高速ストリーム処理のための ビット並列パターン照合手法にもとづくハードウェアアルゴリズム

金田　悠作† 　湊　真一† 　有村　博紀†

† 北海道大学大学院情報科学研究科　〒 060–0814 札幌市北区北 14 条西 9 丁目
E-mail: †{y-kaneta,minato,arim}@ist.hokudai.ac.jp

あらまし　センサーデータやネットワークデータ，ウェブデータのようなストリームデータに対する高速なパターン照合のための，効率よいハードウェアアルゴリズムを提案する．そのために，開発のターゲットを FPGA (Field Programmable Gate Array) とし，ビット並列パターン照合手法にもとづいて，正規表現の部分クラスである線形正規表現に対して文字列パターン照合を行うアーキテクチャを与える．これは，与えられた線形正規表現パターンを非決定性有限オートマトンに変換し，並列ハードウェアで高速に実行するものであり，長さ m のパターン P と長さ n のテキスト T に対して，$O(n)$ 時間でパターン照合を行う．

キーワード　パターン照合，正規表現，ストリームデータ，ビット並列手法，ハードウェアアルゴリズム，FPGA

Fast Hardware Algorithm for High-Speed Stream Processing Based on Bit-Parallel Method

Yusaku KANETA†, Shin-ichi MINATO†, and Hiroki ARIMURA†

† Graduate School of Information Science and Technology, Hokkaido University
N14 W9, Sapporo 060–0814, Japan
E-mail: †{y-kaneta,minato,arim}@ist.hokudai.ac.jp

Abstract　We propose an efficient hardware algorithm for high-speed stream processing based on bit-parallel method. We use FPGA (Field Programmable Gate Array) for this algorithm. FPGA is one of the mostly used reconfigurable hardwares. The algorithm can run in $O(n)$ time, where n is the stream length.

Key words　pattern matching, regular expression, stream data, bit-parallelism, hardware algorithm, FPGA

1. まえがき

1.1 背　　景

センサーデータやネットワークデータなどのストリームデータ処理において，高速な正規表現パターン照合の実現は，応用上重要な問題である．とくに，非常に多数の正規表現に対するパターン照合は，処理全体の負荷が大きい．**ネットワーク侵入検知システム** (NIDS, Network Intrusion Detection System) は，正規表現パターン照合のハードウェア実行の重要な応用である．現在，NIDS の一つである Snort では，1000 個以上の正規表現パターンを扱っている [7].

このような多数のパターンに対する処理をハードウェア上で実行することは，処理速度および，消費電力，コスト面などにおいて，多大な利点をもつと考えられる．ここでは，繰り返し再構成可能なハードウェアである FPGA(Field Programmable Gate Array) によるハードウェア実行を考える．

1.2 本研究の目的

本稿では，ビット並列パターン照合手法 [1], [8] にもとづいた **FPGA による線形正規表現パターン照合ハードウェアアルゴリズム**を提案する．一般的な正規表現を対象とした研究としては，Sidhu らによる [6] があるが，この方法では，合成される回路が複雑になるという問題がある．そこで我々は，対象を**線形正規表現**と呼ばれる正規表現の部分クラスに制限し，任意個の定数文字と可変長ワイルドカード，例外文字の連結からなるパターンを考える．

ビット並列パターン照合手法 [1], [8] は，上記のクラスのパターンを線形の NFA に変換し，ビット演算と数値演算におけるレジスタ内の並列性を利用して，状態遷移計算を汎用 CPU 上で高速に実行する．これに対し，提案手法は，この線形の NFA の状態遷移計算を FPGA 上で高速に実行する．また，コンパクトな回路を実現するために，LUT を用いた論理関数の効率的な実装および，文字アルファベットの分割を行う．

提案手法の有効性を見積もるために，各種パターンに対する回路を構成するのに必要な CLB 数の理論的評価を行った．これにより，先行研究 [6] による一般的な正規表現に対する構成手法と比較した場合に，よく使われるいくつかのパターンに対して，提案手法の有効性が示された．FPGA 上での実装，およ

び，評価実験に関しては今後の課題としたい．

1.3 FPGA

FPGA (Field Programmable Gate Array) は，**CLB** (Configurable Logic Block) と呼ばれる書き換え可能な論理ブロックやメモリなどから構成される．各 CLB は，L 入力 1 出力の任意の論理関数を実現可能な**ルックアップテーブル** (LUT, Look Up Table) と，1 ビットの記憶素子である**フリップフロップ** (FF, Flip Flop) からなる．現在の FPGA では，LUT の入力数 L として，4 入力から 6 入力が一般的である．

この FPGA では，**ハードウェア記述言語** (HDL, Hardware Description Language) で記述した回路データにもとづいて，目的の機能を実現する回路を自由に構成可能である．このため，ASIC などの特定用途向けハードウェアと比較すると，開発期間とコストの面で利点をもつ．

1.4 関 連 研 究

現在の正規表現パターン照合手法のは多くは，**決定性有限オートマトン**(DFA, Deterministic Finite Automaton) や，**非決定性有限オートマトン**(NFA, Nondeterministic Finite Automaton) を用いたものである [2].

正規表現パターン照合回路を実現する方法として，与えられた正規表現を等価な DFA に変換し，その決定的な状態遷移を順序回路上で模倣する手法が考えられる．この手法は，最悪時に DFA の状態数が指数的に爆発するため，パターンによっては回路量が非現実的に大きくなる可能性がある．

一方，Sidhu らは，一般的な正規表現パターンを Thompson オートマトンと呼ばれる NFA に変換し，その NFA 上の状態遷移を FPGA 上で模倣するアーキテクチャを与えた [6]．また，川中らは，正規表現の部分クラスに対してパターン照合を実行するシストリック型のアーキテクチャを与えた [9]．Sourdis と Vassiliadis [7] らは，Sidhu らと同様の NFA にもとづく正規表現パターン照合のアーキテクチャを与えている．また，FPGA 上の正規表現パターン照合に関する従来の結果の概観を与えている．

ビット並列パターン照合手法は，Baeza-Yates と Gonnet [1] および，Wu と Manber [8] によって，独立に提案された文字列パターン照合手法であり，上記のクラスに対して，実用上もっとも高速なアルゴリズムの一つと言われている．

2. 準　　備

2.1 基本的な定義

アルファベットを Σ とする．Σ 上の全順序を $<_\Sigma$ で表わす．文字 $x \in \Sigma$ を，ビット幅 W のビット列 $x = b_1 \cdots b_W \in \{0,1\}^W$ と同一視する．ここに，$W \leqq \lceil \log 2|\Sigma| \rceil$ なる正整数である．Σ 上の文字列全体の集合を Σ^* で表し，文字列 $s \in \Sigma^*$ の長さを $|s|$ で表わす．長さが 0 の文字列を空語といい，ϵ で表わす．文字列 $x, y \in \Sigma^*$ の連結を，$x \cdot y$ で表わす．ただし，とくに混乱が生じない限りはこれを省略し，xy と書く．

2.2 線形正規表現パターン

Σ 上の**正規表現** (Regular Expression) r とその正規表現が表わす言語 $L(r)$ は，r_1, r_2 を正規表現としたとき，以下の式から帰納的に定義される [2]:

(i) 定数文字: $a \in \Sigma \cup \{\epsilon\}$．$L(a) = \{a\}$.
(ii) 連接: $r_1 \cdot r_2$．$L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$.
(iii) 選言: $(r_1|r_2)$．$L((r_1|r_2)) = L(r_1) \cup L(r_2)$.
(iv) 繰り返し: $(r_1)^*$．$L((r_1)^*) = L(r_1)^*$.

以下では，$\mathcal{RES}$ で正規表現全体のクラスを表わす．

正規表現に対して，次の略記法 (syntax sugar) を用いる．

- **オプション文字:** $r? \equiv (\varepsilon|r)$.
- **1 回以上の繰り返し:** $r^+ \equiv (r \cdot r^*)$.
- **文字種:** $[a_1 \cdots a_k] \equiv (a_1|\cdots|a_k)$.

ここで，$a_1, \ldots, a_k \in \Sigma$ である．

- **文字範囲:** $[a_1 \text{ - } a_k] \equiv (a_1|\cdots|a_k)$.

ここで，$a_1 <_\Sigma \cdots <_\Sigma a_k$ は Σ 上の全順序で，a_1 から a_k の範囲の文字全体である．

- **例外文字種:** $[\hat{}a_1 \cdots a_k] \equiv (b_1|\ldots|b_l)$.

ここで，$\{b_1, \ldots, b_l\} = \Sigma - \{a_1, \ldots, a_k\}$ である．また，$k = 1$ の場合をとくに，**例外文字**と呼ぶ．

- **ワイルドカード:** $. \equiv (a_1|\cdots|a_{|\Sigma|})$.

ここで，$a_1 <_\Sigma \cdots <_\Sigma a_{|\Sigma|}$ は Σ 上の全順序で，Σ に含まれる文字全体である．

- **可変長ワイルドカード:** $\Sigma^* \equiv (.)^*$.

［定義 1］ Σ 上の正規表現 $r = \alpha_1 \cdots \alpha_m$ $(m \geqq 0)$ が，**線形正規表現パターン**であるとは, 任意の $i = 1, \ldots, m$ に対して, α_i が下記の定義の (i)–(iii) のいずれかであることをいう．

(i) 定数文字: $a \in \Sigma$. $L(a) = \{a\}$.
(ii) 可変長ワイルドカード: Σ^*．$L(\Sigma^*) = \Sigma^*$.
(iii) 例外文字: $[\hat{}a]$. $L([\hat{}a]) = \Sigma - \{a\}$.

ここで，線形正規表現パターン $r = \alpha_1 \cdots \alpha_m$ の言語を，$L(r) = L(\alpha_1) \cdots L(\alpha_m)$ と定義する．

［補題 1］ 任意の線形正規表現パターン P は，$P = P_1 \cdots P_m (m \geqslant 0)$ と表わせる．ここに，各 $P_i (1 \leqq i \leqq m)$ は，次の (i)–(iii) のいずれかであり，**要素パターン**と呼ぶ．
(i)$P_i = a \in \Sigma$; (ii)$P_i = a(\Sigma^*)(a \in \Sigma)$; (iii)$P_i = [\hat{}a](a \in \Sigma)$

正規表現 α が，テキスト T 中に**位置 p で出現する**とは，長さ 0 以上の文字列 $u, v, w \in \Sigma^*$ が存在して，(i) $T = uvw$ かつ (ii) $v \in L(\alpha)$, (iii) $p = |uv| + 1$ となることをいう．すなわち，P は，パターン P の後端の出現位置である．(注1).

本論文で考察するパターン照合問題を次のように定義する．Σ 上の正規表現のクラス $\mathcal{C}$ に対する**正規表現パターン照合問題**とは，あらかじめ正規表現 P（パターンと呼ぶ）を受け取って前処理を行った後に，文字列 $T = t_1 t_2 \ldots t_n \in \Sigma^*$（テキストと呼ぶ）を受け取り，$T$ における P のすべての出現位置 i を求める問題である．

(注1)：通常，パターンの前端の出現位置 $q = |u| + 1$ を指すことが多いが，本稿では記述の簡略化のために，上記の定義を採用する．よく知られた方法として，後端の出現位置 p から後処理で逆方向に照合することで，前端の出現位置 q を見つけられる [4].

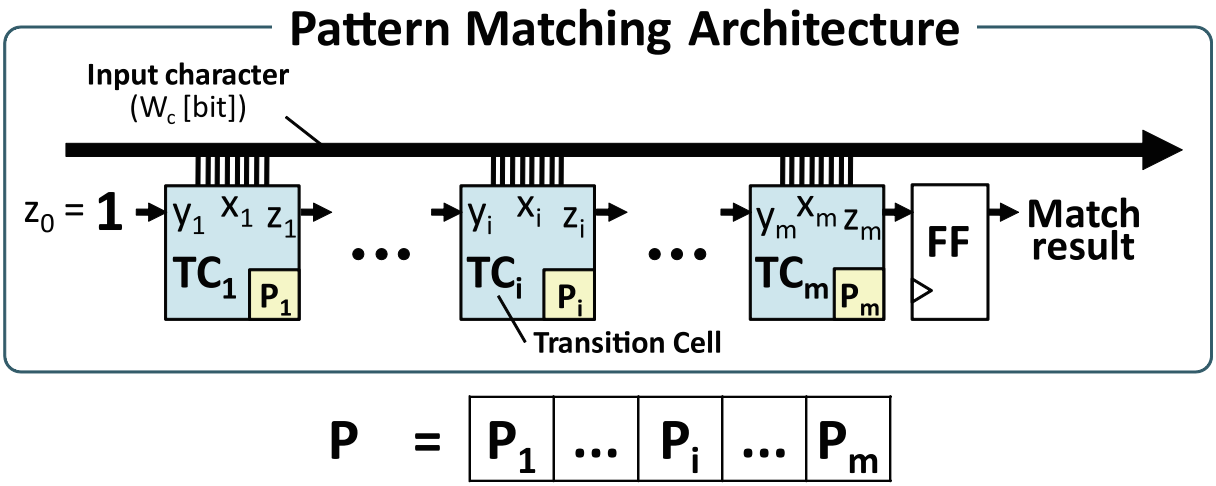

図 1　パターン照合ハードウェアアルゴリズム.

2.3　ビット並列手法

一般に，ビット並列手法とは，汎用 CPU 上でのビット演算と数値演算のレジスタ内並列性を利用して，計算を高速化する手法である.

Shift-And 手法は，ビット並列による文字列パターン照合手法のひとつである．この手法は，前処理時にパターン α を NFA に変換し，この NFA から状態遷移計算に用いるビットマスク M を計算する．パターン照合実行時には，各時点における状態をビット列としてレジスタに保持し，このビット列に対するビット演算と数値演算を使って状態遷移計算を高速に実行する．Shift-And 手法の例として，$\Sigma = \{A, B, C\}$ および，線形正規表現パターン $P = AB[AB]C$，入力テキスト $T = T[1] \cdots T[n]$ の場合を考える．P は，前処理時に 4 状態の NFA に変換される．この NFA から，各文字のビットマスク配列 `M[A] = 0101; M[B] = 0010; M[C]= 1000` が得られる．また，各時点における状態を保持するビット列を $D = d_4 d_3 d_2 d_1 \in \{0,1\}^4$ で，初期状態と受理状態のビットマスク `I = 0001; F = 1000` で表わす．これらを用いると，Shift-And 手法の疑似コードは，次のようになる.

```
D = 0000;
for i = 1,..., n {
  D = (D<<1 | I) & M[T[i]];
  if (D & F) then report i as the occurrence
} /* for */
```

ビット並列パターン照合手法の長所として，線形正規表現のクラスに対して，ハードウェアの並列性をソフトウェア上で自然に利用可能な点が挙げられる．一方で，その短所として NFA の状態数がレジスタのビット長以下に制限されることや，レジスタ内並列性以外のハードウェアの並列性の利用が難しいことが挙げられる.

3.　提案アーキテクチャ

3.1　パターン照合ハードウェアのアーキテクチャ

図 1 に，本稿で提案する線形正規表現パターンのクラスに対するパターン照合ハードウェアのアーキテクチャを示す．このアーキテクチャは，要素パターンで決まる**遷移セル**を複数個連結した構成をとる．提案方式では，与えられたパターンを静的に回路にコンパイルし，FPGA を再構成する．その後，入力テキストに対してパターン照合を行う.

任意の $i = 1, \ldots, m$ に対して，各遷移セル TC_i は，W_C ビット入力 x_i と 1 ビット入力 y_i，1 ビット出力 z_i の 3 つの入出力をもつ．x_i には，クロックに同期して入力テキストの各文字 $T_1, \ldots, T_n$ が入力される．y_i は，ひとつ前の遷移セル TC_{i-1} の出力 z_{i-1} と接続される．z_i は，ひとつ次の遷移セル TC_{i+1} の入力 y_{i+1} と接続される．ただし，左端の遷移セル TC_1 の入力 y_1 には，常に 1 を入力する．また，右端の遷移セル TC_m の出力 z_m には，パターン照合の結果を保持するフリップフロップ FF を接続する.

以下では，各要素パターンに対する遷移セルの詳細を示す．ここで，LUT の入力数 $L = 6$ および，文字のビット幅 $W = W_C = 8$ を仮定する.

3.2　文字コンパレータ

文字コンパレータ (CC, Character Comparator) は，文字の比較を行う論理回路であり，遷移セルの主要な構成要素である.

提案手法で用いる文字コンパレータには，パターン照合の実行前に，比較する文字 $a \in \Sigma$ が静的に与えられる．入力文字 $x = x_W \cdots x_1 \in \{0,1\}^W$ を文字 $a = a_W \cdots a_1 \in \{0,1\}^W$ と比較する文字コンパレータを $CC[a]$ で表わす．パターン照合実行時の $CC[a]$ は，$x \in \Sigma$ に対し，以下の W_C 入力 1 出力論理関数として動作する.

$$CC[a](x) = \begin{cases} 1 & x = a \text{ のとき}, \\ 0 & \text{それ以外のとき}. \end{cases}$$

文字コンパレータ $CC[a]$ は，実際の FPGA 上で次のように実装可能である．$W_C \leqq L$ の場合は，論理関数 $CC[a](x)$ を単一の LUT で直接実現可能である．$W_C > L$ の場合は，LUT の入力ビット幅が足りないため，そのままでは実現できない．

この場合，レジスタを用いたビット並列パターン照合手法と同様に [8]，入力文字 $x = x_8 \cdots x_1$ を幅 L ごとの $K = \lceil \frac{W_C}{L-1} \rceil$ 個のブロックに分割することで，K 個の LUT を用いて実現可能である．入力ビット幅 L の LUT には，入力ビット幅 $L-1$ の文字コンパレータと，その出力と 1 ビット入力の AND をとり出力する回路を格納できる．したがって，全部で K 個の LUT を用いることで W_C ビットの文字コンパレータが実現できる．

3.3 定数文字の遷移セル

定数文字は，$a \in \Sigma$ であるような要素パターンである．定数文字のみの連結からなるパターンは，いわゆる EXACT マッチパターンである [4]． 定数文字に対して，Shift-And 手法では，以下の数値演算で状態集合 $D = z_W \cdots z_1 \in \{0,1\}^W$ の更新を行う．ただし，ビットマスク M は，次のように計算される: $M[x][k] = 1 \iff x = P_k \ (x \in \Sigma, 1 \leqq k \leqq m)$.

```
D = ((D<<1 | I) & M[T[i]]);
```

この更新式から，状態集合の更新を行う論理回路を構成するために，i 番目の遷移セル TC_i は次の計算を行う．

- 左隣りの遷移セル TC_{i-1} のフリップフロップから，前回の状態 $p = z_{i-1} \in \{0,1\}$ を受け取る．
- 左端の遷移セル TC_1 は，前回の状態に関わりなく，ビット $z_0 = 1$ を受け取る．
- 入力文字 $x = T[i]$ を受け取り，マスク値 $r = M[x]$ の値を計算する．マスク値 r の値は，文字コンパレータ $CC[a](x)$ が計算を行う．
- $z_{i-1}\&r$ を遷移セル TC_i の出力側のフリップフロップに格納する．このフリップフロップの値は，クロックに同期して，z_i に出力される．

図 2 に，以上をもとに構成した定数文字のため遷移セルを示す．この遷移セルは，以下の論理関数を計算し，クロックに同期して結果を出力する．

$$TC[a](x,y,z) = \begin{cases} 1 & x = a \text{ かつ } y = 1 \text{ のとき} \\ 0 & \text{それ以外のとき} \end{cases}$$

文字ビット幅 W と LUT の入力ビット幅 L に対して，この遷移セルの CLB 数は，高々$N = \lceil \frac{W+1}{L-1} \rceil$ となる．

3.4 可変長ワイルドカードの遷移セル

可変長ワイルドカードは $a(\Sigma^*)(a \in \Sigma)$ であるような要素パターンである．この要素パターンでは，状態が一度アクティブになったら，以後，その状態はアクティブであり続ける．このアイディアにもとづき，Shift-And 手法では，以下の数値演算で状態集合 $D = z_W \cdots z_1 \in \{0,1\}^W$ の更新を行う．これは，ビットマスク M が，Σ^* の直前の定数文字 a から計算する点を除いて，定数文字の場合と同じである．

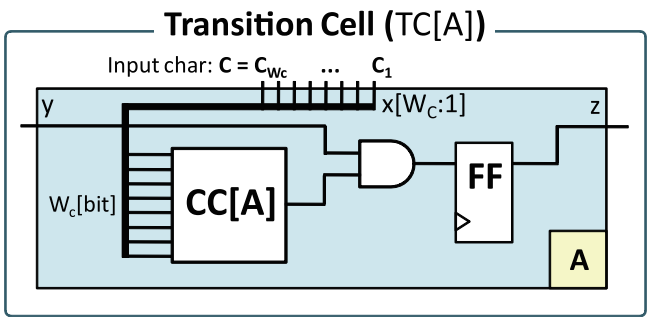

図 2　定数文字の遷移セル．

```
D = D | ((D<<1 | I) & M[T[i]]);
```

図 3 に，以上をもとに構成した可変長ワイルドカードのため遷移セルを示す．この遷移セルでは，FF の出力をループバックさせて，a による EXACT マッチの結果と OR をとり，再度 FF に戻すことで，上記の状態集合の更新を実現している．

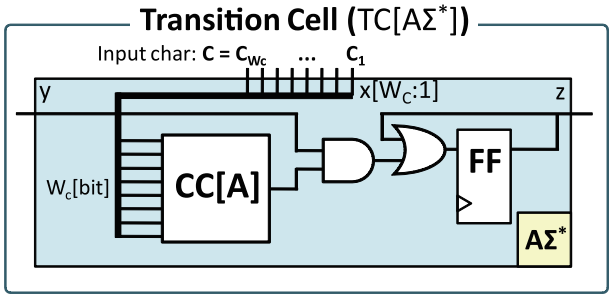

図 3　可変長ワイルドカードの遷移セル．

この遷移セルの CLB 数は，高々$N = \lceil \frac{W+2}{W-1} \rceil$ となる．

3.5 例外文字の遷移セル

例外文字は，$\lceil a](a \in \Sigma)$ であるような要素パターンである．この要素パターンでは，ビットマスク M の構成方法が異なるだけで，状態集合の更新式は，定数文字の EXACT マッチの場合と同一である．例外文字では，ビットマスク $M[a][k]$ が，定数文字の EXACT マッチの場合の否定になっている．

この議論から例外文字のための遷移セルは，図 4 のようになる．この遷移セルは，以下の論理関数を計算する．

$$TC[a](x,y,z) = \begin{cases} 1 & x \neq a \text{ かつ } y = 1 \text{ のとき} \\ 0 & \text{それ以外のとき} \end{cases}$$

この遷移セルの CLB 数は，高々$N = \lceil \frac{W+1}{W-1} \rceil$ となる．

3.6 複数パターン照合

正規表現パターン照合の代表的な応用であるネットワーク侵入検知システムなどでは，1000 個以上の正規表現パターンを同時に照合する．そのため，複数パターン照合を並列かつ効率的に扱えることは，非常に重要である．

提案アーキテクチャの複数パターン照合への拡張は，単一パ

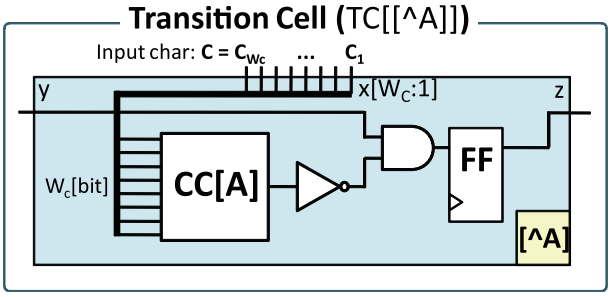

図 4　例外文字の遷移セル．

ターン照合を行う上記の回路を並列にならべることにより容易に実現できる．複数パターン照合では，パターンの出現位置および，どのパターンが出現したのかを出力する必要がある．

また，提案アーキテクチャを複数パターン，あるいは，非常に長いパターンに適用する場合，文字コンパレータを共有化することで回路サイズを大幅に縮小できる [7]．この方法を適用することで，回路全体で必要な文字コンパレータは，高々Σ 個に抑えられる．

4. 提案アークテクチャの解析

4.1 計 算 時 間

線形正規表現パターンを $P = P_1 \cdots P_m$ とし，入力テキストを $T = T_1 \cdots T_n$ とする．

提案アーキテクチャの遷移セル TC_i $(1 \leqq i \leqq m)$ は，各クロック CLK_k $(1 \leqq k \leqq n)$ において，入力文字 T_k を処理し，その結果を CLK_{k+1} と同期して出力する．したがって，提案アーキテクチャは，入力テキスト T に対するパターン照合結果を，CLK_{n+1} と同期して出力する．すなわち，BPRH は，入力テキスト長 $|T| = n$ にのみ依存し，パターン長 $|P| = m$ に依存しない．以上のことから，次の定理が示せる．

［定理 1］ 提案アーキテクチャの時間計算量は，$O(n)$ 時間である．

これは，ソフトウェアによる正規表現の素朴な照合アルゴリズムの時間計算量 $O(mn)$ と比較して，m 倍高速である．

4.2 回　路　量

Sidhu らの手法 [6] で例外文字 $[\hat{}a]$ および，可変長ワイルドカード $\Sigma^* = (.)^*$ の照合を行う回路を実現する場合，それぞれ，$(b_1|\ldots|b_{|\Sigma|-1})$ と $(c_1|\ldots|c_{|\Sigma|})^*$ と見なす必要がある．ここで，$\{b_1, \ldots, b_{|\Sigma|-1}\} = \Sigma - \{a\}$ である．また，$c_1 <_\Sigma \cdots <_\Sigma c_{|\Sigma|}$ は，Σ 上の全順序で，c_1 から $c_{|\Sigma|}$ の範囲の文字全体である．しかし，例外文字および，可変長ワイルドカードの照合を行う回路を上記の方法で実現する場合，$O(|\Sigma|)$ の回路量が必要になり，アルファベットサイズが大きいと，回路量が増大するという問題がある．

提案アーキテクチャでは，Sidhu ら [6] の手法において回路量がアルファベットサイズ $|\Sigma|$ に比例するこれら二つのパターンを特別に扱うことで，この問題を解決している．実際，提案アーキテクチャでは，例外文字遷移セルを高々$N = \lceil \frac{W+1}{L-1} \rceil$ の CLB で，可変長ワイルドカード遷移セルを高々$N = \lceil \frac{W+2}{L-1} \rceil$ の CLB で構成する

以上の議論から，次の定理が示される．

［定理 2］ 任意の線形正規表現パターン $P = P_1 \cdots P_m$ に対して，提案アーキテクチャは，高々$N = m\lceil \frac{W+2}{L-1} \rceil$ 個の CLB で構成される．ここで，W は文字ビット幅であり，L は CLB を構成する LUT の入力ビット幅である．

これらは，入力テキストの文字ビット幅 W と CLB を構成する LUT の入力ビット幅 L のみから決まる．

5. おわりに

本稿では，ビット並列パターン照合手法 [1], [8] にもとづき，FPGA による線形正規表現パターン照合アーキテクチャを提案した．このアーキテクチャは，長さ m の線形正規表現パターン P と長さ n のテキスト T に対し，計算時間 $O(n)$ でパターン照合を実行する．また，この回路は，高々$N = m\lceil \frac{W+2}{L-1} \rceil$ 個の CLB から構成できる．これは，Sidhu らが提案した回路 [6] と比較して，非常に少ない回路サイズである．今後の課題としては，FPGA 上での実装および，実験などが挙げられる．また，関係ストリーム上の述語を含んだ時系列パターンへの拡張も興味深い課題である [3], [5].

文　献

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

[2] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley Publishing Company, 1979.

[3] T. Kida, T. Saito, and H. Arimura. Flexible Framework for Time-Series Pattern Matching over Multi-Dimension Data Stream. In *The 1st International Workshop on Algorithms for Large-Scale Information Processing in Knowledge Discovery (ALSIP 2008)*, 2008.

[4] G. Navarro and M. Raffinot. *Flexible Pattern Matching in Strings: Practical On-Line Search Algorithms for Texts and Biological Sequences*. Cambridge University Press, 2002.

[5] T. Saito, T. Kida, and H. Arimura. An Efficient Algorithm for Complex Pattern Matching over Continuous Data Streams Based on Bit-Parallel Method. In *Databases for Next Generation Researchers, 2007. SWOD 2007. IEEE International Workshop on*, pp. 13–18, 2007.

[6] R. Sidhu and V. K. Prasanna. Fast Regular Expression Matching Using FPGAs. In *The 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01)*, pp. 227–238, 2001.

[7] I. Sourdis, J. Bispo, J. M. P. Cardoso, and S. Vassiliadis. Regular Expression Matching in Reconfigurable Hardware. *Journal of Signal Processing Systems*, Vol. 51, No. 1, pp. 99–121, 2008.

[8] S. Wu and U. Manber. Fast text searching: allowing errors. *Communications of the ACM*, Vol. 35, No. 10, pp. 83–91, 1992.

[9] 川中洋祐, 若林真一, 永山忍. パターンを実行時に設定可能な正規表現ストリングマッチングマシンと FPGA による実装. 情報処理学会研究報告, 2008-SLDM-133, pp. 61–66, 2008.