoft 社が Web エンジン「Bing] の処理を高速化するために、自社のデータセンター FPGA を導入すると発表した。また、中国のネット検索サービス大手の Baidu 社も画像検索サービスの実装に FPGA の導入を検討している。Intel 社でもサーバー CPU「Xeon] のパッケージに FPGA を収める製品を投入する予定である。

データセンターでは、ウイルス対策などのセキュリティ対策が不可欠である。その対策の一つとして、侵入検知システム（IDS : Instruction Detection System) がある。IDS では、ネットワーク型とホスト型がある。ネットワーク型は、通常の振る舞いとの比較により、不正アクセスを検知するため、処理が重い。一方、ホスト型は既存の不正アクセスパターンを記憶し、パターンマッチングにより、不正アクセスを検知する。そのため、ホスト型の処理は比較的に軽く、現在侵入検知システムとして普及している。

現在ホスト型は、既存の不正アクセスパターンを正規表現で記述することが多い。しかし、パターンの複雑度に従い、パターンマッチング回路が大きくなることは問題である。

本研究では、パターンに依存せず、コンパクトなマッチングマシンを実現することを目的としている。そのため、解析表現文法（Parsing Expression Grammar）を用いる。PEG 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。

2. 解析表現文法

ＰＥＧは `A <= e` というルールの集合である。解析表現は図 1 にある値と演算子を組み合わせた式である。

解析表現	意味	解析表現	意味
'hoge'	文字列リテラル	e*	0回以上の繰り返し
[a-zA-Z0-9]	文字クラス	e+	1回以上の繰り返し
.	任意の文字	&e	肯定先読み
A	非終端記号	!e	否定先読み
(e)	グルーピング	e_1 e_2	シーケンス
e?	オプション	e_1 / e_2	優先度付き選択

図 1　PEG

3. 仮想マシン

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 氏が PEG のための Virtual Parsing Machine を提案した。本研究で用いるバーチャルマシンは Medeiros 氏が提案したバー

†

チャルマシンをベースにする。本研究で用いる VM は図 2 となる。

種類	命令名	意味	PEG例
基本命令	Byte	文字リテラル	‘a’
	Set	文字クラス	[1-9]
	Any	任意の文字	.
特化命令	Obyte/ Oset	オプション	‘a’?
	Rbyte/ Rset	0個以上	‘a’*
	Nbyte/ Nset /Nany	否定先読み	!‘a’
制御用命令	Call	呼び出し	
	Alt	Fail stackにpush	
	Fail	強制的にfail信号をハイレベルにする	
	Succ	Fail stackからpop	
	Ret	呼び出し先に戻る	
	Jump	指定された命令にジャンプ	

図 2　命令セット

本研究で用いる命令セットは Medeios 氏の提案した命令セットに特化命令を追加した。特化命令とは、PEG の演算子を実行するための特化した命令である。特化命令では、オプション命令（Obyte, Oset)、0 個以上命令 (Rbyte, Rset）及び先読み命令 (Nbyte, Nset, Nany) がある。こちらの命令は、実行する命令数を削減し、またこれらの命令に特化した回路によって、実行効率を上げるためである。

4. 設　　計

4.1 全　体　図

全体のシステムは図（）となる。ホストとの通信は Ubuntu OS と FPGA が可能にした、Xillybus 社が提供している Xillybus IP コアを用いる。まずホストから命令列のバイトコードを受け取り、メモリに一時的に保存する。NezProcessor のメインメモリは FPGA に搭載しているブロックメモリで実装する。次に文字列をホストから受け取り、解析を行い、結果をホストを返す。

Virtual Machine の全体図は図（）となる。まずは書き換え機能つきプログラムカウンタ（PC) がある。PC は次に実行すべき命令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、分岐命令や割り込み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。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。スタックポインタは、インクリメンタとディクリメンタを持っており、信号によってインクリメントやディクリメントが適時

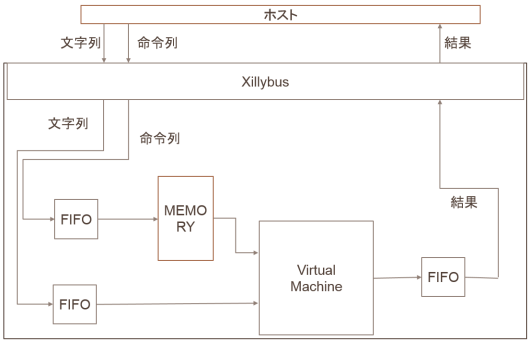


図 3　全体システム

実行される。

命令を解読するデコーダやそれぞれの命令に特化した命令用回路がある。また、メモリから読み込んだ命令データ、FIFO から受け取った文字データはそれぞれ命令データレジスタ（IR)、文字データレジスタ（TR）に一時的に保存される。

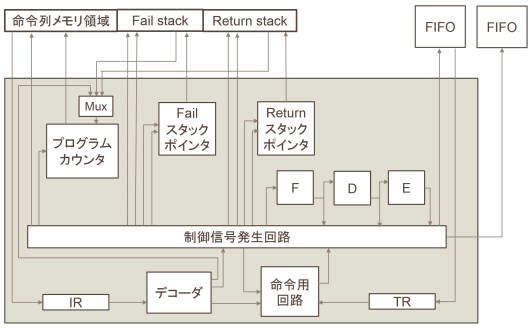


図 4　Virtual Machine の全体図

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。

NezProcessor では、命令フェッチ、文字データ読み込み（F)、命令デコード（D) 及び演算・データ転送を行う命令実行（E) という 3 つの状態がある。一般的に、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。しかし、本実装では、FPGA に搭載している BlockRAM をメモリとして、使用するため、回路とメモリの距離が短く、命令フェッチは 1 クロックサイクルで実行

できる。実際に、メモリアドレスは事前に設定しておき、読み出し信号がある場合、データをメモリデータレジスタを通さず、直接命令レジスタに転送される。D 状態も 1 クロックサイクルで実行される。E 状態は基本的に 1 クロックサイクルで実行されるが、Rbyte、Rset や分岐命令の場合は命令用のフリップフロップがある。具体敵に（）節に説明する。

どこ状態にあるかは制御信号生成回路によって制御される。状態 F,D、E 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 F 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 D 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 F 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、IDR が持っている命令データがデコードされる。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。

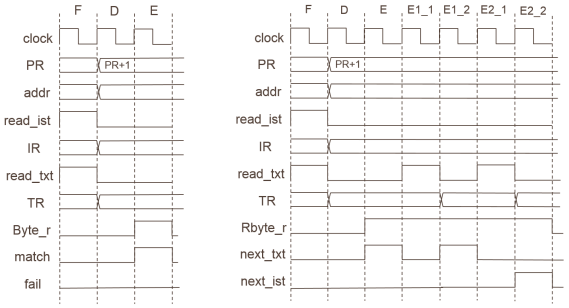
4.2 各命令の実行

4.2.1 基 本 命 令

Byte 命令実行のタイムチャートは図（）に示す。F 状態では、クロックの立ち上がりで read-ist 信号がハイレベルになり、命令が格納されるメモリにアクセスし、データを命令レジスタ（IR）に転送される。アクセスアドレスはプログラムレジスタ（PR) から転送されたアドレスである。同時に read-text 信号もハイレベルになり、FIFO から 1 文字を読み出し、文字レジスタ（TR）に転送される。

次のクロックの立ち上がりで、IR と CR のデータが確立され、このクロックサイクルで IR のデータがデコードされ、どの命令を実行するかが決まる。今回は Byte 命令用回路が実行されることになる。同クロックサイクルで、PR はインクリメントされ、メモリアクセス用のレジスタである addr にデータが転送される。

次のクロックサイクルの立ち上がりで、Byte 命令用回路のトリガーである Byte-r がハイレベルになり、Byte 命令用回路が実行される。IR が持っている文字データと CR の文字データが一致するならば、match 信号がハイレベルになり、一致しなければ、fail 信号がハイレベルになる。match 信号及び fail 信号は、制御信号生成回路の入力であり、match 信号がハイレベルであれば、次のクロックから新たな命令フェッチを実行するように制御信号が生成される。一方、fail 信号がハイレベルの場合、Fail 処理を実行する制御信号が生成される。



(a) Byte　　(b) Rbyte

図 5　基本命令実行のタイムチャート

4.2.2 特 化 命 令

Rbyte 命令実行では、F 状態、D 状態は Byte 命令と同様である。Ex 状態では、IR が持っている文字データと TR の文字データが一致した場合、next-txt 信号がハイレベルとなる。この場合、次のクロックサイクルの立ち上がりで read-txt 信号がハイレベルとなり、FIFO から 1 文字を読み出す。次のクロックサイクルで、IR の持っている文字データと新たな TR の文字データを比較する。一致すれば、また FIFO から新たな文字を読み込まれる。IR の文字データと TR の文字データが一致しなくなるまで、この処理が繰り返される。この時、next-ist 信号がハイレベルとなり、次のクロックから新たな命令フェッチを実行する。

Byte 命令は、IR を持っている 1 文字のデータと TR の文字データを比較するのに対して、Set 命令は TR の文字が複数の文字の中のどれかと一致するかを評価する命令である。Set 命令を実行するために、Set テーブルを使う。Set テーブルは 256 ビットのデータの配列である。ASCII 表の n 番目の文字と一致したらマッチするという場合、n ビット目を'1' にする。このようにして、与えられた文字を Set テーブルに照らし合わせて、そのビットの値は'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（Obyte, Oset）も同様に実行されるが、文字消費信号を持っているところが違う。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（NByte, NSet）の実行も Byte,Set 命令の実行と類似するが、文字を消費しない。

4.2.3 分岐命令

分岐命令には、Jump 命令がある。Jump 命令実行のタイムチャートは図（）に示す。

E 状態では、デコードの結果、Jump 命令の実行のトリガーである Jump-r がハイレベルになり、プログラムレジスタのトリガーである PR-lat 信号もハイレベルになる。この場合、インクリメントのトリガーである PR-inc はローレベルであるため、ジャンプ先のアドレスを持っている PC-data-in の値は PR に置き換える。次のクロックサイクルで PR の値が少し遅れてメモリアドレスレジスタ addr に転送される。また、次のクロックサイクルの立ち上がりで新たな命令フェッチが始まる。

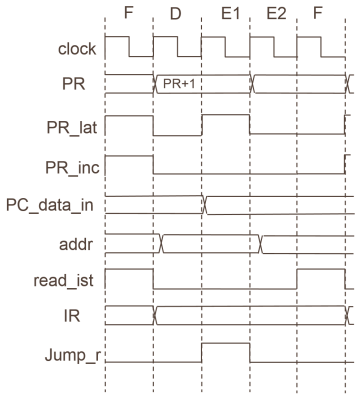


図 6 Jump 命令実行のタイムチャート

4.2.4 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。

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。Call 命令の実行は Jump 命令と類似しており、ただ新たなアドレスを PR に転送している同時に、Return スタックにプッシュダウンを行う。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 命令実行のタイムチャートは図（）に示す。

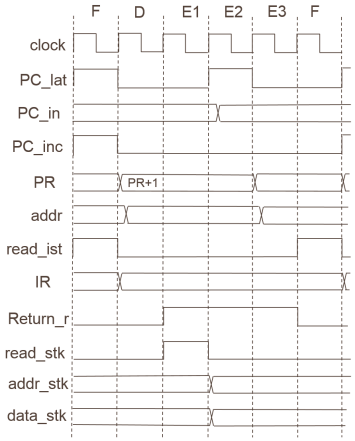


図 7 Return 命令実行のタイムチャート

E1 状態のクロックサイクルの立ち上がりで、Return 命令実行のトリガーである Return-r 信号がハイレベルになる。同クロックサイクルで Return スタックの読み出し信号である read-stk がハイレベルになり、データを data-stk に転送される。次のクロックサイクルで PR の書き換えデータ PC-data-in に少し遅れて転送される。次のクロックサイクルの立ち上がりで PR の新たなアドレスが確立し、少し遅れて命令のアドレスである addr に転送される。次に新たな命令フェッチが始まる。

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。

Fail スタックを操作する命令は Alt 命令と Succ 命令がある。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。AL t 命令と Succ 命令の動作は Call 命令、Return 命令と類似しているが、両者の違いは AL t 命令と Succ 命令がスタックを操作するだけで、PR にデータを転送しない点である。

一方、どこかでマッチが失敗した場合、バックトラック処理が実行される。バックスタック処理は Return 命令の実行と類似している。

5. 性能評価

本研究では、Xilinx 社の Zynq xc7z010-1clg400c を搭載した Zynq-7000 評価ボードで実装した。また、VHDL を使った RLT 設計により、実装し、論理合成などは Vivado Suite Design ツールを用いる。

現時点では、四則演算を表すなどの簡単な PEG に対して、正しく動作することが確認できた。バーチャルマシン本体（ホストとのインターフェースを除く）のリソース使用量は以下となる。

リソース	使用量	使用率（%）
LUT	323	1.84
FF	196	0.56
ロジックセル	565	2.02

上記に記載したリソースは PEG ファイル及び文字列の複雑度に依存しない。ただし、ブロック RAM の使用量は PEG ファイル及び文字列の複雑度に依存する。

ブロック RAM は、命令列領域、スタック領域、Set テーブルで使われている。命令列領域及び Set テーブルに用いられるメモリ量は PEG ファイルに依存する。例えば、四則演算を表す PEG のバーチャルマシンの場合、命令列領域に 34x16bit、Set テーブルに 216x3bit、合計で 1192bit が使われている。

一方、スタックに使われるメモリ量は PEG 及び文字列の長さや構造に依存する。どのぐらいのメモリを確保すればよいかは今後の課題となる。

6. まとめ

本稿では、解析表現文法（PEG）に特化したバーチャルマシンについて述べた。必要最小限の回路しか載せないことで、また PEG 演算子に特化した回路により、コンパクトかつ効率がよいバーチャルマシンが実現できた。複数のバーチャルマシンの co-processor を載せ、並列で動作させることによって、高いスループットのマッチングマシンが期待できる。

現時点では、四則演算を表すなどの簡単な PEG ファイルに対して正しく動作できた。今後の課題はスタックに使われるメモリの見積である。また、より複雑なデータ構造を処理できるように回路を拡張する。

謝辞

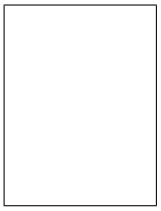
文　献

[1]

付　録

1.

（平成 xx 年 xx 月 xx 日受付）



Abstract

Key words