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.

オートマトンの分解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　NFA (Non-deterministic finite automaton) において, p 文字までの文字列に対する遷移を許すことにより状態数を削減した NFA を MNFA(p) (Modular NFA with p-character-consuming transition) という. また, 文字数 p の制約を除いた MNFA を MNFAU (MNFA with unbounded multi-character transition) という. 本論文では, MNFAU の分解に基づく正規表現マッチング回路の実現法について述べる. まず, MNFAU を文字列検出回路と状態遷移模擬回路に分解する. 次に, 文字列検出回路を DFA (Deterministic finite automaton) で実現し, 状態遷移模擬回路を NFA で実現する. MNFAU の分解に基づく正規表現マッチング回路は, メモリの使用率と LUT の使用効率が優れており, 安価なシステムで実装可能である. 本論文では, 並列ハードウェアにおける, NFA, DFA, 及び分解した MNFAU の回路の面積と時間複雑度の解析を行い, MNFAU を分解することにより計算時間を増やすことなく面積を削減できることを示す. 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 社の FPGA に実装し, 必要なメモリ量と LUT 数を求め, 提案手法が既存手法よりも優れていることを示す.

A Regular Expression Matching Circuit Based on a Decomposed Automaton

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　In this paper, we propose a regular expression matching circuit based on a decomposed automaton. To implement regular expressions on the hardware, first, we convert them to a non-deterministic finite automaton (NFA). Then, to reduce the number of states, we convert the NFA into a modular non-deterministic finite automaton with unbounded multi-character transition (MNFAU). Next, to realize it by a feasible amount of the hardware, we decompose the MNFAU into the deterministic finite automaton (DFA) and the NFA. The DFA part is implemented by an off-chip memory and a simple sequenser, while the NFA part is implemented by a cascade of logic cells. Also, this paper shows that the MNFAU based implementation is superior to the DFA and the NFA based ones, with respect to the area and time complexity.

1. はじめに

1.1 正規表現マッチングとその応用

正規表現 (regular expression) とは、文字列の集合の表現法の一種であり, 入力文字列に対して正規表現で記述されたパターンを検出することを**正規表現マッチング**という. ネットワーク機器 (侵入検知システム [15] [8], スパムメールフィルタ [16], ウィルス検出 [6], L7 フィルタ [10] 等) は 1 Gbps を超える高速なデータ処理を行うため, ハードウェアによる正規表現マッチングが必要である. ネットワーク機器では, 少量多品種生産かつプロトコルの更新が頻繁に行われるため, 再構成可能な FPGA を用いることが多い. 近年, 高速ネットワーク用に設計された高速レシーバを搭載した FPGA がリリースされており, FPGA をネットワーク機器に採用する動きが加速している.

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, まず, 与えられた正規表現と等価な有限オートマトンを構成し, 次に, このオートマトンを用いて入力文字列の受理を調べることにより実行できる. 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毎に分割し, コンパクトに実現する手法 [18], 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上に実現する Prasanna の手法 [14] が知られている. Prasanna の手法の改良法として, 正規表現の共通部を制約を考慮しながら併合する手法 [11], 正規表現の繰返しを Xilinx 社の 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 提案手法の概要

本論文では, FPGA に適した正規表現マッチング回路の実現法について述べる. 本論文の貢献点は以下の 2 点である.

a) 文字列遷移を行う 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) で実現する. しかし, 現在の FPGA は LUT の他に大容量の組込みメモリを持つため, 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, 文字列遷移を行う 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, オートマトンを分解し, LUT と組込みメモリの両方を活用するため, 正規表現マッチング回路を FPGA 上に効率よく実装可能である.

b) 並列ハードウェア上の複雑度の解析

NFA と DFA との複雑度の比較は, Random Access Machine (RAM) モデル (ソフトウェア) 上や [20], 並列ハードウェアモデル (ハードウェア) [12] 上で行われている. 本論文では, 提案する回路の複雑度解析を行い, 他の方法よりも理論的に優れていることを示す. また, 侵入検知システム SNORT のデータを用いた実験により, この解析が正しいことを確認する.

本論文の残りは以下の様に構成される. 第 2 章ではオートマトンに基づく正規表現マッチング回路について述べる. 第 3 章では文字列で遷移する NFA に基づく正規表現マッチング回路について述べる. 第 4 章では複雑度の解析を行う. 第 5 章では実験結果を示す. 第 6 章で本論文のまとめを行う.

2. オートマトンに基づく正規表現マッチング回路

2.1 DFA に基づく正規表現マッチング回路

正規表現は**文字**と文字の集合を表す**メタ文字**から成る. 本論文では表 1 に示すメタ文字を扱う.

表 1 本論文で扱う正規表現のメタ文字.

メタ文字	意味
.	任意の一文字
*	0 文字以上の繰返し
+	1 回以上の繰返し
?	0 回又は 1 回の繰返し
^	文字列の先頭のみにマッチ
$	文字列の末尾のみにマッチ
()	優先度の変更
[]	文字の集合
[^]	文字の補集合
{n,m}	n 回以上 m 回以下の繰り返し
{n,}	n 回以上の繰り返し
{n}	n 回繰り返し
|	文字列の OR

［定義 2.1］ 決定性有限オートマトン (Deterministic finite automaton: DFA) は $M_{DFA} = (S, \Sigma, \delta, s_0, A)$ の 5 組で構成されるオートマトンである. ここで, $S = (s_0, s_1, \ldots, s_{q-1})$ は状態の有限集合; Σ は文字の有限集合であり, 実用的なネットワーク機器では 8 ビットのアスキーコード ($|\Sigma| = 2^8$) である. δ は状態遷移関数 ($\delta : S \times \Sigma \to S$) であり; $s_0 \in S$ は初期状態であり; $A \subseteq S$ は受理状態の集合である.

［定義 2.2］ $s \in S$, $c \in \Sigma$ とするとき, 状態遷移関数が $\delta(s, c) \neq \{\phi\}$ となる c を**状態 s の遷移文字**という.

状態遷移関数 δ を拡張し, DFA が受理する入力文字列を定義する.

［定義 2.3］ Σ^+ を空でない文字列の集合とするとき, 状態遷移関数の定義域を拡張した写像 $\hat{\delta} : S \times \Sigma^+ \to S$ を定義する. ここで, $C \subseteq \Sigma^+$, $s \in S$ とすると, 関数 $\hat{\delta}(s, C)$ は, 状態 s において文字列 C を与えたときの遷移先を表す. このとき, $M_{DFA} = (S, \Sigma, \delta, s_0, A)$ において $C_{in} \subseteq \Sigma^+$, $a \in A$ とすると,

$$\hat{\delta}(s_0, C_{in}) = a \tag{1}$$

が成立するとき, M_{DFA} は文字列 C_{in} を受理する.

拡張した状態遷移関数 $\hat{\delta}$ は, 文字列 $C = (c_0, c_1, \ldots, c_n)$ の文字 c_i と状態遷移関数 δ を用いて

$$\hat{\delta}(s, C) = \hat{\delta}(\delta(s, c_0), (c_1, \ldots, c_n)) \tag{2}$$

と再帰的に記述できる.

式 (1) と (2) より, 遷移文字と現在状態に応じて状態遷移を繰り返すことで, DFA を用いた文字列照合を行うことができる.

［例 2.1］ 図 1 に正規表現 A+[AB]{3}C に対応する DFA を示す. ■

［例 2.2］ 図 1 に示した DFA に入力文字列 (A,A,B,A,C) を与えたときの動作を示す. 初期状態を s_0 とする. $\delta(s_0, A) = s_1$ に遷移. $\delta(s_1, A) = s_2$ に遷移. $\delta(s_2, B) = s_5$ に遷移. $\delta(s_5, A) = s_9$ に遷移. $\delta(s_9, C) = s_{11}$ に遷移. s_{11} は受理状態なので, 文字列 (A,A,B,A,C) が受理される. ■

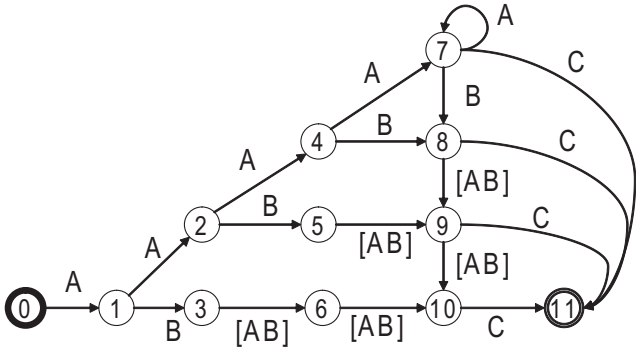


図 1 正規表現 A+[AB]{3}C に対応する DFA.

図 2 に DFA を実現する回路を示す. 図 2 において, レジスタは現在状態を保持し, メモリは状態遷移関数 δ を実現する. 状態数を q, Σ 中の文字数を $|\Sigma| = n$ とすると, DFA を実現する回路のメモリ量は $2^{\lceil log_2 n \rceil + \lceil log_2 q \rceil} \times \lceil log_2 q \rceil$ ビットとなる (注1).

(注1): 図 2 のレジスタのビット数を固定しない場合, 正規表現のクラスになる

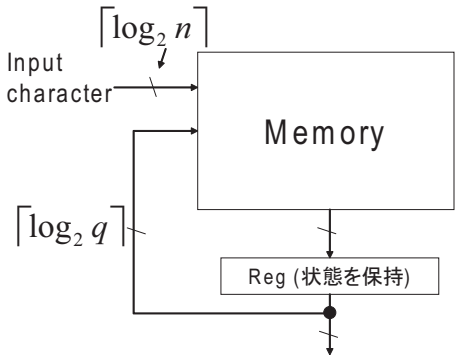


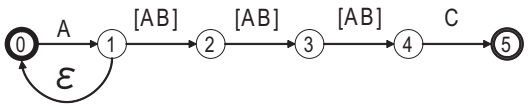
図 2　DFA を実現する回路.

2.2　NFA に基づく正規表現マッチング回路

［定義 2.4］ 非決定性有限オートマトン (Non-deterministic finite automaton: NFA) は $M_{NFA} = (S, \Sigma, \gamma, s_0, A)$ の 5 組で構成されるオートマトンである. S, Σ, s_0, A は定義 2.1 で述べたものと同じであるが, 状態遷移関数が $\gamma : S \times (\Sigma \cup \{\varepsilon\}) \to P(S)$ である点が異なる. ここで, ε は空の文字であり, $P(S)$ は集合 S の冪集合である.

NFA が DFA と異なる点は, 空文字入力 (ε 入力) を受け付けることである. 従って, NFA では状態遷移が一意に決まらないため, 複数の状態に遷移可能である. 本論文では, NFA の状態遷移図中の ε が ε **遷移**を表すものとする.

［例 2.3］ 図 3 に正規表現 A+[AB]{3}C に対応する NFA を示す. また, 入力文字列 (A,A,B,A,C) を与えたときの動作を示す. 図 3 において, 2 値ベクトルは遷移中の状態を表し, アクティブな状態は 1 で表している. ■



Initial	1	0	0	0	0	0
Input 'A'	1	1	0	0	0	0
Input 'A'	1	1	1	0	0	0
Input 'B'	1	1	1	1	0	0
Input 'A'	1	1	1	1	1	0
Input 'C'	1	0	0	0	0	1

accept 'AABAC'

図 3　正規表現 A+[AB]{3}C に対応する NFA.

NFA を実現する手法として, NFA の各状態を小規模回路で実現し, 並列に動作させる方法がある [14]. 図 4 に図 3 に示した NFA を実現する回路を示す (注2). NFA を実現するため, メモリを用いて遷移文字を検出し, 各状態を表現する回路に検出信号を送る. ε 遷移は, OR ゲートとランダムな配線で実現する.

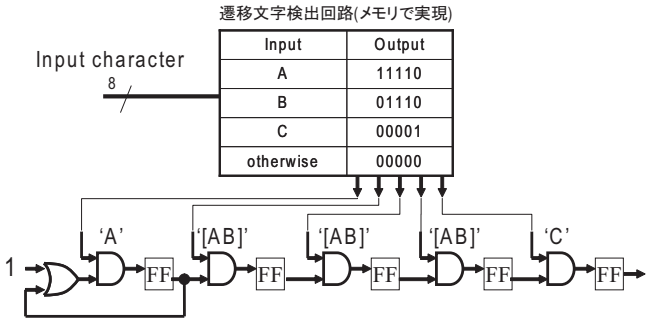


図 4　図 3 に示した NFA を実現する回路.

3.　文字列で遷移する NFA に基づく正規表現マッチング回路

3.1　MNFAU

既存の NFA を実現する手法は 1 文字遷移を行う NFA に基づくため, AND ゲートやシフタで実現できる. FPGA で実現する場合, AND ゲートやシフタは 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を持ち, チップ面積のかなりの部分を占めるため, 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, **文字列で遷移する NFA(Modular nondeterministic finite automaton with unbounded multi-character transition: MNFAU)** に基づく正規表現マッチング回路を提案する. 1 文字だけを扱う NFA の連続する状態を縮約する. ただし, 元の NFA との等価性を保持するため, 以下に示す条件を満たす場合のみ, 状態を縮約する.

［定義 3.5］ NFA の状態 $S = (s_0, s_1, \ldots, s_{q-1})$ を, $\{S\} = \{S_1\} \cup \{S_2\} \cup \cdots \{S_u\}$ かつ $\{S_i\} \cap \{S_j\} = \phi (i \neq j)$ となる分割 $(S_1, S_2, \ldots, S_u)$ を考える. このとき, $S_i = (s_k, s_{k+1}, \ldots, s_{k+p})$ は $r = k, k+1, \ldots, k+p-1$ に対して $e_r = 0$ のとき, **MNFAU の 1 状態に縮約可能**である. ここで, e_r は NFA の状態 s_r の ε 遷移の入出力数を表す.

［定義 3.6］ $(s_k, s_{k+1}, \ldots, s_{k+p})$ を MNFAU の 1 つの状態に縮約するとき, 状態 s_j の遷移文字 $c_j \in \Sigma$ を連結した文字列 $C = (c_k, c_{k+1}, \ldots, c_{k+p})$ を**遷移文字列**という.

［例 3.4］ 図 3 に示す NFA の状態集合 (s_2, s_3, s_4, s_5) は MNFAU の 1 状態に縮約可能である. このとき, 縮約した状態への遷移文字列は, 正規表現を用いて [AB][AB][AB]C と表現できる. しかし, 状態集合 (s_1, s_2) は, $e_1 \neq 0$ であるため, 縮約できない. ■

［例 3.5］ 図 5 に図 3 に示す NFA と等価な MNFAU の例を示す. ■

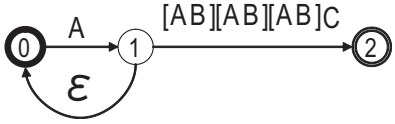


図 5　図 3 に示した NFA と等価な MNFAU の例.

MNFAU を用いることで NFA の状態数を削減できるため, LUT 数を削減できる. しかしながら, 従来手法とは異なる実装法が必要である.

が, 現実的に実現不可能である. 一方, レジスタのビット数を固定すると, 受理できる文字列のクラスが制限される. Yu らも [20] はクラスを制限し, DFA の解析を行っている. 本論文でも, Yu らが扱ったクラスを考慮する. この場合, 状態を保持するレジスタは, 状態遷移関数を模擬するメモリと比較して無視できるほど小さいので, メモリ量には考慮しない.

(注2)：文献 [14] では, 遷移文字検出回路を LUT で構成している.

3.2 MNFAU の実現

提案手法では MNFAU を

1. 遷移文字列検出回路
2. MNFAU の状態遷移模擬回路

に分解し, それぞれを単純な回路で実現する．遷移文字列はメタ文字を含まず (注3) 長さが p 以下なので, **厳密マッチング**で検出できる．厳密マッチングは正規表現マッチングよりも簡単なクラスなので, DFA でも現実的なサイズの回路で実現できる．一方, 状態遷移模擬回路は ε 遷移を含むため, NFA の回路に一部変更を加えて実現する．従って, MNFAU は図 6 に示す回路に分解できる．

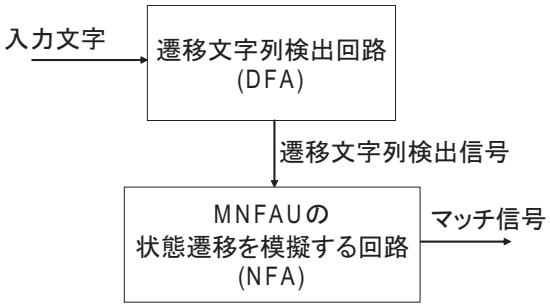


図 6　MNFAU の分解.

c）遷移文字列の検出

NFA の状態を縮約して MNFAU を構成する場合, 一般的には縮約できる状態数は異なる．このため, MNFAU の各状態の遷移文字列長も異なる．本論文では, 読み戻し (バックトラッキング) が不要で, 長さが異なる複数の文字列を受理できる **Aho-Corasick DFA (AC-DFA)** [1] を用いる．AC-DFA は図 2 に示した回路で実現できる．

［例 3.6］ 図 7 に, 図 5 に示した MNFAU の遷移文字列 (A) と ([AB],[AB],[AB],C) を受理する AC-DFA を示す． ■

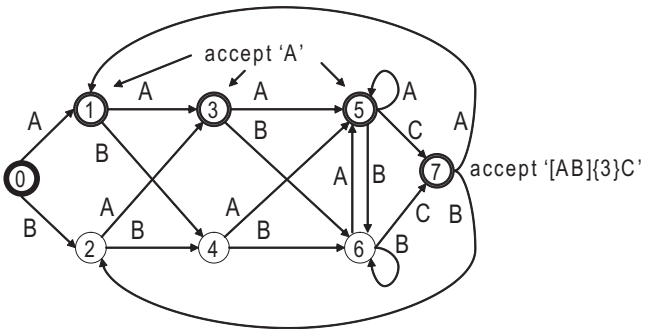


図 7　遷移文字列 (A) と ([AB][AB][AB]C) を受理する AC-DFA.

d）MNFAU の状態遷移の模擬 [13]

MNFAU では, 遷移文字列が検出されたときに, 状態遷移を行う．図 8 に文字列遷移を行う回路を示す．AC-DFA で遷移文字列 (図 8 では, ABC) を検出すると検出信号が送られる．AC-DFA は 1 文字毎に遷移を行うため, 同期をとるために, MNFAU の状態間に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 [19]．図 9 に Xilinx 社の FPGA の LUT のモードを示す．SRL16 モードを用いることで, 16 個のフリップ・フロップを 1 個の LUT で構成できる．

AC-DFA と状態遷移模擬回路で MNFAU を実現する．図 10 に MNFAU を実現する回路を示す．これは, 図 6 に示した MNFAU の分解を実現しているともいえる．AC-DFA では $\lceil log_2 q \rceil$ ビットで現在状態を表す．ここで, q は状態数を表す．$\lceil log_2 q \rceil$ ビットの状態番号から, MNFAU が遷移する検出信号にデコードするメモリ (デコーダメモリ) を用意する．AC-DFA のメモリと比較してデコーダメモリは小さく, かつ出力ビット数が多いので, FPGA 内の組込みメモリで実現する．

［例 3.7］ 図 10 において, デコーダメモリの入力は図 7 に示した AC-DFA の受理状態に割当てた番号である．デコーダメモリの出力は, 検出した遷移文字列によって遷移する状態を表すベクトルである． ■

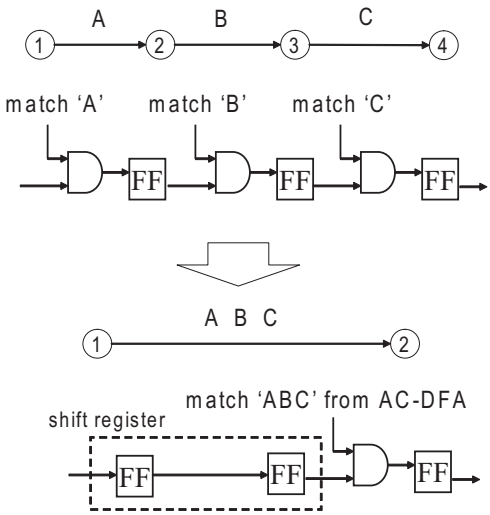


図 8　文字列による遷移.

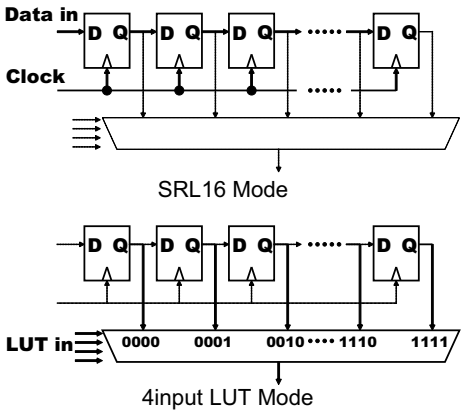


図 9　Xilinx 社の FPGA の LUT のモード.

4. 正規表現マッチング回路の複雑度

MNFAU を遷移文字列検出回路と状態遷移模擬回路に分解することで, 全体のハードウェア量を削減可能なことを示す．まず, 正規表現を DFA のみで実現した場合と, NFA のみで実現した場合の実行時間とハードウェア量の複雑度の解析を行う．次に, MNFAU について同様の解析を行う．そして, 実験的に検証を行う．

(注3)：ただし, 文字の集合'[]' のみ許容する．

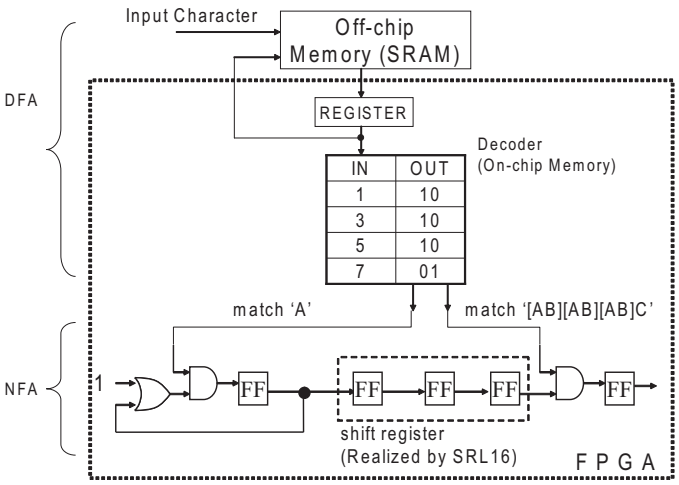


図 10　MNFAU を実現する回路の例.

Xilinx 社の FPGA は, 4 入力 LUT1 個と FF1 個で**ロジック・セル (LC)** [17] を構成している. 本論文では, 2 つのモードの LUT と FF を組合せて状態遷移模擬回路を構成するため, 等価な LC 数で複雑度の解析を行う.

4.1　面積と実行時間

e)　DFA の解析

図 2 に示すように, DFA は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回路で構成できる. DFA を実現する回路は, 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 2 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, このシーケンサを実現する場合, メモリ以外の面積はほぼ一定であるとみなせる. 従って, LC 数に関する面積複雑度は $O(1)$ である. Yu [20] らは DFA のメモリ量に関する解析を行い, m 個の長さ s の正規表現に対する DFA を実現する回路のメモリ量は $O(|\Sigma|^{sm})$ となることを示した.

f)　NFA の解析

図 4 より, NFA は 3 入力 LUT1 個 (OR ゲートと AND ゲートを実現) とフリップフロップ 1 個で構成された回路を直列に並べたもので実現できる. 従って, m 個の長さ s の正規表現に関する LC 数の複雑度は, $O(ms)$ である. 一方, 1 文字を検出するメモリは $m \times |\Sigma| \times s$ ビットあれば十分であるから, メモリ量に関する複雑度も $O(ms)$ である. 図 4 の回路は, NFA の s 個の状態と s 個の ε 遷移を 1 クロックで並列に実行する. また, m 個の正規表現は図 4 の回路を m 個並列にした回路で実現できる. 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.

g)　MNFAU の解析

MNFAU は, 遷移文字列検出回路と状態遷移模擬回路に分解できる. 遷移文字列検出回路は AC-DFA で実現できる. AC-DFA の解析 [7] より, p_{max} を MNFAU の遷移文字列の最大長とすると, メモリ量は $O(|\Sigma|^{p_{max}})$ となる. p_{ave} を MNFAU の遷移文字列の平均長とすると, MNFAU の状態遷移模擬回路は, 図 4 に示す NFA を実現する回路の FF を平均して p_{ave} 個縮約したものであるから, LC 数は $O(\frac{ms}{p_{ave}})$ となる. MNFAU を表現する回路は, m 個の正規表現を実現する回路を m 個並列にした回路で実現できる. また, この回路は一定クロック内に動作が終了するため, 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.

表 2　DFA, NFA, 及び MNFAU に基づく正規表現マッチング回路の並列ハードウェアの複雑度.

	Time	Area	
		Memory	#LC
Baeza-Yates's NFA	$O(1)$	$O(ms)$	$O(ms)$
Aho-Corasic DFA	$O(1)$	$O(\|\Sigma\|^{ms})$	$O(1)$
MNFAU (Aho-Corasic + NFA)	$O(1)$	$O(\|\Sigma\|^{p_{max}})$	$O(\frac{ms}{p_{ave}})$

表 2 に DFA, NFA, 及び MNFAU に基づく正規表現マッチング回路の並列ハードウェアの複雑度を示す. 表 2 より, MNFAU を用いることでメモリ量を $\frac{1}{|\Sigma|^{ms-p_{max}}}$ に, LC 数を $\frac{1}{p_{max}}$ に削減できる. 実際の回路における複雑度を解析するため, オープンソースの侵入検知システム SNORT [15] から 80 個の正規表現を抽出し, DFA, NFA, 及び MNFAU を合成した. そして, 必要 LC 数とメモリ量を求めた. 図 11 に正規表現長 s と LC 数の関係を, 図 12 に正規表現長 s とメモリ量の関係を示す. ここで, 図 12 の縦軸が対数になっていることに注意. 図 11 に示すように, LC 数に関しては正規表現長 s に対して複雑度の比が一定になっており, また, 図 12 に示すように, メモリ量に関しては正規表現長 s に対して複雑度の比が指数関数になっている.

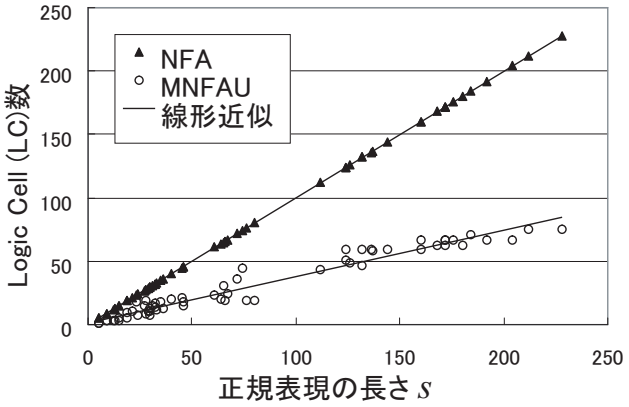


図 11　正規表現長 s と LC 数の関係.

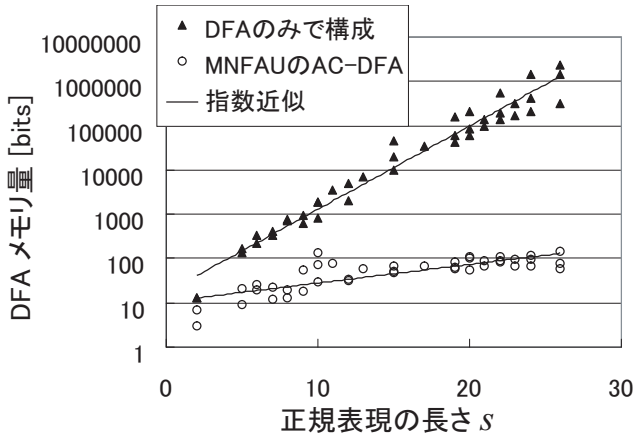


図 12　正規表現長 s とメモリ量の関係.

表 3　他の手法との比較.

	オートマトン	デバイス	Th (Gbps)	#LC	MEM (Kbits)	#Char	#LC/ #Char	MEM/ #Char	特徴
Brodie et al. (ISCA'06) [5]	DFA	Virtex 2	4.0	247,000	3,456	11,126	22.22	3182.2	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	N/A	367.5	MPU とビット分割 DFA
Vassiliadis et al. (FPT'06) [4]	NFA	Virtex 4	2.9	25,074	0	19,580	1.28	0	Prasanna 法の改良
MNFA(p) (SASIMI'10) [12]	MNFA(p)	Virtex 6	3.2	4,717	441	12,095	0.39	37.3	NFA を 3 文字まで縮約した MNFA(3)
提案手法	MNFAU	Spartan 3	1.6	19,552	1,585	75,633	0.25	21.4	MNFAU

5. 実験結果

オープンソース進入検知システム SNORT の正規表現から MNFAU を生成し, Xilinx 社の FPGA (Spartan III, XC3S4000: Logic Cell (LC) 数 62,208 個, BRAM 1,728K ビット) 上に実装した. 実装に用いた SNORT のルール数は 1,114 個 (文字数は 75,633 個) である. MNFAU を生成した結果, MNFAU の状態数は 12,673 個であり, 遷移文字列を検出する AC-DFA の状態数は 10,066 個であった. 実装結果から, 必要な LC 数は 19,552 個であり, AC-DFA を実装するのに必要な外付け SRAM は 16Mbit であった. また, デコーダ用の FPGA メモリは 1,585K ビット必要であった. 外付け SRAM のアクセスがボトルネック (注4) となるので, SRAM の動作周波数 200MHz で正規表現マッチング回路を動作させた. 1 クロックで 1 文字 (8bit) 処理するので, システムのスループットは 1.6 Gbps である.

MNFAU と他の手法との比較を行った結果を表 3 に示す. 表 3 において, *Th* はスループット (Gbps); *#LC* はロジックセル数; *MEM* は FPGA の組込みメモリ量 (Kbits); *#Char* は正規表現の文字数を表す. 表 3 より, MNFAU は DFA を用いる手法と比較して 1 文字当りのメモリ量を 17.17-148.70 分の 1 にでき, NFA を用いる手法と比較して 1 文字当りの LC 数を 1.56-5.12 分の 1 に削減できた.

6. まとめ

本論文では, NFA (Non-deterministic finite automaton) の状態を縮約した MNFAU を考え, MNFAU を 2 つの回路に分解することにより, 正規表現マッチング回路を実現した. まず, MNFAU を遷移文字列検出回路と状態遷移模擬回路に分解する. 次に, 遷移文字列検出回路を DFA で, 状態遷移模擬回路を NFA で実現する. MNFAU に基づく正規表現マッチング回路は, メモリの使用率と LUT の使用効率が優れており安価なシステムで実装可能である. 本論文では, 並列ハードウェアにおける, NFA, DFA, 及び MNFAU の面積と時間複雑度を解析した. MNFAU の分解に基づく実現により, 計算時間を増やすことなく面積を削減できることを示した. 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 に実装し他の手法と比較を行った結果, MNFAU は DFA を用いる手法と比較して 1 文字当りのメモリ量を 17.17-148.70 分の 1 に, NFA を用いる手法と比較して 1 文字当りの LC 数が 1.56-5.12 分の 1 に削減できた.

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.

(注4)：FPGA 内部の回路は 371.2 MHz で動作可能.

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] "Clam Anti Virus: open source anti-virus toolkit," http://www.clamav.net/lang/en/
[7] 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.
[8] "Firekeeper: Detect and block malicious sites," http://firekeeper.mozdev.org/
[9] Z. Kohavi, *Switching and Finite Automata Theory, McGraw-Hill Inc.*, 1979.
[10] "Application Layer Packet Classifier for Linux," http://l7-filter.sourceforge.net/
[11] 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.
[12] H. Nakahara, T. Sasao, and M. Matsuura, "A regular expression matching circuit based on a modular non-deterministic finite automaton with multi-character transition," *The 16th Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI-2010)*, Taipei, Oct. 18-19, 2010, pp. 359-364.
[13] 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.
[14] 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.
[15] "SNORT official web site," http://www.snort.org.
[16] "SPAMASSASSIN: Open-Source Spam Filter," http://spamassassin.apache.org/
[17] "Spartan III data sheet," http://www.xilinx.com/
[18] 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.
[19] "Using Look-up tables as shift registers (SRL16)," http://www.xilinx.com/support/documentation/application_notes/xapp465.pdf
[20] 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.