社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

多文字遷移を行うNFAに基づく正規表現マッチング回路について

中原　啓貴† 　笹尾　勤† 　松浦　宗寛†

† 九州工業大学 情報工学部 〒820–8502 福岡県飯塚市大字川津 680–4

あらまし　本論文では, NFA (Non-deterministic finite automaton) に基づく正規表現回路の実現法について述べる. 正規表現の長さと個数から, NFA の面積複雑度と時間複雑度を求め, NFA に基づく回路が DFA に基づく回路よりも優れていることを示す. 提案手法は以下の手順で正規表現回路を生成する. まず, 与えられた正規表現を NFA に変換する. 次に, NFA の状態数を削減するために, p 文字遷移する MNFA(p) (Modular NFA with p-character-consuming transition) に変換する. 最後に, p 文字を検出する FIMM と, MNFA(p) の状態を模擬するマッチングエレメント (ME) を生成する. オープンソースの侵入検知ソフトウェアである SNORT の正規表現の一部を Xilinx 社 FPGA に実装し, 効率よく実現する MNFA(p) を実験的に求めた. 面積当りの性能で比較した結果, 提案手法は DFA に基づく手法よりも 6.2-18.6 倍優れており, 通常の NFA に基づく手法よりも 1.8 倍優れていることがわかった. 提案手法では FPGA のリソース (LUT と組込みメモリ) の使用効率が良いため, 安価な FPGA で高性能なシステムが実現可能である.

A Regular Expression Matching Circuit Based on an NFA with Multi-Character Consuming

Hiroki NAKAHARA†, Tsutomu SASAO†, and Munehiro MATSUURA†

† Department of Computer Science and Electronics, Kyushu Institute of Technology
680–4, Kawazu, Iizuka, Fukuoka, 820–8502 Japan

Abstract　This paper shows an implementation of a regular expression circuit based on an NFA (Non-deterministic finite automaton). Also, it shows that the NFA based one is superior to the DFA (Deterministic finite automaton) based one, with respect to the area complexity and the time complexity. A regular expression matching circuit is produced as follows: First, the given regular expressions are converted into a non-deterministic finite automaton (NFA). Then, to reduce the number of states, the NFA is converted to a modular non-deterministic finite automaton (MNFA(p)) with p-character-consuming transition. Finally, a finite-input memory machine (FIMM) to detect p-characters is generated, and the matching elements (MEs) realizing the states for the MNFA(p) are generated. We designed MNFA(p) for different p on Xilinx FPGA. As for the performance per area, our method is 6.2-18.6 times better than the DFA-based methods, and is 1.8 times better than the NFA-based method. Then, we derive an optimal value p that efficiently uses both LUTs and embedded memories of the FPGA.

1. はじめに

1.1 侵入検知システム

侵入検知システム (IDS: Intrusion Detection System) とは, ネットワークを監視し, 不正アクセスを検知して管理者に通報するシステムである. IDS はネットワーク型とホスト型に大別される. ネットワーク型 IDS は通常のふるまいとの比較を行うことで不正アクセスを検知する. 一方, ホスト型 IDS は既知の不正アクセスパターンを記録しておき, ネットワークに流れるデータとの**パターンマッチング**を行うことで不正アクセスを検知する. ネットワーク型 IDS は未知の不正アクセスを検知できるが, データの統計処理を行うため, 処理が重い. 一方, ホスト型 IDS は既知の不正アクセスのみしか検知できないが, ネットワーク型 IDS と比較して検知処理が軽い. 現在は, パターンマッチングを行うホスト型 IDS が普及している. 近年は, ファイアウォールと IDS を組み合わせ, 不正アクセスを遮断する **IPS (Intrusion Prevention System)** も実用化されている.

ホスト型 IDS では, 既知の不正アクセスパターンを**正規表現**で記述していることが多い. IDS の処理のボトルネックは, ネッ

トワーク上のデータと合致する不正アクセスパターンを検出する**正規表現マッチング**である．オープンソースの IDS として，サーバに用いられる SNORT [13] や個人のブラウザ (FireFox) にインストールする Firekeeper [7] が有名である．どちらも，正規表現マッチングに基づくホスト型 IDS である．

1.2 関 連 研 究

入力文字列に対する正規表現マッチングは，与えられた正規表現と等価な有限オートマトンを用いた入力文字列の受理を調べることと等価である．入力に対し，遷移状態が一意に決まらないオートマトンを**非決定性オートマトン (NFA)** といい，遷移状態が一意に決まるオートマトンを**決定性オートマトン (DFA)** という．DFA を用いた手法は，Aho-Corasick アルゴリズム [1] に基づく手法が一般的である．Aho-Corasick オートマトンをビット分割し，コンパクトに実現する手法 [15]，Aho-Corasick オートマトンと MPU を組合わせ，正規表現マッチングを行う手法 [3] が提案されている．Brodie [5] らは，パイプライン化 DFA を用いて正規表現回路を実現した．一方，NFA を用いた手法として，NFA を PC のシフト演算と AND 演算で模擬するアルゴリズム [2] を並列ハードウェア常に実現する Prasanna の手法 [12] が知られている．Prasanna の手法の様々な改良法として，正規表現の共通部を制約を考慮しながら併合する手法 [9] や正規表現の繰返しを Xilinx 社 FPGA の素子 (SRL16) にマッピングする手法 [4] が提案されている．

1.3 本論文の貢献点

本論文の貢献点は 2 点である．

a） ハードウェア実現した場合の NFA と DFA の比較

文献 [17] で，Random Access Machine (RAM) モデル (ソフトウェア) での NFA と DFA との複雑度の比較が行われている．しかしながら，並列ハードウェア上の複雑度の比較は行われていない．本論文では，NFA ハードウェアと DFA ハードウェアの計算複雑度と面積複雑度を正規表現の長さ s に関して求め，比較を行う．

b） 多文字遷移を行う NFA に基づく正規表現マッチング回路

既存の NFA ハードウェアは，1 文字遷移を行う Baeza-Yates らの NFA に基づくため，AND やシフタで実現できる．FPGA で実現する場合，AND やシフタは Look-Up テーブル (LUT) で実現できる．しかし，現在の FPGA は大容量の組込みメモリの両方を持つため，従来の LUT のみを用いる手法では，FPGA のリソース利用効率が悪い．本論文では，多文字遷移を行う NFA に基づく正規表現マッチング回路を提案する．提案手法は，LUT と組み込みメモリを利用するため，FPGA のリソース利用効率が良い．

本論文の残りは以下の様に構成される．第 2 章では NFA に基づく正規表現マッチング回路について述べ，第 3 章では NFA と DFA のハードウェア実現の比較を行い，第 4 章では正規表現マッチング回路の簡単化について述べ，第 5 章では実験結果を示し，第 6 章で本論文のまとめを行う．

2. NFA に基づく正規表現マッチング回路

2.1 正 規 表 現

正規表現は**文字**と文字の集合を表す**メタ文字**から成る．提案手法は以下のメタ文字を扱う: ‘*’ (0 文字以上の繰返し); ‘?’ (0 回又は 1 回の繰返し); ‘.’ (任意の一文字); ‘+’ (1 回以上の繰返し); ‘()’ (優先度の変更); ‘文字列 | 文字列’ (文字列の OR)．

2.2 NFA と DFA

DFA に入力文字に関係なく遷移する ε **遷移**を付加することで NFA を得る．Yu らが行った Random Access Machine (RAM) モデルに対する解析 [17] では，多くの場合，DFA では一意に遷移先が決まるので高速に評価できるが，正規表現を表現した場合，巨大なメモリが必要である．一方，NFA では RAM モデルに基づくため，ε 遷移を逐次模擬する必要があり，低速となるが，正規表現をコンパクトに表現できる．

2.3 Baeza-Yates らの NFA に基づく正規表現マッチング回路

図 1 に正規表現から NFA に変換する手法を示す．図 1 において ε は ε 遷移を表し，灰色の状態は受理状態を表す．図 2 に正規表現 “abc(ab)*a” を NFA に変換した結果を示す．また，入力文字列 ‘abca’ に対する NFA の状態遷移を示す．図 2 において，ベクトルの要素は NFA の各状態に対応し，遷移している状態を’1’ で表す．図 3 に図 2 に示した NFA を模擬する回路を示す．NFA を模擬するため，メモリを用いて 1 文字を検出し，**マッチング・エレメント (ME)** に検出信号を送る．ME は状態遷移を模擬し，マッチ信号を出力する．ME 中の *FF* は，図 2 のベクトルの各要素を保持する．i は前状態からの遷移信号; o は次状態への遷移信号; c はメモリからの文字検出信号入力; ei, eo は ε 遷移の入出力信号を表す．

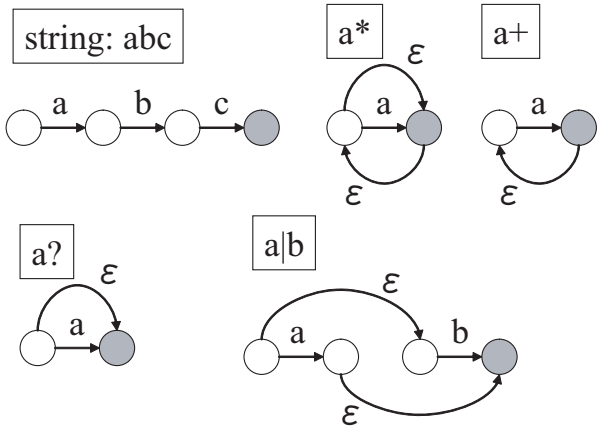

図 1 正規表現から NFA への変換．

3. NFA と DFA を模擬する並列ハードウェアの比較

ここでは NFA と DFA を模擬する並列ハードウェアの比較を行う．正規表現中の文字数を正規表現長と呼び，s で表す．本章では，NFA と DFA を模擬するハードウェアの面積複雑度と計算時間複雑度を m 個の長さ s の正規表現に対して求め，比較を行う．

［例 3.1］ 図 2 に示した，正規表現 “abc(ab)*a” を表す NFA では $s = 6$ である．正規表現長は NFA の状態数と必ずしも一致しない． ■

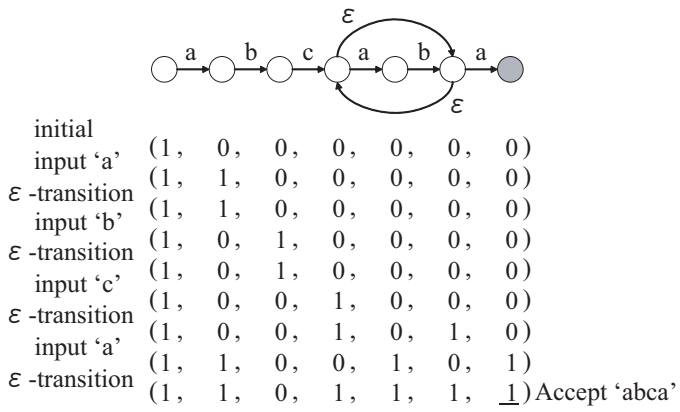

図 2　NFA と状態遷移の例.

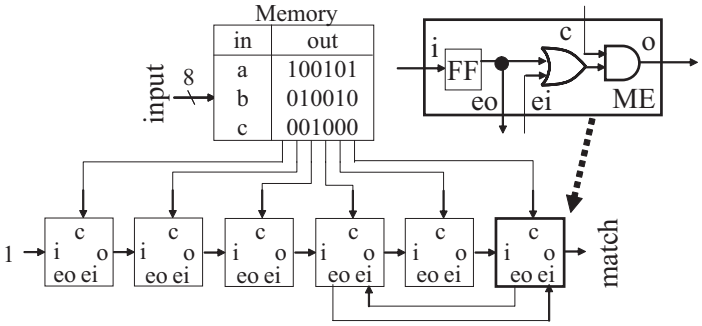

図 3　NFA を模擬する回路 [12].

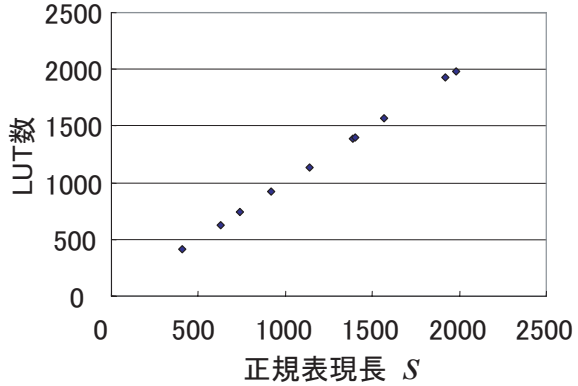

図 4　正規表現長と LUT 数の関係.

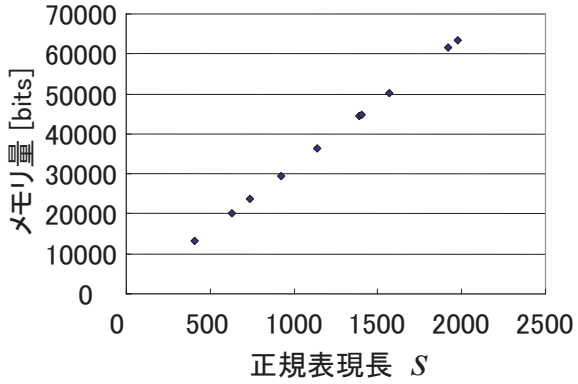

図 5　正規表現長とメモリ量の関係.

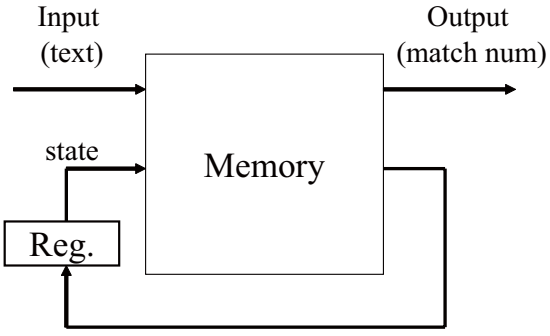

図 6　DFA を実現するシーケンサ.

3.1　NFA に基づく正規表現マッチング回路の複雑度解析

図 3 より, NFA を模擬する回路は OR ゲートと AND ゲート, 及びフリップフロップで構成される ME と 1 文字を検出するメモリで構成可能である. ME は 3 入力 LUT1 個とフリップフロップ 1 個で構成できる. 現在の多くの FPGA では 4 入力 LUT とフリップフロップがペアで配置されており, ME はこの素子を用いて構成できる. 従って, m 個の長さ s の正規表現に関する ME の面積複雑度は, 必要 LUT 数より $O(ms)$ である. 一方, 1 文字を検出するメモリは $m \times 2^8 \times s$ ビットあれば十分であるから, メモリ量に関する面積複雑度も $O(ms)$ である.

図 3 に示した回路は, NFA の s 個の状態と s 個の ε 遷移を 1 クロックで並列に模擬する. また, m 個の正規表現は図 3 に示した回路を m 並列で模擬できる. 従って, NFA に基づく並列正規表現マッチング回路の時間複雑度は $O(1)$ である.

オープンソースの侵入検知システム SNORT [13] から幾つかの正規表現を抽出し, 図 3 に示した回路を合成した. そして, 必要 LUT 数とメモリ量を求めた. 図 4 に正規表現長 s と LUT 数の関係を, 図 5 に正規表現長 s とメモリ量の関係を示す. 図に示すように, LUT 数, メモリ量共に正規表現長 s に対して線形に増加している.

3.2　DFA との比較

DFA は DFA の状態を保持するレジスタと遷移先を保持するメモリ, 及びそれらを制御する簡単なシーケンサで構成できる. 本論文でも, 図 6 に示したシーケンサを DFA を実現するハードウェアとする. DFA を実現するシーケンサは, 1 クロックで次の状態が確定するので, 時間複雑度は $O(1)$ である. また, 図 6 に示したシーケンサの面積の大部分は遷移先を保持するメモリであり, FPGA でこのシーケンサを構成する場合, メモリを除く部分の面積はほぼ一定であるとみなせる. 従って, LUT に関する面積複雑度は $O(1)$ である.

Yu [17] らは DFA のメモリ量に関する解析を行い, m 個の長さ s の正規表現を実現する DFA のメモリ量は $O(\sum^{sm})$ となることを示した. ここで $\sum$ は正規表現で使用する異なる文字数を表し, 一般的な正規表現では $\sum = 2^8 = 256$ である. Tan らは DFA シーケンサのメモリ量を削減するビット分割 DFA シーケンサを示した [15]. また, Sherwood らはビット分割 DFA シーケンサの状態数が分割前の DFA の状態数を超えないことを証明した [6]. ビット分割を行えばメモリ量を削減できることは保証されているが, 複雑度の $O(\sum^{sm})$ は変わらない. 今後, 正規表現で記述された IDS のルール数は増加することが予想されるため, DFA を用いる方法ではメモリ量が指数関数的に増加する点が問題となる. 表 1 に NFA と DFA を実現する並列ハード

表 1　NFA と DFA を実現する並列ハードウェアの複雑度の比較.

		ビット分割 DFA	Prasanna-NFA
面積複雑度	LUT 数	$O(1)$	$O(ms)$
	メモリ量	$O(\sum^{ms})$	$O(ms)$
時間複雑度		$O(1)$	$O(1)$

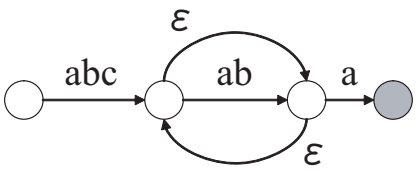

図 7　図 2 に示した NFA と等価な MNFA(3).

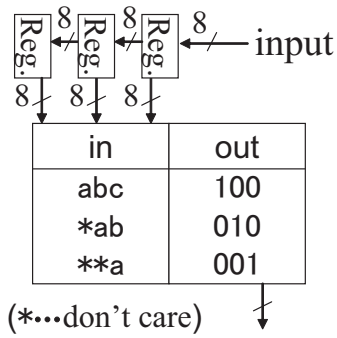

図 8　FIMM.

ウェアの複雑度の比較を示す.

4.　正規表現マッチング回路の簡単化

4.1　多文字遷移を行う NFA [10]

NFA を並列に実現する回路では, 必要 LUT 数は状態数に比例する. p **文字を扱う NFA (MNFA(p): NFA with p-character-consuming transition modular NFA)** を用いると NFA の状態数を削減できる. 本論文では, 1 文字を扱う NFA (MNFA(1)) を単に NFA と表記する. p 文字を扱う MNFA(p) に変換するには, 1 文字だけを扱う NFA の連続する状態間の遷移文字を連結する. ただし, ε 遷移を保持するため, 連結する文字間の状態には ε 遷移が存在しない, という制約を設ける. 図 7 に図 2 に示した NFA と等価な MNFA(3) を示す.

4.2　多文字遷移を行う NFA に基づく正規表現マッチング回路

p 文字を扱う MNFA(p) では, 遷移文字列が入力されたときに, 状態遷移が起こる. 図 7 では, 遷移文字列は {abc,ab,a} である. 遷移文字列の長さは有限であるため, 有限長の文字列を検出する**有限入力メモリ機械 (FIMM)** [8] を用いる. 図 8 に {abc,ab,a} を検出する FIMM を示す. 図 9 に多文字遷移を行う回路を示す. 図 9 では, FIMM で 'abc' を検出すると検出信号が送られる. FIMM で処理中に, 前の状態からの遷移信号を保持するため, シフトレジスタを挿入する. Xilinx 社 FPGA は 4 入力 LUT のモードを SRL16 に切り替えることで 16 ビットまでのシフトレジスタを構成できる [16]. 図 10 に Xilinx 社の FPGA の LUT のモードを示す. FIMM が検出する文字列の最大長を p とすると, FIMM のメモリを直接実現する場合, $p2^{8p}$ ビットのメモリが必要である. 図 11 の実現では $p = 3$ なので, 必要メモリ量は 48 メガビットとなり, 現実的ではない. よって, FIMM のメモリを複数のメモリとビットワイズ AND で実現する [11]. 実装では FPGA の組込みメモリのビット長に適合させるため, p 個のメモリで FIMM を実現する.

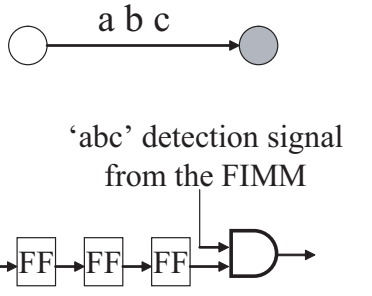

図 9　多文字遷移を行う回路.

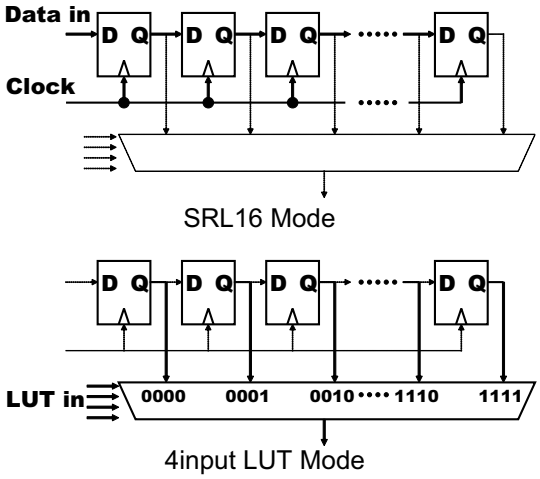

図 10　Xilinx 社 FPGA の LUT のモード.

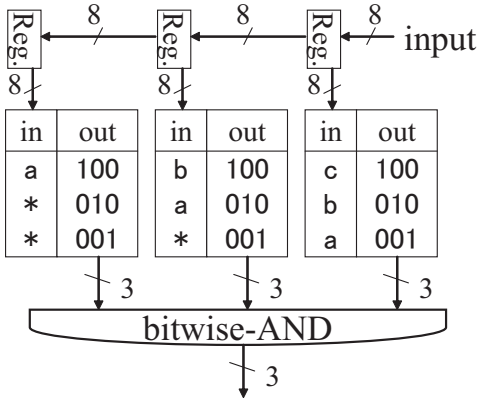

図 11　FIMM の分割.

[例 4.2]　図 12 に図 9 に示した MNFA(3) を模擬する回路を示す. 多文字遷移を実現するため, シフトレジスタを ME 内に挿入する. FIMM で多文字の遷移文字列を受理すると, ME に検出信号が送られ多文字遷移を実行する. ■

4.3　FIMM のメモリの分割

図 11 では s 出力の FIMM のメモリを 1 文字 (8 ビット) 毎に分割したが, さらに分割することができる. 1 文字は 8 ビットなので, 4 通りの分割 (図 13) が考えられる. 8 入力 s 出力の

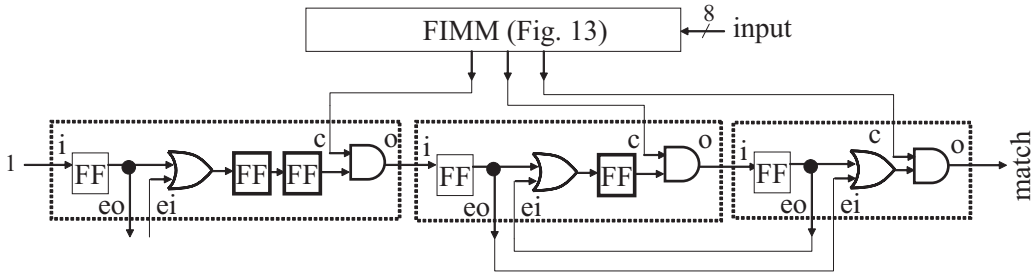

図 12　MNFA(3) を模擬する回路.

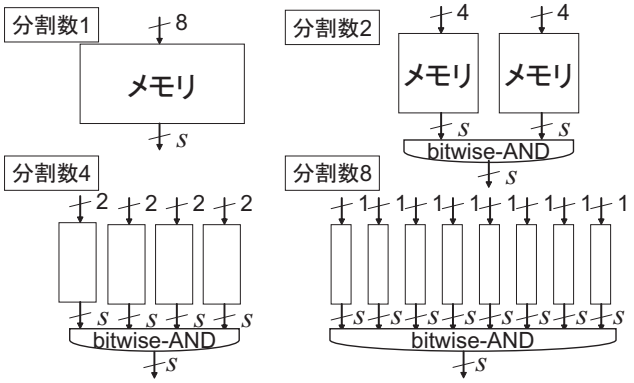

図 13　FIMM のメモリの分割.

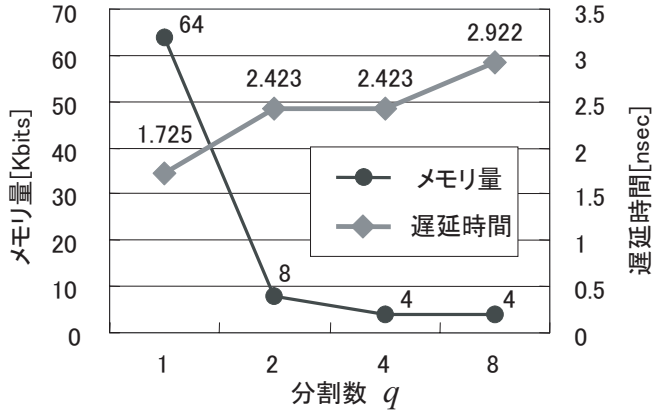

図 14　メモリ量と遅延時間.

FIMM のメモリを q 分割した場合, 総メモリ量 $M_{FIMM}(q)$ は

$$M_{FIMM}(q) = q \times 2^{\frac{s}{q}} \times s \qquad (1)$$

となる. 一方で, 分割を行うとビットワイズ AND 回路が必要であり, 分割数 q を増やすとビットワイズ AND 回路が複雑になる. つまり, q を増加すると総メモリ量は削減できるが, ビットワイズ AND 回路が複雑化し遅延時間が増大する. メモリ量と遅延時間のトレード・オフを調べるため, Xilinx 社の FPGA に図 13 に示した回路を実装し, メモリ量と遅延時間を求めた. ただし, $s = 256$ としている. 図 14 にメモリ量と遅延時間を示す. 分割数 q を増やすとメモリ量 $M(q)$ は減る. ただし, $M_{FIMM}(4) = M_{FIMM}(8)$ である. 一方, 遅延時間は $q = 2$ のときと $q = 4$ のとき, 同じである. これは Xilinx 社 FPGA が 4 入力 1 出力 LUT で構成されているからである. $q = 2$ のとき, s ビットのビットワイズ AND は s 個の 2 入力 AND で構成される. 一方, $q = 4$ のときは s 個の 4 入力 AND で構成される. どちらも 4 入力 1 出力 LUTs 個で構成されるため, 遅延時間は等しい. 図 14 より, $q = 4$ のとき, メモリ量と遅延時間がバランスするが Xilinx 社の FPGA の構成を考えると $q = 2$ のほうがよい. なぜならば, Xilinx 社 FPGA の組込みメモリ (BRAM) は 1 個の容量が 18 Kbit と決まっており, ポート数も 2 個までしか構成できないため, $q = 4$ の場合は, BRAM を十分に使いきれないからである. 一方, $q = 2$ の場合は, BRAM を二分割しデュアルポートを利用すれば, 見かけ上 2 個のメモリを構成できる. 従って, 実装では $q = 2$ とし, FIMM のメモリを 2 分割した.

5. 実 験 結 果

5.1 最適な p の決定

MNFA(p) に基づく正規表現マッチング回路において, FIMM のメモリの分割数を q とし, 正規表現の長さを s, 状態数を $s(p)$ とすると, 必要なメモリ量 $M(p)$ は

$$M(p) = q \times 2^{\frac{s}{q}} \times p \times s(p) \qquad (2)$$

である. 予備実験より, 分割数は $q = 2$ と決定した. ε 遷移が無い場合, p を 2 倍にすると, 状態数 $s(p)$ は半分になり, FIMM のアドレス数 $q \cdot 2^{\frac{s}{q}} \cdot p$ は 2 倍に増える. 従って, $M(p)$ は一定となる. 一方で, ME を実現するのに必要な LUT 数は状態数 $s(p)$ である. よって, ε 遷移が無い場合, p を増加させるほど LUT を削減でき, このときメモリ量は変化しない.

しかしながら実際には ε 遷移による制約のため, 多くの場合, p を 2 倍にしても状態数 $s(p)$ は半分にはならない. すなわち, $\frac{1}{2}s(p) \leqq s(2p)$ であるから, $M(p) \leqq M(2p)$ となり, まとめる文字数 p を増やすとメモリ量が増加する. 一方で, ME を実現する LUT 数は減少するので, メモリ量と LUT 数がバランスする最適な p が存在する.

SNORT から正規表現をいくつか抽出し, p を変化させ MNFA(p) を模擬する回路を生成し, Xilinx 社の Virtex6 FPGA (XC6VLX75T, LUT 数: 74,496 個, BRAM メモリ量: 5,616 Kbits) に実装した. MNFA(p) でまとめる文字数 p に対する FPGA のリソース使用率を図 15 に示す. 図 15 より $p = 2$ または $p = 3$ のとき, 組込みメモリと LUT の使用率がバランスする.

5.2 他の正規表現マッチング回路との比較

提案手法の MNFA(3) を Xilinx 社 Virtex 6 に実装した. 実装結果から, 最大動作周波数は 400 MHz であった. 1 クロックで 1 文字 (8 ビット) 処理できるため, スループットは 3.2 Gbps である. Vassiliadis らは, Virtex の組込みメモリ (BRAM)12

表 2　他の手法との比較.

	オートマトン	デバイス	スループット (Gbps)	ロジックセル数	メモリ量 (Kbits)	文字数	PEM	特徴
Brodie et al. (ISCA'06) [5]	DFA	Virtex 2	4.0	247,000	3,456	11,126	0.66	パイプライン化 DFA
Baker et al. (FPL'06) [3]	DFA	Virtex 4	1.4	N/A	6,000	16,715	0.22	MPU とビット分割 DFA
Vassiliadis et al. (FPT'06) [4]	NFA	Virtex 4	2.9	25,074	0	19,580	2.27	Prasanna 法の改良
提案手法	NFA	Virtex 6	3.2	4,707	441	12,095	4.11	MNFA(3)

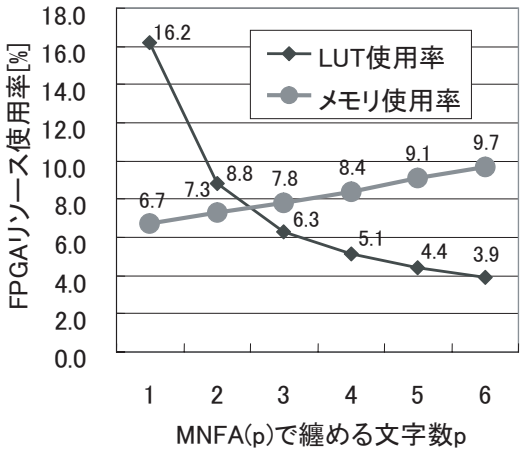

図 15　FPGA のリソース使用率.

バイトとロジックセル 1 個の面積が等しいという結果 [14] を用いて, 単位面積コスト当りのスループット (PEM: Performance Efficiency Metric) を定義し, 正規表現マッチング回路の比較を行っている [4]. PEM は

$$PEM = \frac{\text{スループット } (Gbps)}{\text{面積コスト}} \tag{3}$$

$$= \frac{\text{スループット } (Gbps)}{\frac{\text{ロジックセル数} + \frac{\text{メモリ量 } (Bytes)}{12}}{\text{文字数}}} \tag{4}$$

で得られる. 表 2 に他の正規表現マッチング回路と PEM を比較した結果を示す. 表 2 より, 提案手法は PEM に関して DFA に基づく手法よりも 6.2-18.6 倍優れており, NFA に基づく手法よりも 1.8 倍優れていることがわかる.

6. ま と め

本論文では, p 文字遷移を行う NFA (MNFA(p)) に基づく正規表現回路の実現法および, 正規表現から MNFA(p) に変換する手法を述べた. また, 本論文では, 正規表現の長さと個数から, 面積複雑度と時間複雑度を求め, NFA に基づく手法が DFA に基づく手法よりも優れていることを示した. Xilinx 社 FPGA に MNFA(p) に基づく正規表現回路を実装した結果, p が 2 または 3 のとき, FPGA の LUT と組込みメモリの使用効率がバランスすることを実験的に示した. 面積当りの性能で比較した結果, 提案手法は DFA を用いた手法よりも 6.2-18.6 倍優れており, NFA を用いた手法よりも 1.8 倍優れていることがわかった. 提案手法では FPGA のリソース (LUT と組込みメモリ) の使用効率が良いため, 安価な FPGA を用いて高性能な回路が実現できる.

今後の課題の一つは, 提案手法に第 1 章の関連研究で述べた, NFA に基づく正規表現回路を簡単化する手法との併用を検討することである.

7. 謝　　辞

本研究は, 一部, 日本学術振興会・科学研究費補助金, および, 文部科学省・知的クラスター創成事業 (第二期) の補助金による.

文　　献

[1] A. V. Aho, and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," *Comm. of the ACM*, Vol. 18, No. 6, pp. 333-340, 1975.

[2] R. Baeza-Yates, and G. H. Gonnet, "A new approach to text searching," *Communications of the ACM*, Vol. 35, No. 10, pp. 74-82, Oct., 1992.

[3] Z. K. Baker, H. Jung, and V. K. Prasanna, "Regular expression software deceleration for intrusion detection systems," *16-th Int. Conf. on Field Programmable Logic and Applications (FPL'06)*, pp. 28-30, 2006.

[4] J. Bispo, I. Sourdis, J. M. P. Cardoso, and S. Vassiliadis, "Regular expression matching for reconfigurable packet inspection," *Proc. IEEE International conference on Field Programmable Technology (FPT 2006)*, pp.119-126, 2006.

[5] B.C.Brodie, D.E.Taylor, and R.K.Cytron, "A scalable architecture for high-throughput regular-expression pattern matching," *Proc. 33rd Int'l Symp. on Computer Architecture (ISCA 2006)*, pp. 191-202, 2006.

[6] R. Dixon, O. Egecioglu, and T. Sherwood, "Automata-theoretic analysis of bit-split languages for packet scanning," *Proc. 13th Int'l conf. on Implementation and Application of Automata (CIAA 2008)*, pp.141-150, 2008.

[7] "Firekeeper: Detect and block malicious sites," http://firekeeper.mozdev.org/

[8] Z. Kohavi, *Switching and Finite Automata Theory, McGraw-Hill Inc.*, 1979.

[9] C. Lin, C. Huang, C. Jiang, and S. Chang, "Optimization of regular expression pattern matching circuits on FPGA," *Proc. of the Conference on Design, automation and test in Europe (DATE 2006)*, pp.12-17, 2006.

[10] H. Nakahara, T. Sasao, and M. Matsuura, "A regular expression matching using non-deterministic finite automaton," *Proc. of Eighth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE 2010)*, Grenoble, France, July 26-28, 2010. (to be published)

[11] H. Nakahara, T. Sasao, M. Matsuura, and Y. Kawamura, "A virus scanning engine using a parallel finite-input memory machines and MPUs," *Proc. Int'l Conf. on Field Programmable Logic and Applications (FPL 2009)* Aug. 31 - Sept. 2, 2009.

[12] R. Sidhu, and V. K. Prasanna, "Fast regular expression matching using FPGA," *Proc. of the 9th Annual IEEE symp. on Field-programmable Custom Computing Machines (FCCM 2001)*, pp. 227-238, 2001.

[13] "SNORT official web site," http://www.snort.org.

[14] T. Sproull, G. Brebner, and C. Neely, "Mutable codesign for embedded protocol processing," *Proc. of 15th Int'l Conf. on Field Programmable Logic and Applications*, 2005.

[15] L. Tan, and T. Sherwood, "A high throughput string matching architecture for intrusion detection and prevention," *Proc. 32nd Int'l symp. on Computer Architecture (ISCA 2005)*, pp.112-122, 2005.

[16] "Using Look-up tables as shift registers (SRL16)," http://www.xilinx.com/support/documentation/application_notes/xapp465.pdf

[17] F. Yu, Z. Chen, Y. Diao, T. V. Lakshman, and R. H. Katz, "Fast and memory-efficient regular expression matching for deep packet inspection," *Proc. of the 2006 ACM/IEEE symp. on Architecture for Networking and Communications Systems (ANCS 2006)*, pp. 93-102, 2006.