広島市立大学審査博士学位論文

ネットワーク侵入検知のための
パターン非依存正規表現マッチングハードウェア
に関する研究

2014年3月

広島市立大学大学院 情報科学研究科
博士後期課程 情報科学専攻

若葉 陽一

概要

インターネットの普及に伴い，コンピュータウィルス等の脅威も増加している．その対策の 1 つとしてネットワーク侵入検知システム（NIDS）がある．NIDS はネットワークセキュリティの向上のため，ネットワーク上を流れる通信パケットを監視するシステムである．NIDS はコンピュータウィルスを正規表現で定義したパターン（ウィルスパターン）と通信パケットに対する正規表現マッチングを行うことで，ウィルスの検知を行う．ここで，正規表現とは文字列の集合を 1 つの文字列で表現する方法であり，正規表現マッチングとは，正規表現を用いて記述されたパターンと一致する部分文字列をテキスト中から検索する操作である．

現在のネットワーク環境下における NIDS の正規表現マッチングにおいて，以下の要件を全て満たすことが望ましい．(要件 1）高速ネットワーク (ギガビットイーサネット) 上でリアルタイムなウィルス検知ができる，(要件 2）どんなウィルスパターンでも扱うことができる，(要件 3）多数のウィルスパターンをコンパクトな回路で扱ことができる，(要件 4）新しいウィルスに素早く対応するため，パターン更新を瞬時に行うことができる．

ソフトウェアによる正規表現マッチングでは，(要件 1）を満たす事が難しいため，様々な正規表現マッチングハードウェアが提案されている．既存のハードウェアは大きく分けてパターン依存型とパターン非依存型に分類される．パターン依存型は与えられたパターンに特化した回路構成を持ち，高速かつコンパクトな回路規模で正規表現マッチングを実現できる．しかしパターンが更新される度に回路を再設計する必要があり，パターン更新に時間がかかる．パターン非依存型は任意のパターンに対応できる回路構成を持ち，パターンの更新を瞬時に行えるという利点を持つ．しかし，任意の正規表現を扱うマッチングハードウェアは非現実的な回路規模と性能となるため，扱う正規表現を制限したマッチングハードウェアが提案されている．

本論文では，上述の全ての要件を満たす正規表現マッチングハードウェアを実現するために，(1) シストリックアルゴリズムと非決定性有限オートマトン（NFA）を組み合せた新しいマッチングハードウェア，(2)FPGA の部分再構成機能に基づく回路面積の削減手法，(3) 拡張正規表現に対するマッチング手法，を提案する．本論文の成果により，NIDS においてパフォーマンスの低下を招くことなくウィルス検知が可能となり，ネットワークのセキュリティが向上することが期待される．

本論文では4章で，任意の正規表現を扱うことができるパターン非依存正規表現マッチングハードウェアを提案する．既存のパターン非依存正規表現マッチングハードウェアのほとんどは，回路構造の制約からマッチング可能な正規表現が制限されている．そのため，パターン更新時に検知可能なウィルスが制限される．また任意の正規表現を扱うことができる既存のパターン非依存正規表現マッチングハードウェアは，複雑な回路構造を持つため実用的な回路規模ではない．そこで，本論文では単純な回路構造を持つシストリックアルゴリズムに基づくハードウェアと任意の正規表現を扱うことができるNFAに基づくハードウェアの利点を活かしながら組み合わせることで，任意の正規表現パターンに対応可能であり，しかも回路規模がコンパクトなパターン非依存正規表現マッチングハードウェアを提案する．実験的評価から，任意の正規表現を扱うことができる既存のパターン非依存ハードウェアと比べ実用的な回路規模でマッチングを実現できることを示す．

4章で提案するハードウェアは任意の正規表現を扱うことができ，回路規模は実用的ではあるが，やはり大きい．そこで5章において，任意の正規表現を扱うことができかつ瞬時にパターン更新が可能であるという特長を保ちつつ，回路面積を削減する手法を提案する．提案手法では，FPGA の部分再構成機能を用いて与えられた正規表現パターンに合わせた回路構成を瞬時に生成する．実験的評価から，提案手法は4章で示すハードウェアと比べ，63 %の回路面積を削減できる事を示す．

NIDS のパターン記述においては，ユニオン等の基本演算子だけでなく量指定子等の様々な演算子を導入した拡張正規表現もよく用いられる．拡張正規表現を用いると，パターンをより簡潔に記述することができるだけでなく，正規表現では表せないパターンを記述することもできる．そこで，6章において，拡張正規表現に対するハードウェアマッチング手法を提案する．拡張正規表現の中で，量指定子やクラス文字に対するハードウェアマッチング手法は既に提案されているが，先読み演算や後方参照に対するマッチング手法は提案されていない．そこで，これらの演算子に対するハードウェアマッチング手法を提案する．実験的評価から，これらの演算子をハードウェアで効率的に扱える事を示す．

A Study on Pattern-Independent Regular Expression Matching Hardware for Network Intrusion Detection

Abstract

A threat of computer viruses has been increasing with the diffusion of the Internet. One of the methods to guard against the threat is introduction of network intrusion detection systems (NIDSs). NIDSs are systems for monitoring packet payloads on a network to improve network security. NIDSs perform regular expression matching for packet payloads and regular expression patterns defining computer viruses (virus patterns) to detect malicious traffic such as computer viruses and worms. Regular expression (RE) is a method to represent a set of strings as a string in a compact manner. Regular expression matching (REM) is an operation to find substrings which match with a pattern described using an RE in a text.

For NIDSs in the current network environment, it is desirable for REM to satisfy all of the following requirements. (Requirement 1) Real time virus detection on a fast speed network (e.g., Gigabit Ethernet), (Requirement 2) Any RE pattern can be handled (i.e., any virus which can be described with a RE can be detected), (Requirement 3) Many virus patterns can be handled with compact circuit size, (Requirement 4) An RE pattern can be updated quickly to detect new virus immediately.

Many REM hardware engines have been proposed, since software implementation for REM is difficult to satisfy (Requirement 1). The existing REM hardware engines can be classified into two types: pattern dependent hardware engines and pattern independent hardware engines. The pattern dependent hardware engine has a dedicated circuit structure for a given pattern. Therefore, it can realize fast REM with compact circuit size. However, it needs to redesign the circuit whenever a pattern is updated. As a result, it takes a long time to update a pattern. On the other hand, the pattern independent hardware engine has a circuit structure which can handle any pattern. Therefore, the pattern independent hardware engine can update a pattern immediately. However, an existing engine which can handle any RE has impractical circuit size and is impractical performance. Another existing engine handles restricted RE classes.

To realize a REM hardware engine for NIDSs which can satisfy all requirements mentioned above, this dissertation proposes the following (1), (2) and (3); (1) A new REM

hardware engine combining systolic algorithm and Non-Deterministic Finite Automaton (NFA), (2) The method based on partial reconfiguration of FPGA to reduce the circuit area, and (3) A matching method for extended regular expressions. This dissertation contributes for realizing an NIDS which can detect computer viruses without causing performance degradation. As a result, the improvement of network security can be expected.

First, Chapter 4 proposes a pattern independent REM hardware engine which can handle any RE. In most existing pattern independent REM hardware engines, the class of REs to be used as a pattern is restricted by the constraint of each circuit structure. As a result, the class of viruses which can be detected is restricted when updating patterns. The existing pattern independent REM hardware engine which can handle any RE has a complicated circuit structure and the circuit size is impractical. Thus, this dissertation proposes a compact pattern independent REM hardware engine, which can handle any RE, by combining the systolic based and the NFA based hardware engines, keeping their respective advantages. Experimental results show that the circuit size of the proposed hardware engine is extremely compact compared to that of the existing hardware engine.

The REM hardware engine proposed in Chapter 4 can handle any RE with practical circuit size. However, the circuit size is still large. Therefore, Chapter 5 proposes a method to reduce the circuit size keeping the advantages that any RE can be handled and a pattern can be updated immediately. The proposed method produces a dedicated circuit for a given RE pattern quickly using partial reconfiguration of FPGA. Experimental results show that the proposed method can reduce 63 % circuit size compared to the REM hardware engine proposed in Chapter 4.

For pattern description in NIDSs, extended REs including not only basic operators such as union but also various operators such as quantifier are used. Using extended REs, a pattern can be compactly described and a pattern which cannot be represented by normal REs can be described. Therefore, Chapter 6 proposes a hardware matching method for extended REs. In extended REs, hardware matching methods for quantifier and character class have been proposed. However, hardware matching methods for look ahead assertion and back reference have not been proposed yet. Thus, this dissertation proposes matching methods for those operators. Experimental results show that the proposed methods can handle those operators efficiently.

目次

図 目 次

表 目 次

第1章　序論

1.1　研究背景と目的

近年，インターネットは社会インフラを支える重要な役割を担っている．インターネットの歴史は古く，1969 年に米国防総省の高等研究計画局が導入した Advanced Research Projects Agency Network（ARPANET）[1] が起源と言われている．1995 年頃，検索エンジン Yahoo や OS Microsoft Windows95 等の登場により，インターネットは急速に普及し，現在，日本人口の約 8 割がインターネットを利用しているという結果が総務省の通信利用動向調査によって示されている [2].

インターネットの普及は現代社会に大きな利益をもたらした反面，コンピュータウィルスや不正アクセス等によるサイバー攻撃の被害も増大している．サイバー攻撃によって個人や企業等が機密データの詐取や破壊，改ざん等の被害を受け，その被害額は 2013 年現在，全世界で 1,130 億ドルにも達していることがアンチウィルスソフトウェア会社 Symantec の調査によって示されている [10]．そのため，サイバー攻撃対策は必要不可欠となっている．

サイバー攻撃対策の 1 つとして，ネットワーク侵入検知システム（NIDS）の導入が挙げられる．NIDS はネットワーク上を流れる通信パケットを監視することで，ネットワークに対する不正侵入や攻撃を検知し，ネットワークセキュリティを向上させるシステムである [20]．日本ネットワークセキュリティ協会（JNSA）の 2012 年の調査報告 [21] によると，ネットワーク脅威対策製品の 24% が NIDS であり，NIDS の設置は一般的になってきていることを示している．NIDS はコンピュータウィルスを定義したパターン（ウィルスパターン）と通信パケットに対するストリングマッチングを行うことで，ウィルスの検知を行っている．

ストリングマッチングとは与えられた文字列（パターン）と一致する部分文字列を入力系列（テキスト）から検索する操作である [3,4]．NIDS においては、ウィルスパターンは正規表現を用いて記述されることが一般的である．正規表現は，文字列の集合を文字列に

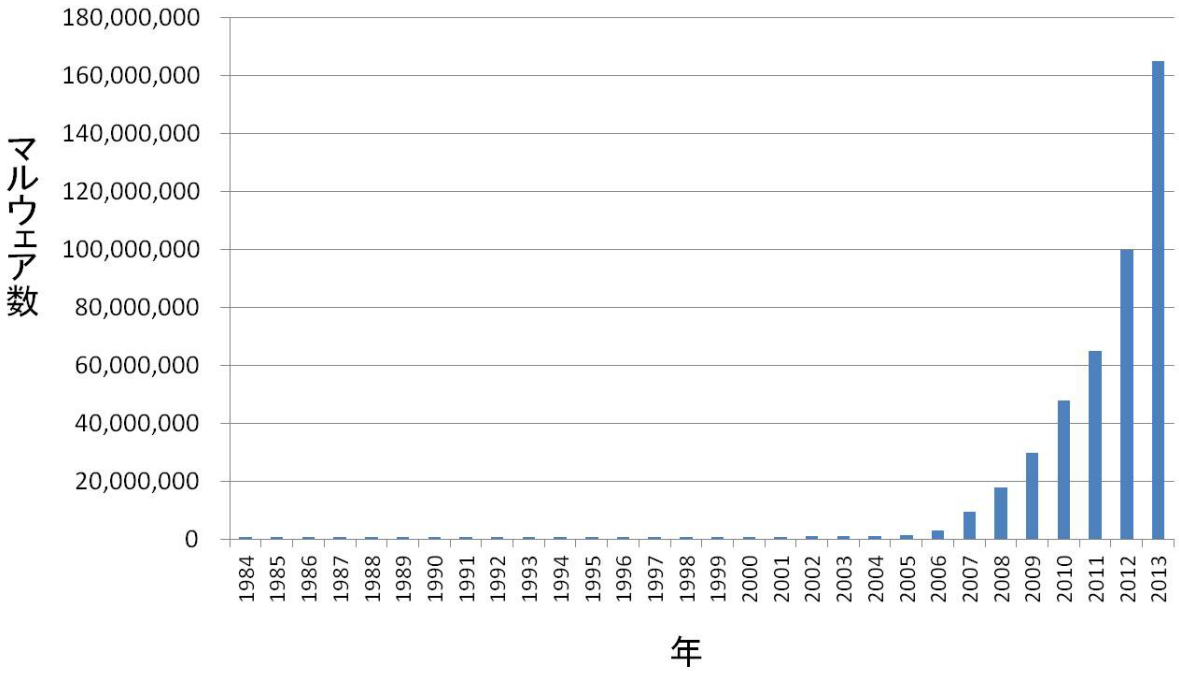


図 1.1: 1984 年から 2013 年までのマルウェア数の統計情報 [9]

対する演算を含む 1 つの文字列でコンパクトに表現する方法であり [19]，ストリングマッチングにおいて，パターンに正規表現を用いる場合，正規表現マッチングと呼ぶ．ストリングマッチングは，データベースやバイオインフォマティクス，NIDS など様々な分野で応用されている．ストリングマッチングは，情報科学において主要な問題の 1 つであり，これまでに多くの研究が行われてきている [5]．特に，NIDS を前提とした正規表現マッチングに関してはこれまで多くの研究がある [8, 44–47, 49–68]．

NIDS において，Snort [26] や Bro [39] のように，正規表現マッチングは従来ソフトウェアによって実現されることが一般的であった．しかし，近年，NIDS を取り巻く環境が変化している．第一に，ネットワークの通信速度の向上が挙げられる．現在では 1Gbps のインターネット回線が低コストで提供されるようになり，一般家庭においても，1Gbps のインターネット回線が普及しつつある．また現在，100Gbps イーサネットが標準化されており [6]，近い将来，10Gbps や 100Gbps のインターネット回線が普及していくと考えられる．第二に，ネットワークトラフィックの増加が挙げられる．[7] によると，2012 年の世界のインターネットユーザー数は 23 億人と推定されており，2017 年までには世界のインターネットユーザー数は約 36 億人になると予想されている．また，数年前はデータサイズの小さいテキストや画像等の通信が多かったが，近年ではデータサイズの大きい動画等が頻繁に通信されるようになってきている．2012 年から 2017 年の間に世界のイン

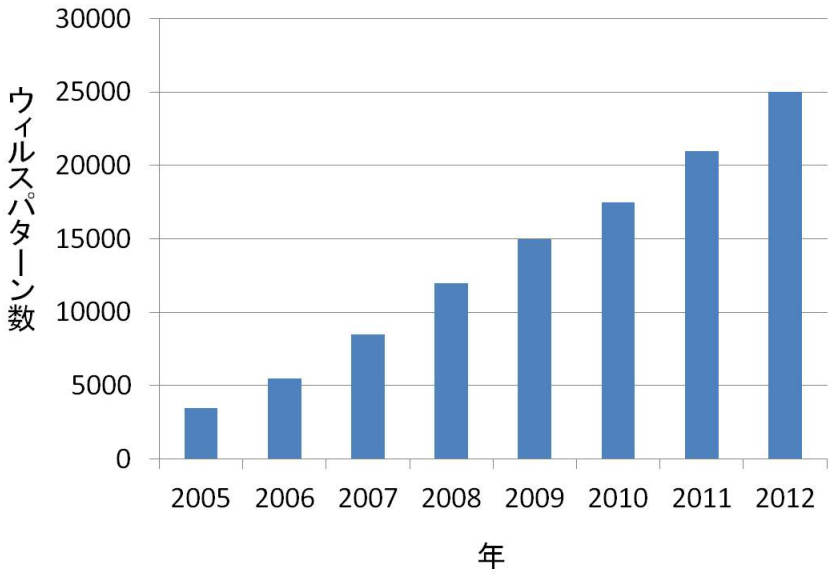


図 1.2: Snort における 2005 年から 2012 年までのウィルスパターン数の傾向 [8]

ターネットプロトコル（IP）トラフィック量は 3 倍に増加すると予想されている [7]．第三に，不正侵入や攻撃の種類の増加や巧妙化が挙げられる．不正侵入や攻撃種類の増加に伴い（図 1.1），ウィルスパターンの種類とウィルスパターンの更新頻度が増加している．[8] による Snort における 2005 年から 2012 年までのウィルスパターン数の傾向調査によると（図 1.2），ウィルスパターン数は 8 年間で約 7 倍に増加している．AV-TEST 社による Symantec や Trend Micro [11] 等の有名なアンチウィルスソフトウェア会社の製品 15 種類における 1 日のパターン更新頻度に関する調査によると，製品によっては，200 回以上のパターン更新が行われ，全ての製品で数回のパターン更新が行われていることが示されている [9]．また，当初，ウィルスパターンは文字列で記述されることが多かったが，攻撃の巧妙化によって，近年では正規表現や拡張正規表現（Perl Compatible Regular Expression, PCRE）[28] を用いて記述されることが多くなってきている．Snort において，正規表現や拡張正規表現を用いたパターンは 2009 年で 390 種類であったのに対し，2012 年では 3966 種類まで増加している．

このような状況下において，ソフトウェアによる正規表現マッチングではリアルタイムなウィルス検知は困難になってきた．実際，Snort における PCRE ライブラリを用いた正規表現マッチングでは，0.9Kbps から 10.9Mbps の処理性能であることが示されている [8]．そのため，正規表現マッチングのハードウェア実装に関する研究が盛んに行われている．NIDS のための正規表現マッチングハードウェアにおいて，高速ネットワーク (ギガビットイーサネット) 上でのリアルタイムなウィルス検知ができるだけでなく，以下の要件を

全て満たすことが望ましい.（要件 1）NIDS 導入の低コスト化のため，多数のウィルスパターンをコンパクトな回路で扱ことができる,（要件 2）正規表現で記述できるどんなウィルスでも検知するため，どんな正規表現パターンでも扱うことができる,（要件 3）新しいウィルスに素早く対応するため，頻繁に更新されるウィルスパターンを瞬時に更新できる.

既存の正規表現マッチングハードウェアは大きく分けてパターン依存型 [52–62] とパターン非依存型 [63–68] に分類される.

パターン依存型は与えられた正規表現パターンを非決定性有限オートマトン（NFA）に変換し，その NFA を FPGA などの再構成デバイスを用いて回路として実現する．パターンに特化した回路構成であるため，高速かつコンパクトな回路規模で正規表現マッチングを実現できる．しかしパターンが更新されるたびに，マッチングハードウェアのハードウェア記述言語（HDL）ファイルの生成と論理合成，配置配線，FPGA コンフィギュレーションファイルの生成，ダウンロードといった設計手順を踏む必要があり，パターン更新に一定時間必要となる.

一方，パターン非依存型は任意のパターンに対してマッチングを実現できるように回路を構成したマッチングハードウェアであり，マッチング実行時にパターン設定が可能であり，パターン更新時に回路の再構成が不要という利点がある．しかし，任意の正規表現を扱う既存のパターン非依存マッチングハードウェアは回路規模や処理性能が非現実的であったため，扱う正規表現クラスを制限したパターン非依存マッチングハードウェアが提案されている.

本論文の目的は，上述の全ての要件を満たす正規表現マッチングハードウェアを実現し，NIDS においてパフォーマンスの低下を伴わないウィルス検知を可能とすることで，ネットワークのセキュリティ向上に貢献することである．そのため，本論文では，パターン非依存正規表現マッチングハードウェアに焦点を当て，(1) シストリックアルゴリズムと非決定性有限オートマトン（NFA）を組み合せた新しいマッチングハードウェア，(2) FPGA の部分再構成機能に基づく回路面積の削減手法，の 2 つの手法を提案することで，上述の全ての要件を満たす正規表現マッチングハードウェアを実現する．また，本論文では，巧妙化するウィルスに対応可能なマッチングハードウェアを実現するため，(3) 拡張正規表現に対するマッチング手法，を提案する.

1.2　論文の概要と構成

本論文ではまず，任意の正規表現を扱うことができるパターン非依存正規表現マッチングハードウェアを提案する．既存のパターン非依存正規表現マッチングハードウェア [63–67] は，コンパクトな回路でマッチングを行うことができるが，ウィルスパターンとしては正規表現の部分クラスしか扱うことができない．近年，NIDS において，ウィルスの種類の増加によりウィルスパターンが複雑化しており，[63–67] のハードウェアでは扱えない正規表現で記述されるウィルスパターンも増加している．そのため，ウィルスを検知できない可能性が増えるため，セキュリティ面で問題となる．この課題を解決するため，[68] において任意の正規表現を扱うことができる NFA に基づくマッチングハードウェア（DPA 手法）が提案されているが，実用的でない回路規模になってしまうという問題点があった．そこで本論文では，任意の正規表現を扱うことができるが回路構造が複雑な DPA 手法 [68] と回路構造は単純であるが正規表現の部分クラスしか扱うことができないシストリックアルゴリズムに基づくマッチングハードウェア [66] に注目し，2 つのハードウェアを組み合わせた新しいマッチングハードウェアを提案する．提案ハードウェアはシストリックアルゴリズムに基づく単純文字列の比較機能と NFA に基づく状態遷移機能の組み合わせにより，文字列に対して状態遷移する有限状態オートマトンをシミュレートする．これにより，任意の正規表現パターンに対応でき，かつ DPA 手法より回路構造がコンパクトなパターン非依存正規表現マッチングハードウェアを実現する．また，提案ハードウェアを FPGA 上に実装し，実験的評価により提案ハードウェアの有効性を示す．

上記で示した提案ハードウェア（提案ハードウェア 1）は実用的な回路規模で任意の正規表現を扱うことができるが，ハードウェアコストの面から回路規模はできる限り小さい方が望ましい．そこで，本論文では次に，FPGA の部分再構成機能を用いた回路規模削減手法を提案する．提案手法は，正規表現の異なる部分クラスを扱う部分回路を複数用意しておき，与えられた正規表現パターンに適したコンパクトな回路構成をこれらの部分回路の組み合わせにより自動生成する．生成された回路構成は，パターン更新に伴って変更が必要となる箇所のみが部分再構成される．提案手法は，全てのパターンに対応可能になるように設計された提案ハードウェア 1 に比べ，コンパクトな回路を生成できる．またパターン依存型と違いパターンが更新された際に，時間のかかる回路の再設計を必要とせず，適切な部分回路の選択と，部分再構成のみが必要であるため，高速なパターンの更新が可能である．実験的評価から，提案手法は提案ハードウェア 1 と比べ，わずかにパターン更新時間を増やすだけで，63 %の回路面積を削減できることを示す．

本論文では最後に，NIDS のパターン記述によく用いられる拡張正規表現に対するハードウェアマッチング手法を提案する．拡張正規表現を用いると，パターンをより簡潔に記述することが可能となるだけでなく，正規表現では表せないパターンを記述することもできる．拡張正規表現の中で，量指定子やクラス文字に対して様々なハードウェアマッチング手法 [53, 60, 61, 64, 67] が提案されている．そこで本論文では，既存手法が知られていない先読み演算 [71] と後方参照 [71, 72] に対するマッチング手法を提案する．先読み演算に対する提案手法では，先読みを扱うために，正規表現マッチングハードウェアに前処理回路を導入する．前処理回路は文字列の末尾から文字列の先頭へマッチングを行う．また高スループットを達成するために，スタックメモリを用いた新しいバッファ機構を提案する．後方参照に対する提案手法では，後方参照を含むパターンに対して，ソフトウェアとハードウェアの両方を用いてマッチングを行う．提案マッチング手法では，パケットが入力されると，まずハードウェアによって前処理を行い，後方参照を含むパターンと一致する可能性のあるパケットと可能性のないパケットを高速に分類する．一致する可能性のないパケットはそのまま通過させ，一致する可能性のあるパケットはソフトウェアで深さ優先探索とバックトラックによるマッチングを行い，後方参照を含むパターンと正確に一致するか調べる．

本論文の章構成は以下のとおりである．2 章では準備として，ストリングマッチングと正規表現，有限オートマトン，FPGA，NIDS について説明する．3 章では，関連研究を概観する．4 章では，任意の正規表現を扱うことができるシストリックアルゴリズムと NFA に基づくパターン非依存正規表現マッチングハードウェアを提案する．5 章では，FPGA の部分再構成機能に基づく高速パターン更新可能な正規表現マッチングハードウェアを提案する．6 章では，拡張正規表現に対するハードウェアマッチング手法を提案する．最後に 7 章で，本論文をまとめる．

第2章　準備

本章では，本論文の議論の準備として，ストリングマッチング，正規表現，正規表現マッチング，有限オートマトン，FPGA，ネットワーク侵入検知システムについて説明する．

2.1　ストリングマッチング

ストリングマッチングとは文字列に関する基本操作の 1 つである．ストリングマッチングは長さ n の文字列（テキスト）と長さ m の文字列（パターン）が与えられると（m≤n），テキスト中にパターンと一致する部分文字列の現れる箇所があればその 1 つを示し，なければないと答える操作である [3, 4].

ストリングマッチングに対して様々なアルゴリズムが研究されている．力まかせ法と呼ばれる単純なアルゴリズムは，テキストの先頭から順に位置を 1 つずつ右にずらしながらパターンと一致する文字列が現れるか調べる方法であり，最大計算量は O(mn) である．ラビン-カープ法（RK 法）[32] は力まかせ法にハッシュ法を加えた手法である．最大計算量は O(mn) であるが，実用的には，ほとんど確実に O(n+m) の計算量で解くことができる．クヌース・モーリス・プラット法（KMP 法）[31] やボイヤー-ムーア法（BM 法）[33] は力まかせ法のように常にテキストの位置を 1 つずつずらすのではなく，ずらす距離を予め計算しておく方法である．これらの手法の最大計算量は O(m+n) となる．

上記は単一のパターンに対するストリングマッチングアルゴリズムである．複数のパターンに対するストリングマッチングアルゴリズムも提案されており，エイホ-コラッシック法（AC 法）[13] と呼ばれている．AC 法は与えられたパターン集合からマシン AC と呼ばれる一種の有限オートマトンを構成し，この有限オートマトンにテキストを入力として与えることによりストリングマッチングを実行する．AC 法は複数のパターンに対するストリングマッチングを O(n) で解くことができる高速なアルゴリズムである．また Wu-Manber 法 [14] も複数のパターンに対するマッチングアルゴリズムである．[14] は BM 法を複数のパターンのマッチングに対しても適用する手法を提案している．Wu-Manber

法は AC 法よりもメモリ効率がよい [15].

ストリングマッチングアルゴリズムにおいて，パターンの出現箇所をすべて見つけるように容易に拡張できるものも多い．これは， アルゴリズムの多くはテキストを先頭から順に調べるため，発見したパターンと一致する文字列 (match) の先頭位置の直後から同じアルゴリズムを適用することで次の match を見つけることができるからである．

ストリングマッチングの一種である近似ストリングマッチング [16] も広く研究されている．近似ストリングマッチングはテキスト内からパターンと類似する部分文字列を見つける操作である．文字列間の類似度を表す編集距離等の指標を用いてマッチングが行われる [17]．近似ストリングマッチングに対するアルゴリズムとして，動的計画法 (DP 法) を使って編集距離を計算する方法が有名であり，O(mn) の編集距離計算で問題を解くことができる．

ストリングマッチングはパターンをキーとする一種の検索問題である．しかし，テキスト中からパターンを探す際，テキストは文字の大小順などにソートされていないため，2 分探索法などの探索アルゴリズムをそのまま適用することはできない．ストリングマッチングはそれ自身が独立した興味深い問題である．

また，ストリングマッチングの応用としては，データベースにおける検索やバイオインフォマティクスにおける DNA 配列の解析，ネットワーク侵入検知システムにおけるコンピュータウィルスの検知などが挙げられ，情報科学において主要な問題の 1 つと言える．

2.2　正規表現と正規表現マッチング

正規表現とは，文字列の集合を 1 つの文字列で表現する方法，およびそのような文字列自体を言う [19]．正規表現が表す文字列の集合を正規言語と言う．以下に正規表現の定義を示す.

[定義 1] 記号の有限集合（アルファベットという）を Σ=$\{a_1, a_2, \cdots, a_s\}$ とする．Σ 上の正規表現 R は以下のように定義される．

1. ϕ は正規言語 ϕ（空集合）を表す正規表現である．
2. 空語 ε は正規言語 $\{\varepsilon\}$ を表す正規表現である．
3. 記号 a_i は正規言語 $\{a_i\}$(ただし，$a_i \in \Sigma$）を表す正規表現である．
4. R_1 と R_2 をおのおの Σ 上の正規言語 $\mathrm{L(R_1)}$, $\mathrm{L(R_2)}$ を表す正規表現としたとき，

(a) ユニオン $(R_1|R_2)$ は正規言語 $\mathrm{L(R_1)}\cup\mathrm{L(R_2)}$ を表す正規表現である．

(b) 連接 (R_1R_2) は正規言語 $\mathrm{L(R_1R_2)}=\{\mathrm{xy}|\mathrm{x}\in\mathrm{L(R_1)},\ \mathrm{y}\in\mathrm{L(R_2)}\}$ を表す正規表現である．

(c) クリーネ閉包 $(R_1)^*$ は正規言語 $\{\varepsilon\}\cup\mathrm{L(R_1)}^1\cup\mathrm{L(R_1)}^2\cup\ldots$ を表す正規表現である．ここで，$\mathrm{L(R_1)}^1=\mathrm{L(R_1)}$，$\mathrm{L(R_1)}^{i+1}=\mathrm{L(R_1)L(R_1)}^i=\{\mathrm{xy}|\mathrm{x}\in\mathrm{L(R_1)},\ \mathrm{y}\in\mathrm{L(R_1)}^i\}$，$i\geq 1$ とする．

5. 以上の 1. から 3. の正規表現から出発して，4. における演算を有限回施して得られる表現のみが (Σ 上の) 正規表現である．

正規表現において，各演算子の優先順位（強いものから，クリーネ閉包 '$*$'，連接，ユニオン '|' の順）を決めることで，括弧は省略できる．

例　“$(ab)|(a(b)^*)$” ⇒ “$ab|ab^*$”

正規表現の例とそうでない例を以下に示す．

正規表現の例	“$ab|ab^*a|(ab)^*ba$”，“$((a^*b^*)^*|ba^*)^*$”
正規表現でない例	$(a|b)||b^*$，“$b^*a|b|^*a$”

なお，1 つの正規言語を表す正規表現は一般に幾通りもあり得る．例えば，正規言語 $\{011, 111\}$ を表す正規表現としては “(01|11)1” や “(0|1)11” などがある．2 つの正規表現 R_1，R_2 において $\mathrm{L(R_1)}=\mathrm{L(R_2)}$ であるとき，R_1 と R_2 は等価であるといい $R_1=R_2$ と表す．

またここでは，4 章で示す提案ハードウェアで必要となる正規表現 R に対する極大単純部分表現を定義する．

[定義 2] 文字列 w に対し，w の任意の部分文字列 x を w の部分語と呼び，$x \preceq w$ で表す．正規表現 R に対し，R の部分語 X が正規表現であるとき，X を R の部分表現と呼ぶ．特に，部分表現 X が Σ の要素のみから構成されているとき，単純部分表現という．さらに，R の単純部分表現 X について，X を部分語として真に含むような R の単純部分表現 Y が存在しないとき，X を**極大単純部分表現**という．

正規表現 R のすべての極大単純部分表現の集合を $Str(R)$ で表わし，$Str(R)=\{SE_1, SE_2, \ldots, SE_t\}$，$SE_i=e_1^ie_2^i\ldots e_{l_i}^i$，$e_j^i\in\Sigma$，$1\leq j\leq l_i$，$\Sigma_{i=1}^t l_i=p$，$p$ は R に現れる Σ の要素数（文字数）とする．例えば，$R'=(abc|def)^*ghi$ に対して，$Str(R')=\{abc, def, ghi\}$ である．

次に正規表現マッチングについて説明する．正規表現マッチングとは，正規表現で表されたパターンと一致する文字列をテキストから検索する操作である．正規表現マッチングはパターンが文字列集合であるため，パターンが 1 つの文字列であるストリングマッチン

グと比べると問題が複雑であり，解くために時間がかかる．正規表現マッチングに対するアルゴリズムとして，有限オートマトンをシミュレートする方法が有名である [34]．有限オートマトンについては，次節で説明する．

本論文では，正規表現マッチングにおいて，Σ 上のテキスト T=$\mathrm{t}_1 \ldots \mathrm{t}_m$ と正規表現 R に対して，2 通りのマッチングモードを定義する．1 つ目はアンカーモードマッチングと呼ばれ，$\mathrm{t}_1 \ldots \mathrm{t}_i$=r $\in \mathrm{L}(R)$ $(1 \leq i \leq m)$ の場合に $\mathrm{t}_1 \ldots \mathrm{t}_i$ が R に一致したと判断するマッチングである．もう一方は，アンアンカーモードマッチングと呼ばれ，$\mathrm{t}_i \ldots \mathrm{t}_j$=r $\in \mathrm{L}(R)$ $(1 \leq i \leq j \leq m)$ の場合に一致したと判断するマッチングである．

2.3　有限オートマトン

有限オートマトンとは，有限個の状態と遷移と動作の組み合わせからなる数学的に抽象化された「ふるまいのモデル」である．有限オートマトンは初期状態から入力系列に従って状態遷移を行い，受理状態に達成可能かどうかで入力系列が受理できるかを判定する [19,34,35]．有限オートマトンによって受理される文字列の集合を有限オートマトンが表現する言語と言う．有限オートマトンは大きく分けて決定性有限オートマトンと非決定性有限オートマトンに分類できる．次節で各オートマトンの定義を述べる．

2.3.1　決定性有限オートマトン

決定性有限オートマトン（Deterministic Finite Automaton, DFA）は 5 項組 (Q, Σ, δ_D, q_0, F) で定義される．ここで各項の定義は以下となる．

- 状態の有限集合 Q=$\{\mathrm{q}_0, \mathrm{q}_1, \mathrm{q}_2, \ldots, \mathrm{q}_D\}$,
- 記号の有限集合 Σ=$\{a_1, a_2, \cdots, a_s\}$,
- 状態遷移関数 δ_D : Q$\times$ $\Sigma \rightarrow$ Q,
- 初期状態 $\mathrm{q}_0 \in$ Q,
- 受理状態の集合 F $\subseteq$ Q.

DFA は現状態と入力記号の任意の対に対して遷移先状態が一意に必ず定まるという特徴を持つ．部分文字列 “abc” を含む文字列を受理する DFA の例を図 2.1 に示す．図において，Σ=$\{a, b, c\}$ とする．

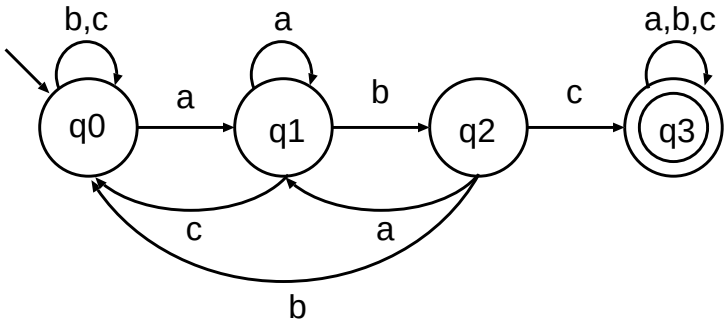


図 2.1: 部分文字列 “abc” を含む文字列を受理する DFA

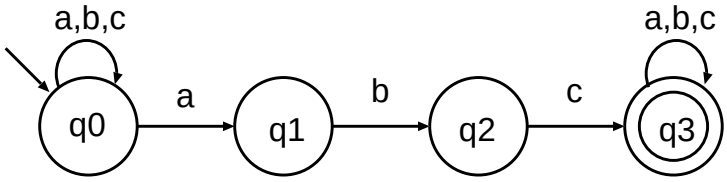


図 2.2: 部分文字列 “abc” を含む文字列を受理する NFA

2.3.2　非決定性有限オートマトン

非決定性有限オートマトン（Non-deterministic Finite Automaton, NFA）は 5 項組 (Q, Σ, δ_N, q_0, F) で定義される．ここで各項の定義は以下となる．

- 状態の有限集合 Q=$\{q_0, q_1, q_2, \ldots, q_N\}$,
- 記号の有限集合 Σ=$\{a_1, a_2, \cdots, a_s\}$,
- 状態遷移関数 $\delta_N : \mathrm{Q} \times (\Sigma \cup \{\varepsilon\}) \to P(\mathrm{Q})$,
- 初期状態 $q_0 \in \mathrm{Q}$,
- 受理状態の集合 $\mathrm{F} \subseteq \mathrm{Q}$.

ここで $P(\mathrm{Q})$ は Q のすべての部分集合からなる集合（冪集合）であり，ε は空語である．

NFA は現状態と入力記号に対して複数の状態に遷移することができ，また，入力記号を読み込まず次状態に遷移する ε 遷移が可能である．また，現状態と入力記号の対に対して，遷移先状態が未定義であることも許す．部分文字列 “abc” を含む文字列を受理する NFA の例を図 2.2 に示す．図において，Σ=$\{a, b, c\}$ とする．

正規表現と NFA，DFA の言語の表現能力は等価である．ある正規表現に対する等価な NFA や，ある DFA に対する等価な正規表現が存在し，その変換方法も広く知られている [34]．正規表現から NFA への変換方法として，Thompson が提案したアルゴリズム [37]

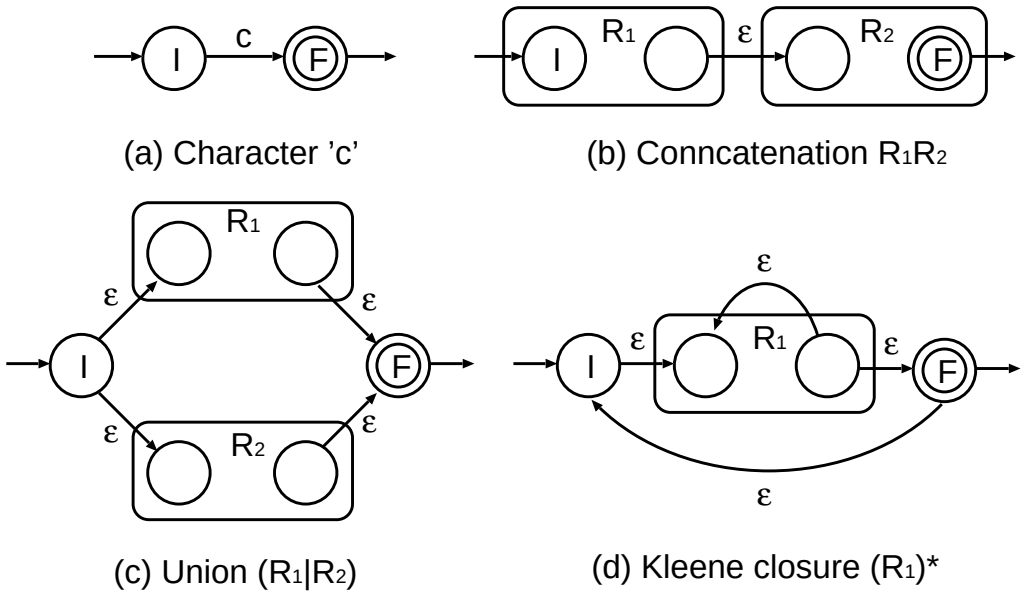


図 2.3: 文字 'c'，R_1 R_2，$(R_1|R_2)$，$(R_1)^*$ に対する NFA

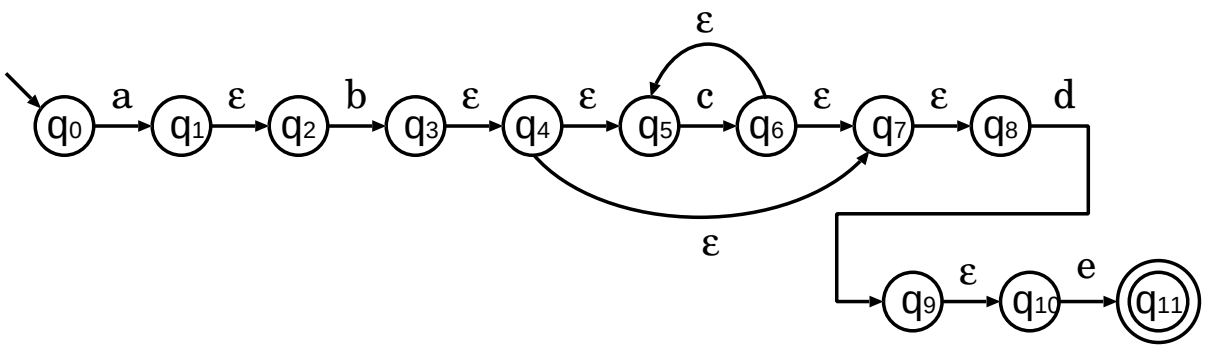


図 2.4: 正規表現 "$ab(c)^*de$" を受理する Thompson オートマトン

が有名である．正規表現の演算子（連接，ユニオン，クリーネ閉包）に対して図 2.3 で示す等価な NFA を用いて，与えられた正規表現に対して再帰的に適用することで，NFA（Thompson オートマトンと呼ぶ）を構築する．図 2.3 中の R_1 と R_2 は正規表現，I と F はそれぞれ初期状態と受理状態を示している．例えば，正規表現 "$ab(c)^*de$" を受理する Thompson オートマトンは，図 2.4 となる．

2.3.3　双対位置オートマトンと文字列遷移双対位置オートマトン

ε 遷移を持つ NFA に対して，これと等価であり，かつ ε 遷移を含まない NFA（ε-$free$ NFA）を構築できる [29, 30]．この ε 遷移を除去する操作を ε 閉包（ε-closure）と呼ぶ．

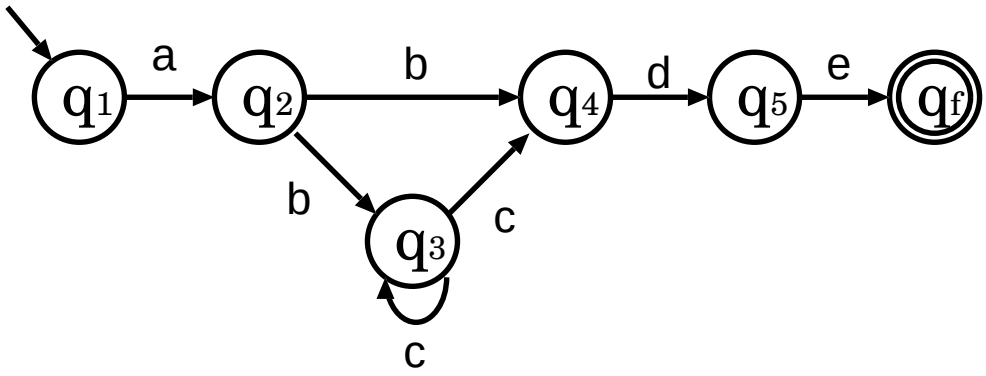


図 2.5: 正規表現 “$ab(c)^*de$” に対する DPA

[38] と [42] はそれぞれ ε-$free$NFA として，Glushkov オートマトン（Position Automaton,PA）や双対位置オートマトン（Dual Position Automaton, DPA）を提案している．PA は Thompson オートマトンと同様，正規表現から構築することができ [36]，DPA は PA または Thompson オートマトンから構築できる [42]．PA や DPA は一般的に，Thompson オートマトンと比べ状態数が少ないという特徴を有する．

ここでは，DPA について説明する．正規表現 R に対する DPA の特徴は以下となる．

1. 状態数は $p+1$ 個（p は R に現れる Σ の要素数），
2. 1 つ以上の初期状態と唯一の受理状態をもつ，
3. 各状態 q において，q から次状態への全ての遷移は同一の文字（記号）で生じる．

例として，正規表現 “$ab(c)^*de$” を受理する DPA を図 2.5 に示す．

各状態から次状態への全ての遷移が同一の文字列のみで生じるように DPA を拡張したオートマトンを，本論文では文字列遷移双対位置オートマトン（String Transition Dual Position Automaton, STDPA）と呼ぶ．4 章で説明する提案ハードウェアは STDPA の動作をシミュレートすることで，正規表現マッチングを実現する．正規表現 R に対する STDPA A は 5 項組 $(Q, \Sigma, \delta_{ST}, Q_s, q_f)$ で定義される．ここで各項の定義は以下となる．

- 状態集合 Q=$\{q_1, q_2, \cdots, q_t, q_f\}$（$t$ は R の極大単純部分表現の集合 $Str(R)$ の要素数），
- 記号の有限集合 Σ=$\{a_1, a_2, \cdots, a_s\}$，
- 初期状態集合 $Q_s \subseteq Q$，
- 受理状態 $q_f \in Q$，
- 状態遷移関数 δ_{ST}: $Q \times \Sigma^+ \rightarrow Q$．

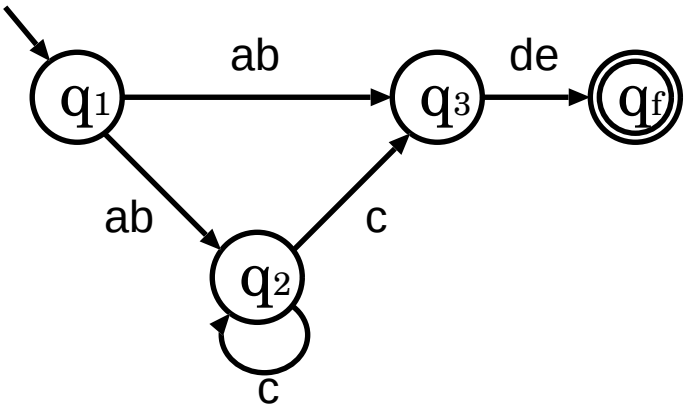


図 2.6: 正規表現 “$ab(c)^*de$” に対する STDPA

また各状態 $q_i \in Q(1 \leq i \leq t)$ から次状態への遷移は R の極大単純部分表現 SE_i でのみで生じるとする．例として，正規表現 “$ab(c)^*de$” を受理する STDPA を図 2.6 に示す.

以下では，STDPA において，状態 $q \in Q$ から入力文字列 w で次状態 r に遷移するとき（$\delta(q, w) = r$），r と w をそれぞれ $Next(q)$ と $Label(q)$ で表す（$Next(q) = r$，$Label(q) = w$）.

2.4　FPGA と部分再構成

FPGA(Field Programmable Gate Array) とは回路の書き換えが可能なロジック・デバイスであり，1985 年に Xilinx 社によって初めて製品化された．有名な FPGA としては Xilinx 社の Virtex シリーズ，アルテラ社の Stratix シリーズがある.

2.4.1　FPGA の構造

本節では Xilinx 社の FPGA について簡単に説明する．最も一般的な FPGA として，図 2.7 で示すアイランドスタイル FPGA が有名である [40].

FPGA 内部には，論理回路を実現するための論理ブロック（LB），配列状に配置された論理ブロック間を縦横方向に通る配線，論理ブロック間の接続を制御するスイッチブロック（SB），回路の入出力機能を実現する I/O ブロック（IOB) が用意されている.

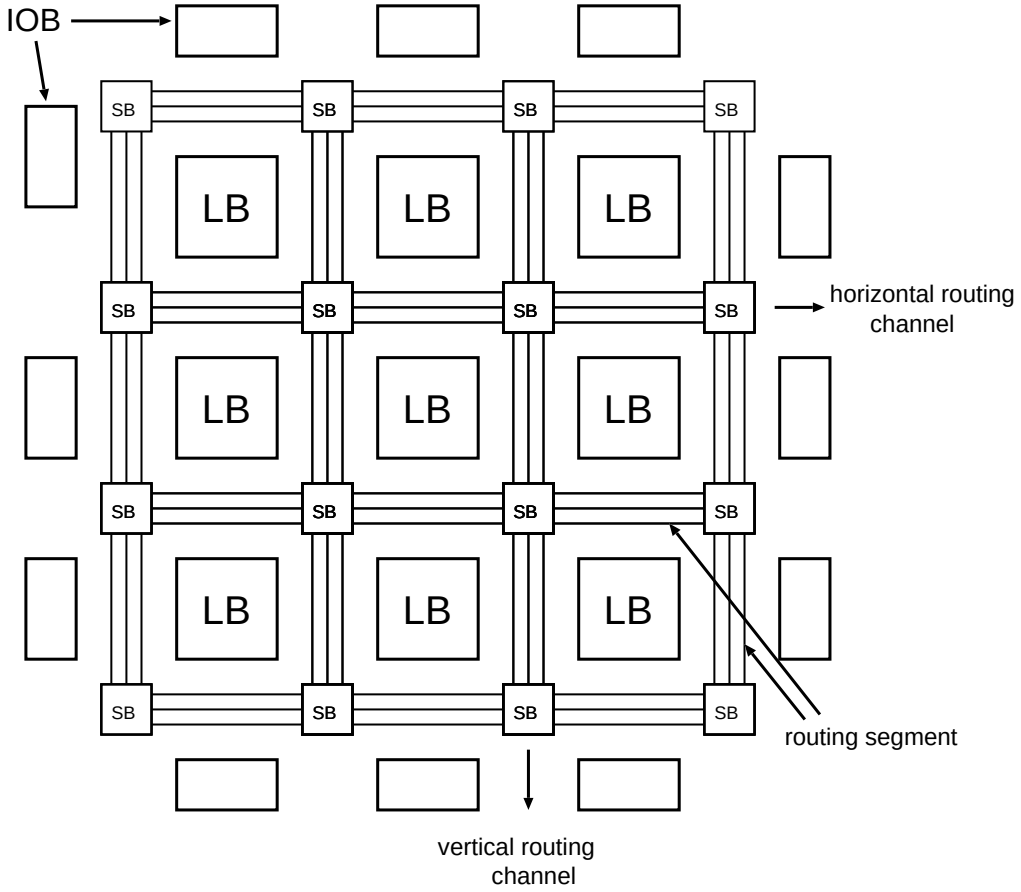


図 2.7: FPGA の構造

2.4.2　FPGA の設計工程

FPGA の設計の流れを図 2.8 に示す．まずユーザによって FPGA 上に実現したい回路をハードウェア記述言語（Verilog-HDL や VHDL など）で記述する．生成したファイルを設計ツール（Xilinx 社の ISE 等）に与えることで，設計ツール内で論理合成と配置配線が行われ，FPGA にダウンロード可能な回路の情報ファイル（ビットストリームデータ）が自動的に出力される．このビットストリームデータを FPGA にダウンロードすることで FPGA 上に回路を実現できる．

2.4.3　FPGA の部分再構成

最新の FPGA には，チップ全体ではなく特定の領域だけを書き換えることができる部分再構成機能や，特定領域以外の回路の動作を停止せずに部分再構成を行うことができる

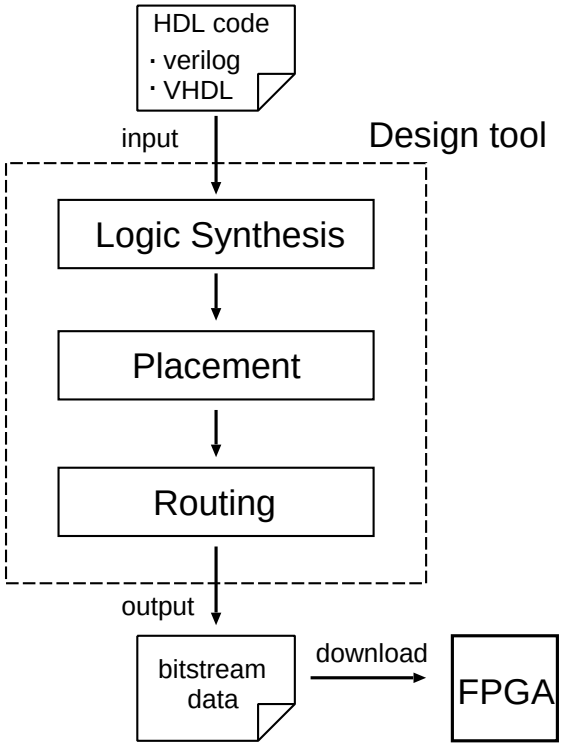


図 2.8: FPGA 設計フロー

動的部分再構成機能が備えられている．近年，Xilinx 社から PlanAhead と呼ばれる動的部分再構成を用いた回路の設計をサポートするツールが提供された．動的部分再構成機能の 1 つの利点として，面積効率の向上があげられる．暗号化処理やメディア処理では，入力に対して順に一連の処理を施していく場合が多く [40]，暗号化やメディア処理回路ではこれらの処理に対する部分回路は同時に動作しない．[43] は，暗号化回路に対して動的に部分回路を切り替えていくことで小規模の回路面積で実装している．

部分再構成の概略を図 2.9 に示す．部分再構成のために確保される FPGA 内の領域は PRR(Partial Reconfiguration Region) と呼ばれ，PRR に実装されるモジュールは PRM(Partial Reconfiguration Module) と呼ばれる．また，PRR 以外の領域を SR(Static Region) と呼び，SR に実装するモジュールは SRM(Static Region Module) と呼ぶ．PlanAhead を使った回路設計では，まず FPGA 内に PRR の大きさと配置する位置をユーザが指定する．ここで，PRR の形は長方形であり，指定できる PRR の大きさの最小単位は Xilinx 社 Virtex-6 の場合，40 個の論理ブロック (Configure Logic Block, CLB) となる [41]．次に，PlanAhead は 1 つの PRM と SRM の組み合わせに対して配置配線を行う．その後，SRM の配置配線結果に合わせて他の PRM の配置配線が行われる．他の PRM の配置配線は PRR を持たない回路の配置配線と比べると難しくなるため，面積オーバヘッドが生じ

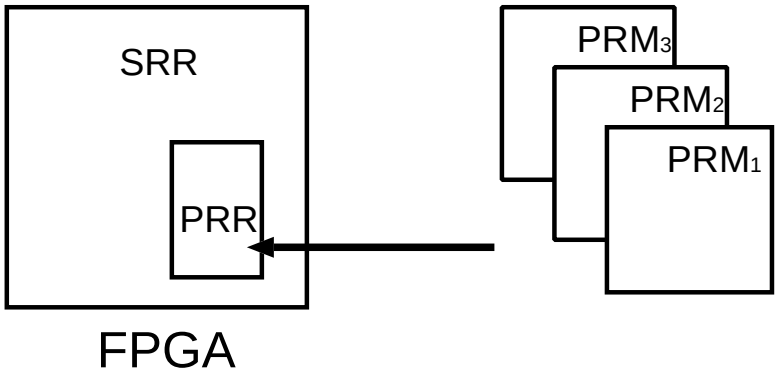


図 2.9: 部分再構成の概略

る．複数の PRR を持つ回路も実装可能であるが，PRR を増やすにつれて面積オーバヘッドも大きくなる．

2.5　侵入検知システム

コンピュータシステムおよびコンピュータネットワークにおける侵入検知システム（Intrusion Detection System，IDS）とは，侵入検知を行うことを目的として設計されたシステムである [20]．ここでの侵入検知とは，コンピュータおよびネットワークに対するセキュリティ侵害の検出，通知，検出情報の管理に関する一連のプロセスを指す．侵入検知の目的は，組織などがセキュリティを考慮していくときに前提となる目的，対象，条件などを明示的に規定したセキュリティポリシーや，対象とする環境などによって変わることがある．その目的と対応としては，主に以下の 4 つが考えられる．

1. システムセキュリティに対する脅威の評価

システムが受けている攻撃について監視および記録を実施し，どの程度の脅威が存在しているかを確認する．ここで得られたデータをもとに，以後のセキュリティ計画の立案やセキュリティポリシーの設定などを行う．

2. 侵入の抑止

システムを監視しているということを，内部と外部のユーザに知らしめ，心理的な効果から不正な行為を抑制する．

3. 被害の防止および軽減

システムへの侵入を受けた際に早期に侵入を発見し，素早く対処することで，被害の発生の防止，軽減を図る．

4. 侵入者の特定・被害状況の記録

システムに残された記録から侵入者を特定したり，保管した記録を法的な証拠として確保する．

侵入検知システムはセキュリティ侵害を発見するために，コンピュータ，ユーザ，あるいはネットワークの状態を監視する．セキュリティ侵害を発見した場合には，発見内容を管理者へ通知するほかに，ファイアウォールの設定を変更したり，ネットワーク接続の切断などの機能を持つものもある．

カリフォルニア大学デービス校による共通侵入検知フレームワーク（Common Intrusion Detection Framework, CIDF）では，4 種類の機能からなるコンポーネントを用いた侵入検知システムの構成を提案している [25]．実際にはすべての侵入検知システムがこのような構成をとっているわけではないが，侵入検知システムに必要とされる機能が概念的に整理されている．このフレームワークに基づく一般的な侵入検知システムのモデルを図 2.10 に示す.

図 2.10 の侵入検知システムでは，各構成要素は以下の 4 つに分類できる．

● **イベント生成部（Event generator）**

システム環境の中から，侵入検出に必要なイベント情報を入力として獲得する．

● **イベント分析部（Event analyzer）**

侵入を検出するためにイベント解析を行うモジュール．さまざまなタイプの分析手法をこのモジュールに適用できる．

● **イベントデータベース（Event database）**

取得したイベント情報を格納しておくデータベース．

● **レスポンスユニット（Response unit）**

検出結果に基づいて，プロセスの停止，接続の遮断，ファイル設定変更，管理者への通知などの侵入に対する対処を実施するためのモジュール．

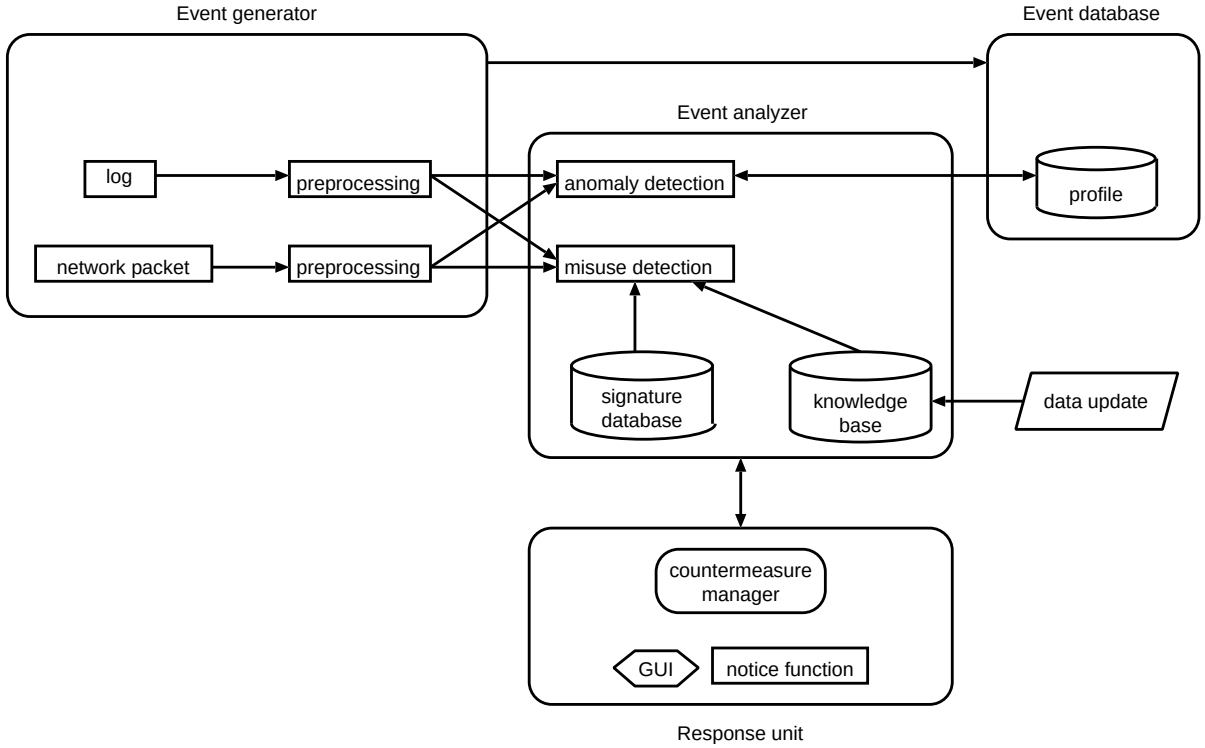


図 2.10: 侵入検知システムの構成

現在，侵入検知システムにはさまざまなタイプのものが存在しており，決定的な方法論やメカニズムというものは確立されていない．多種多様な侵入検知システムは，大きく分けるとホストベース IDS（Host-based Intrusion Detection System, HIDS）とネットワークベース IDS（Network-based Intrusion Detection System，NIDS）の 2 つのタイプに分類することができる．

2.5.1　ホストベース侵入検知システム

ホストベース侵入検知システム（HIDS）はオペレーティングシステム（OS）およびアプリケーションが生成するログデータやコマンドなど単一のホストのシステム上で生成されるイベント情報を監視することで，不正侵入を検知する．代表的なホストベース IDS としては，Tripwire [22] が挙げられる．Tripwire は監視対象のファイルやディレクトリの状態をデータベースとして保存し，システムの現在の状態をデータベースと照らし合わせ，変化がないかを比較する．不正アクセスなどによりファイルの改ざんや削除，パーミッションの変更などがある場合，エラーとして管理者に報告する．

2.5.2　ネットワーク侵入検知システム

ネットワーク侵入検知システム（NIDS）は接続したネットワーク上のパケットデータを監視することで，不正侵入を検知する．NIDS には，自ホスト宛のパケットのみを監視するものと同一ネットワーク内の他ホスト宛のものを含む全トラフィックを監視するものとの 2 つのタイプがある．現在，後者のタイプが主流である．パケットデータの監視においては，パケットのヘッダ情報のみで不正侵入を判断するものと，トラフィックの内容（payload）までを監視対象とするものがある．

不正侵入の検知方法としては**不正検出（Misuse　Detection）**手法と，**異常検出（Anomaly　Detection）**手法の 2 つに分類することができる．以下に各手法について説明する．

1. 不正検出（Misuse Detection）

「不正な操作」や「不正な接続」に際して発生する特徴情報をあらかじめ登録しておき，この特徴情報と同じものを入力イベント中から検出し，侵入を発見する手法を不正検出と呼ぶ．一般に，この不正検出のための特徴情報をシグネチャと呼び，あらかじめ定義されたデータベースとして保存しておく．システムはこのデータベースに基づいて同じ特徴情報を発生したイベントと比較検証する．したがって，この手法ではあらかじめ登録可能な不正（既知の攻撃）のみを検出することが可能となる．代表的なシグネチャ型 NIDS としては Snort [26] や Bro [39] などが挙げられる．

2. 異常検出（Anomaly Detection）

システムやユーザの正常時における振る舞いの傾向をプロファイルデータとしてコンピュータが記憶し，実際のシステム上での振る舞いがこれと大きく異なる状態を検出することによって侵入を発見する手法を異常検出と呼ぶ．異常検出は不正そのものを検出するのではなく，異常な状態を見つけるものであり，未知の攻撃（ゼロデイ攻撃）や DoS 攻撃にも対応することが可能である．ただし，ユーザの操作やシステムの変動に大きく影響を受けるという欠点があり，誤警報の発生確率が高いといわれている．代表的な異常検出型 NIDS としては PAYL [23] や SigFree [24] などが挙げられる．

2.5.3　Snort

Snort は 1988 年 Martin Roesch によって開発された，NIDS の代表的なオープンソースソフトウェアである [26]．Snort はバッファオーバーフローやステルススキャン，CGI 攻撃，SMB 探査，OS fingerprinting attempt などの攻撃を発見するために用いられ，検索とマッチングを含むプロトコル解析を行うことでこれらの攻撃を検知する．Snort には以下の 3 つの主な用途がある．

- sniffer ：　ネットワークを流れるパケットを単に読み込み、入ってくるパケットを入出力装置のセット上で表示.
- packet logger ：　ネットワークを流れるパケットをディスクに保存.
- **ネットワーク不正侵入検知システム** ：　ネットワーク上のトラフィックに対しユーザが定義したルールセットにマッチするかを解析し，その解析結果に基づいて何らかのアクションを実行.

Snort はシグネチャ型 NIDS であり，PCRE（Perl Compatible Regular Expression）[27, 28] で記述された攻撃パターンをシグネチャ（Snort ルール）として，入力されたパケットとのマッチングを行っている．以降では PCRE を拡張正規表現と呼ぶ.

2.5.4　拡張正規表現

拡張正規表現は正規表現と比べ，より簡潔にパターンを記述することができ，正規表現では記述できないパターンも記述できる.

以下に拡張正規表現の演算子を**定義 1** で示した正規表現演算子を用いて定義する．定義において，記号の有限集合を $\Sigma = \{a_1, a_2, \cdots, a_s\}$，$R_1$ と R_2，R_3 を正規表現，R_1^n=$R_1 R_1^{n-1}$ とする.

1. $R_1^? = (R_1|\varepsilon)$,
2. $R_1^+ = (R_1|R_1^2|\ldots)$,
3. $. = (a_1|a_2|\ldots|a_s)$,
4. $R\{M\} = R^M$,
5. $R\{M, N\} = (R^M|R^{M+1}|\ldots|R^N), M < N$,
6. $R\{M, \} =(R^M|R^{M+1}|\ldots)$,
7. $[a_i a_j a_k]= (a_i|a_j|a_k), 1 \leq i < j < k \leq s$,

8. $[a_i - a_j] = (a_i|a_{i+1}|\ldots|a_j), 1 \leq i < j \leq s$,
9. $[\hat{}\ a_i] = (a_1|a_2|\ldots|a_{i-1}|a_{i+1}|\ldots|a_s), 1 \leq i \leq s$,
10. ‘ ^ ’ は文字列の先頭，‘$’ は文字列の終端もしくは改行の前にそれぞれマッチする，
11. 以下の演算子は次のように定義される．また，各演算子の大文字はその演算の否定を意味する．
    - \n　改行文字
    - \r　リターン文字
    - \w　英数字または ‘ _ ’
    - \s　空白文字
    - \d　数字
    - \b　単語境界 (片方を \w，もう片方を \w 以外で挟まれた間の位置)
12. ‘(’ と ‘)’ で囲まれた部分正規表現をグループと呼ぶ．パターンの左から p($1 \leq p \leq 99$) 番目のグループにマッチした文字列を ‘$p’ とした時，‘\p’ は文字列 ‘$p’ を意味する演算子である．
    例）拡張正規表現 “$(cat|dog)\ and\ \backslash 1$” は 2 つの文字列 “cat and cat” と “dog and dog” を表現している．
13. “$R_1\ (?{=}R_2)\ R_3$” は R_1 と一致したすぐ後に R_2 と R_3 の両方と一致するような文字列の集合を意味する．ここで，R_2 と一致する文字列は R_3 と一致する文字列の部分文字列，またはその逆であることに注意されたい．
14. “$R_1\ (?!R_2)\ R_3$” は R_1 と一致したすぐ後に R_2 には一致せず R_3 と一致するような文字列の集合を意味する．
15. 以下の修飾子を設定することでマッチングの実行方法が変更される．
    - i　大小文字を区別しない．
    - s　‘ . ’ が改行文字にもマッチする．
    - m　‘ ^ ’ と ‘ $ ’ が各行の行頭・行末とマッチする．
16. 2. と 10. から 12. の各演算子の後ろに ‘?’ を付けることによって，なるべく少ない回数繰り返した文字列を見つけるようにマッチングを実行する．
    例）パターン $test.^{*}test$ とテキスト “test hard test easy test” に対しては “test hard test easy test” がマッチする．一方，パターン $test.^{*}\mathbf{?}test$ と同テキストに対しては “test hard test” がマッチする．

ここで，1. を選択文字，3. を任意の文字，4. から 6. を量指定子，7. から 9. を文字クラス，12. を後方参照，13 を肯定的先読み演算，14. を否定的先読み演算と呼ぶ．

第3章　関連研究

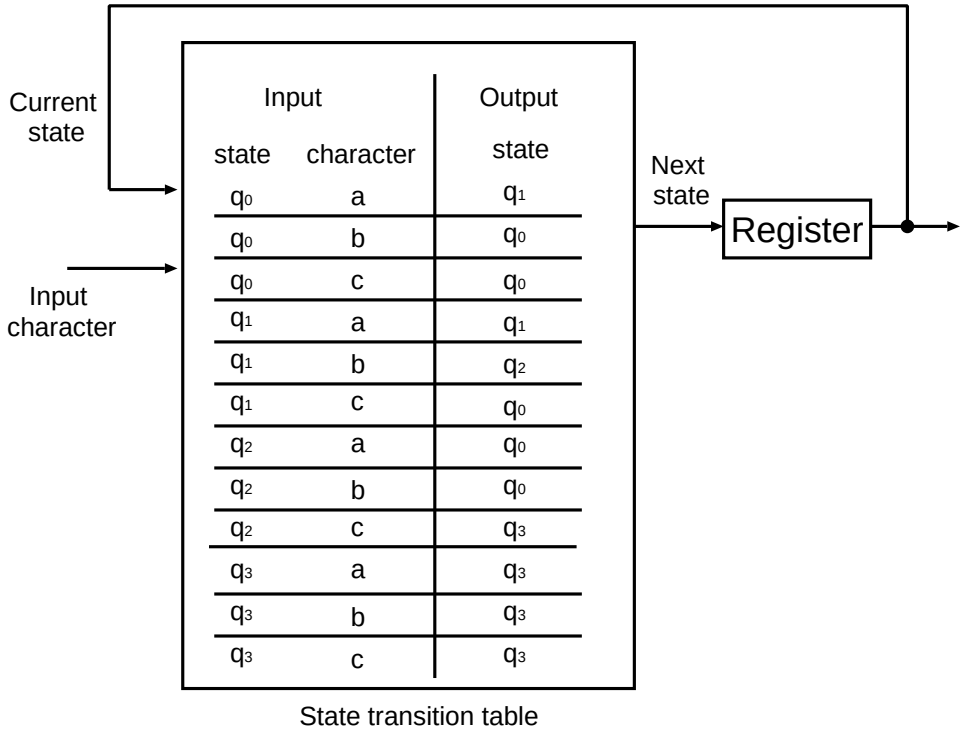
本章では，正規表現マッチングハードウェアに関する従来研究について概観する．

3.1　正規表現マッチングの実現手法

正規表現マッチングは決定性有限オートマトン（DFA）または非決定性有限オートマトン（NFA）の動作をシミュレートすることで実現されることが一般的である．Snort ではソフトウェアによって NFA または DFA をシミュレートすることで，コンピュータウィルスを検知している．

DFA のシミュレーションモデルを図 3.1 に示す．DFA のシミュレーションでは状態遷移表がメモリ上に実装される．図 3.1 の状態遷移表は図 2.1 に示した DFA に対するものである．入力文字と現状態をアドレスとしてメモリに入力することで，次状態がメモリから出力される．DFA のシミュレーションはテキスト 1 文字に対して 1 回のメモリアクセスだけで状態遷移ができるので，高速な正規表現マッチングを実現できる．しかし特定の正規表現に対するマッチングや複数の正規表現（ウィルスパターン）を同時にマッチングする場合，DFA の状態数が爆発し，大量のメモリ容量が必要になる可能性がある [44–46]．DFA の状態数の削減のため，Delayed DFA (D^2FA) [44, 45] や extended FA (XFA) [46] の提案や，状態遷移表の圧縮手法 [47] などが提案されている．

一方，NFA によるシミュレーションでは，NFA の状態数は少ないため，必要となるメモリ容量は少量である．しかし，NFA は現状態が複数あるため，ソフトウェアでは各現状態からの遷移を逐次的に計算する必要があり，時間がかかる．Snort では，NFA の状態遷移を深さ優先的に行い，遷移に失敗するとバックトラックを行い次の状態遷移を試している [27]．図 2.2 の NFA とテキスト “abababc” に対するソフトウェアシミュレーションの動きを示す．横軸には入力文字と処理される状態遷移を示している．例えば 1 文字目の a に対して NFA では現状態から q_0 と q_1 に遷移するが，ソフトウェアでは，まず q_1 への遷移を試していき，遷移に失敗したときにバックトラックし，q_0 への遷移を試してい



state	character	state
q_0	a	q_1
q_0	b	q_0
q_0	c	q_0
q_1	a	q_1
q_1	b	q_2
q_1	c	q_0
q_2	a	q_0
q_2	b	q_0
q_2	c	q_3
q_3	a	q_3
q_3	b	q_3
q_3	c	q_3

図 3.1: DFA のシミュレーションモデル

く．Snort における正規表現マッチングの処理性能は，0.9Kbps から 10.9Mbps と非常に低い性能であることが [8] で示されている．近年では，頻繁にバックトラックがおこるパケットを送りつける NIDS の性能低下を狙った攻撃 algorithmic complexity attack [48] があることから，[8] では，バックトラックを行わず NFA をシミュレートする方法として，NFA を OBDD (Ordered Binary Decision Diagram) を用いて表現する方法を提案している．この手法の処理性能は 0.6Mbps から 4.7Mbps となり，バックトラックを行う方法と比べ，最悪処理性能は約 600 倍改善している．しかしソフトウェアの処理では数十 Mbps の性能しか出すことができない．近年では，並列処理を行うことができる正規表現マッチングの GPU 実装や正規表現マッチングハードウェアに関する研究が盛んに行われている．

正規表現マッチングの GPU 実装として，[49–51] が提案されている．[49] は，GPU 内の全てのプロセッサが異なる入力パケットに対して同じ DFA に対するシミュレーションすることで（SIMD 処理)，最大で 16Gbps の処理性能を達成している．しかし，この手法は DFA をシミュレートするため，状態数が爆発する可能性がある．そのため，状態数が爆発するようなパターンはソフトウェアで NFA をシミュレートすることで正規表現マッチングが行われる．[50,51] は各プロセッサで同じ NFA と異なる入力パケットに対してシミュレーションを行うことで，9.9Mbps から 46.4Mbps の処理性能を達成している．

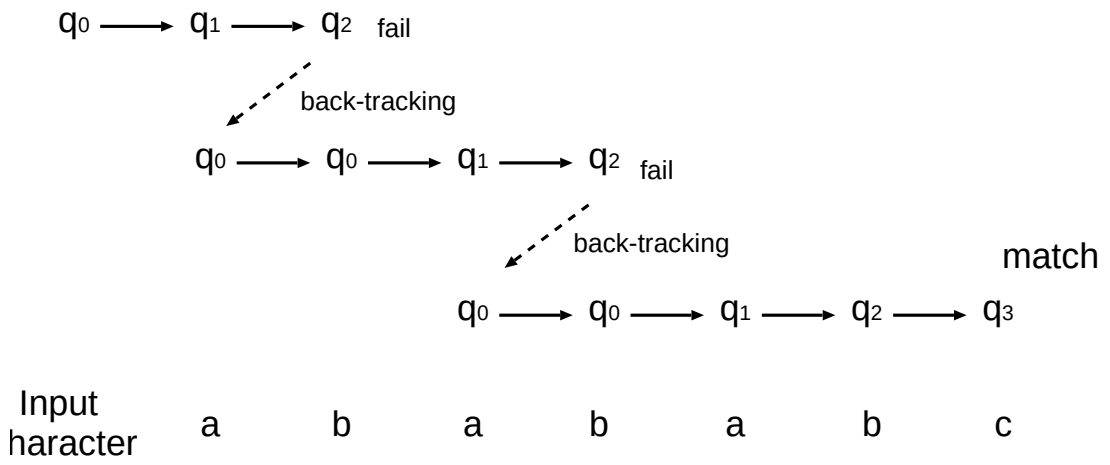


図 3.2: NFA のソフトウェアシミュレーション

正規表現マッチングハードウェアは大きく分けてパターン依存型とパターン非依存型に分類される．次節からはこれらのハードウェアについて説明していく．

3.2　パターン依存正規表現マッチングハードウェア

パターン依存正規表現マッチングハードウェアは，与えられたパターンに特化した回路構成を持つ [52–62]．[52] は与えられた正規表現パターンに対応する NFA から NFA を実現する回路の生成方法を提案した．図 2.3 で示した 4 つの基本 NFA に対応した回路（図 3.3）を用いて全体回路が構成される．図 3.3 の’c’ は 入力文字が’c’ と一致するかを比較する比較器を意味する．図 3.4 は，正規表現 “ $a(bc)$*$(d|e)$ ” に対する NFA 回路である．

この手法の拡張として，[54] は正規表現から NFA への変換と NFA から回路への変換の最適化によって，回路規模と処理速度の改善を行っている．[58, 59] は複数の正規表現を扱う際，接頭辞や接尾辞，接中辞をまとめることで回路規模を削減する手法を提案している．[55] は 6 入力 LUT を持つ FPGA に対する回路の最適化手法を提案している．[56, 57] は文字列遷移 NFA を導入し，処理速度の向上と回路規模の削減を行っている．[53, 60, 61] は量指定子やクラス文字に対する回路規模の削減のために，これらの演算子に対する回路構成を提案している．[62] は回路規模の削減のため，FPGA の動的部分再構成機能を用いて Snort に予め用意されている通信プロトコルごとのパターンに対する専用ハードウェアを動的に切り替える手法を提案している．[74] は複数のパターン依存正規表現マッチングハードウェアを FPGA 上に並列実装し，10Gbps の性能で正規表現マッチングを実現できることを示している．

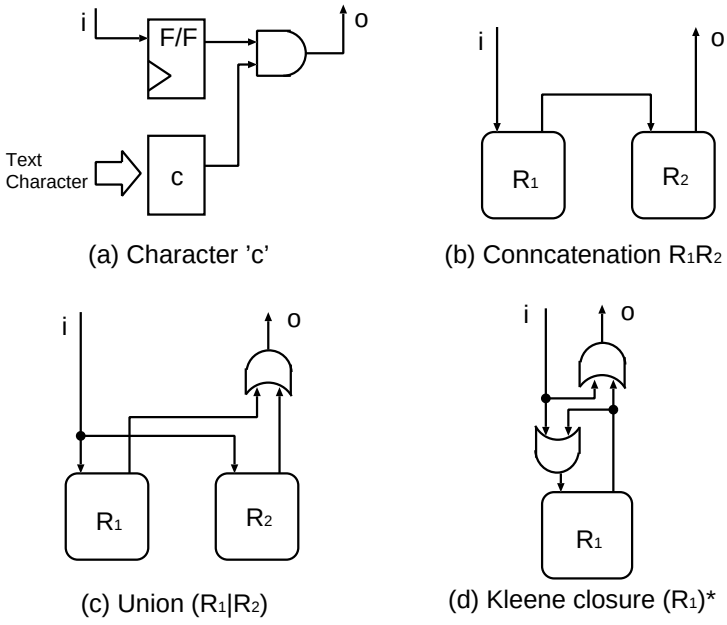


図 3.3: 文字 c，R_1R_2，$(R_1|R_2)$，$(R_1)^*$ に対する NFA 回路

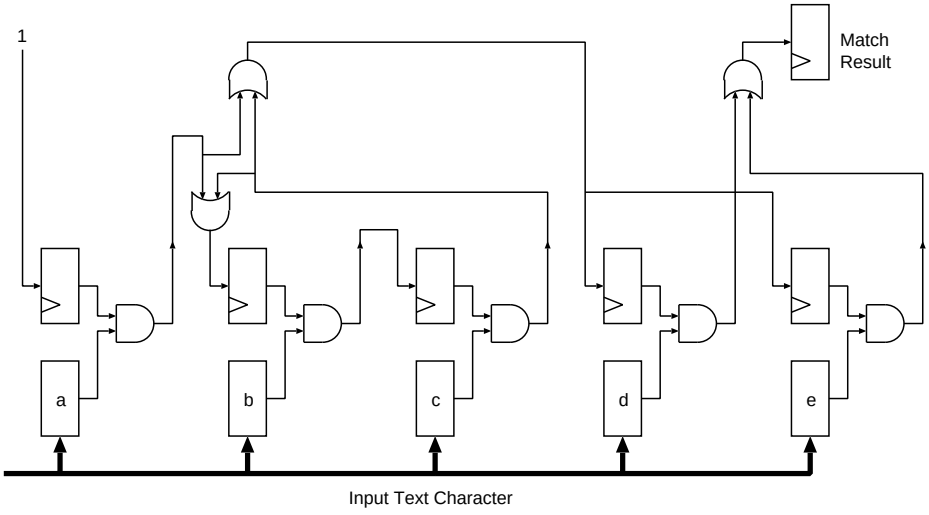


図 3.4: 正規表現 " $a(bc)^*(d|e)$ " に対する NFA 回路

パターン依存正規表現マッチングハードウェアはパターンが更新される度に，回路を再構成する必要があるため，FPGA のような再構成可能なハードウェアデバイス上に実装される．

3.3　パターン非依存正規表現マッチングハードウェア

パターン依存正規表現マッチングハードウェアはパターンに特化した回路構成を持つため，高速かつコンパクトな回路で実現できる．しかし，パターンが更新される度に回路の再合成と再構成が必要となり，パターン更新に一定時間必要となる．

もうひとつの正規表現マッチングハードウェアとして，パターン非依存正規表現マッチングハードウェア [63–68] が提案されている．パターン非依存正規表現マッチングハードウェアは，任意のパターンに対してマッチングを実現できるように回路を構成したマッチングハードウェアである．パターン非依存正規表現マッチングハードウェアは回路面積や処理速度の面で，パターン依存正規表現マッチングハードウェアには劣るが，パターンに依存しないハードウェア構成であるため，パターンの更新に瞬時に対応できるという利点がある．[63–67] は，正規表現のクラスを制約することで，コンパクトで高速な回路を提案している．[63] と [64] はそれぞれ NFA とビット並列 NFA [36] に基づく正規表現マッチングハードウェアを提案している．これらのハードウェアは入力として，文字列に対するユニオンを禁止している．[65–67] において，シストリックアルゴリズム [69] に基づく正規表現マッチングハードウェアを提案している．[66] は文字の連接 abc や 1 文字のクリーネ閉包 a^* のような正規表現クラスのみを対象としたハードウェアである．[65] ではクリーネ閉包のネストを除く全ての正規表現クラスを扱うための拡張が提案された．ここで，クリーネ閉包のネストとは $(R)^*$ において，R にさらにクリーネ閉包を含む正規表現であり，例として $(a(bc)^*d)^*$ があげられる．[67] では量指定子を直接扱うための拡張が提案された．任意の正規表現を扱うことができるパターン非依存正規表現マッチングハードウェアとして，[68] では DPA に基づくマッチングハードウェアを提案している．しかし回路構造が複雑で回路規模が非常に大きくなるという欠点を持つ．

次節では本論文で提案するハードウェアの基となるシストリックアルゴリズムに基づく正規表現マッチングハードウェア [65–67] と DPA に基づく正規表現マッチングハードウェア [68] について説明する．

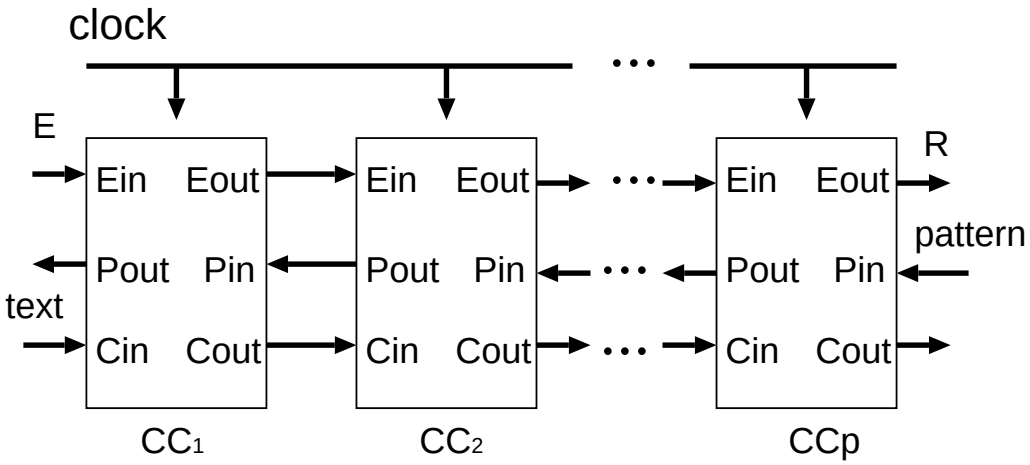


図 3.5: シストリックアルゴリズムに基づく正規表現マッチングハードウェアの構成

3.4　シストリックアルゴリズムに基づく正規表現マッチングハードウェア

3.4.1　アーキテクチャ

シストリックアルゴリズム [69] に基づく正規表現マッチングハードウェアは比較セル (Comparison Cell，CC) と呼ばれるプロセッシングユニットを 1 次元配列状に相互接続した構成をもつ (図 3.5)．各 CC は 1 文字単位のマッチングを行う同期回路である．マッチング開始前にあらかじめ，パターンを 1 文字ずつ入力していき，各 CC にパターンの各文字を記憶する．マッチングを開始すると，左側の CC から 1 クロックごとにテキストが 1 文字ずつ入力され，各 CC で 1 文字単位のマッチングが並列に実行される．各 CC の比較結果は右隣の CC に伝えられ，最終的に右端の CC からマッチング成功が出力されると，テキスト内の部分文字列がパターンと一致したことを示す．この構成により，各 CC にセットされているパターン文字を書き換える事で，パターンの更新を瞬時に行うことができる．このハードウェアアーキテクチャは規則正しい構造を持ち，クロック信号に同期して動作する．クロック信号以外の信号はすべて隣り合ったセル間で伝送されるローカルな信号であることから，このアーキテクチャはシストリックアーキテクチャとみなすことができる．

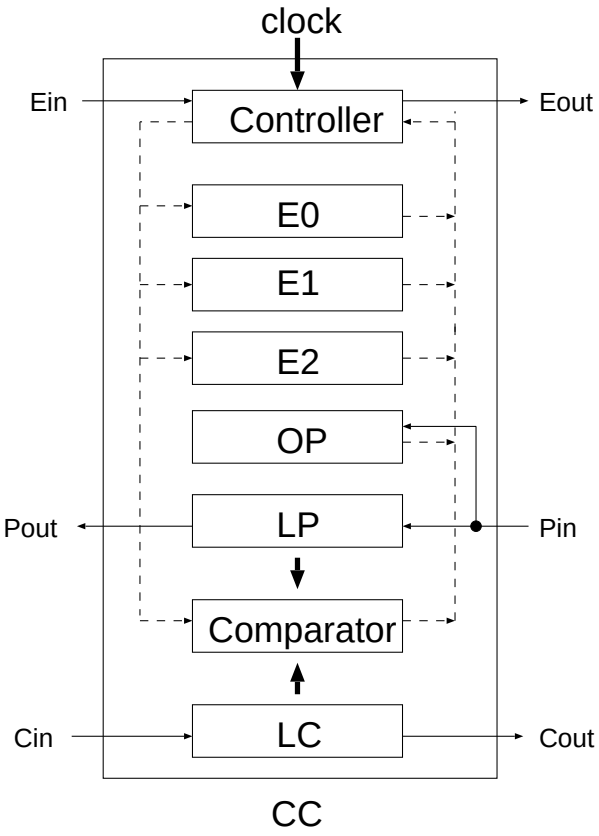


図 3.6: 比較セルの構造

3.4.2　比較セル（CC）の基本構造

この節では比較セル（CC）の基本構造を示すため，正規表現演算子である‘ε’と 1 文字に対するクリーネ閉包‘$*$’，‘.’，‘?’を扱うことができるセルについて説明する．

CC は 1 文字単位のマッチングを行う回路である．図 3.6 に CC の構造を示す．CC は LP と LC，E0，E1，E2，OP レジスタと制御回路 Controller，比較器 Comparator からなる．LP，LC レジスタはパターン中の 1 文字とテキスト中の 1 文字をそれぞれ記憶する 8 ビットレジスタである．OP は正規表現演算子（‘ε’や‘$*$’など）を示すための n ビットレジスタである．ここで n はこのハードウェアがサポートする正規表現演算子の種類数となる．‘ε’と‘$*$’，‘.’，‘?’を扱う比較セルの場合，4 ビットレジスタとなる．全 CC は単一クロック信号に同期して動作する．マッチング開始信号（Ein=true）が入力されると，正規表現演算子を考慮しつつ LP と LC の比較が行われる．ここで，Ein=true は CC で比較が行われるテキスト内の文字の 1 つ前までの文字がそこまでのパターンと一致したことを意味する．LP と LC の比較が成功した場合は 1 クロック後に E1 の値が true となる．さら

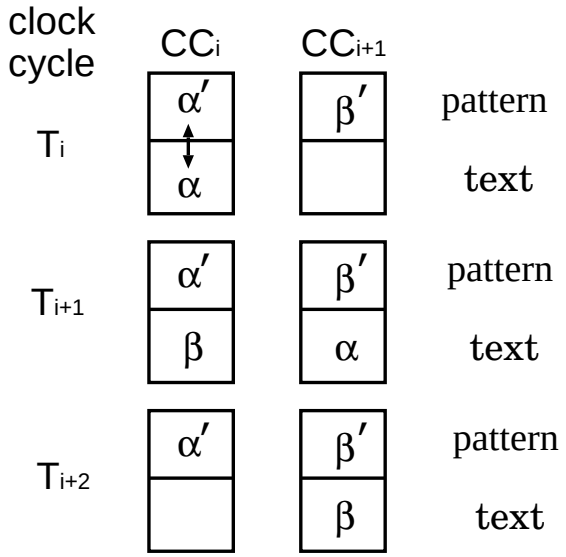


図 3.7: 各 CC におけるテキストの流れ

t_0: begin
  MOVE_INPUT_STRING;
  E:=Ein or $E0$;
  $E2$:=$E1$;
  $E0$:=false;
  $E1$:=false
  $Eout$:=$E2$;
end;
t_1: begin
  if E then begin
    R:= $CMP(LP, LC)$;
    if R or (OP=‘.’) then begin
      $E1$:=true;
      if (OP=‘$*$’) then $E0$:=true
    end;
    if (OP=‘$*$’) or (OP=‘ε’) or (OP=‘?’)then
      $E2$:=true
  end
end;

図 3.8: 比較セルの動作アルゴリズム

に 1 クロック後に E1 の値は E2 に伝えられ，E2 の値はそのまま Eout の値となる．以降では，Eout=true を Eout を出力すると言い換えている．アーキテクチャの特性上，テキ

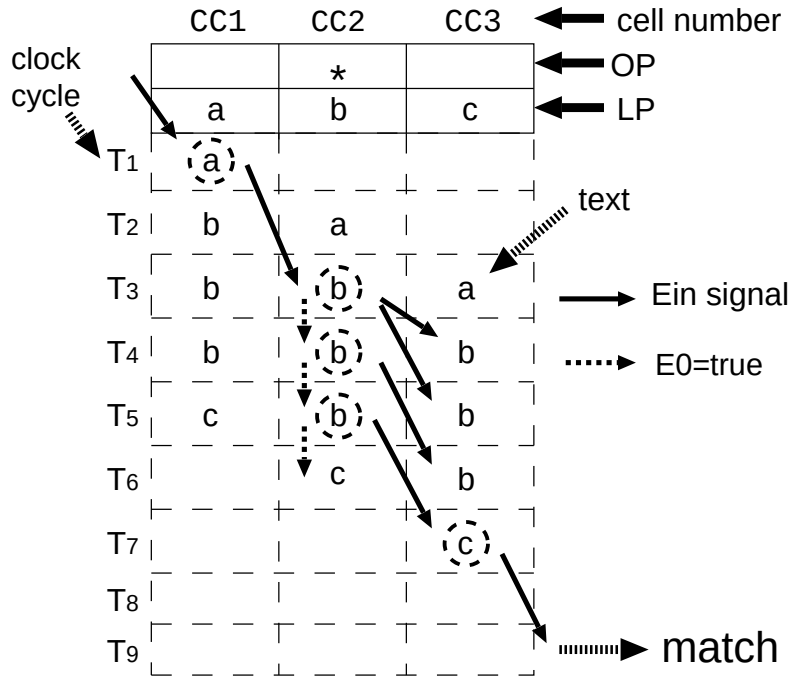


図 3.9: パターン “ab^*c” に対する比較セルの動作例

スト内のある任意の文字 α の次の文字 β が，文字 α が一致したパターン文字 α' の次のパターン文字 β' を持つ CC （i.e., α' を持つ CC の右隣の CC）に到着するまで 2 クロック必要であることから（図 3.7），各 CC は 2 クロック後に比較結果を右隣の CC に伝える必要があり，出力タイミングを遅らせるために E1 と E2 の 2 つのレジスタを用いている．レジスタ E0 は 1 文字に対するクリーネ演算のために用いられる．E0 は OP に ‘$*$’ の演算子が記憶されているとき，マッチング開始信号（Ein=true）が入力されると E0=true となる．これは，その CC での文字の比較が不一致となるまで保持され，E0=true である限りマッチングが繰り返し実行される．比較が不一致となった後は，次の開始信号が入力されるまで，マッチングは実行されない．

比較セルの動作アルゴリズムを図 3.8 に示す．この図では便宜上，内部変数 E と R が使われているが，図 3.6 では対応するレジスタはないことに注意されたい．またレジスタ LP と OP にはパターンの 1 文字と正規表現演算子が既に設定されているものとする．さらに説明の簡単化のため，2 相クロックフェーズ t_0,t_1 を仮定して動作を記述している．図の “MOVE_INPUT_STRING;” はクロックに同期して右隣の CC にテキスト文字をシフトすることを意味し，各 CC は与えられたテキストが右端の CC から出力されるまでこの動作を繰り返す．また，$CMP(LP, LC)$ は LP と LC の文字が一致する場合 $true$ であり，そうでない場合は $false$ である．

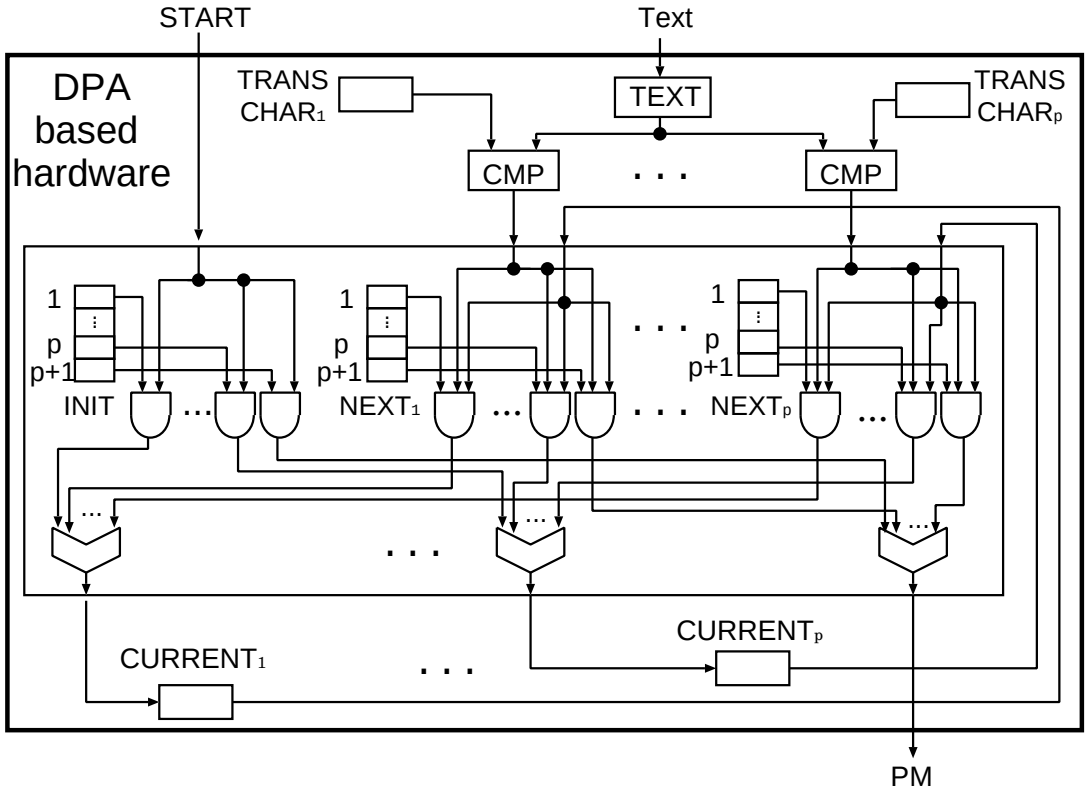


図 3.10: DPA に基づく正規表現マッチングハードウェアの構成

3.4.3　動作例

図 3.9 にパターン “*ab*c*” とテキスト “abbbc” に対する各 CC の動作例を示す．図は各クロックサイクルにおける各 CC の出力信号と各 CC が保持しているテキスト内の 1 文字を表している（テキストは左側から 1 クロックごとに入力されていくことに注意）．T1 クロックでは，CC1 にテキストの 1 番目の文字 ‘a’ と Ein が入力される．このとき，CC1 で比較が行われる．ここでは比較に成功するため，2 クロック後の T3 クロックで CC2 に Eout が出力される．CC2 において T3 クロックでは，テキストの 2 番目の文字 ‘b’ と Ein が入力される．CC2 は OP に ‘∗’ が記憶されているため，空語とのマッチング成功として次のクロックで CC3 に Eout を出力する．また CC2 では文字の比較が不一致になるまで，文字の比較が行われるため，T3 から T5 に入力されたテキストの 2 から 4 番目の文字 ‘b’ とのマッチング成功に対して Eout が出力される．CC3 では T4 から T7 クロックに Ein が入力され，比較が行われる．T7 クロックで入力されたテキストの 5 番目の文字 ‘c’ と比較に成功するため T9 クロックで CC3 が Eout を出力する．ここで，右端の CC である CC3 が Eout を出力したことは，テキスト内の部分文字列がパターンと一致した事を示している．

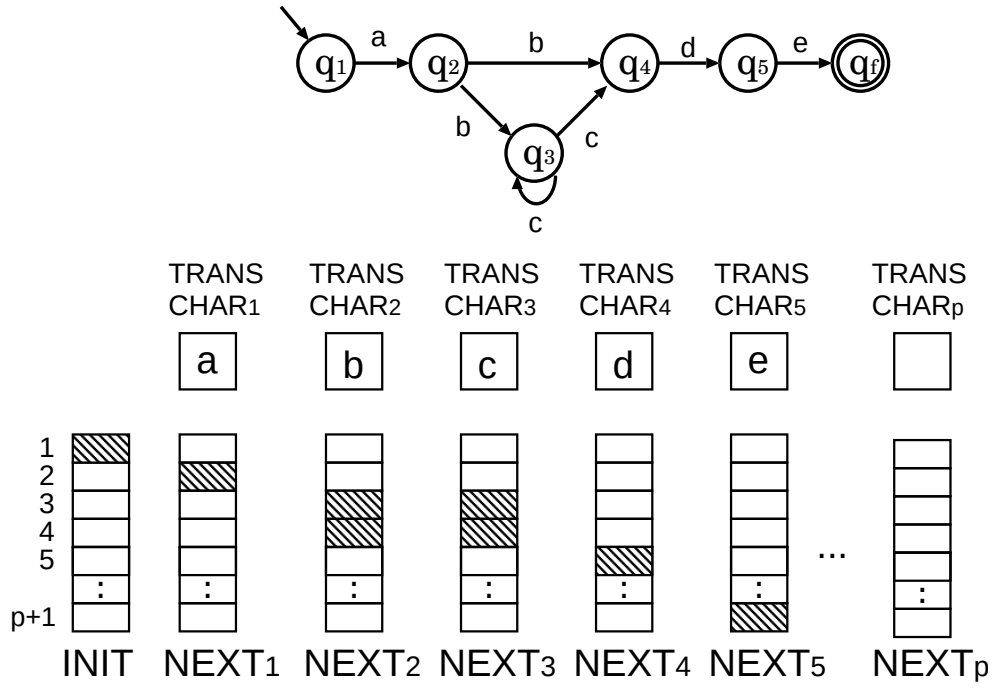


図 3.11: DPA と各レジスタの値

本節では [66] で提案された基本 CC について説明した．[65, 67] はこの CC を拡張し，様々な正規表現演算子を扱っている．拡張 CC の詳細については [65, 67] を参照されたい．

3.5　DPA に基づく正規表現マッチングハードウェア

DPA（2.3.3 節で前述）に基づく正規表現マッチングハードウェアの構成を図 3.10 に示す．DPA 基づく正規表現マッチングハードウェアはクロックに同期して動作を行い，1 クロックサイクルごとにテキストが先頭から 1 文字づつ入力される．またアンカーモードでは，テキストの先頭文字が入力されるクロックサイクルに $start$=1 が入力され，アンアンカーモードでは，全てのテキスト文字が入力されるクロックサイクルで $start$=1 が入力される．ハードウェアは入力に基づき，DPA の状態遷移をシミュレートし，受理状態に達したとき，パターンマッチ信号 PM=1 を出力する．

構成として，レジスタ $TEXT$ と p 個のレジスタ $CURRENT$，p 個のレジスタ $NEXT$，p 個のレジスタ $TRANS\_CHAR$，レジスタ $INIT$ と比較器 cmp からなる．p はハードウェアで扱うことができる DPA の状態数の最大値である．

$INIT$ と $NEXT_i(1 \leq i \leq p)$ はそれぞれ初期状態と状態 q_i の次状態を示すための p+1

ビットレジスタである．$TRANS\_CHAR_i$ は状態 q_i の遷移文字を示すための 8 ビットレジスタである．$INIT$ と $NEXT_i$，$TRANS\_CHAR_i$ の値を設定することで，DPA を表現する．例として，図 2.5 の DPA に対する各レジスタの値を図 3.11 に示す.

このハードウェアは，任意の正規表現を実現でき，柔軟性が高いという利点がある．しかし，各状態の次状態をレジスタで記憶しているため，レジスタの総 bit 数と配線数が状態数に対して二乗オーダーとなる．DPA の状態数は正規表現パターン内の文字数に対応するので，長いパターンを扱う場合，大量のハードウェアリソースが必要となる．

第4章　シストリックアルゴリズムとNFAに基づくパターン非依存正規表現マッチングハードウェア

本章では，任意の正規表現を扱うことができるシストリックアルゴリズムとNFAに基づくパターン非依存正規表現マッチングハードウェアを提案する．提案ハードウェアはシストリックアルゴリズムに基づく単純文字列の比較機能とDPAに基づく状態遷移機能の組み合わせにより，文字列に対して状態遷移するSTDPA（2.3.3節で定義）をシミュレートする．これにより，任意の正規表現パターンに対応でき，かつDPAに基づくマッチングハードウェアより回路構造がコンパクトなパターン非依存正規表現マッチングハードウェアを実現する．

本章の流れとして，まず提案ハードウェアの構成を示し，その後，ハードウェアの正当性を示す．次に，提案ハードウェアの拡張を示し，最後に実験的評価を述べる．

4.1　基本アーキテクチャ

図4.1に提案マッチングハードウェアの基本アーキテクチャを示す．基本アーキテクチャはシストリックアルゴリズムに基づくストリングマッチングユニット（SMU）とNFAに基づく状態遷移ユニット（STU）からなる．SMUは与えられた正規表現Rの遷移文字列（極大単純部分表現（2.2節で前述））の比較を行い，比較結果をSTUに伝える．STUはその比較結果に基づき状態遷移を管理する．

基本アーキテクチャはクロックに同期して動作を行い，1クロックサイクルごとにテキストT= $\mathrm{t}_1 \ldots \mathrm{t}_m$が先頭から1文字づつ入力される．ここでは，クロックで決まるクロックサイクルを$T_1, T_2, \ldots$で表し，Tのt_iがSMUに入力されるクロックサイクルをT_iとする．またアンカーモードでは，T_1のみ$START$=1が入力され，アンアンカーモードでは，全てのクロックサイクルで$START$=1が入力される．以下の節で，SMUとSTUについ

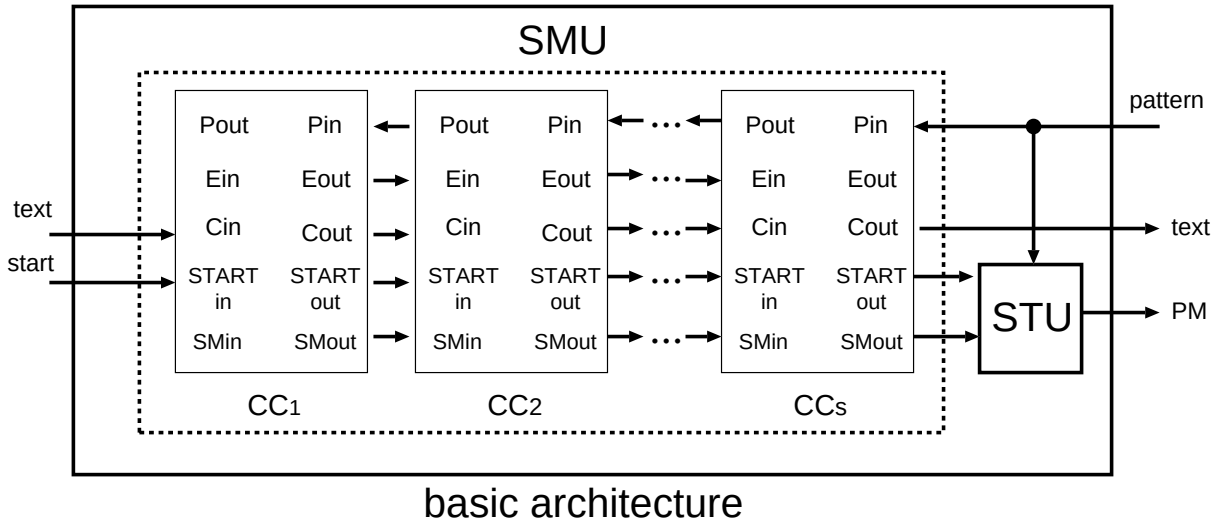


図 4.1: 基本アーキテクチャ

て詳しく述べる．

4.2　ストリングマッチングユニット（SMU）

本節では STDPA の遷移文字列の比較を実現するストリングマッチングユニット（SMU）について説明する．SMU は正規表現 R を構成する極大単純部分表現 SE_i，$1 \leq i \leq t$，とテキスト T= $\mathrm{t}_1 \ldots \mathrm{t}_m$ が入力され，SE_i と一致する T 中の部分文字列を見つける．以下では，R において，$Str(R)$=$\{SE_1, SE_2, \ldots, SE_t\}$，$SE_i$=$e_1^i e_2^i \ldots e_{l_i}^i$，$e_j^i \in \Sigma$，$1 \leq j \leq l_i$，$\Sigma_{i=1}^{t} l_i = p$，$p$ は R に現れる Σ の要素数とする．SMU は SE_i に対するストリングマッチング信号 SM_i とマッチング開始信号 $START$ を STU に出力する．ここで，SE_i は STDPA の状態 q_i からの遷移文字列（$Label(q_i)$）であることに注意する．以下では，テキストを構成するアルファベット Σ の要素（文字）は 8 ビットで表現されていると仮定する．

SMU は s 個の比較セル（CC）を 1 次元配列状に接続した構造を持つ．ここで，$s \geq p$ である．各 CC は 1 文字単位のマッチングを行う同期回路である．マッチング開始前に，SE_i を構成する各文字 e_j^i が各 CC に設定される．$s > p$ の場合，CC_{s-p+1},CC_{s-p+2},...,CC_s には空文字 ε が設定される．マッチングが開始されると，CC_1 に 1 クロックサイクルごとにテキストが 1 文字づつ入力され，各 CC で 1 文字単位のマッチングが並列に行われる．

CC の構成（図 4.2）について説明する．各 CC は LP と LC，E1，E2，START，SM，ST，INDEX レジスタと制御回路 Controller，比較器からなる．LP と LC は 8 ビットレジ

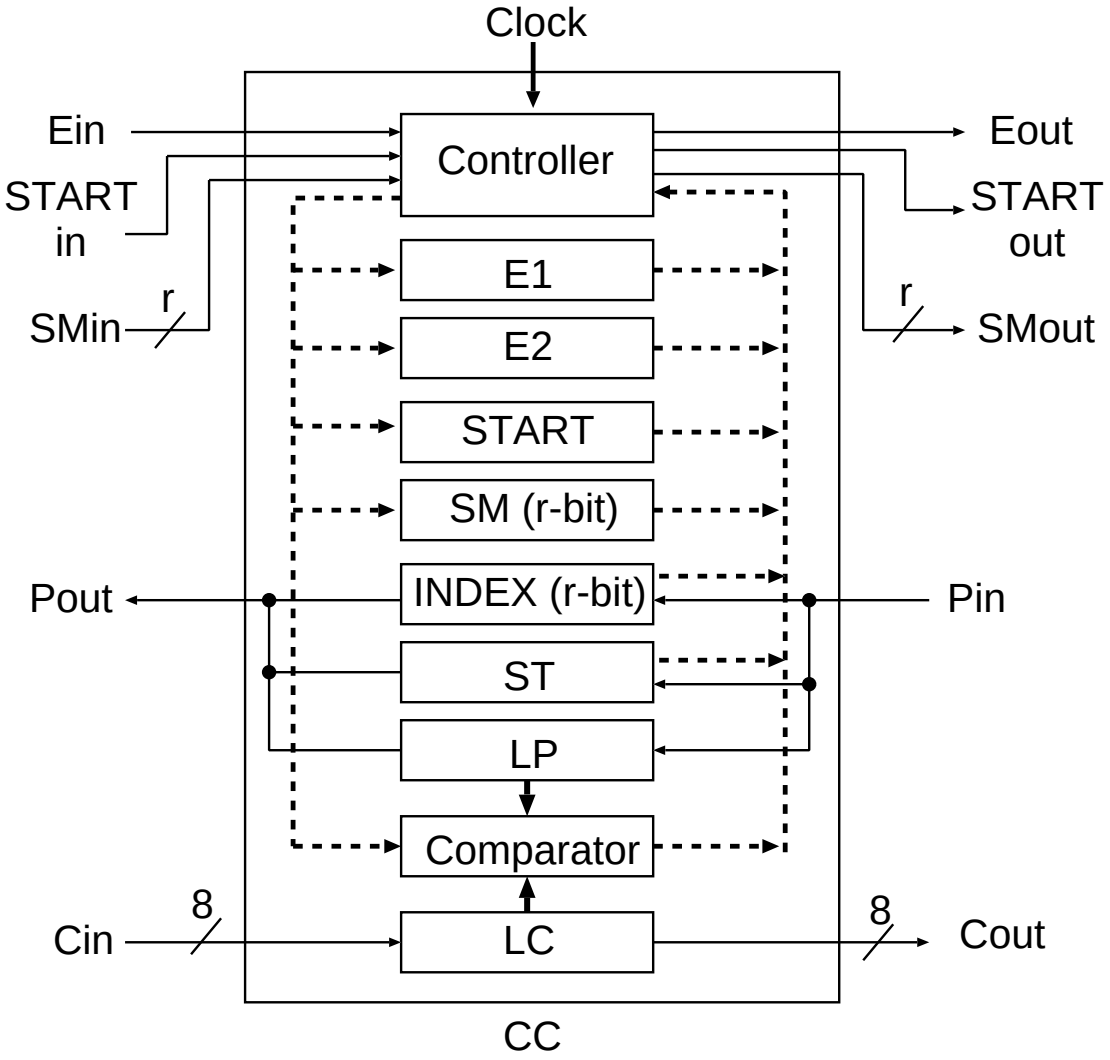


図 4.2: CC の構成

スタで，LP は SE_i を構成する文字 e^i_j を記憶し，LC はテキスト内の 1 文字を記憶する．E1 と E2 は文字 e^i_j の比較成功を表す信号 $Eout$ を 2 クロックサイクル後に右隣の CC に伝えるための 1 ビットレジスタである．START はマッチング開始信号 $START$ を STU に伝えるための 1 ビットレジスタであり，SM は SM_i を STU に伝えるための r ビットレジスタである．r は提案ハードウェアで扱うことができる R に対する遷移文字列の最大要素数（STDPA の状態数）である．ST は CC の動作モードを指定する 3 ビットレジスタである．CC の動作モードについては以下で説明する．INDEX は CC が担当する遷移文字列 SE_i の番号 i を示す r ビットレジスタである．LP と ST，INDEX レジスタの値をマッチング開始前に設定することで，任意の遷移文字列を扱うことができる．

次に CC の動作について説明する．各 CC は設定された文字に依存して 5 通りの動作

モードを持つ.

[$e^i_k(k = 1\ \&\ k = l_i)$[1]**が設定された CC（モード 1）**] e^i_k=$\mathrm{t}_j(1 \le j \le m)$ ならば，2 クロックサイクル後に右隣の CC に SM_i=1 を出力する.

[$e^i_k(k = 1\ \&\ k \neq l_i)$ **が設定された CC（モード 2）**] e^i_k=t_j ならば，2 クロックサイクル後に右隣の CC に $Eout$=1 を出力する．ここで $Eout$=1 は右隣の CC の Ein=1 を意味する.

[$e^i_k(k \neq 1\ \&\ k = l_i)$ **が設定された CC（モード 3）**] Ein=1 が入力されかつ，e^i_k=$\mathrm{t}_j(1 \le j \le m)$ ならば，2 クロックサイクル後に右隣の CC に SM_i=1 を出力する.

[$e^i_k(k \neq 1\ \&\ k \neq l_i)$ **が設定された CC（モード 4）**] Ein=1 が入力されかつ，e^i_k=t_j ならば，2 クロックサイクル後に右隣の CC に $Eout$=1 を出力する.

[ε **が設定された CC（モード 5）**] 比較動作は行わず，$Eout$=0 を出力する.

[全ての CC に共通の動作] $START$=1 または SM_i=1 が入力されると，次のクロックサイクルで $START$=1 または SM_i=1 を右隣の CC に出力する．ここで，ε が設定された CC はこの動作のみ行うことに注意する.

図 4.3 に正規表現 “$ab(c)^*de$” とテキスト “abccdefg” のアンカーモードマッチングに対する SMU の動作例を示す．例において，s=5，r=3 としている．図は各クロックサイクルにおける各 CC の出力信号と保持しているテキスト内の 1 文字を示している．クロックサイクル T_1 で，CC_1 にテキストの 1 番目の文字 ‘a’ と $START$=1 が入力される．CC の共通の動作より，クロックサイクル T_2 で，$START$=1 を CC_2 に出力する．この $START$ は，クロックサイクル T_6 で STU に入力される．また CC_1 は e^i_1 が設定されているため，クロックサイクル T_1 で ‘a’ の比較を行う．比較に成功するので，2 クロックサイクル後に CC_2 に $Eout$ を出力する．クロックサイクル T_3 で CC_2 に 2 文字目の ‘b’ と Ein が入力され，‘b’ の比較が行われる．‘b’ の比較成功は，遷移文字列 “ab” の比較が成功したことを意味する．そのため，クロックサイクル T_5 で SM_1 を右隣の CC に出力する．この SM_1 は，クロックサイクル T_8 で STU に入力される．他の遷移文字列に対するマッチングについても同様である.

CC の動作から SMU は以下の事実が成り立つ.

[事実 1] アンカーモードにおいて，SMU が t_1 を出力するクロックサイクル T_{s+1} においてのみ $START$=1 を STU に出力する．アンアンカーモードにおいて，全てのクロックサイクルで $START$=1 を STU に出力する.

[事実 2] テキスト T の任意の部分語 $U \preceq T$ について，U が SE_i と一致するとき（すなわち，$U = SE_i = e^i_1 e^i_2 \ldots e^i_{l_i} = u_1 u_2 \ldots u_{l_i}$），SMU が u_{l_i} を出力するクロックサイクルの次

[1] SE_i が 1 文字のみの場合

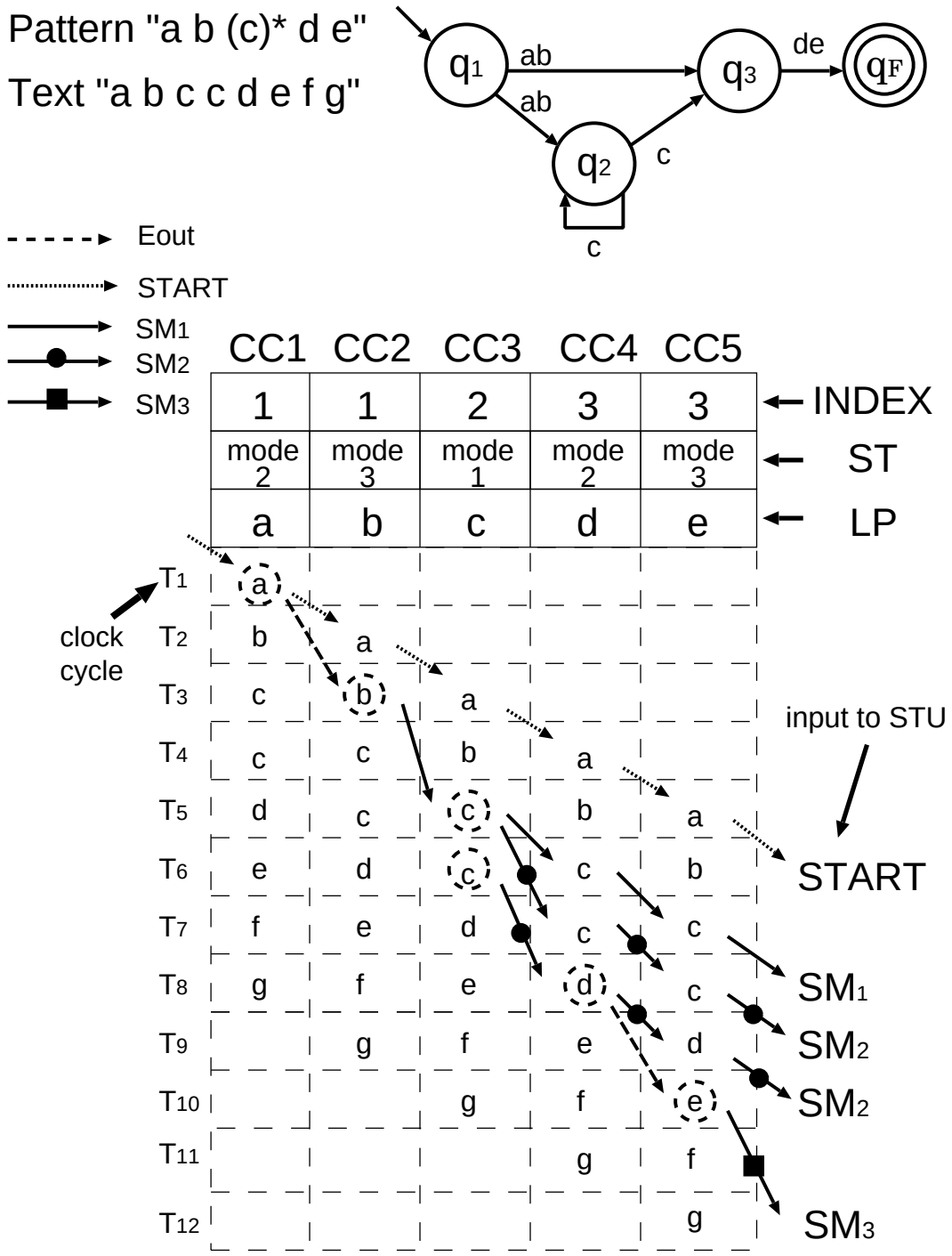


図 4.3: SMU の動作例

のクロックサイクルで SM_i=1 を STU に出力する.

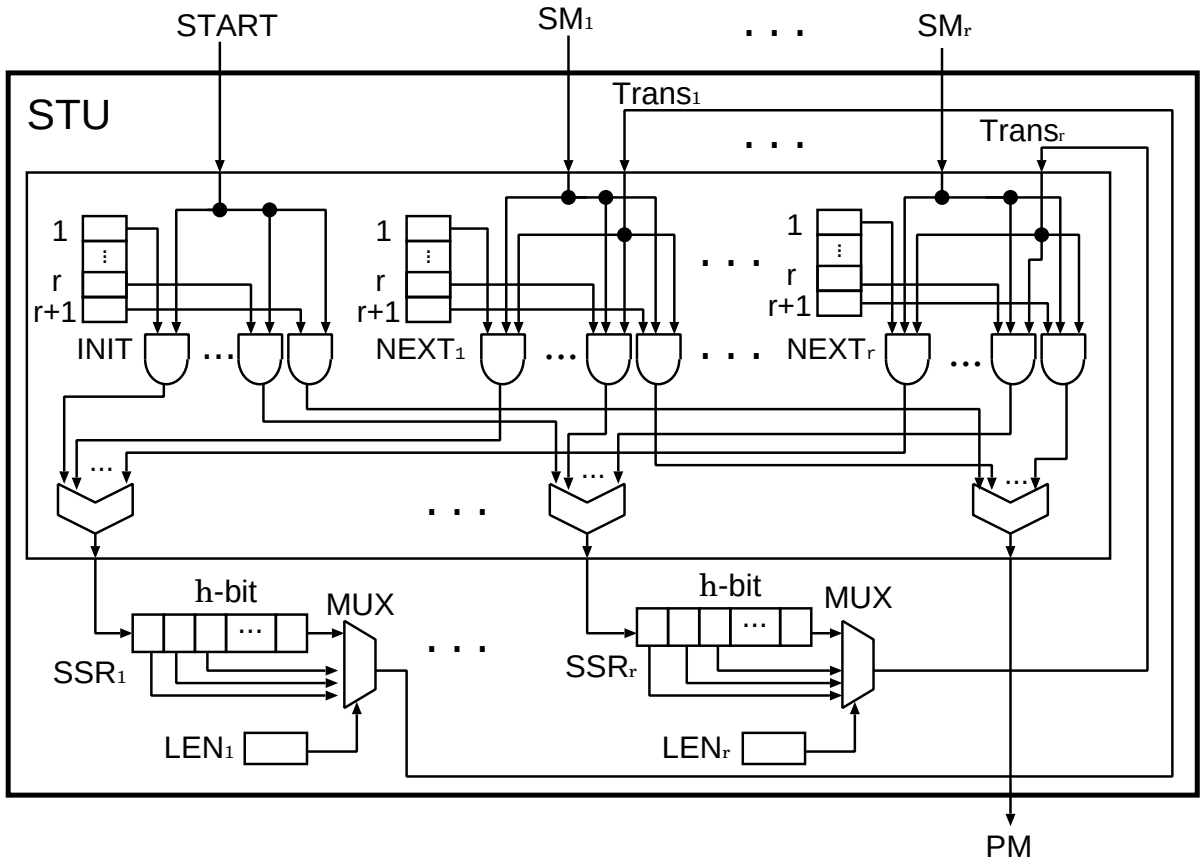


図 4.4: STU の構造

4.3　状態遷移ユニット（STU）

本節では，状態遷移を管理する状態遷移ユニット（STU）について説明する．STU は SMU から $START$=1 と SM_i=1 が入力され，それらの信号に基づき STDPA の状態遷移をシミュレートする．受理状態に達したとき，パターンマッチ信号 PM=1 を出力する．

STU の構造を図 4.4 に示す．STU は，r 個のシフトレジスタ SSR と r 個のレジスタ LEN，r 個のレジスタ $NEXT$，レジスタ $INIT$ から構成される．r は提案ハードウェアで扱うことができる STDPA の状態数の最大値である．

$INIT$ と $NEXT_i(1 \leq i \leq r)$ はそれぞれ初期状態と状態 q_i の次状態を示すための r+1 ビットレジスタである．LEN_i は q_i の遷移文字列の長さ $|SE_i|$=l_i を示すための $\lceil log_2 h \rceil$ ビットレジスタである．ここで，h は提案ハードウェアで許容する遷移文字列の最大文字数である．LEN_i と $INIT$，$NEXT_i$ の値を設定することで，STDPA を表現する．例として，

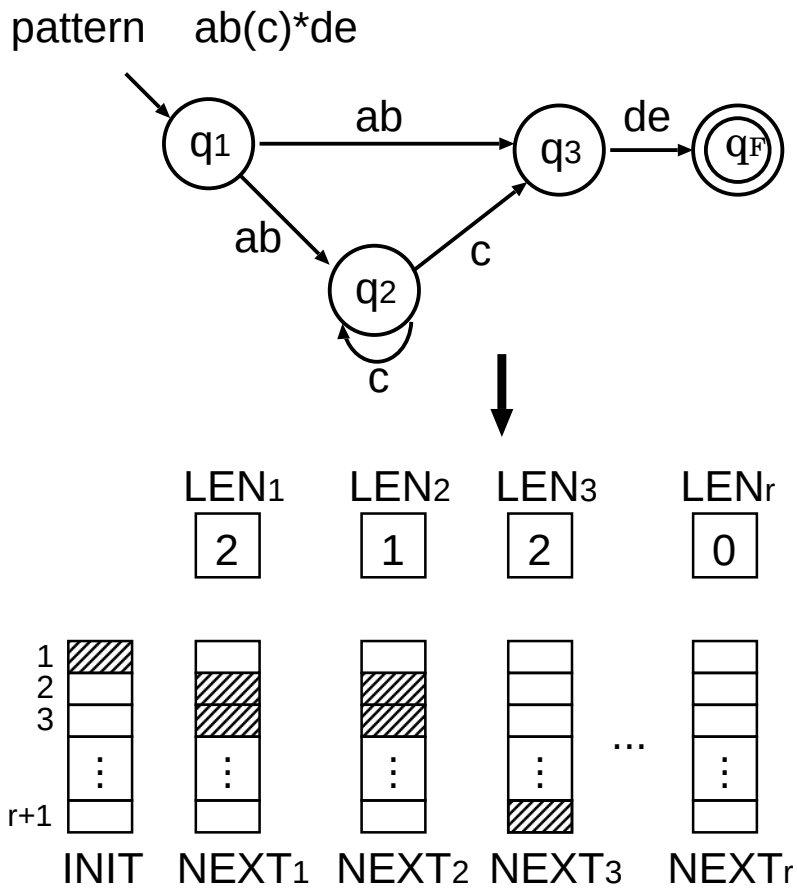


図 4.5: STDPA と各レジスタの値

STDPA に対する各レジスタの値を図 4.5 に示す.

SSR は現状態を記憶するための h ビットシフトレジスタである．STDPA の状態 q_i に対応する STU の SSR を SSR_i で表し，SSR_i の各ビットを $SSR_i[1]$，$SSR_i[2]$，...，$SSR_i[h]$ で表す．シフト動作は右シフトであり，クロックに同期して $SSR_i[p]$ の値が $SSR_i[p{+}1]$ にセットされる．LEN_i で示す SSR_i のビットの値すなわち $SSR_i[l_i]$ の値は 1 クロックサイクル後に $Trans_i$ 信号にセットされる．$SSR_i[1]$ の値が 1 にセットされる条件は以下のどちらかを満たすクロックサイクルである．**(1)** 与えられた STDPA A において $q_i \in Q_s$（初期状態）でありかつ，$START{=}1$．**(2)** 以下の論理式の値が 1 である：$OR(AND(SM_{j_1}, Trans_{j_1}), AND(SM_{j_2}, Trans_{j_2}), \ldots, AND(SM_{j_s}, Trans_{j_s}))$，ただし A において $Next(q_{j_1}) = Next(q_{j_2}) = \ldots = Next(q_{j_s}) = q_i$ である．論理式において OR と AND はそれぞれ論理和と論理積を意味する.

条件 (2) において，$Next(q_{j_1}) = Next(q_{j_2}) = \ldots = Next(q_{j_s}) = q_f$（受理状態）の場合,

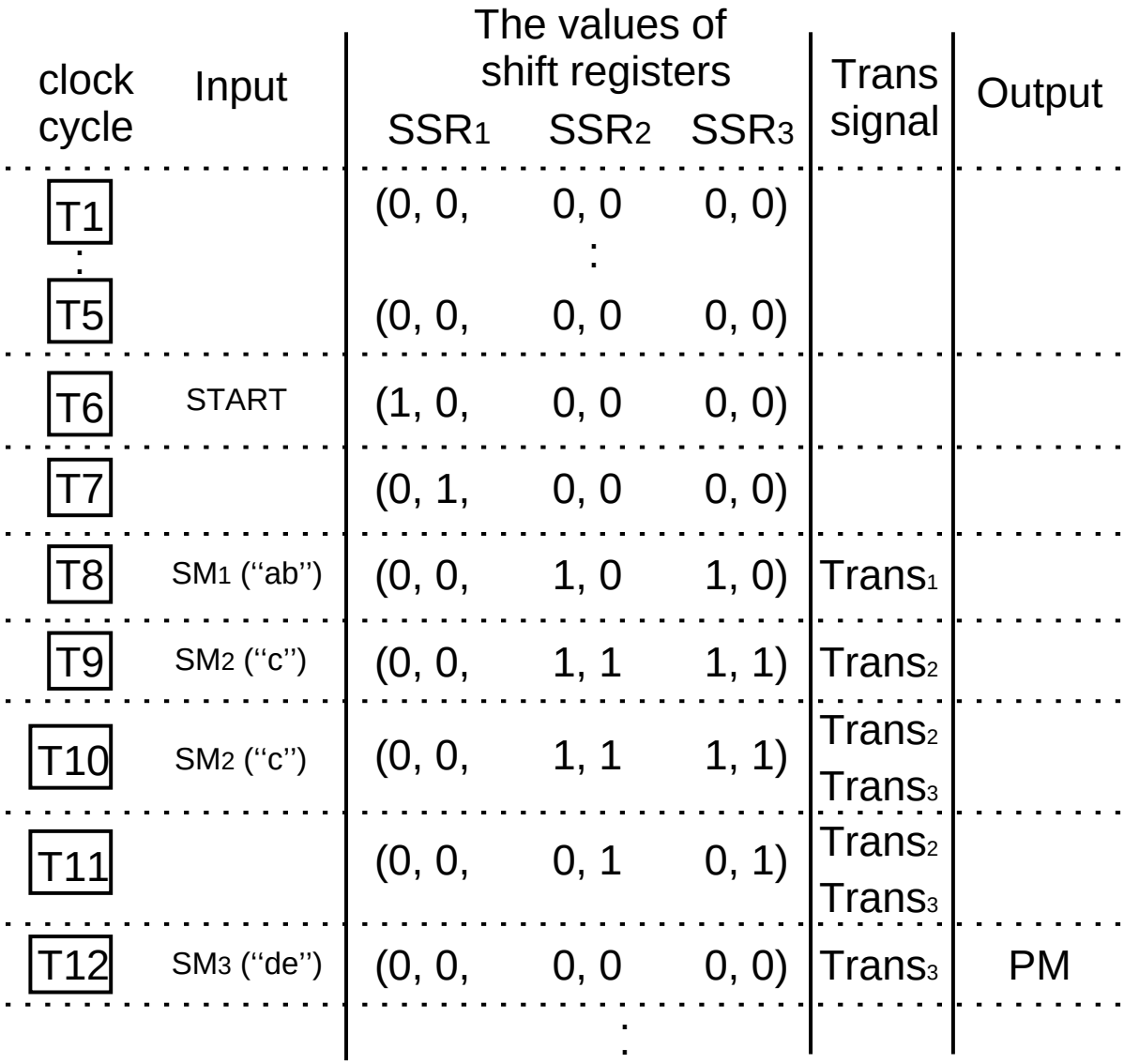


図 4.6: STU の動作例

パターンマッチ信号 PM=1 を出力する．

上記の動作より，STU は以下の事実が成り立つ．

[事実 3] クロックサイクル T_z において $SSR_i[1]$=1 であるとき，クロックサイクル T_{z+l_i} で SM_i=1 であれば，$SSR_j[1]$=1 となる．ただし，$Next(q_i)$=q_j，$Label(q_i)$=SE_i，$|SE_i|$=l_i とする．

図 4.6 に図 4.3 で示した SMU の入力に対する STU の動作例を示す．例において，r=3, h=2 としている．また $NEXT_i$ と $INIT$，LEN_i の値は図 4.5 で示したものと同じである．図 4.6 は各クロックサイクルにおける各 SSR の値をタプルで表現している．クロックサ

イクル T_1 から T_5 において STU に SMU からの入力がないため，状態遷移は起こらず，各 SSR の値は変わらない．クロックサイクル T_6 で $START$=1 が入力される．このとき，状態 q_1 が初期状態であるため，$SSR_1[1]$ が 1 となる．クロックサイクル T_7 で $SSR_1[2]$ が 1 となる．LEN_1=2 であるため，クロックサイクル T_8 で，$Trans_1$=1 となる．同クロックサイクルで，SM_1=1 であるため q_1 の次状態 q_2 と q_3 に対応する $SSR_2[1]$ と $SSR_3[1]$ が 1 となる．以降同様に遷移し，クロックサイクル T_{12} で PM=1 を出力する．これは，部分テキスト “abccde” に対するマッチング成功を示している．

4.4　基本アーキテクチャの正当性

前節で示した基本アーキテクチャの動作が STDPA の動作を正しくシミュレートすることを示す．

[補題] STDPA A において，現状態が q_s で，Σ 上の文字列 U を入力後，状態 q_t に遷移するとき，かつ，その時に限り，基本アーキテクチャ M において，クロックサイクル T_{i+1} で $SSR_s[1]$=1 であり，文字列 U を入力後，クロックサイクル $T_{i+1+|U|}$ において $SSR_t[1]$=1 である．

（証明）STDPA A の状態遷移と文字列 U を以下のように仮定する．

$\delta(q_{i_0}, U_1)$=q_{i_1}，$\delta(q_{i_1}, U_2)$=q_{i_2}，…，$\delta(q_{i_{r-1}}, U_r)$=q_{i_r}，U=$U_1U_2 \ldots U_r$．

r に関する帰納法で補題の証明を行う．

(r=1) A において q_{i_0} は初期状態で，状態遷移は $\delta(q_{i_0}, U_1)$=q_{i_1} である．アンカーモードに対して，事実 1 よりクロックサイクル T_{s+1} で $START$=1 であり，$SSR_{i_0}[1]$=1 である．U_1 の最後の文字は SMU からクロックサイクル $T_{s+|U_1|}$ で出力されるので，事実 2 より，クロックサイクル $T_{s+1+|U_1|}$ で SM_{i_0}=1 である．ただし，U_1=SE_{i_0} である．よって事実 3 より，クロックサイクル $T_{s+1+|U_1|}$ において，$SSR_{i_1}[1]$=1 となる．アンアンカーモードに対して，全てのクロックサイクルで $SSR_{i_0}[1]$=1 である．U_1=SE_{i_0} であるので，事実 2 より，あるクロックサイクル $T_{i+1+|U_1|}$ で SM_{i_0}=1 である．よって事実 3 より，クロックサイクル $T_{i+1+|U_1|}$ において，$SSR_{i_1}[1]$=1 となる．逆も同様に証明できる．

(r=k+1) r=k で補題が成り立つと仮定し，r=k+1 の証明を行う．帰納法の仮定より，A において状態 q_{i_0} から $U_1U_2 \ldots U_k$ が入力されると状態 q_{i_k} に遷移している．また，M においてはクロックサイクル T_{i+1} で $SSR_{i_0}[1]$=1 であり，文字列 $U_1U_2 \ldots U_k$ を入力後，クロッ

クサイクル T_{i+1+L_k} において $SSR_{i_k}[1]$=1 である．ただし，L_k=$|U_1U_2 \ldots U_k|$ である．A において U_{k+1} が入力されると $q_{i_{k+1}}$ に遷移する．ただし，$\delta(q_{i_k}, U_{k+1}) = q_{i_{k+1}}$ である．一方，M において U_{k+1} が入力されると，事実 2 からクロックサイクル $T_{i+1+L_k+|U_{k+1}|}$ で，SM_{i_k}=1 となる．ただし，U_{k+1}=SE_{i_k} である．よって，事実 3 からクロックサイクル $T_{i+1+L_k+|U_{k+1}|}$ において，$SSR_{i_{k+1}}[1]$=1 となる．逆も同様に証明できる．

[定理] STDPA A が入力テキスト T の部分系列 U を受理するとき，かつ，そのときに限り，M は U に対してパターンマッチ信号 PM=1 を出力する．

（証明）補題において，q_{i_0} を初期状態，q_{i_r} を受理状態とすればよい．

4.5　提案ハードウェアの拡張

4.5.1　比較セルの拡張

NIDS の代表的なシステムである Snort [26] においては，正規表現を用いたルール（ウィルスパターン）の記述を容易にするため，正規表現の基本演算子（連接，ユニオン，クリーネ閉包）に加えて，様々な演算子の追加や記法の拡張を行なっている．本節で提案する正規表現マッチングハードウェアにおいては，NIDS や文献データベースにおけるテキスト全文検索などの応用において一般的に使われる記法として文字クラスを導入し，4.2 節で提案した比較セル（CC）を文字クラスに対して拡張する．

アルファベット $\Sigma = \{a_1, a_2, \ldots, a_s\}$ に対し，Σ の文字クラスの集合を Σ_{class} とし，提案マッチングハードウェアの正規表現パターンを $\Sigma \cup \Sigma_{class}$ 上で定義される正規表現とする．すなわち，SE_i を構成する各文字 e^i_j を $e^i_j \in \Sigma \cup \Sigma_{class}$ とする．文字クラスを導入することでパターンの記述が容易になる．また，文字クラスを導入することで，正規表現の基本演算子のみを使ってパターンを記述する場合と比較して，状態遷移ユニット STU の状態遷移数の大幅な削減も可能になる．

文字クラスを扱えるようにするために，4.2 節で説明した比較セル（CC）を拡張する．文字クラス “$[p_1p_2 \ldots p_k]$”（$1 \leq k$, $p_i \in \Sigma$, $1 \leq i \leq k$）あるいはその否定 “$[\hat{}\ p_1p_2 \ldots p_k]$” に対してそれぞれ k 個の CC を用いてマッチングを実現する．また，“$[a_i$-$a_j]$”（$i < j$）あるいはその否定 “$[\hat{}a_i$-$a_j]$” については，それぞれ 2 個の CC を用いてマッチングを実現する．クラス文字とテキスト文字が一致したかどうかの判定は p_k または，a_j が設定された CC で行われる．

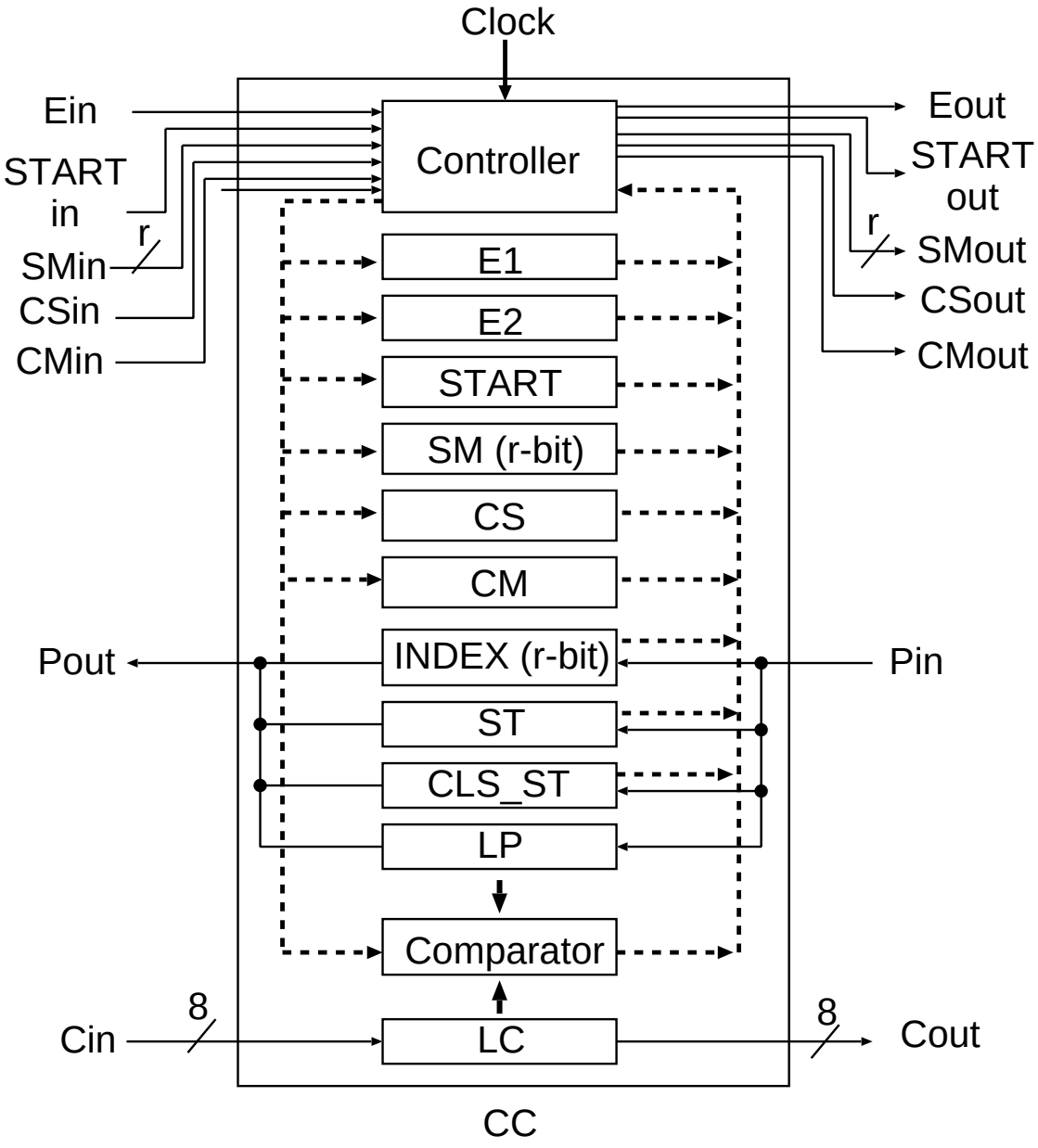


図 4.7: 拡張 CC の構成

拡張した CC の構造を図 4.7 に示す．文字クラスを扱うため，図 4.2 の CC にレジスタ CS と CM，CLS_ST を追加し，比較器 $Comparator$ が $=$，$\leq$，$\geq$ に対する結果を出すように拡張している．CS は文字クラスの比較開始を p_2 から p_k が設定されている CC に伝えるための 1 ビットレジスタである．CM は p_1 から p_{k-1} のどれかの文字クラスがテキスト文字と一致したことを p_k が設定された CC に伝えるためまたは，テキスト文字 $\mathrm{t}_q(1 \leq q \leq m)$ が $\mathrm{t}_q \geq a_i$ であることを a_j が設定されている CC に伝えるための 1 ビットレジスタである．CLS_ST は文字クラスに対する動作モードを指定するための 4 ビットレジスタである．

次に文字クラスに対する動作モードを以下に示す．動作モードは 9 つあり，各モード内においても 4.2 節で示したモード（基本モードと呼ぶ）によって動作が異なる．また 4.2 節で示した共通の動作は拡張 CC でも同じである．

[否定文字クラスである $p_i(i = 1\ \&\ i = k)$ が設定された CC[2]（モード 1）]

{ 基本モード 1}

$p_i \neq \mathrm{t}_q(1 \leq q \leq m)$ ならば，2 クロックサイクル後に右隣の CC に SM_i=1 を出力する．

{ 基本モード 2}

$p_i \neq \mathrm{t}_q(1 \leq q \leq m)$ ならば，2 クロックサイクル後に右隣の CC に $Eout$=1 を出力する．

{ 基本モード 3}

Ein=1 が入力されかつ，$p_i \neq \mathrm{t}_q(1 \leq q \leq m)$ ならば，2 クロックサイクル後に右隣の CC に SM_i=1 を出力する．

{ 基本モード 4}

Ein=1 が入力されかつ，$p_i \neq \mathrm{t}_q(1 \leq q \leq m)$ ならば，2 クロックサイクル後に右隣の CC に $Eout$=1 を出力する．

[$p_i(i = 1\ \&\ i \neq k)$ が設定された CC（モード 2）]

{ 基本モード 1,2}

常に，1 クロックサイクル後に右隣の CC に CS=1 を出力する．また，p_i=$\mathrm{t}_q(1 \leq q \leq m)$ ならば，1 クロックサイクル後に右隣の CC に CM=1 を出力する．

{ 基本モード 3,4}

Ein=1 が入力されるならば，1 クロックサイクル後に右隣の CC に CS=1 を出力する．また，Ein=1 が入力されかつ，p_i=$\mathrm{t}_q(1 \leq q \leq m)$ ならば，1 クロックサイクル後に右隣の CC に CM=1 を出力する．

[否定文字クラスでない $p_i(i \neq 1\ \&\ i = k)$ が設定された CC[3]（モード 3）]

{ 基本モード 1,3}

CS=1 が入力されかつ，p_i=$\mathrm{t}_q(1 \leq q \leq m)$ または，CM=1 が入力されるならば，2 クロックサイクル後に右隣の CC に SM_i=1 を出力する．

[2]すなわち，[^ p_1] に対して p_1 が設定された CC
[3]すなわち，[$p_1 p_2 \ldots p_k$] に対して p_k が設定された CC

{ 基本モード 2,4}

CS=1 が入力されかつ，p_i=$\mathrm{t}_q(1 \leq q \leq m)$ または，CM=1 が入力されるならば，2 クロックサイクル後に右隣の CC に $Eout$=1 を出力する．

[否定文字クラスである $p_i(i \neq 1 \ \& \ i = k)$ が設定された CC[4]（モード 4）]

{ 基本モード 1,3}

CS=1 が入力されかつ，$p_i \neq \mathrm{t}_q(1 \leq q \leq m)$ かつ，CM=1 が入力されないならば，2 クロックサイクル後に右隣の CC に SM_i=1 を出力する．

{ 基本モード 2,4}

CS=1 が入力されかつ，$p_i \neq \mathrm{t}_q(1 \leq q \leq m)$ かつ，CM=1 が入力されないならば，2 クロックサイクル後に右隣の CC に $Eout$=1 を出力する．

[$p_i(i \neq 1 \ \& \ i \neq k)$ が設定された CC（モード 5）]

{ 基本モード 1,2,3,4}

CS=1 が入力されるならば，1 クロックサイクル後に右隣の CC に CS=1 を出力する．また，CS=1 が入力されかつ，p_i=$\mathrm{t}_q(1 \leq q \leq m)$ または，CM=1 ならば，1 クロックサイクル後に右隣の CC に CM=1 を出力する．

[a_i が設定された CC（モード 6）]

{ 基本モード 1,2}

常に，1 クロックサイクル後に右隣の CC に CS=1 を出力する．また，$a_i \leq \mathrm{t}_q(1 \leq q \leq m)$ ならば，1 クロックサイクル後に右隣の CC に CM=1 を出力する．

{ 基本モード 3,4}

Ein=1 が入力されるならば，1 クロックサイクル後に右隣の CC に CS=1 を出力する．また，Ein=1 が入力されかつ，$a_i \leq \mathrm{t}_q(1 \leq q \leq m)$ ならば，1 クロックサイクル後に右隣の CC に CM=1 を出力する．

[否定文字クラスでない a_j が設定された CC[5]（モード 7）]

{ 基本モード 1,3}

CS=1 と CM=1 が入力されかつ，$a_j \geq \mathrm{t}_q(1 \leq q \leq m)$ ならば，2 クロックサイクル後

[4]すなわち，[ˆ $p_1 p_2 \ldots p_k$] に対して p_k が設定された CC
[5]すなわち，$[a_i - a_j]$ に対して a_j が設定された CC

に右隣の CC に SM_i=1 を出力する．

{ 基本モード 2,4}

CS=1 と CM=1 が入力されかつ，$a_j \geq \mathrm{t}_q (1 \leq q \leq m)$ ならば，2 クロックサイクル後に右隣の CC に $Eout$=1 を出力する．

[否定文字クラスである a_j が設定された CC[6]（モード 8）]

{ 基本モード 1,3}

CS=1 が入力されかつ，CM=1 が入力されないまたは，$a_j < \mathrm{t}_q (1 \leq q \leq m)$ ならば，2 クロックサイクル後に右隣の CC に SM_i=1 を出力する．

{ 基本モード 2,4}

CS=1 が入力されかつ，CM=1 が入力されないまたは，$a_j < \mathrm{t}_q (1 \leq q \leq m)$ ならば，2 クロックサイクル後に右隣の CC に $Eout$=1 を出力する．

[文字クラス以外の文字が設定された CC（モード 9）]

基本モードを行う．

図 4.8 に正規表現 “$a[b-z](c)^*de$” とテキスト “axccdefg” のアンカーモードマッチングに対する拡張 SMU の動作例を示す．この例で用いる正規表現は図 4.3 で示した正規表現と比べ，‘b’ を “$[b-z]$” に変更している．ここでは，CC_2 と CC_3 の動作のみ説明する．クロックサイクル T_3 で CC_2 に 2 文字目の ‘x’ と Ein が入力される．CC_2 はモード 6，基本モード 3 であるので，クロックサイクル T_4 で $CS = 1$ を右隣の CC に出力する．また，$b \leq$ ‘x’ であるので，1 クロックサイクル後に右隣の CC に CM=1 を出力する．クロックサイクル T_4 で CC_3 に 2 文字目の ‘x’ と CS=1，CM=1 が入力される．CC_3 はモード 7，基本モード 3 である．そのため，$z \geq$ ‘x’ であるため，2 クロックサイクル後に右隣の CC に SM_1=1 を出力する．

図 4.3 の動作例と比較し，STU に伝わる全ての信号が 1 クロックサイクルずれているだけである．文字クラスはテキストの 1 文字にマッチングする点では通常の文字とのマッチングと同じであるため，SMU を上記のように文字クラスに対して拡張したセルで構成しても，4.4 節で示した提案ハードウェアの動作の正当性の証明には影響を与えない．従って，拡張したセルを用いた提案ハードウェアは文字クラスを含む正規表現マッチングを実現する．

[6]すなわち，$[\hat{}\ a_i - a_j]$ に対して a_j が設定された CC

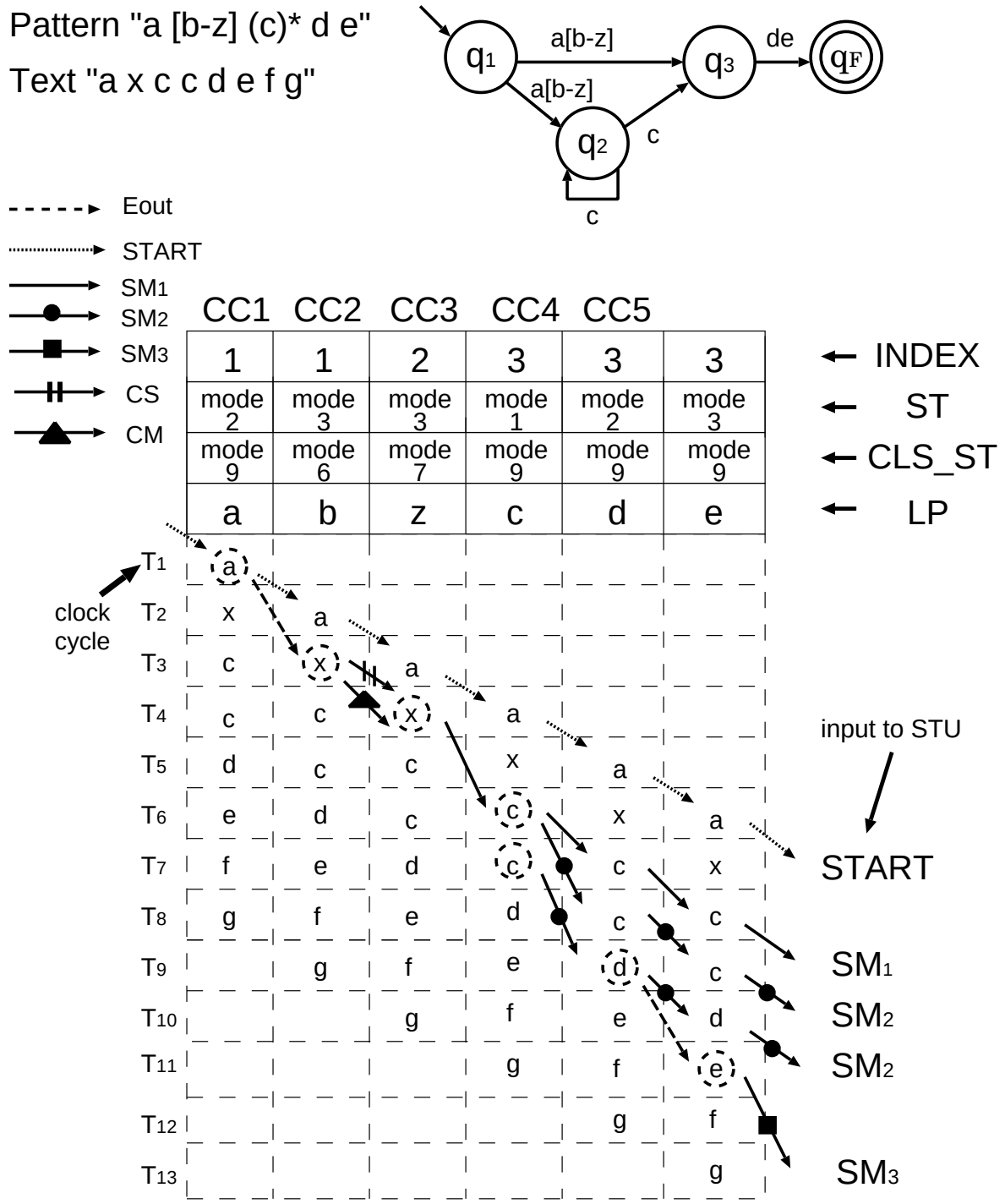


図 4.8: 拡張 SMU の動作例

4.5.2　システムの全体構成

本節では，提案ハードウェアの全体構成について説明する．NIDS 等の正規表現マッチングの応用においては，正規表現パターン R は $R{=}R_1R_2 \ldots R_m$ のように部分表現 R_i の

連接で表されるとことが一般的である．このような応用においては，R 全体を基本アーキテクチャで実現すると回路規模の点で無駄が多い．そこで入力正規表現パターンとして部分表現の連接を仮定し，それに合ったシステム構成を提案する．提案ハードウェアの全体構成を図 4.9 に示す．提案ハードウェアは 4.1 節で示した基本アーキテクチャ（図 4.1）（以下ではマッチングモジュール MM と呼ぶ）を m 個直列に並べた構成となる．例えば，R=R_1R_2 の場合，R_1 と R_2 に対してそれぞれ MM_1 と MM_2 を用いてマッチングを行うことができる．その際，MM_1 の STU からのパターンマッチング信号 PM_1 は，MM_2 の CC_1 の $STARTin$ に入力される．

また各マッチングモジュールの $STARTin$ 入力と text 入力は前段のマッチングモジュールのパターンマッチング信号 PM_{i-1} とテキスト出力 $\mathrm{text\_out}_{i-1}$，と全体のスタート信号 $START$ と全体の text 入力をセレクタで切り替えるようにしている．これによって R=$R_1|R_2|\ldots|R_m$ のような部分表現のユニオンも実現可能にしている．

提案ハードウェアは FPGA 上に実装する．FPGA を用いることで，提案ハードウェアは様々な正規表現パターンに対して柔軟に対応できる．提案ハードウェアの基本アーキテクチャは状態数の最大値 r，遷移文字列の最大文字数 h，CC 数という 3 つのパラメータを持つ．提案ハードウェアはこれらのパラメータによって扱える正規表現パターンの長さが制限される．そこで，あらかじめ様々なパラメータに対する提案ハードウェアを FPGA のビットファイルとして作成しておき，与えられた正規表現に対して適切なビットファイルを FPGA 上に実装する．

正規表現パターンを提案ハードウェアに設定する際，変換ソフトウェアによって正規表現パターンを提案ハードウェアで扱えるパターン形式に変換して設定を行う．

4.6　実験的評価

本節では提案パターン非依存正規表現マッチングハードウェアの評価を行う．

論理合成ツールとして Xilinx ISE 13.1，対象 FPGA として Xilinx 社の Virtex-4 (XC4VLX200-11FF1513) を使用した．対象 FPGA の Slice 数は 89,088 個である．比較対象として，DPA に基づく正規表現マッチングハードウェア（DPA 手法 [68]）を用いる．このハードウェアは，提案ハードウェアと同様，任意の正規表現を扱うことができるパターン非依存正規表現マッチングハードウェアである．

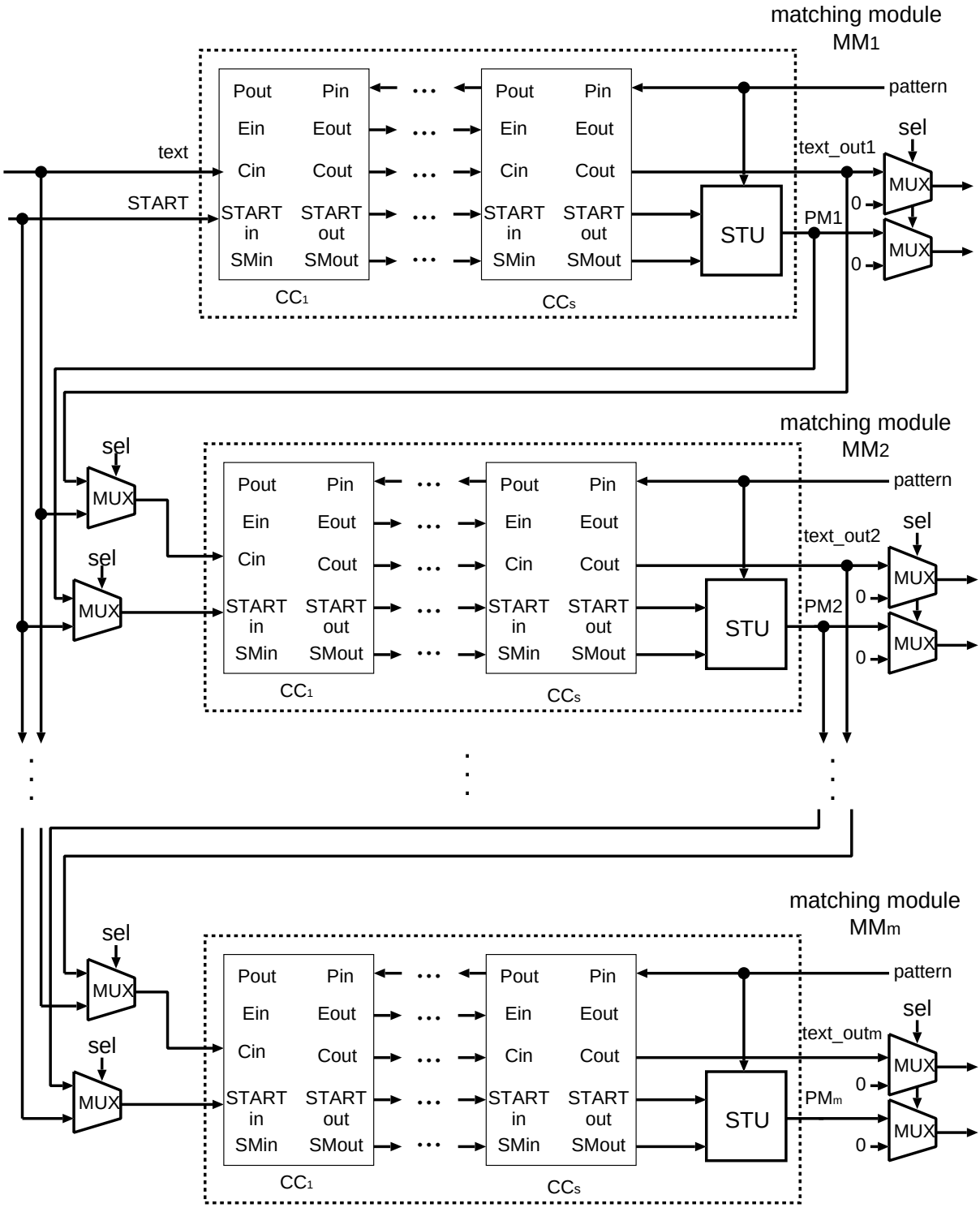


図 4.9: 提案ハードウェアの全体構成

提案ハードウェアのマッチングモジュール MM（基本アーキテクチャ）は STU の状態数の最大値 r，遷移文字列の最大文字数 h，SMU の CC 数の 3 つパラメータを持つ．表 4.1

表 4.1: 様々なパラメータに対する MM の面積と性能

CC=25				CC=50			
	r=10	r=20	r=30		r=10	r=20	r=30
h=10	1,371 (Slice) 339 (MHz)	2,156 206	3,087 232	h=10	2,484 302	3,629 208	4,920 231
h=20	1,516 (Slice) 296 (MHz)	2,461 210	3,542 209	h=20	2,649 294	3,949 243	5,385 205
h=30	1,682 (Slice) 257 (MHz)	2,742 211	3,997 201	h=30	2,814 275	4,210 210	5,850 197

CC=75				CC=100			
	r=10	r=20	r=30		r=10	r=20	r=30
h=10	3,598 339	5,103 226	6,754 196	h=10	4,713 350	6,578 210	8,589 179
h=20	3,733 298	5,428 202	7,229 219	h=20	4,898 280	6,913 214	9,074 227
h=30	3,948 250	5,679 218	7,704 207	h=30	5,083 263	7,149 235	9,559 214

と表 4.2 に様々なパラメータに対する MM と拡張 MM（CC が文字クラスを扱うことができる）の使用 Slice 数と最大動作周波数を示す．拡張 MM は MM と比べ，回路面積が約 7% 増加し，性能は約 80% に低下した．

次に NIDS として著名な Snort におけるウィルスパターン Snort ルール v2.9 (2012 年) に提案ハードウェアを適用した結果を示す．拡張正規表現（後方参照と先読み演算）を含むパターンを除いた全てのパターンに対応可能となるように MM と拡張 MM のパラメータ値を決定する．MM の場合，状態数の最大値 r は 296 となり，対象 FPGA には実装できなかった[7]．拡張 MM の場合，状態数の最大値 r : 30，遷移文字列の最大文字数 h : 25，CC 数 : 75，拡張 MM の数 : 12 のパラメータ値で全てのパターンに対応可能であった．このパラメータ持つ提案ハードウェアを対象 FPGA に実装した場合，使用 Slice 数は 89,086 個（99%）であり，最大動作周波数は 186MHz であった．比較対象として，DPA に基づくマッチングハードウェア（DPA 手法 [68]）で実現した場合，最大で 10428 個の状態が必要となり，対象 FPGA には実装できなかった[8]．よって提案ハードウェアは DPA

[7] r が 150 の時点でスライス使用率が 98% であったため

[8] 3.5 節で述べたように，DPA に基づく正規表現マッチングハードウェア（DPA 手法 [68]）において必要となるレジスタの総 bit 数は状態数の二乗オーダである．そのため，今回の場合では，約 1 億 bit のレジスタが必要となる．一方，対象 FPGA（総 Slices 数 89,088 個）の各 Slice には 2 個の 1bit レジスタが含ま

表 4.2: 様々なパラメータに対する拡張 MM の面積と性能

CC=25				CC=50			
	r=10	r=20	r=30		r=10	r=20	r=30
h=10	1,501 (Slice) 217 (MHz)	2,297 233	3,220 227	h=10	2,764 220	3,946 235	5,256 230
h=20	1,672 (Slice) 243 (MHz)	2,603 207	3,677 222	h=20	2,930 220	4,265 196	5,727 211
h=30	1,828 (Slice) 211 (MHz)	2,910 214	4,135 197	h=30	3,100 216	4,584 194	6,191 202

CC=75				CC=100			
	r=10	r=20	r=30		r=10	r=20	r=30
h=10	4,014 210	5,577 208	7,265 223	h=10	5,274 208	7,222 204	9,295 204
h=20	4,189 212	5,902 203	7,740 213	h=20	5,459 209	7,557 196	9,780 189
h=30	4,364 213	6,227 190	8,250 186	h=30	5,644 214	7,892 197	10,265 183

手法と比べてコンパクトな回路で正規表現マッチングを実現している．我々の知る限り，DPA 手法は NFA に基づいて任意の正規表現を扱うことができるパターン非依存正規表現マッチングハードウェアの唯一のものであり，提案ハードウェアはこれまでで最もコンパクトな回路構造であることが示された．また中規模サイズの FPGA でも実現可能であり，実用上，十分面積の小さい回路構造であると言える．

最後に従来研究で提案されたパターン非依存正規表現マッチングハードウェア [63–65] との性能比較を示す．表 4.3 に各ハードウェアのスループットを示す．[63, 64] は提案ハードウェアと使用デバイスが異なるので厳密な比較はできないが，[65] と比較して提案ハードウェアのスループットは 29%低い．しかし [63–65] のハードウェアはパターンとして制限された正規表現のクラスしか扱えないのに対して，提案ハードウェアは扱える正規表現のクラスに制限はないという利点がある．

れているため，約 5000 万個の Slice が必要である．よって，ハードウェアの実現に必要な Slice が大幅に不足しているため，DPA 手法を対象 FPGA に実装することは不可能である．

表 4.3: パターン非依存正規表現マッチングハードウェアの性能比較

	デバイス	スループット
マッチングハードウェア [63]	Virtex-2Pro	0.8Gbps
マッチングハードウェア [64]	Virtex-5LX300	1.6Gbps
マッチングハードウェア [65]	Virtex-4LX200	2.1Gbps
提案ハードウェア	Virtex-4LX200	1.5Gbps

4.7　まとめ

本章では，シストリックアルゴリズムと NFA を組み合わせた新しいパターン非依存正規表現マッチングハードウェアを提案した．提案マッチングハードウェアは，シストリックアルゴリズムに基づく単純文字列の比較機能と DPA に基づく状態遷移機能の組み合わせによって，文字列に対して状態遷移する STDPA をシミュレートする．また NIDS やテキスト全文検索などの応用において一般的に使われる文字クラスを直接扱うための拡張を示した．実験では，Snort ルールに基づき FPGA 上に提案ハードウェアを実装し，DPA に基づく従来パターン非依存正規表現マッチングハードウェアよりもコンパクトな回路で正規表現マッチングを実現できることを示した．

今後の課題として，提案ハードウェアの回路規模の削減があげられる．例えば，SMU の比較セル（CC）の機能を文字クラスだけではなく，CC のマッチング機能をユニオンのネストなどを許すように拡張することで，STU の回路規模を縮小することができ，全体の回路規模の削減につながる可能性がある．正規表現で遷移するオートマトン [70] も知られているため，ユニオンで遷移するオートマトンの実現も可能であると考えられる．

また他の課題として，提案ハードウェアが持つパラメータ（状態数の最大値や遷移文字列の最大文字数等）に合わせて正規表現パターンを変換するソフトウェアの開発があげられる．正規表現パターンは等価な正規表現がいくつか存在する場合があり，それらの正規表現を実現するためのパラメータはそれぞれ異なる．例えば，正規表現パターン “$a(b|cd)e$” は，最大状態数 4 と最大文字数 2 のパラメータを持つハードウェアで実現でき，等価な正規表現パターン “$(abe|acde)$” や “$(ab|acd)e$” に対してはそれぞれ，最大状態数 2 と最大文字数 4，最大状態数 3 と最大文字数 3 のパラメータを持つハードウェアで実現できる．そのため，与えられた正規表現パターンを等価な正規表現パターンに変換する事で，最大状態数と最大文字数のパラメータが小さい提案ハードウェアでも扱うことができる可能性があるので，提案ハードウェアの回路規模の削減につながると考えられる．

第5章　FPGAの部分再構成機能に基づく高速パターン更新可能な正規表現マッチングハードウェア

本章では，任意の正規表現を扱うことができ，かつ高速にパターン更新を行うことができるという利点を保ちつつ，4 章で提案した正規表現マッチングハードウェアよりも回路規模がコンパクトな FPGA の部分再構成機能に基づく正規表現マッチングハードウェアを提案する．

本章の流れとして，まず提案手法の概要を説明し，その後，提案回路の構成について述べる．最後に実験的評価を示す．

5.1　概要

4 章で提案したパターン非依存正規表現マッチングハードウェアは，任意の正規表現を扱うことができ，高速にパターンを更新することができるという利点を持つ．しかし回路規模は非常に大きく，実用化するためにはハードウェアリソースの削減が課題となる．

正規表現パターン R は 4.5.2 節で説明したように，R=$R_1R_2 \ldots R_m$ のように部分表現 R_i の連接または，R=$R_1|R_2| \ldots |R_m$ のような R_i のユニオンで表されることが一般的である．これに対して 4 章で提案したハードウェアは図 4.9 で示したように，どんな R_i に対しても全て同じ回路構成（ホモジニアス構成）でマッチングを行っている．また，R_i に対する回路はクリーネ閉包のネスト（“$(ab(c|d)*ef)*$”）のような複雑な正規表現でも扱えるように多くの機能を持っている．

このような回路構成において，R_i がアルファベット Σ と連接のみで構成される単純な部分表現である場合，R_i のマッチングを行う回路は過剰な機能を持っていると言える．本章では，各部分表現のマッチングを行う回路の構造がそれぞれ異なる構成（ヘテロジニア

ス構成）を実現し，過剰な機能を減らすことで回路規模の削減を目指す．

過剰な機能を減らす最も良い方法としては，3.2 節で説明した与えられた正規表現パターンに完全に特化した回路構成を生成する方法である．しかしこの方法だと，パターンの更新時に時間のかかる回路の再設計が必要となり高速なパターン更新を実現できない．そこで本章では，FPGA の部分再構成機能を用いた新しい手法を提案する．提案手法では，予め異なる正規表現クラスを扱う複数種類の部分回路（テンプレート）を用意しておく．正規表現パターンが与えられると，そのパターンに適したコンパクトな回路構成をテンプレートを組み合わせることで自動生成する．生成された回路構成は，変更箇所のみが FPGA 上に部分再構成される．テンプレートによる設計では，全ての組み合わせに対する全体回路をあらかじめ設計しておくことで部分再構成を用いなくても再設計を必要としないパターン更新を実現できるが，この方法では，後述するように，莫大な数の回路を設計しておく必要があるため現実的ではない．

異なる正規表現クラスを扱う複数種類のテンプレートを作成するため，[65–67] で提案されたシストリックアルゴリズムに基づくパターン非依存正規表現マッチングハードウェアを用いる．

提案手法は，与えられたパターンに適したテンプレートの選択と，変更箇所に該当する FPGA ビットファイルのダウンロードのみでパターンを更新できる．これらの処理は短時間で行うことができるため，パターンの更新を瞬時に行うことができる．また提案ハードウェアは，特定の正規表現クラスの機能のみを持つテンプレートを組み合わせて構成されるため，4 章で提案したパターン非依存正規表現マッチングハードウェアと比べ，コンパクトな回路が得られる．

5.2　提案回路構成

部分再構成により回路を書き換えるためには，FPGA 内に部分再構成領域（PRR）を予め確保し，回路を PRR とそれ以外の領域（SR）に分割する必要がある．部分再構成を用いた提案回路構成を図 5.1 に示す．提案回路構成は図 4.9 で示した各マッチングモジュール部分を PRR，マッチングモジュール間の配線を SR として FPGA 上に実装されている．各 PRR からの信号テキスト $Text$ とパターン $Pattern$，マッチング結果 PM はレジスタを挟んで次の PRR に出力される．パターン更新時には，各 PRR に実装されるテンプレートが独立に書き換わるだけで，これらのレジスタや全体の回路構成を書き換える必要は

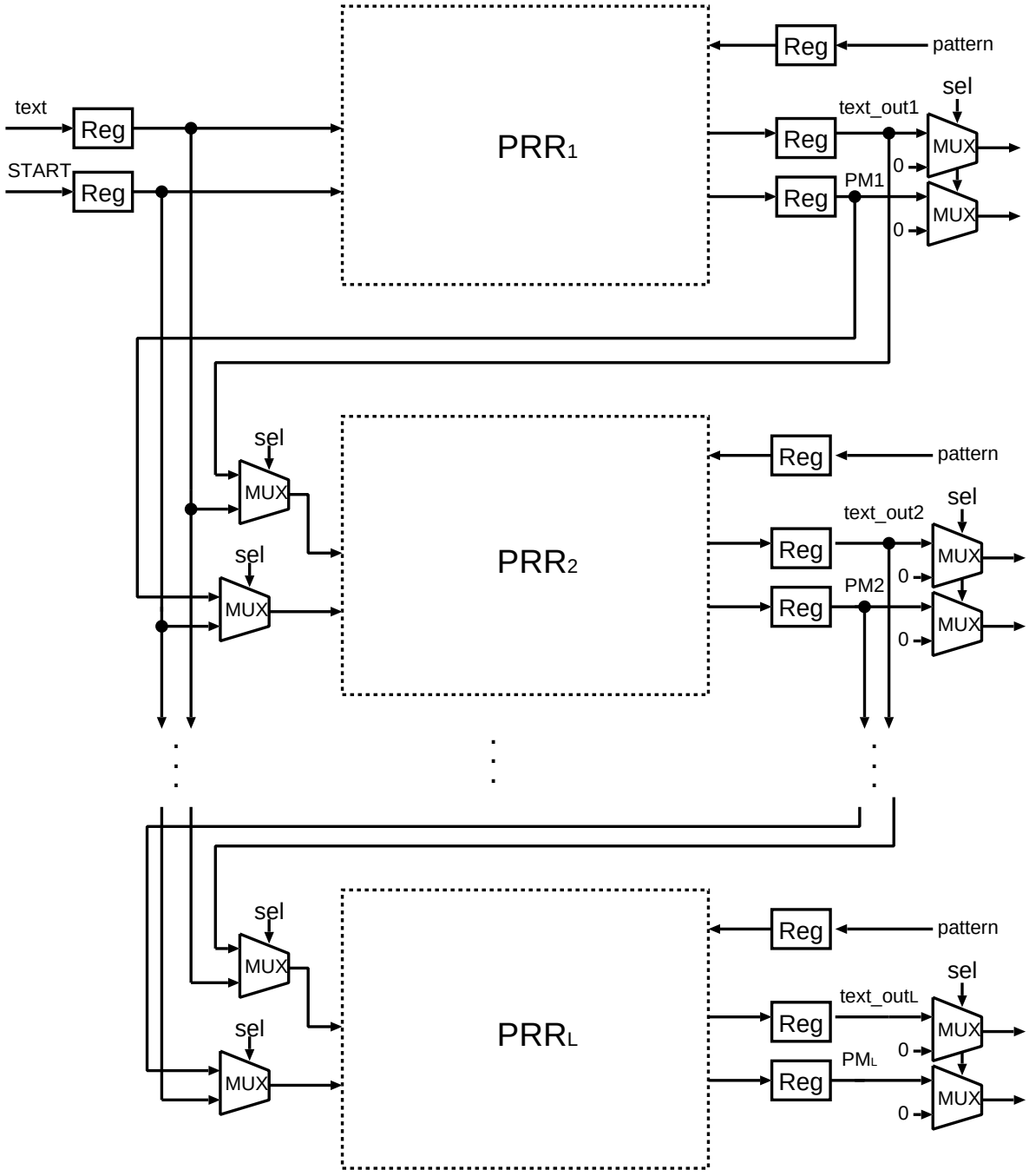


図 5.1: 部分再構成を用いた提案回路構成

ない．1 つの PRR に対して K 種類のテンプレートを用意したとすると，これらのテンプレートは全ての PRR で共通して利用できるため，わずか K 種類のテンプレートを設計するだけで[1]，K^L 種類もの異なるマッチングハードウェアを生成できる．部分再構成を用い

[1]FPGA ツールの都合上，ビットファイルは各 PRR 毎に生成する必要があるため，KL 個のビットファイルが必要となる．

表 5.1: テンプレートに用いる比較セル（CC)

タイプ	扱う正規表現クラス	パターン例	回路規模
1	文字, 任意の文字, 空語または それらに対する選択文字の連接	abc, ε, $x.z^?$	1.0
2	クラス文字の連接, 文字またはクラス文字に対するクリーネ閉包	$[a\text{-}z]b^+c$, $[\hat{}a\text{-}z]^*bc.^?$,	1.4
3	ユニオンの 2 重までのネスト	$(a\|b^*(c\|de)[f\text{-}u])$, $ab(c\|(d\|e)fg)h(i\|j)$	1.8
4	文字またはクラス文字に対する量指定子 を含むユニオンの 2 重までのネスト (10 回までの繰り返し)	$a\{10\}bc$, $(a\{2,\}b\|c^*)$ $(ab\|c[k\text{-}z]\{5,8\})$	2.3
5	文字またはクラス文字に対する量指定子 を含むユニオンの 2 重までのネスト (30 回までの繰り返し)	$a\{30\}bc$, $(a\{22,\}b\|c^*)$ $(ab\|c[k\text{-}z]\{4,15\})$	3.0

ずに全体の再構成だけで同じことを行うには，K^L 種類のテンプレートを予め設計しておく必要がある．そのため，本提案手法は，パターン更新時間だけではなく，設計時間の短縮にも有効である．

5.3　テンプレート

テンプレートの構造はシストリックアルゴリズムに基づくマッチングハードウェア（図 3.5）または，4 章で提案したマッチングモジュール（図 4.1）となる．図 3.5 の比較セル（CC）には [65–67] で提案された様々な正規表現クラスを扱う CC を用いることで，異なる正規表現クラスを扱う様々なテンプレートを作成する．

テンプレートには 5 タイプの CC を用いる．タイプ a（$1 \leq a \leq 5$）の CC が扱うことができる部分正規表現 SR_a を以下に定義する．

1. SR_1=$r_1^{S_1}r_2^{S_2}\ldots r_i^{S_i}$, $S_j \in \{?\} \cup \{\varepsilon\}$, r_i^{ε}=r_i, $r_j \in \{.\} \cup \Sigma \cup \{\varepsilon\}$, $1 \leq i$, $1 \leq j \leq i$.
2. SR_2=$r_1^{S_1}r_2^{S_2}\ldots r_i^{S_i}$, $S_j \in \{?\} \cup \{*\} \cup \{+\} \cup \{\varepsilon\}$, r_j^{ε}=r_j, $r_j \in \{.\} \cup \Sigma \cup \{\varepsilon\} \cup \Sigma_{class}$, $1 \leq i$, $1 \leq j \leq i$.
3. SR_3=$A_1A_2\ldots A_i$, $A_j \in B \cup SR_2$, B=$(C_1|C_2|\ldots|C_k)$, C_l= $D_{(l,1)}D_{(l,2)}\ldots D_{(l,m)}$, $D_{(l,n)} \in E \cup SR_2$, E=$(F_1|F_2|\ldots|F_o)$, $F_p \in SR_2$, $1 \leq i$, $1 \leq j \leq i$, $1 \leq k$, $1 \leq l \leq k$, $1 \leq m$, $1 \leq n \leq m$, $1 \leq o$, $1 \leq p \leq o$.

4. SR_4=$A_1A_2 \ldots A_i$, $A_j \in B \cup T$, B=$(C_1|C_2| \ldots |C_k)$, C_l= $D_{(l,1)}D_{(l,2)} \ldots D_{(l,m)}$, $D_{(l,n)} \in E \cup T$, E=$(F_1|F_2| \ldots |F_o)$, $F_p \in T$, T=$r_1^{S_1}r_2^{S_2} \ldots r_t^{S_t}$, $S_u \in \{?\} \cup \{*\} \cup \{+\} \cup \{\varepsilon\} \cup \{(M)\} \cup \{(M,N)\} \cup \{(M,)\}$, r_u^{ε}=r_u, $r_u^{(M)}$=$r_u\{M\}$, $r_u^{(M,N)}$=$r_u\{M,N\}$, $r_u^{(M,)}$=$r_u\{M,\}$, $r_u \in \{.\} \cup \Sigma \cup \{\varepsilon\} \cup \Sigma_{class}$, $1 \leq i$, $1 \leq j \leq i$, $1 \leq k$, $1 \leq l \leq k$, $1 \leq m$, $1 \leq n \leq m$, $1 \leq o$, $1 \leq p \leq o$, $1 \leq t$, $1 \leq u \leq t$, $0 \leq M < N \leq 10$.
5. SR_5=$A_1A_2 \ldots A_i$, $A_j \in B \cup T$, B=$(C_1|C_2| \ldots |C_k)$, C_l= $D_{(l,1)}D_{(l,2)} \ldots D_{(l,m)}$, $D_{(l,n)} \in E \cup T$, E=$(F_1|F_2| \ldots |F_o)$, $F_p \in T$, T=$r_1^{S_1}r_2^{S_2} \ldots r_t^{S_t}$, $S_u \in \{?\} \cup \{*\} \cup \{+\} \cup \{\varepsilon\} \cup \{(M)\} \cup \{(M,N)\} \cup \{(M,)\}$, r_u^{ε}=r_u, $r_u^{(M)}$=$r_u\{M\}$, $r_u^{(M,N)}$=$r_u\{M,N\}$, $r_u^{(M,)}$=$r_u\{M,\}$, $r_u \in \{.\} \cup \Sigma \cup \{\varepsilon\} \cup \Sigma_{class}$, $1 \leq i$, $1 \leq j \leq i$, $1 \leq k$, $1 \leq l \leq k$, $1 \leq m$, $1 \leq n \leq m$, $1 \leq o$, $1 \leq p \leq o$, $1 \leq t$, $1 \leq u \leq t$, $0 \leq M < N \leq 30$.

表 5.1 は各 CC が扱う部分正規表現クラスの簡単な説明とタイプ 1 の CC に対する回路規模の相対比を示している．CC の補足説明を以下に記述する．

- タイプ i の CC はタイプ $j(j < i)$ の CC で扱う正規表現クラスも扱える．
- タイプ 4 と 5 の CC は，繰り返し回数が 10 または 30 回以下の量指定子を直接扱うことができる．
- CC の回路規模が小さければ小さいほど，1 つの PRR により多くの CC を実装できる．

これら 5 タイプの CC を組み合わせてテンプレートを作成する．本研究で用意したテンプレートは 4 章で提案したマッチングモジュールのテンプレート 1 種類とタイプ 1 からタイプ 5 の CC それぞれのみから構成されるテンプレート 5 種類，タイプ 1 からタイプ 4 までの CC を 2 種類ずつ組み合わせたテンプレート ${}_4P_2$=12 種類の計 18 種類である（図 5.2）．テンプレートの特徴を以下に述べる．

特徴 1 扱う正規表現クラスが小さいテンプレートは，扱えるパターン長は長い．

特徴 2 扱う正規表現クラスが大きいテンプレートは，扱えるパターン長は短い．

与えられたパターンに対し，各テンプレートにおいて，パターンが設定されない CC には図 5.3 のように ε を設定する．

5.4　テンプレートの選択

提案手法において，与えられたパターンに対して最適なテンプレートの組み合わせを選択する必要がある．与えられたパターンに対する最適なテンプレートの選択問題を以下の

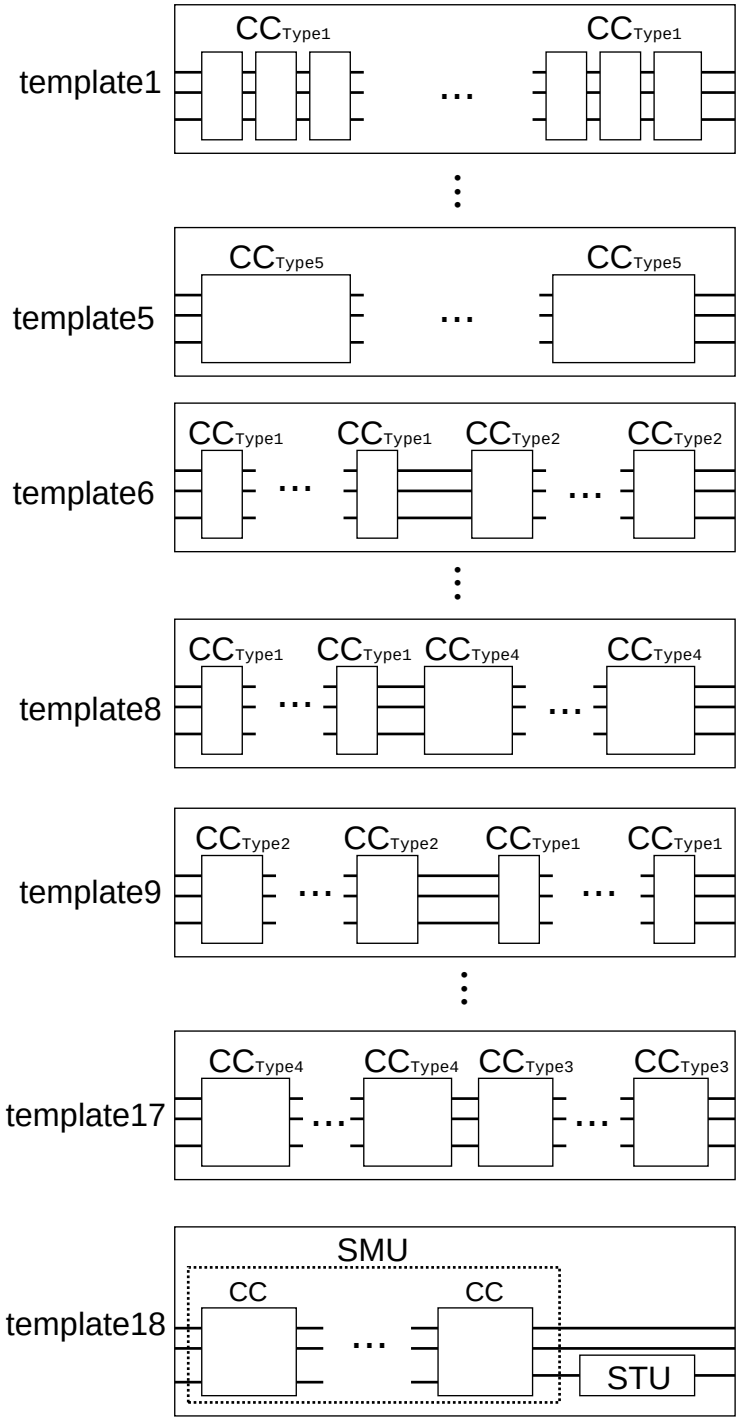


図 5.2: 用意するテンプレート

ように定式化する.

入力 正規表現パターンと L 種類のテンプレート.

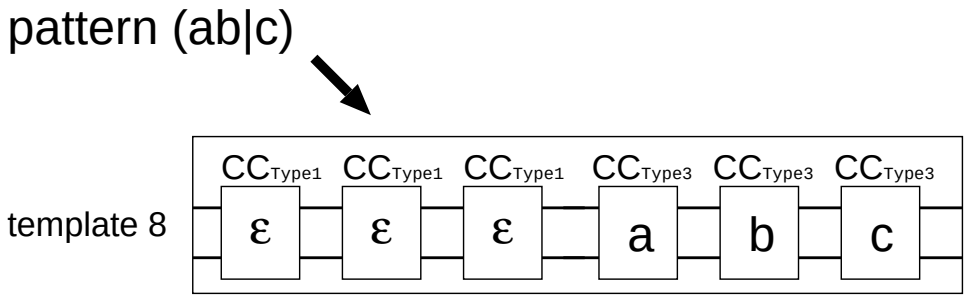


図 5.3: ε の設定

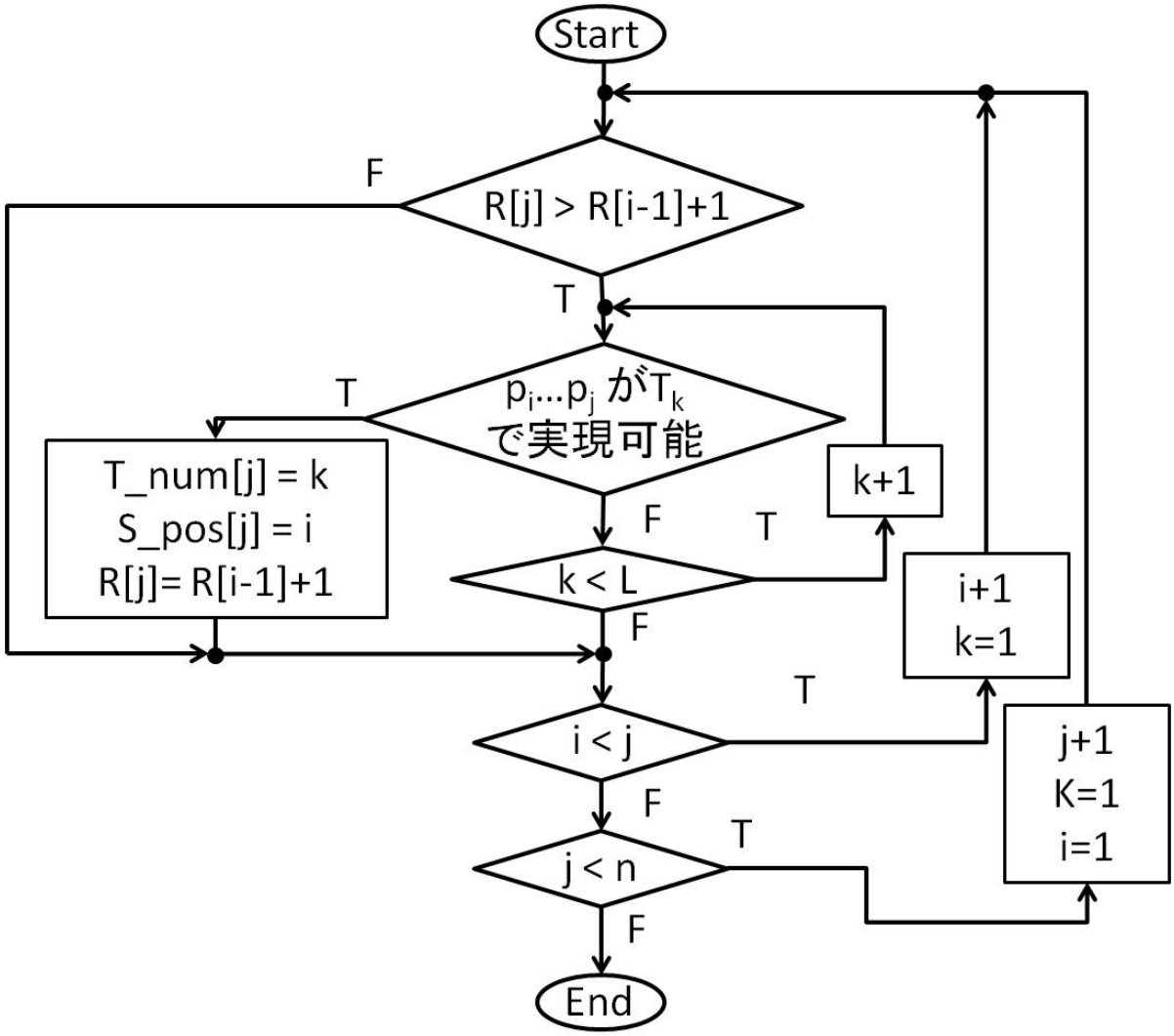


図 5.4: テンプレート選択のフローチャート

目的関数 与えられたパターンを実現するための PRR 数 R を最小化.

出力 R が最小となるテンプレートの組み合わせ.

上記の最適化問題を解くため，動的計画法を用いる．正規表現パターン P=$p_1p_2 \ldots p_n$ と L 種類のテンプレート $T_1, T_2, \ldots, T_L$ が与えられたとき，i 文字目までのパターン $p_1p_2 \ldots p_i$ $(1 \leq i \leq n)$ が必要とする PRR 数の最小値 $f(i)$ は動的計画法用いて以下のように計算される．

$f(0) = 0.$

$f(i) = min(f(0)+D(1,i), f(1)+D(2,i), f(2)+D(3,i), \ldots, f(i-1)+D(i,i)), 1 \leq i \leq n.$

$$D(x,y) = \begin{cases} 1 & (p_x \ldots p_y \text{があるテンプレート} T_k \\ & \text{で実現可能}) \\ \infty & (p_x \ldots p_y \text{がどのテンプレート} \\ & \text{でも実現不可能}) \end{cases} \tag{5.1}$$

式 (5.1) において，$p_x \ldots p_y$ が T_k で実現可能かどうかの判定方法について説明する．テンプレートは比較セル（CC）を 1 次元配列状に並べたものであるため，$p_x \ldots p_y$ を左端の CC から 1 文字づつ順に設定していき，そのテンプレート内の CC に入った場合は実現可能，入らない場合は実現不可能と判断する．$p_j (x \leq j \leq y)$ が CC で扱えない正規表現のクラスであるとき，その CC には図 5.3 のように，ε を設定する．

提案する動的計画法では，まず 1 文字目のパターン p_1 に必要な PRR 数の最小値を求め，1 文字づつパターン文字を増やしながら順に最小値を求めていく．$p_1 \ldots p_i$ に必要な PRR 数の最小値の計算では，p_1 から p_{i-1} までのそれぞれの結果が用いられる．

図 5.4 に提案する動的計画法のフローチャートを示す．このフローチャートでは 3 個の変数 i, j, k と 3 つの配列 $R[0 \ldots n], T\_num[0 \ldots n], S\_pos[0 \ldots n]$ が使われている．$R[j]$ は $p_1 \ldots p_j$ に必要な PRR 数の最小値 $f(j)$ を記憶する．このとき，$p_i \ldots p_j$ がテンプレート T_k で実現されるとすると，$T\_num[j]$ と $S\_pos[j]$ はテンプレート番号 k と先頭文字のインデックス番号 i を記憶する．変数の初期値として，i=j=k=1，$R[0]$=0，$R[1 \ldots n]$=∞ とする．

図 5.4 のフローチャートは，式 (5.1) を効率よく計算するために，時間のかかる $D(x,y)$ の計算をできるだけ避けるように設計されている．式 (5.1) をそのまま実装すると，$D(1,i)$ から $D(i,i)$ までの i 個の全ての組み合わせについて，各部分パターンが T_k で実現可能かを調べた後，最小となる組み合わせを選択することになる．しかし最小となる組み合わせさえわかればいいので，フローチャートでは，その組み合わせが最小となる見込みがあるか否かを先に判別し，見込みがある時のみ，T_k で実現可能か否かを調べることで，計算を効率化している．

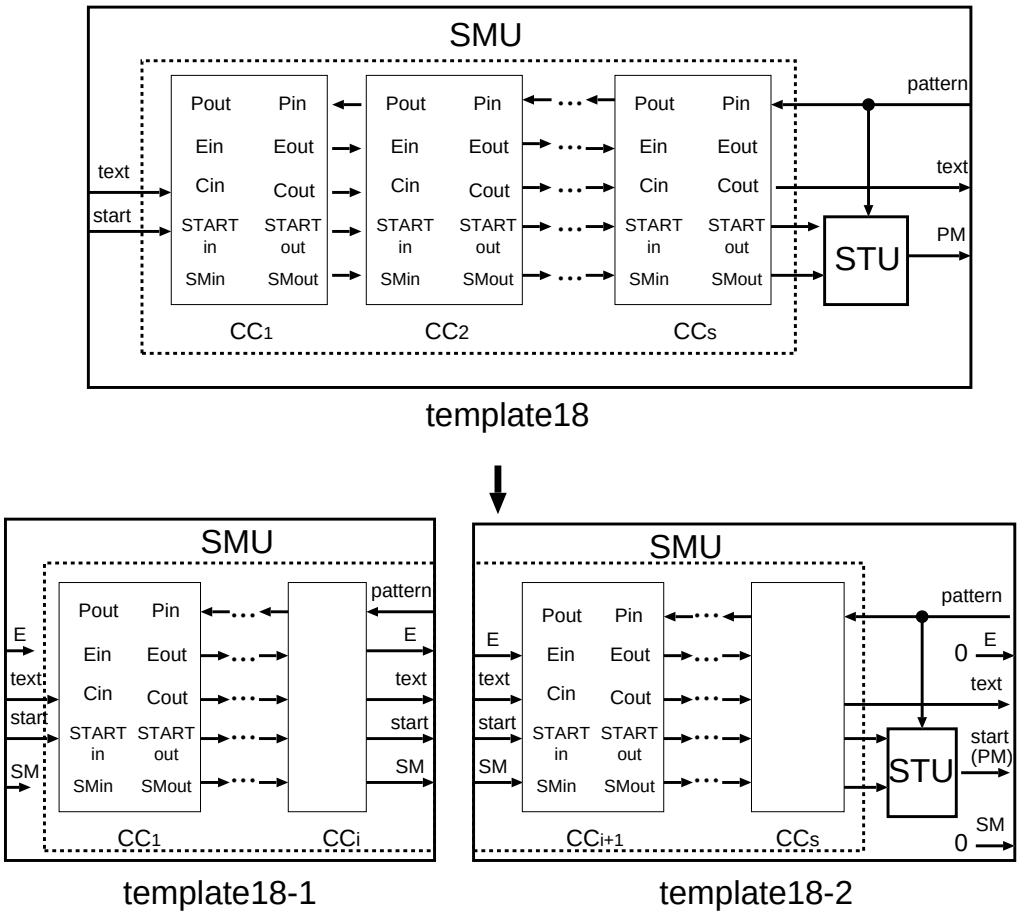


図 5.5: マッチングモジュールのテンプレート分割

正規表現パターン P に対する必要な PRR 数の最小値は $R[n]$ に記憶されており，テンプレートの組み合わせは $T\_num[0 \ldots n]$ と $S\_pos[0 \ldots n]$ を用いたトレースバックによって求めることができる．

5.5　細粒度なテンプレート設定

5.2 節で説明した回路構成は，マッチングモジュールの回路規模に合わせて PRR の領域を確保している．マッチングモジュールの回路規模は大きいため，あまり多くの PRR を FPGA 上に確保できない．一方，PRR の数が多いほど，細かな粒度で与えられたパターンに適した回路構成を生成できる．そこで，1 つのマッチングモジュールを複数の PRR に実装することで PRR の領域サイズを小さくし，多くの PRR を確保する．図 5.5 にマッチングモジュールを 2 つのテンプレートに分解する例を示す．SMU の CC_i で 2 つの回路に分割する．テンプレートの入出力信号は CC_i と CC_{i+1} 間の信号 E と $text$，$Start$，SM，

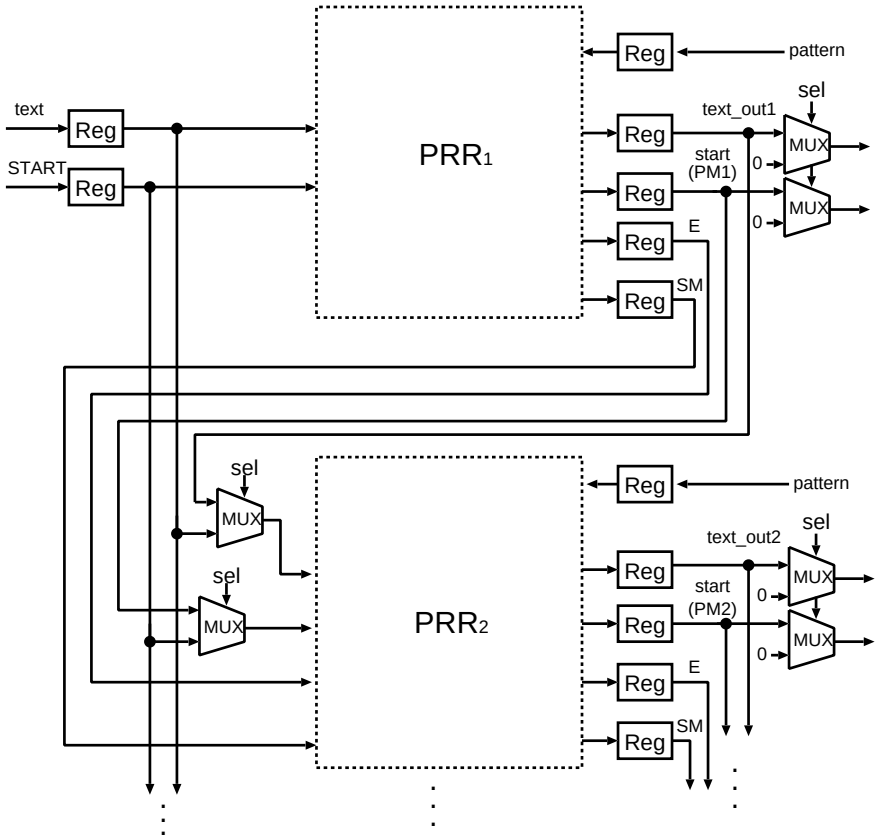


図 5.6: 提案回路構成の改良

pattern となる．template 18-1 では，E 信号と SM 信号は受け取る必要がないので，CC と接続せず，template 18-2 では，E 信号と SM 信号は出力する必要がないので，常に 0 を出力する．また他のテンプレートでも同様で，出力する必要のない信号は常に 0 を出力し，受け取る必要のない信号は CC と接続しない．静的回路（SR）には，それらの信号に対する配線を実装する（図 5.6）．

テンプレートの選択では，式 (5.1) においてマッチングモジュールのテンプレートに対する $D(x, y)$ の値を 2 に変更するだけで対応できる．

5.6　マッチング開始までの流れ

図 5.7 は提案手法におけるマッチング開始までの流れを示している．実線で囲まれた処理はパターン非依存正規表現マッチングハードウェアでも必要な処理であり，点線で囲まれた処理が提案手法で新たに必要となる追加処理である．

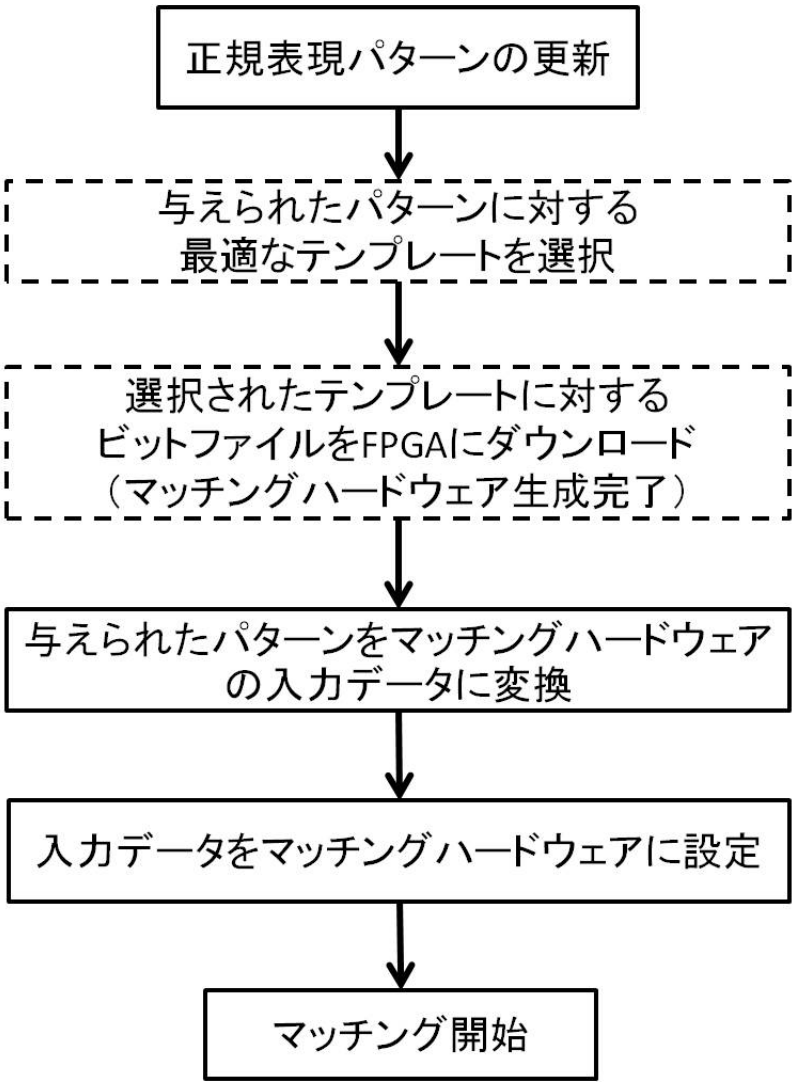


図 5.7: マッチング開始までの流れ

5.7　実験的評価

本節ではまず実験環境について述べる．提案回路構成とテンプレートの設計では，Xilinx ISE13.1 と PlanAhead13.1 を使用した．対象 FPGA は Virtex-6(xc6vlx240tff1156-11)（Slice 数： 37,680）と Virtex-4(xc4vlx200ff1513-11)（Slice 数： 89,088）を用いた．テンプレート選択ための C プログラムは以下の PC 上で実行した．OS: Ubuntu12.04 (VMware 上)，CPU: Intel(R) Core i7-3770K 3.50GHz，メモリ: 4GB．テストベンチとして，Snort ルール v2.9 における拡張正規表現（先読みと後方参照）を含むパターンを除いた 2052 種類のパターンを使用した．

5.7.1　面積評価

PRR 数（L=3, 6, 9, 12, 15, 18）を変えながら，提案ハードウェアの回路面積について評価する．用意するテンプレートは 5.3 節で述べた 18 種類とした．また L が 9 以上の場合，マッチングモジュールに対するテンプレートは 2 つの PRR に実装される．

図 5.8 に各提案ハードウェアと 4 章で示したハードウェア（ここでは，従来ハードウェアと呼ぶ）が，2052 種類の各パターンを実現するために必要となる Slice 数の最大値を示す．Virtex-4 と Virtex-6 それぞれにおいて，3 個の PRR を持った提案ハードウェアは従来ハードウェアと比べ，約半分の Slice 数で実装できる．また PRR 数を 6 にすることで，さらに使用面積を削減することができる．PRR 数が 9 から 15 までの提案ハードウェアは PRR 数が 6 の提案ハードウェアとほぼ同じ面積であり，18 個の PRR を持つ提案ハードウェアは使用面積が逆に増加した．提案手法は PRR 数を増やせば増やすほどより細かな粒度で，与えられたパターンに適した回路構成を生成できるが，2.4.3 節で述べたように PRR 数の増加に伴い，部分再構成のための面積オーバヘッドも増加する．18 個の PRR を持つ提案ハードウェアはこの面積オーバヘッドが細粒度による最適化の効果より大きくなったため，使用面積が増加したと考えられる．このように，PRR 数を極度に増やさない限り，部分再構成を用いることで面積効率を改善することができる．

最小の使用面積となった提案ハードウェアの構成は Virtex-4 では L=6，Virtex-6 では L=9 だった．各回路構成は従来ハードウェアに比べ，42% および 37% の回路面積でそれぞれマッチングを行うことができる．

5.7.2　パターン更新時間の評価

提案手法ではパターン更新に，最適なテンプレートの選択とそのテンプレートに対するビットファイルのダウンロードが必要であるため，その時間について評価する．

最適なテンプレートの選択において，2052 種類の中でパターン長が 10,428 文字のパターンが最も時間を要し，3.9 秒であった．これは，全体回路の再設計に約 5 分間必要とするパターン依存手法に比べ，非常に短い．

次に，パターン長と最適なテンプレート選択の計算時間の関係について述べる．図 5.9 は 10,428 文字のパターン長を定数倍した時のそれぞれのパターンに対する計算時間を示している．Snort ルールには，10,428 文字以下のパターンしか存在しないため，より長い

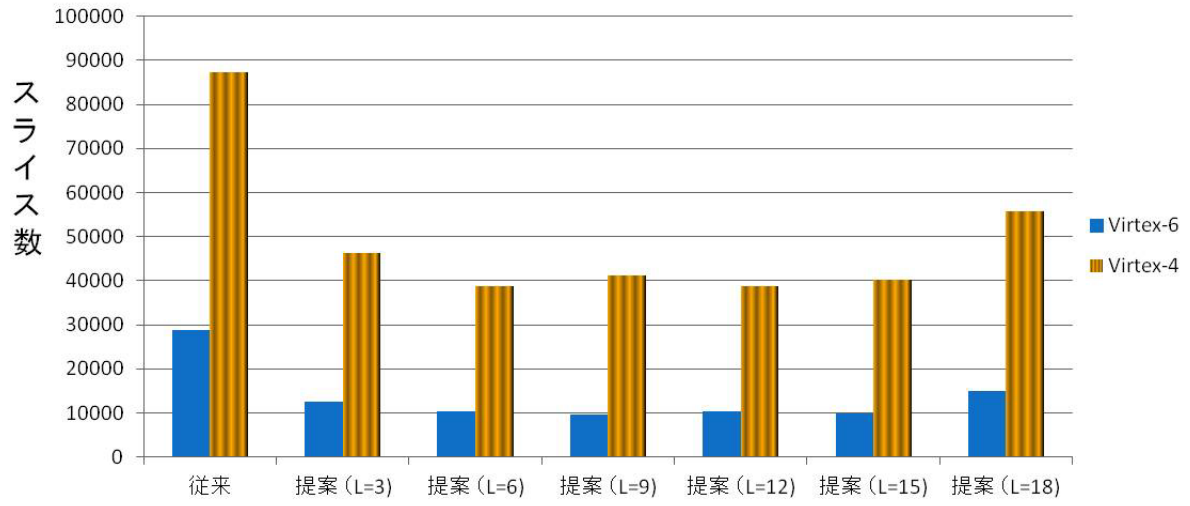


図 5.8: 各提案ハードウェアと従来ハードウェアの使用 Slice 数

図 5.9: パターン長に対するテンプレート選択の計算時間

パターンに対する計算時間の関係を調べるために，この実験では，10,428 文字のパターンを定数個複製し連接でつなぐことで，長いパターンを生成した．図 5.9 より，最適なテンプレート選択の計算時間はパターン長の二乗オーダで増加することがいえる．また，提案手法では，パターン依存手法で Snort ルールに対する回路の再設計で要した 5 分という時間を使えば，8 倍のパターン長に対する回路の生成が完了する．このように，提案手法

は，非常に長いパターンに対しても高速に最適なテンプレートを選択できる．

次にビットファイルのダウンロード時間について評価する．JTAG を用いたダウンロード速度は 66Mbps である．提案ハードウェアにおいて，各 PRR のビットファイルサイズは最大でも 1.5M bit なので，1 秒足らずでビットファイルを FPGA にダウンロードできる．

以上のように，提案手法は，Snort ルールのような実用的なパターンに対して，既存のパターン非依存手法同様，わずか数秒でパターン更新が完了する．また評価のために意図的に生成した非常に長いパターンに対しても，パターン依存手法より遥かに高速にパターンを更新できる．

5.7.3　提案ハードウェアの性能評価

6 個の PRR を持つ提案ハードウェア（Virtex-6）において，最大動作周波数は 159MHz であった．一方，4 章で示したハードウェアは 188MHz である．この性能低下は FPGA の部分再構成によるオーバヘッドであると考えられる．しかし，提案ハードウェアのスループットは 1.3 (= 8×159MHz) Gbps であるため，依然としてギガビットイーサネット上に配置される NIDS に必要な性能は保っている．

5.8　まとめ

本章では，FPGA の部分再構成機能を正規表現マッチングハードウェアに適用する手法を提案した．提案手法では，部分再構成領域 (PRR) があらかじめ FPGA 内に確保され，またテンプレートと呼ぶ PRR に実装する回路があらかじめ提供される．パターンが与えられるとパターンに合わせて最適なテンプレートの組み合わせが選択され，選択されたテンプレートが各 PRR に実装される．実験的評価から，提案手法は 4 章で提案したハードウェアと比べ，同程度のパターン更新時間で回路面積を 37% に削減できる事を示した．

提案ハードウェアの回路規模を更に削減するためにはあらかじめ用意しておくテンプレート集合自体を最適化する必要があると考えられる．よって，今後の課題として，最適なテンプレート集合の設計が挙げられる．

第6章　拡張正規表現に対するハードウェアマッチング手法

NIDS として有名な Snort では，ウィルスパターンを 2.5.4 節で示した拡張正規表現を用いて簡潔に記述している．量指定子や文字クラス等のほとんどの拡張正規表現演算子に対するハードウェアでのマッチング手法は提案されているが，先読み演算と後方参照に対するハードウェアマッチング手法は知られていない．本章ではこれらの 2 つの演算子に対するハードウェアマッチング手法を提案する．

6.1　先読み演算に対するハードウェアマッチング手法

本節では，先読み演算に対するハードウェアマッチング手法を提案する．

6.1.1　概要

先読み演算は特定の部分文字列を含まない文字列や特定の 2 種類の部分文字列を含む文字列と一致するパターンをコンパクトに記述することができる [71]．先読み演算には肯定的先読み “R_1 (?=R_2) R_3” と否定的先読み “R_1 (?!R_2) R_3” の 2 種類がある．ここで R_1，R_2，R_3 は任意の正規表現であり，‘ε’ の場合もある．肯定的先読みは R_1 と一致したすぐ後に R_2 と R_3 の両方と一致するような文字列の集合を表している．ここで，R_2 と一致する文字列は R_3 と一致する文字列の部分文字列，またはその逆であることに注意されたい．否定的先読みは R_1 と一致したすぐ後に R_2 には一致せず R_3 と一致するような文字列の集合を表している．本節では，先読み演算はパターン全体 “R_1 (?=R_2) R_3” または “R_1 (?!R_2) R_3” を指し，先読みパターンは R_2 だけを指す用語とする．

先読み演算は，例えば NIDS において，登録されていないドメイン名からのメールを検知するために使用される．このようなパターンは先読みを用いて，“.*@(?! $hiroshima$-

$cu\backslash.ac\backslash.jp$)" のように記述される．これで "@hiroshima-cu.ac.jp" 以外のドメイン名のアドレスから送られたメールを検知できる．

図 6.1 に先読み演算に対するマッチングの例を示す．図 6.1 の (a) は通常の正規表現，(b) と (c) は先読み演算のマッチング例を示している．

先読み演算を用いたパターンは次節で示すように，正規表現の基本演算子（連接とユニオン，クリーネ閉包）のみを使ったパターンに変換することはできる．しかし変換されたパターンはとても長く，マッチングを行うために非常に大量のハードウェア資源を必要とし，かつ長い変換時間を要する可能性がある．

本節では，正規表現マッチングハードウェアにおける先読み演算のためのマッチング手法を提案する．提案手法では，先読みを扱うために，正規表現マッチングハードウェアに前処理回路を導入する．前処理回路は文字列の末尾から文字列の先頭へマッチングを行う．また高スループットを達成するために，スタックメモリを用いた新しいバッファ機構を提案する．実験結果から，提案手法が効率的に先読みのマッチングを実行できることを示す．

6.1.2　先読み演算から等価な正規表現への変換

先読み演算は正規表現の基本演算子 ('·' と '|'，'∗') を使ったパターンに変換できる．例えば，"$abc(?=\backslash.exe)$" は正規表現 "$abc.exe$" に変換でき，"$n(?=ae).^*g$" は "$nae.^*g$" に変換できる．一般的な先読み演算の変換の手順としては，まず与えられた先読み演算を決定性有限オートマトン（DFA）に変換する．その際，肯定的先読み "$R_1\ (?=R_2)\ R_3$" は R_2 と R_3 の部分が積オートマトンで表され，否定的先読み "$R_1\ (?!R_2)\ R_3$" は R_2 と R_3 の部分が R_2 の否定に対するオートマトンと R_3 の積オートマトンで表される．DFA は正規表現に直接変換できるため [34]，上記のようにして得られた DFA を正規表現に変換することで，先読み演算と等価な正規表現を生成できる．

例として，"$(?=.^*b).^*c$" の変換を考える．まず，"$.^*b$" と "$.^*c$" に対する DFA をそれぞれ生成し，その 2 つの DFA から積オートマトンを生成する（図 6.2）．そして，その積オートマトンで表された DFA から等価な正規表現 "$(\bar{b}|\bar{c})^*(b(\bar{c})^*c \mid c(\bar{b})^*b)$" が生成できる．

しかし，一般に DFA から正規表現の変換は長いパターンを生成してしまう [34]．例えば，現実的な先読み演算を用いたパターンとしては，"$(?!\ .^*medical(care|treatment)).^*virus$" のような複雑なパターンがしばしば利用される．このパターンは，コンピュータウィルス

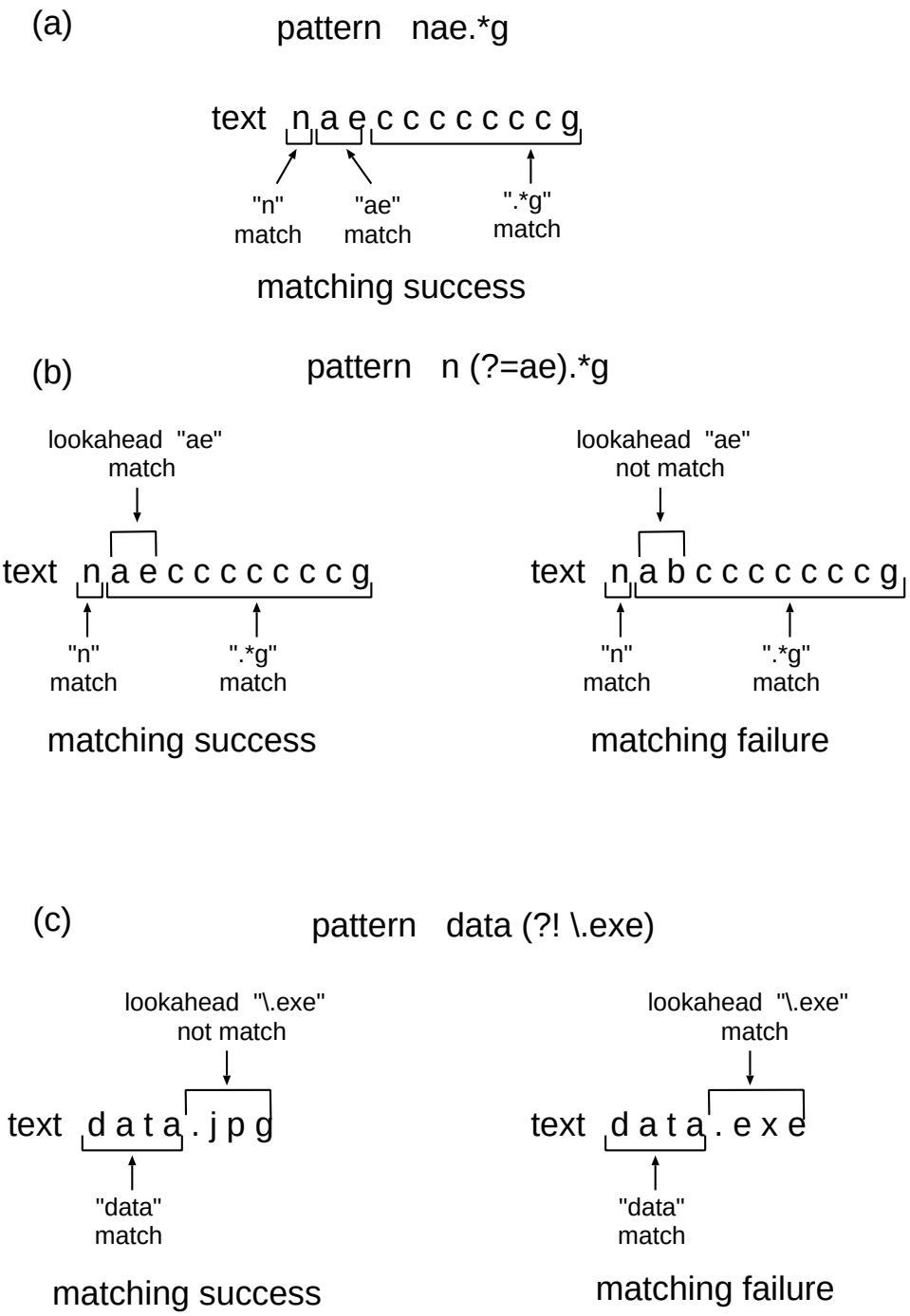


図 6.1: 先読み演算に対するマッチング例

の疑いがある “virus” の単語を含むメールを検知したいが，“medical care” または “medical treatment” の単語が同時に含まれる場合は，医療分野のウィルスを意味するので，こういったメールは検知したくないという場合に利用できる．しかし，このパターンを等価な正規表現に変換すると，とても長いパターンとなり，大規模な正規表現マッチングハー

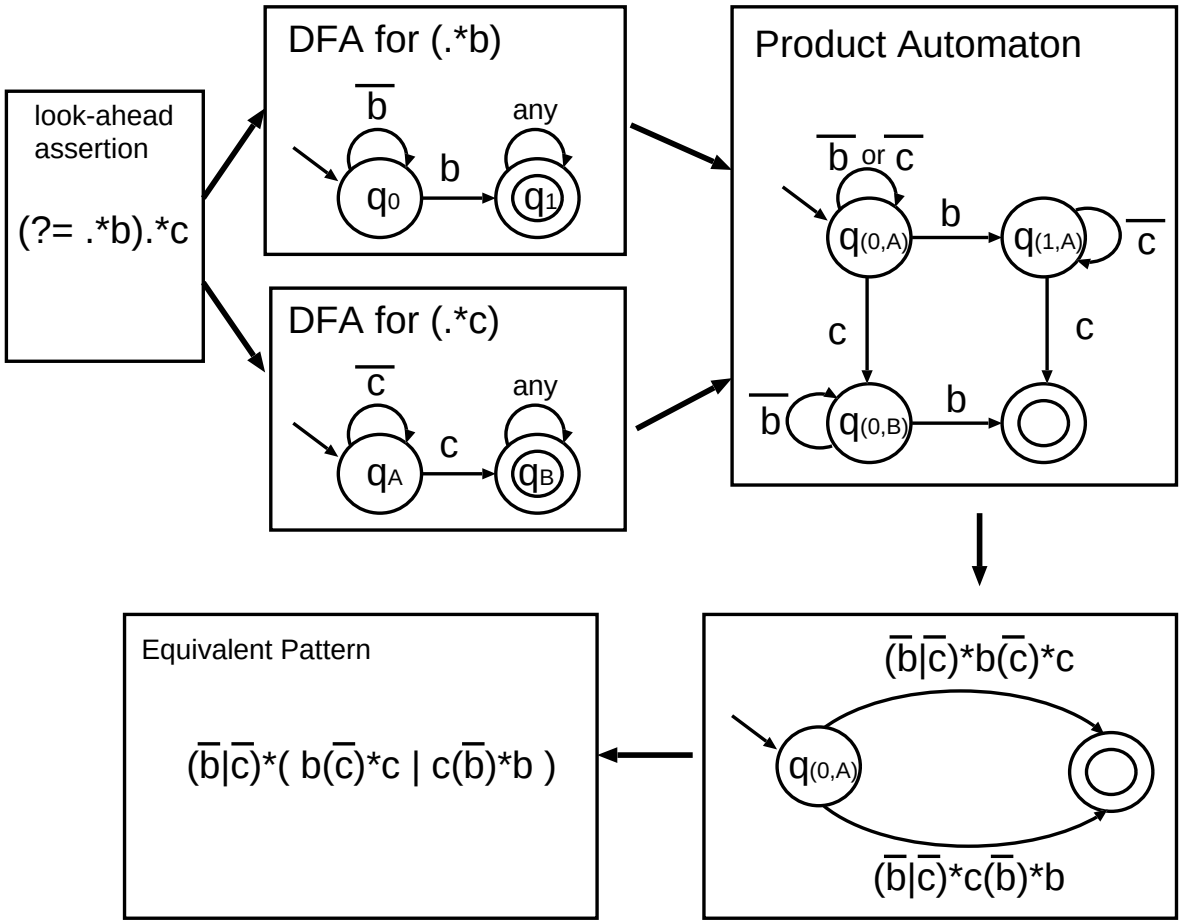


図 6.2: “(?=.*b).*c” の等価な正規表現への変換

ドウェアが必要となる．そのため，先読み演算は，正規表現に変換することなく，短いパターンのまま直接実現できることが望ましい．

6.1.3　先読み演算に対する提案マッチング手法

この節では，先読み演算に対するマッチング手法を提案する．提案手法では，まず先読みパターン R_2 に対するマッチングを行い，次にそのマッチング結果を用いて，先読みパターンを除いた残りのパターンに対するマッチングを行う．

提案マッチング手法の動作例を図 6.3 に示す．先読み演算のマッチングにおいて，まず先読みパターン R_2 と一致する部分文字列の先頭文字を見つける．そして同じテキストに対し R_2 を除くパターンでのマッチングを行い，R_3 のマッチング開始位置が R_2 と一致した部分文字列の先頭文字と同じなら，R_3 のマッチングを行う．そうでなければ，マッチ

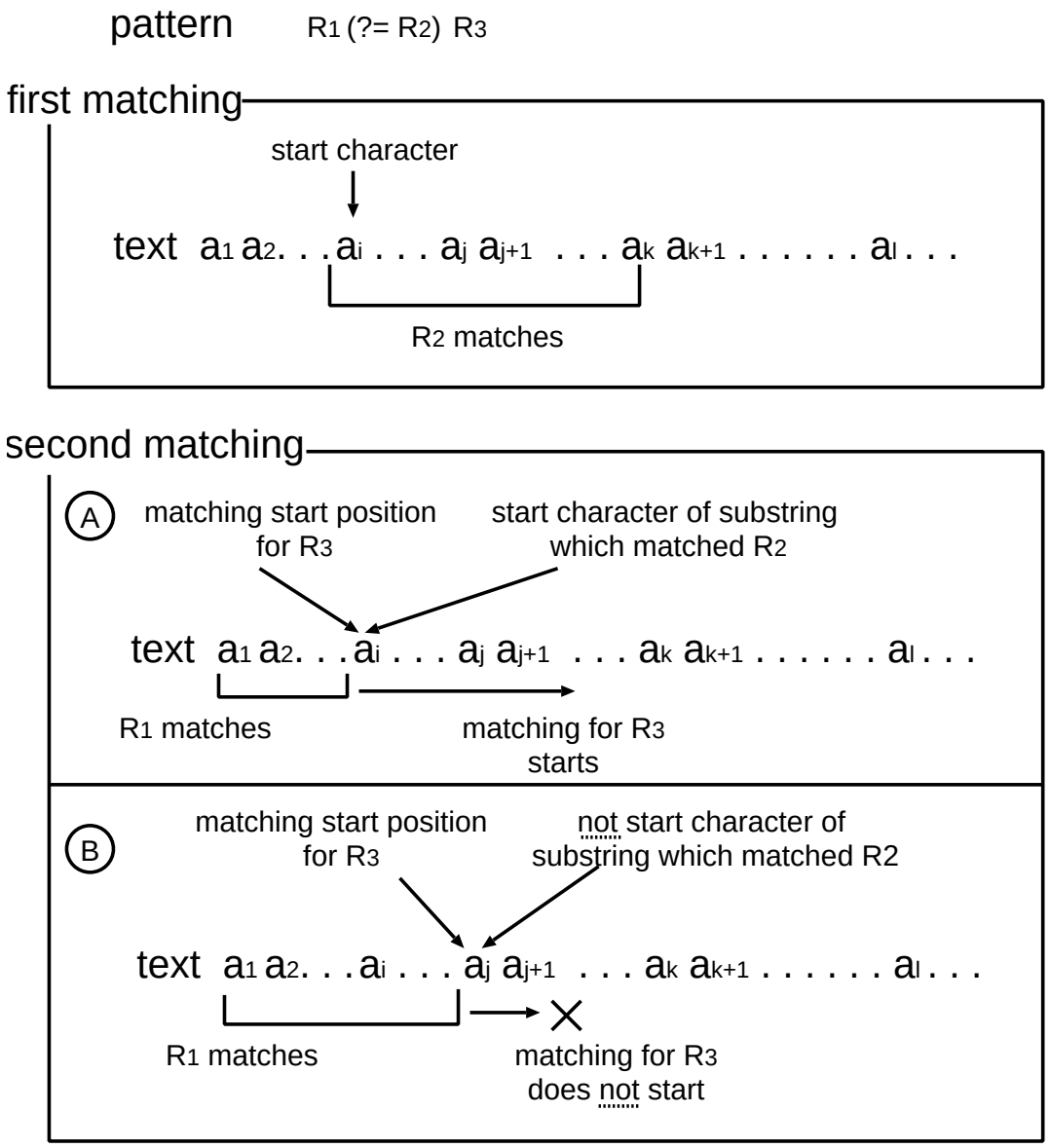


図 6.3: 提案マッチング手法の動作

ングは既に失敗しているので，R_3 のマッチングは行わない．上記は肯定的先読みの場合であり，否定的先読みの場合は，R_3 のマッチング開始位置が先読みパターン R_2 と一致した部分文字列の先頭文字でない場合にマッチングを行う．本提案手法において，R_1，R_2，R_3 のそれぞれに対する正規表現マッチングは，本論文あるいは従来研究で提案された正規表現マッチングハードウェアで行う．

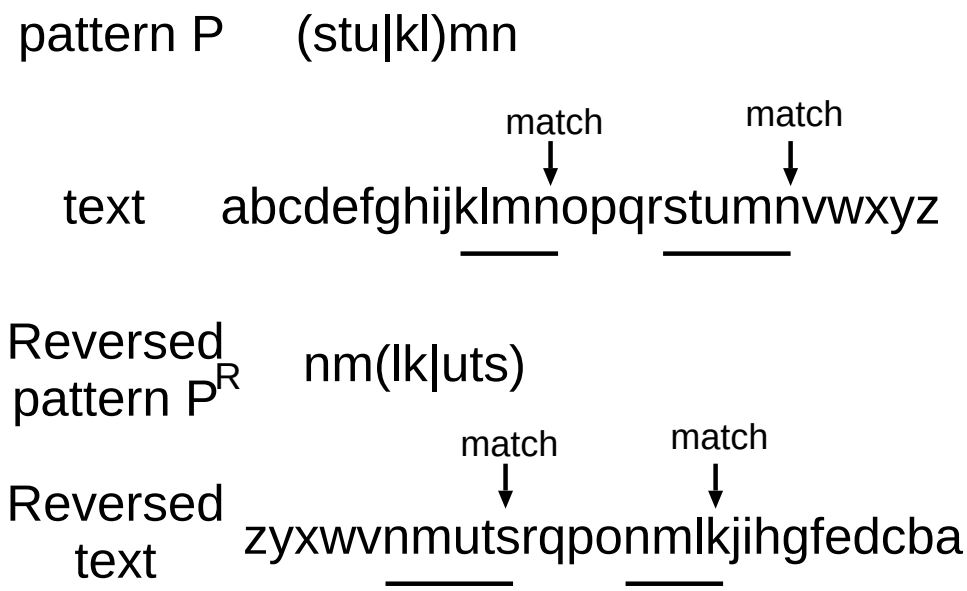


図 6.4: 逆順パターンに対するマッチング

先読みパターンと一致する部分文字列の先頭文字の検索

提案手法において，正規表現マッチングハードウェアは，まず先読みパターンと一致する部分文字列の先頭文字を見つける必要がある．しかしながら，正規表現マッチングハードウェアはテキストの先頭からマッチングを行うと，パターンと一致する部分文字列の末尾文字は検出できるが，先頭文字は検出できない．

よって，元のパターン P の文字や演算子の順番を逆にした逆順パターン P^R をハードウェアに入力し，その後，テキストも逆順に入力していく．この時も，ハードウェアは P^R と一致する部分文字列の末尾文字を見つけることになるが，図 6.4 に示すように，この末尾文字は元の P における先頭文字に他ならない．このように，パターン P と一致する全ての部分文字列 Q_i における先頭文字は，逆順パターン P^R によって，逆順部分文字列 Q_i^R の末尾文字として漏れなく見つけることができる [34].

アーキテクチャ

以下では，提案手法を実現するためのアーキテクチャについて説明する．提案アーキテクチャを図 6.5 に示す．提案アーキテクチャは先読みパターンを除いたパターンに対する正規表現マッチングを行うマッチングハードウェア B と先読みパターンの逆順に対するマッチングを行う前処理回路からなる．また前処理回路は 2 つのスタックメモリと正規表

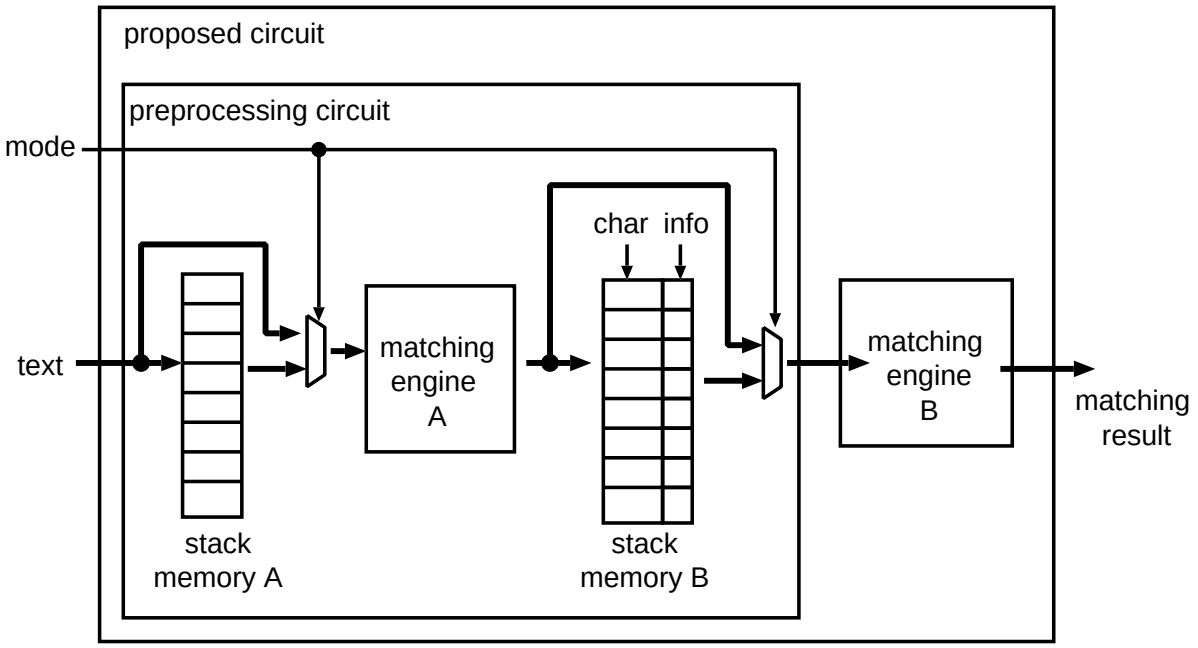


図 6.5: 提案アーキテクチャ

現マッチングハードウェア A からなる．パターンと一致する部分文字列の末尾文字を検出できる正規表現マッチングハードウェアであれば，パターン依存とパターン非依存のどちらでも，提案アーキテクチャに拡張することができる．また，先読み演算を含まないパターンに対しては，2 つのハードウェア A と B を直接つなぐことで，従来通り 1 つのマッチングハードウェアとして正規表現マッチングを行うことができる．以下に提案アーキテクチャの動作を示す．以下の動作はパターン非依存正規表現マッチングハードウェアの場合である．

1. マッチングが開始される前に，逆順先読みパターン P^R がマッチングハードウェア A に設定され，先読みパターンを除くパターンはマッチングハードウェア B に設定される．
2. マッチングが開始されると，まずテキストは逆順テキストを作るためにスタックメモリ A に入力される．
3. テキストの末尾文字がスタックメモリ A に入力された後に，テキストの末尾からマッチングハードウェア A に文字が入力される．
4. マッチングハードウェア A は先読みパターンと一致する部分文字列の末尾文字を検出し，その情報を末尾文字とともにスタックメモリ B に格納する．
5. テキストの先頭文字がスタックメモリ B に入力されると，テキストの先頭からマッチングハードウェア B に文字が入力される．

仮定:	パケットの最大長: n. 提案メモリのサイズ: $2n$.
変数:	$Write\_address$. $Current\_read\_address$. $Current\_end\_address$. $Next\_start\_address$. $Next\_end\_address$.
初期値:	全ての変数は 0 に設定.

Step:

1. メモリアドレス [$Write\_address$] に入力パケット文字を保存.
2. もし入力パケット文字がパケットの末尾文字の場合, $Next\_start\_address = Write\_address$.
3. $Write\_address = Write\_address + 1 \pmod{2n}$.
4. もし $Write\_address$ が n の場合， Step 5 へ． そうでない場合，Step 1 へ.
5. $Current\_read\_address = Next\_start\_address$.
6. $Current\_end\_address = Next\_end\_address$.
7. $Next\_end\_address = Next\_start\_address + 1 \pmod{2n}$.
8. メモリアドレス [$Current\_read\_address$] のパケット文字を出力.
9. メモリアドレス [$Write\_address$] に入力パケット文字を保存.
10. もし入力パケット文字がパケットの末尾文字の場合, $Next\_start\_address = Write\_address$.
11. $Write\_address = Write\_address + 1 \pmod{2n}$.
12. $Current\_read\_address = Current\_read\_address$ - 1 $\pmod{2n}$.
13. もし $Current\_read\_address$ が $Current\_end\_address$ - 1 $\pmod{2n}$ の場合, Step 5 へ．そうでない場合，Step 8 へ.

図 6.6: 提案バッファ機構のアルゴリズム

6. マッチングハードウェア B は前処理回路で得られた情報を使って先読みパターンを除くパターンに対してマッチングを行う.

提案手法は与えられたテキストの長さに上限がない場合，上記 2. が終了しないため，正しく動作しない．しかし本研究で対象とするアプリケーションは NIDS であるため，与えられるテキストはパケットであり，長さは有限である．よって提案手法は NIDS においては有効である.

提案バッファ機構

提案手法を NIDS に適用するためには，次々に到着するパケットを滞りなく，順次，逆順にする必要がある．そのため，提案アーキテクチャのスタックメモリに，パケットをバッファリングする機能を追加する．本節では，スタックとバッファの両方の機能を兼ね備えた新たなバッファ構造を提案する.

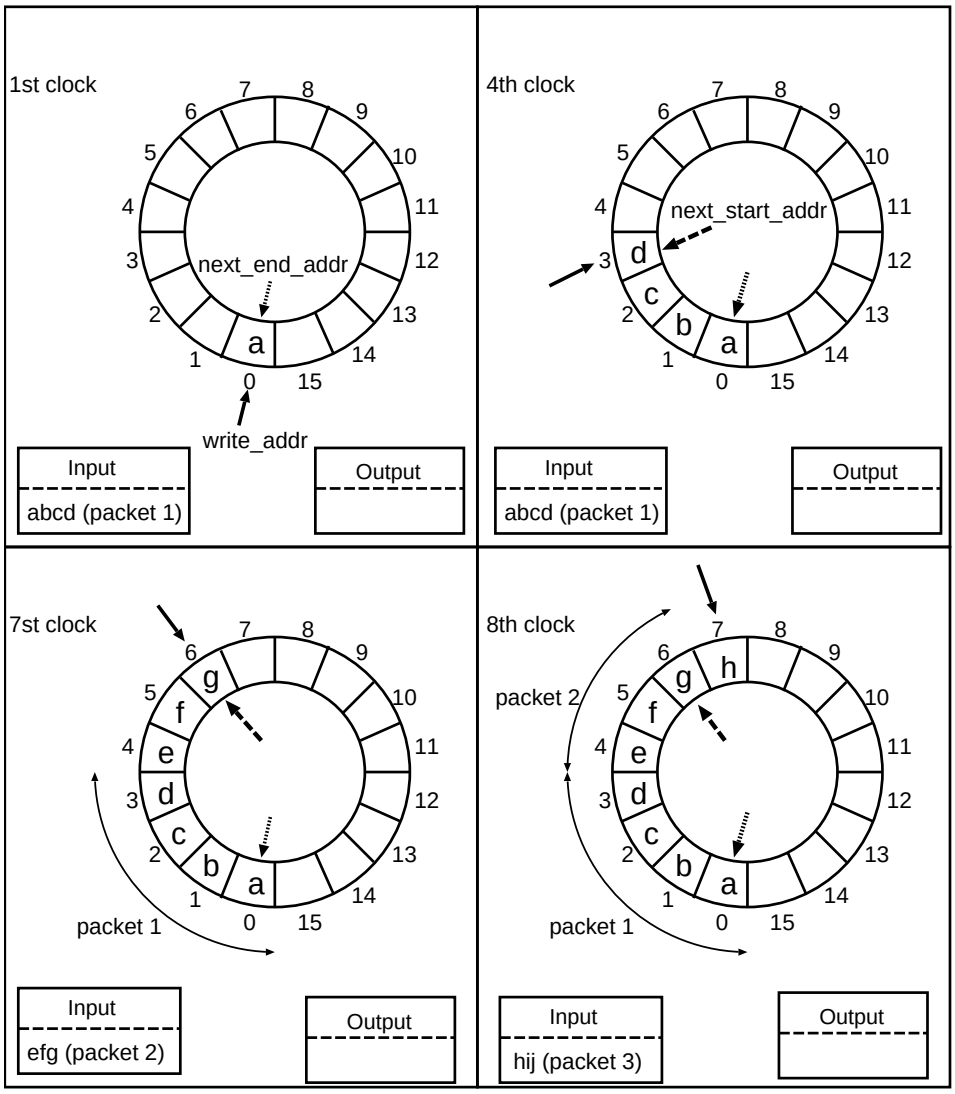


図 6.7: 提案バッファ機構の動作例 1

パケットの最大長を n とすると，提案バッファ機構のメモリサイズは $2n$ であり，構造はリングバッファと同じである．また提案機構を実現するために，デュアルポートメモリを用いる．図 6.6 に提案機構のアルゴリズムを示す．まず，n 文字が各パケットのサイズに関係なくメモリに格納される．この時，複数のパケットがメモリに格納される場合もある．またこの時点で，最後に到着したパケットは，全ての文字が格納されていない場合もあるが，少なくとも 1 つのパケットは完全に格納されていることに注意されたい．n 文字が格納された後，メモリに完全に格納されているパケット P_1 は最後に格納されたパケットから順に最初に格納されたパケットまで，メモリから逆順テキストとして出力される．これらのパケット P_1 が出力されている間，同時に，新たなパケット P_2 がメモリに保存される．P_1 の全てのパケットがメモリから出力し終わると，新たに保存された P_2 が同様に

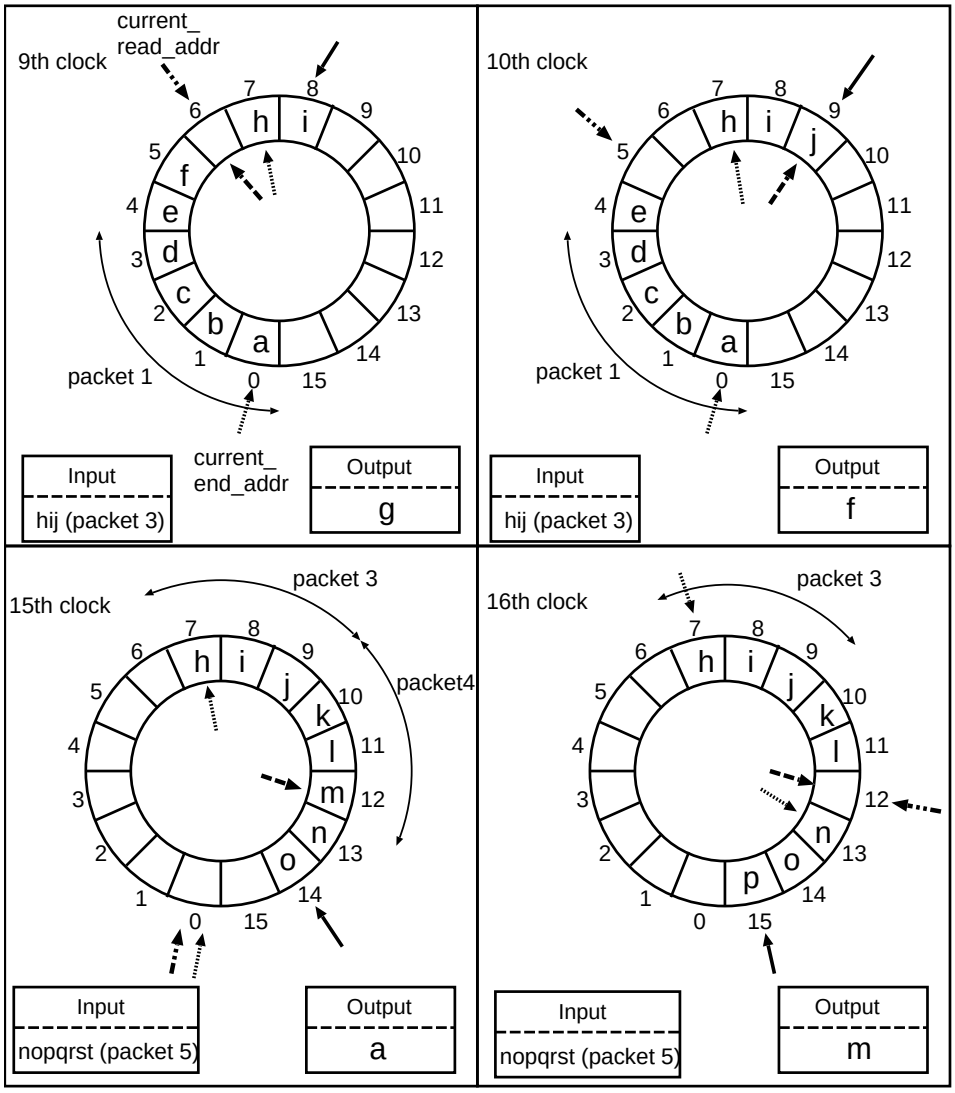


図 6.8: 提案バッファ機構の動作例 2

メモリから逆順テキストとして出力される．以降，パケットの保存と逆順テキストの出力が同時かつ連続的に行われる．

図 6.7 と図 6.8 に提案機構の動作例を示す．ここで，パケットの最大長は 8 となっている．

6.1.4　実験的評価

本節では，提案マッチング手法の面積効率と性能について評価を行う．提案手法を評価するために，FPGA 設計ツールとして ISE13.1，対象デバイスとして Xilinx Virtex6 (XC6VLX240T-1FFG1156) を使用した．実験では，ネットワーク層でのウィルス検知す

表 6.1: 提案ハードウェアの面積と性能評価

	面積オーバヘッド		性能オーバヘッド	
	Slice 数	ブロック RAM 数	最高動作周波数（性能低下率）	スループット
パケットサイズ 1,500bytes に対する提案エンジン	52	3	180 MHz (-4%)	1.4Gbps
パケットサイズ 65,536bytes に対する提案エンジン	240	92	145 MHz (-23%)	1.2Gbps

なわち，IPv4 パケットに対して正規表現マッチングを行うことを前提とし，2 つのパケットサイズに対する提案エンジンを実装した．1 つはイーサーネットの MTU(Maximum Transmission Unit) のパケットサイズ 1,500bytes，もう 1 つは IP パケットの最大サイズである 65,536bytes とした．本節では，4 章で提案したパターン非依存正規表現マッチングハードウェア（本節では，従来ハードウェアと呼ぶ）に対して提案手法を適用した結果を示す．

先読み演算を実現するための面積オーバヘッドと性能オーバヘッドについて評価する．表 6.1 に，2 つのパケットサイズ 1,500bytes と 65,536bytes に対する提案ハードウェアの面積オーバヘッドと性能オーバヘッドを示す．提案ハードウェアは従来ハードウェアに対してリングバッファを実現するためのブロック RAM とアドレス計算回路を追加した回路構成となる．これらを実現するための Slice 数とブロック RAM 数を示す．2 つのパケットサイズ 1,500bytes と 65,536 bytes の提案ハードウェアにおいてアドレス計算回路を実現するための Slice 数はそれぞれ，52 と 240 であった．またブロック RAM に関しては 3 個と 92 個必要となった．

従来ハードウェアで先読み演算のマッチングを実現するためには，等価な正規表現に変換してから実現する必要があり，これには，6.1.2 節で述べたように，長いパターンのために，過度に多くの回路面積が必要になる場合がある．例えば，Snort ルール v2.9 の先読み演算，“`\x3Cobject(?![^\x3E]+?src)[^\x3E]+?data\s*\x3D\s*\x22\x22`” を [34] で示された一般的な変換方法で，等価な正規表現に変換した場合，パターン長が 3000 文字以上となり従来ハードウェアで扱うことはできなかった．一方，提案ハードウェアはリングバッファを実現するためのわずかな回路面積の増加だけで，先読み演算を実現できる．

次に，最高動作周波数を評価する．2 つのパケットサイズ 1,500 bytes と 65,536 bytes に対する提案ハードウェアはそれぞれ 180MHz と 145MHz で動作する．65,536bytes に対するエンジンは大幅に性能が低下している．これは，2 つのスタックメモリのために使用

した 92 個のブロック RAM 間における複雑な配線によるものだと考えられる．しかしながら，65,536 bytes に対する提案ハードウェアのスループットはまだ 1Gbps を超えているため，ギガビットイーサネット上でもウィルスを検知可能である．

以上の結果より，提案ハードウェアは効率的に先読み演算を扱うことができるといえる．

6.2　後方参照に対する提案マッチング手法

本節では，後方参照に対するハードウェアマッチング手法を提案する．

6.2.1　概要

後方参照はテキスト内の部分文字列をパターンとして使用できる演算子であり，正規表現を超える表現能力を持つ [71, 72]．後方参照を含むパターンは R_1 (RB_i) R_2 $\backslash i$ R_3 のように記述される．ここで，R_1 と R_2,R_3,RB_i は任意の正規表現であり，後方参照を含んだ正規表現の場合もある．変数 i は括弧で囲まれた正規表現に対する番号であり，RB_i はパターンの先頭から i 個目の括弧に囲まれた正規表現を意味する．例えば，後方参照を含むパターン “$tea(ab)jfkd(cd)ads$” の場合，RB_1 は ab であり，RB_2 は cd である．$\backslash i$ は RB_i と一致したテキスト中の部分文字列を表す演算子である．

後方参照は，例えばアカウント登録において，ユーザが ID とパスワードを同じもので登録しようとした場合を検知するために使用できる．この時，“$ID : (\backslash w+); PASSWORD : \backslash 1;$” のようなパターンを記述することで，ID とパスワードが同じユーザを検知できる．

後方参照を含んだパターンが表わす言語は一般的には非正規言語となるため，NFA や DFA で表わすことができない．そのため，[73] は後方参照を含んだパターンに対する拡張有限オートマトンを提案している．例として “$([a\text{-}z]^+)ak\backslash 1y$” に対する拡張有限オートマトンを図 6.9 示す．このオートマトンにおいて，初期状態は q0，受理状態は q5 である．実線で示す状態遷移の矢印は通常のオートマトンと同様，入力文字が遷移文字と一致する場合，次状態に遷移する．状態 q1 への遷移に対する点線の矢印は特殊な遷移であり，入力文字が遷移文字と一致する場合，次状態に遷移し，同時に一致した入力文字が記憶される．状態 q3 から次状態への遷移に対する点線の矢印も特殊な遷移である．入力文字が記憶された文字列 S の先頭文字と一致する場合，次状態に遷移する．次状態に遷移する際，

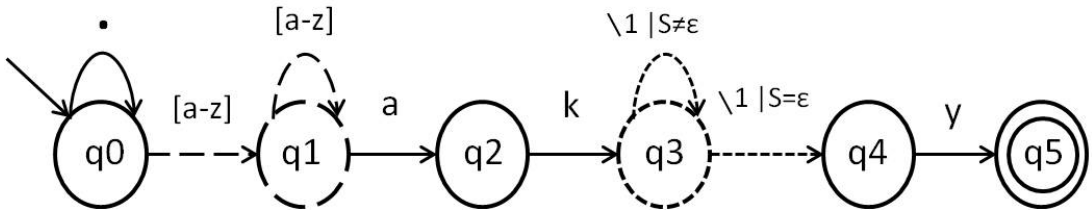


図 6.9: “([a-z]$^+$)ak\1y” に対する拡張有限オートマトン

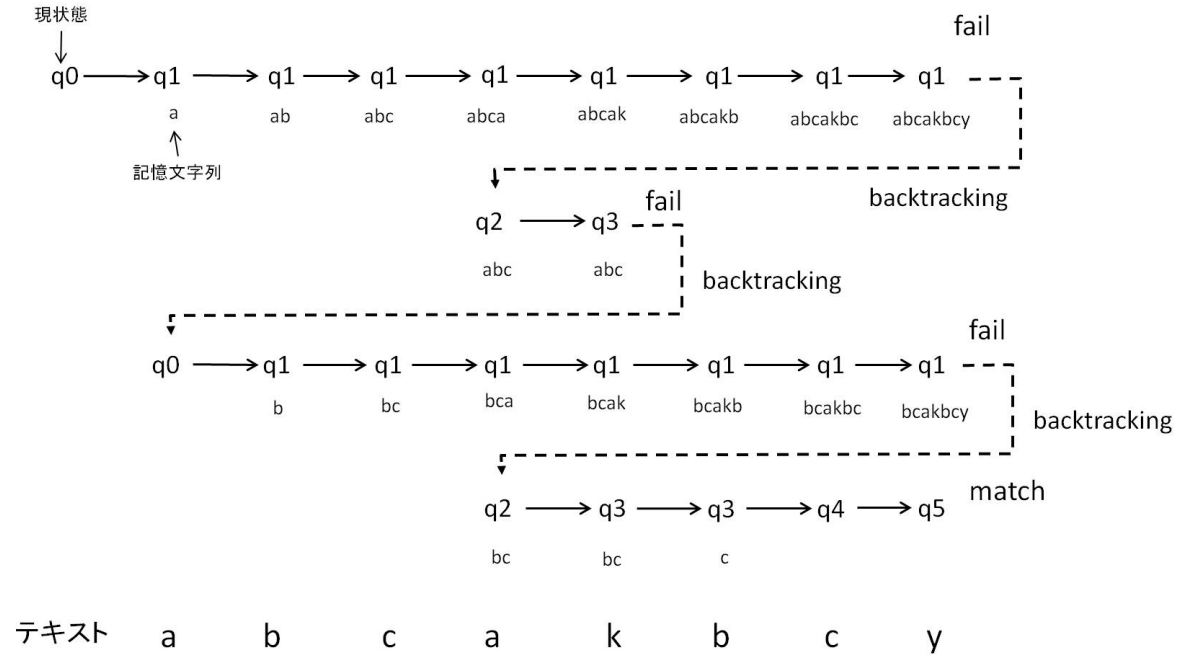


図 6.10: 動作例

S の先頭文字を消去し，S が空語になった場合，遷移条件が $\backslash 1|\ S{=}\varepsilon$ で示された次状態に遷移し，そうでない場合，遷移条件が $\backslash 1|\ S \neq \varepsilon$ で示された次状態に遷移する．

この拡張有限オートマトンに対して深さ優先探索とバックトラックを行うことで，後方参照を含むパターンに対するマッチングを行うことができる．図 6.9 の拡張有限オートマトンにテキスト “abcakbcy” を入力したときのオートマトンの動作例を図 6.10 に示す．

Snort では，深さ優先探索とバックトラックを行うことで，後方参照を含むパターンに対するマッチングを行っている [28]．しかし，ソフトウェアによる処理は非常に遅く，パケットの内容によってはバックトラック回数が非常に多くなり，かなりの処理時間を要する場合がある．例えば，図 6.9 のオートマトンとテキスト “b{100,000}”（b が 10 万文字

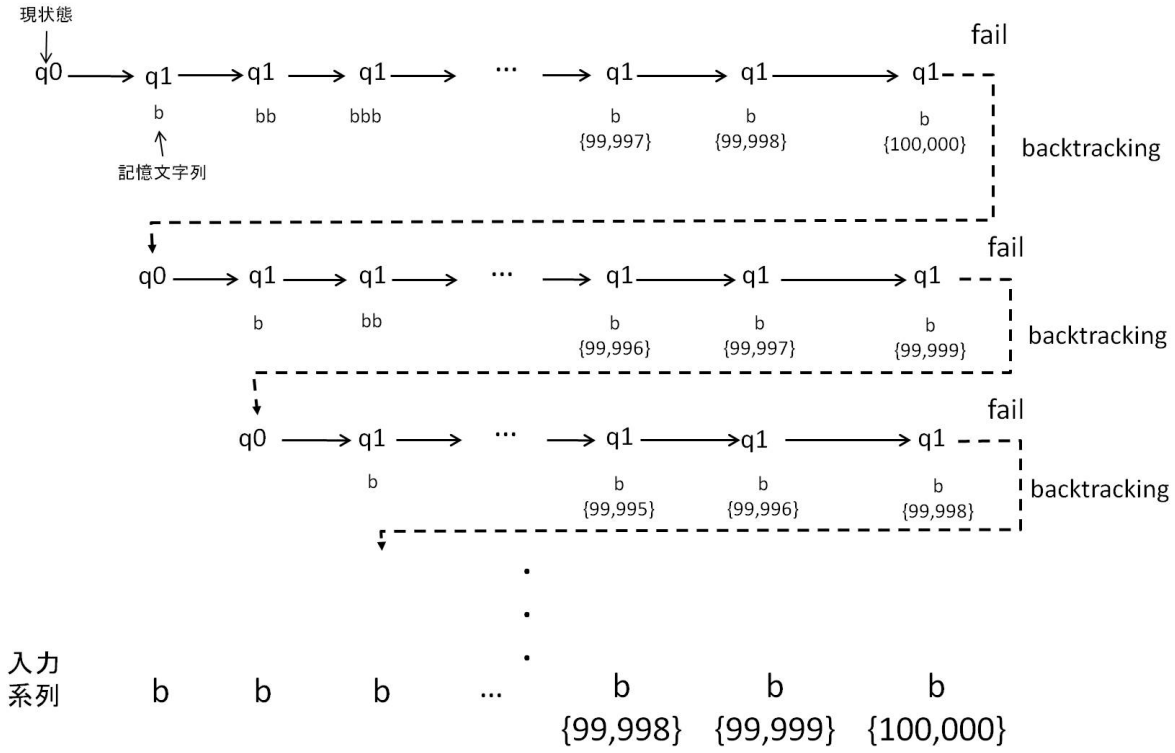


図 6.11: テキスト “b{100,000}” に対するアンアンカーモードのマッチング

の文字列）に対するアンアンカーモード（2.2 節で前述）のマッチングの場合，10 万回のバックトラックが必要となる（図 6.11）.

そこで，本節では，正規表現マッチングハードウェアを用いて後方参照を含むパターンのマッチングの高速化手法を提案する.

6.2.2　正規表現マッチングハードウェアを用いた提案マッチング手法

提案マッチング手法では，後方参照を含むパターンに対して，ソフトウェアとハードウェアの両方を用いてマッチングを行う．提案マッチング手法の流れを図 6.12 に示す．提案マッチング手法では，パケットが入力されると，まずハードウェアによって前処理を行い，後方参照を含むパターンと一致する可能性のあるパケットと可能性のないパケットを高速に分類する．一致する可能性のないパケットはそのまま通過させ，一致する可能性のあるパケットはソフトウェアで深さ優先探索とバックトラックによるマッチングを行い，後方参照を含むパターンと一致するか調べる.

ハードウェアの前処理では，後方参照を含むパターン　“$R_1\ (RB_i)\ R_2\ \backslash i\ R_3$” に対し

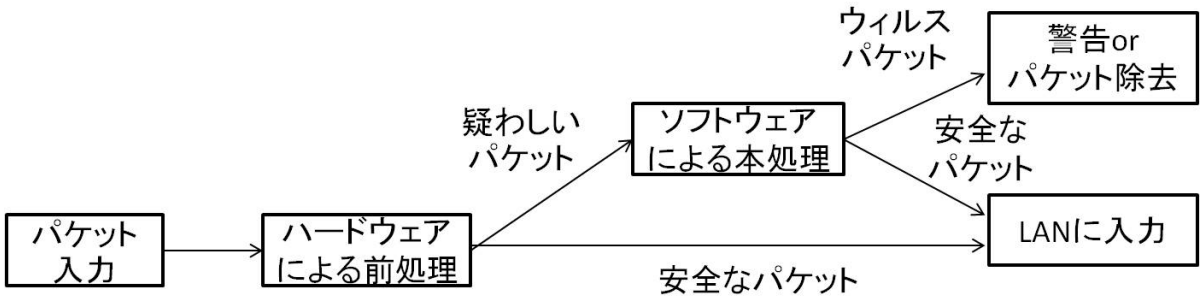


図 6.12: 提案マッチング手法の流れ

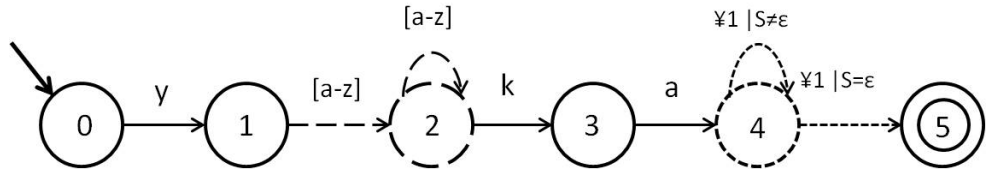


図 6.13: 逆パターン “$y([a\text{-}z]^{+})ka\backslash 1$” に対する拡張有限オートマトン

て，変換パターン “$R_1\ (RB_i)\ R_2\ (RB_i)\ R_3$” と入力パケットに対する正規表現マッチングを行う．後方参照を含むパターンの場合，$\backslash i$ は (RB_i) と一致した文字列と同じ文字列の場合に一致するが，変換パターンの場合，$\backslash i$ を (RB_i) と置き換えてマッチングを行うため，同じ文字列でない場合も一致する．例えば，後方参照を含むパターン “$(A|B)\ and\ \backslash 1$” の場合，”A and A” または ”B and B” のどちらかと一致するが，変更パターン “$(A|B)\ and\ (A|B)$” の場合，”A and A” と ”B and B”，”A and B”，”B and A” のどれでも一致する．そのため，ハードウェアでは，実際のパターンと一致しないパケットも検知してしまう場合（False Positive）はあるが，実際のパターンと一致するパケットを検知しない場合（False Negative）はないことに注意されたい．

この手法によって，ソフトウェアで処理するパケットの数を減らすことができるので，ソフトウェアだけで処理する場合と比べると，パケットを高速に処理することができる．例として後方参照を含むパターン “$([a\text{-}z]^{+})ak\backslash 1y$” に対して，ハードウェアでは “$([a\text{-}z]^{+})ak([a\text{-}z]^{+})y$” に対するマッチングを行った場合，パケット “b{100,000}” に対しては元のパターンと一致する可能性がないので，ソフトウェアでのマッチング処理を避けることができる．パケット “b{100,000}akwy” に対しては元のパターンと一致する可能性があると判断し，ソフトウェアによって実際のパターンのマッチングを行い，一致するかどうかを調べる．

また本節では，ハードウェアによるパケットの分類だけでなく，ソフトウェアによるマッチング処理を高速にする手法も提案する．

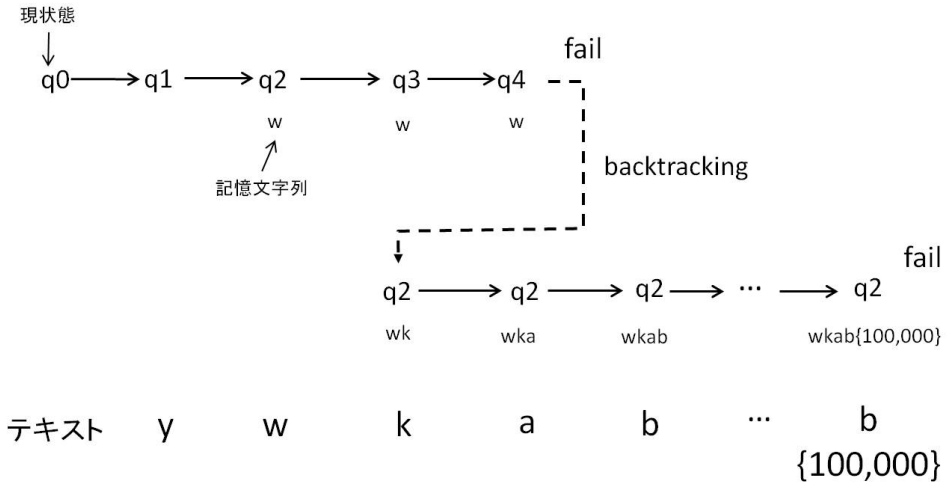


図 6.14: 動作例

ハードウェアでの前処理において，“R_1 (RB_i) R_2 (RB_i) R_3” と入力パケットに対する正規表現マッチングを行う際，パターンとパケットが一致するかどうかだけでなく，6.1.3 節で述べたように，パターンと一致する部分文字列の末尾文字を見つけることができる．例えば，“$([a\text{-}z]^+)ak([a\text{-}z]^+)y$” とパケット “b{100,000}akwy” に対するマッチングでは，’y’ が末尾文字であることをハードウェアで見つけることができる．この情報を用いて，ソフトウェアでのマッチング処理を高速化する．

ソフトウェアでは 6.1.3 節で述べたような後方参照を含むパターンの逆パターンと逆パケットに対してマッチングを行う．例えば，“$([a\text{-}z]^+)ak\backslash 1y$” とパケット “b{100,000}akwy” に対して，逆パターン “$y([a\text{-}z]^+)ka\backslash 1$” と逆パケット “ywkab{100,000}” に対するマッチングを行う．逆パターン “$y([a\text{-}z]^+)ka\backslash 1$” に対する拡張有限オートマトンと動作例を図 6.13 と図 6.14 に示す．初期状態 q0 から状態 q 1 への遷移は入力文字が’y’ かつハードウェアで見つけた末尾文字であることが条件となる．

一般的に，パターンと一致する部分文字列の末尾文字はあまり多くないため，探索回数を大幅に減らすことができ，ソフトウェアでのマッチング処理を高速化できると考えられる．

6.3　おわりに

本章では，拡張正規表現の先読み演算と後方参照に対するハードウェアマッチング手法を提案した．

先読み演算に対する提案マッチング手法では，テキストの末尾からテキストの先頭へ検索を行うことで先読み演算に対するマッチングを可能にする前処理回路を正規表現マッチングハードウェアに導入した．また，NIDS に適した新たなバッファ機構を提案した．実験結果において，4 章で提案したパターン非依存正規表現マッチングハードウェアに対して提案手法を適用し，効率的に先読み演算を扱えることを示した．

後方参照に対する提案マッチング手法では，マッチングハードウェアにより前処理によってソフトウェアによる後方参照マッチング処理が必要なパケットとそうでないパケットを高速に分類する．全てのパケットをソフトウェアで処理する従来の方法（Snort）と比べて，NIDS のウィルス検知処理の高速化が見込めることを示した．後方参照に対する提案マッチング手法の今後の課題として，実際のパケットに対して提案手法を適用し，提案手法の有効性を確かめることがあげられる．また，拡張有限オートマトンをハードウェアで実現することで，後方参照に対するマッチングをハードウェアのみで実現することも今後の課題である．

第7章　結論

7.1　まとめ

ネットワーク侵入検知システム（NIDS）はネットワーク上を流れる通信パケットを監視することで，不正侵入や攻撃を検知するためのシステムであり，現在のインターネット社会におけるセキュリティ向上のために必要不可欠なシステムの 1 つである．NIDS はコンピュータウィルスを定義したパターン（ウィルスパターン）と通信パケットに対する正規表現マッチングを行うことで，ウィルスの検知を行っている．

現在のネットワーク環境下における NIDS の正規表現マッチングにおいて，以下の要件を全て満たすことが望ましい．(要件 1）高速ネットワーク (ギガビットイーサネット) 上でリアルタイムなウィルス検知ができる，(要件 2）どんなウィルスパターンでも扱うことができる，(要件 3）多数のウィルスパターンをコンパクトな回路で扱ことができる，(要件 4）新しいウィルスに素早く対応するため，パターン更新を瞬時に行うことができる．

ソフトウェアによる正規表現マッチングでは，(要件 1）を満たす事が困難であるため，これまでパターン依存型とパターン非依存型に大きく分類される様々な正規表現マッチングハードウェアが提案された．しかし既存のマッチングハードウェアでは全ての要件を満たすことができなかった．

そこで，本論文では，ネットワークのセキュリティ向上に貢献することを目標とし，上述した全ての要件を満たす正規表現マッチングハードウェアを実現するための 3 つの手法を提案した．

まず第一に，本論文では，任意の正規表現を扱うことができるパターン非依存正規表現マッチングハードウェアを提案した．既存のパターン非依存正規表現マッチングハードウェアの多くは，正規表現の部分クラスしか扱うことができず，ウィルスを検知できない可能性が増えるという，セキュリティ面での問題があった．この課題を解決するため，任意の正規表現を扱うことができる NFA に基づくマッチングハードウェア（DPA 手法）が

提案されているが，実用的でない回路規模になるという問題点があった．そこで本論文では，任意の正規表現を扱うことができるが回路構造が複雑な DPA 手法と回路構造は単純であるが正規表現の部分クラスしか扱うことができないシストリックアルゴリズムに基づくマッチングハードウェアに注目し，2 つのハードウェアを組み合わせた新しいマッチングハードウェアを提案した．提案ハードウェアはシストリックアルゴリズムに基づく単純文字列の比較機能と NFA に基づく状態遷移機能の組み合わせにより，文字列に対して状態遷移する有限状態オートマトンをシミュレートする．実験的評価により提案ハードウェアが，任意の正規表現パターンに対応でき，かつ DPA 手法より回路構造が非常にコンパクトであることを示し，提案ハードウェアの有効性を示した．

上記で示した提案ハードウェア（提案ハードウェア 1）は実用的な回路規模で任意の正規表現を扱うことができるが，ハードウェアコストの面から回路規模はできる限り小さい方が望ましい．そのため，次に本論文では，FPGA の部分再構成機能を用いた回路規模削減手法を提案した．提案手法は，正規表現の異なる部分クラスを扱う部分回路をあらかじめ複数用意しておき，与えられた正規表現パターンに適したコンパクトな回路構成をこれらの部分回路の組み合わせにより自動生成する．パターンが更新された場合は変更が必要となった部分回路のみが部分再構成される．提案手法は，全てのパターンに対応可能になるように設計された提案ハードウェア 1 に比べ，コンパクトな回路を生成できる．またパターン依存型と違いパターンが更新された際に，時間のかかる回路の再設計を必要としないため，高速なパターンの更新が可能である．実験的評価から，提案手法は提案ハードウェア 1 と比べ，わずかにパターン更新時間を増やすだけで，63 %の回路面積を削減できることを示した．

NIDS のパターン記述においては，ユニオン等の基本演算子だけでなく量指定子等の様々な演算子を導入した拡張正規表現もよく用いられる．拡張正規表現を用いると，パターンをより簡潔に記述することができるだけでなく，正規表現では表せないパターンを記述することもできる．本論文では，最後に，拡張正規表現の中で既存のハードウェアマッチング手法が知られていない先読み演算と後方参照に対するマッチング手法を提案した．実験的評価から，これらの演算子をハードウェアで効率的に扱える事を示した．

本論文で提案した手法により，現在のギガビットイーサネット上において，NIDS においてパフォーマンスの低下を伴わないウィルス検知が可能となり，ネットワークのセキュリティ向上に寄与すると考えられる．

7.2　今後の課題

本節では，今後に実現されるインターネットの回線速度の向上，ウィルスパターンの増加，ウィルスの巧妙化に対してそれぞれ解決すべき課題を述べる．

まず第一として，近い将来に普及が進むと予想されている 10Gbps や 100Gbps のネットワークにおいても，NIDS においてパフォーマンスの低下を伴わずにウィルス検知を行って行かなければならない．マルチバイト処理可能なパターン非依存正規表現マッチングマッチングハードウェアの開発が挙げられる．現在，1 クロックサイクルでテキスト 1 文字（1 バイト）を処理している．これを 1 クロックで多文字（マルチバイト）処理することでマッチングハードウェアの性能向上につながり，将来のネットワーク速度に対応していきたいと考えている．

第 2 に，今後さらにウィルスの種類も増加し，それに伴いウィルスパターンも増加すると考えられるため，大量のウィルスパターンをうまくハードウェアで扱う方法が必要となる．これに対応するための 1 つの方法として，インターネットが複数のルーターを経由して情報をやり取りするという特徴を活かして，各ルータが異なる少数のウィルスパターンを扱い，ルータを経由する度にウィルスが排除されていく方法が考えられる．こうすることで，マッチングハードウェアの大規模化を避けることができると考えられる．この方法を実現するために，ルーティングを考慮したウィルスパターンの分配手法が必要となってくるため，その手法の考案も今後の課題としてあげられる．

最後にウィルスのさらなる巧妙化に対応するためには，文脈自由言語等の正規表現より強力な表現能力を持つ言語によるパターン記述も必要となると考えられる．そのため，そのような言語で記述されたパターンを扱うことができるマッチングハードウェアの提案も今後の課題としてあげられる．

謝辞

本研究を行うにあたり懇切な御指導と御教授を賜り，また本論文をまとめるにあたって親身になって御助言を頂いた，広島市立大学論理回路システム研究室の若林真一教授に深甚なる謝意を表します.

本論文の審査過程において，様々な御助言と御指導いただいた同大学井上智生教授と弘中哲夫教授に深謝申し上げます.

また，これまでの研究過程中に，様々な御助言と御指導をして頂いた論理回路システム研究室の永山忍准教授と稲木雅人助教，様々な御助言を頂いた同研究室の高橋隆一准教授と上土井陽子講師に深く感謝いたします。

また，プロジェクト進行に際し，有用な御意見を頂いた論理回路システム研究室の諸氏に心から感謝いたします.

最後に，現在まで温かく見守ってくれた両親に感謝いたします.

参考文献

[1] Wikipedia, ARPANET, http://ja.wikipedia.org/wiki/ARPANET.

[2] 総務省, 情報通信統計データベース, http://www.soumu.go.jp/johotsusintokei/.

[3] R. セジウィック, アルゴリズム 第 2 巻=検索・文字列・計算幾何, 近代科学社, 1992.

[4] 北研二, 津田和彦, 獅々堀正幹, 情報検索アルゴリズム, 共立出版, 2001.

[5] J. Aoe, Computer Algorithms: String Pattern Matching Strategies, *IEEE Computer Society Press Los Alamitos*, 1994.

[6] IEEE-SA, IEEE P802.3ba 40Gb/s and 100Gb/s Ethernet Task Force, http://www.ieee802.org/3/ba/.

[7] Cisco Systems, Cisco Visual Networking Index（VNI）: 予測と方法論、2012 〜 2017 年, http://www.cisco.com/web/JP/solution/isp/ipngn/literature/pdf/white_paper_c11-481360.pdf.

[8] L. Yang, "New pattern matching algorithms for network security applications," *Ph.D Thesis, Rutgers, The State University of New Jersey, USA*, May 2013.

[9] AV-TEST - The Independent IT-Security Institute, http://www.av-test.org

[10] Symantec Corporation, http://www.symantec.com/content/ja/jp/about/presskits/2013_Norton_Report.pdf.

[11] Trend Micro Incorporated, http://www.trendmicro.co.jp/jp/index.html.

[12] Y. Kawanaka, S. Wakabayashi, S. Nagayama, "A fast regular expression matching engine for an FPGA-based network intrusion detection system," *Proc. the 16th Workshop on Synthesis and System Integration of Mixed Information technologies*, pp.88-93, March 2009.

[13] A. V. Aho, M. J. Corasick, “Efficient string matching: An aid to bibliographic search,” *Communication of ACM, Vol.18, No.6*, pp.333-340, June 1975.

[14] S. Wu, U. Manber, “A fast algorithm for multi-pattern searching,” *TR 94-17, Department of Computer Science, University of Arizona*, 1994.

[15] N. Tuck, T. Sherwood, B. Calder, G. Varghese, “Deterministic memory-efficient string matching algorithms for intrusion detection,” *Proc. IEEE Infocom, vol.4*, pp.2628-2639, March 2004.

[16] P.A.V.Hall, G.R.Dowling, “Approximate string matching,” *ACM Computing Surveys, Vol.12, No.4*, pp.381-402, December 1980.

[17] 宇丹裕一郎, 若林 真一, 永山 忍, “近似正規表現マッチングのためのシストリックアルゴリズムとそのFPGA実装,” 電子情報通信学会論文誌D, Vol.J94-D, No.6, pp.935-944, June 2011.

[18] 高橋恒介, テキスト検索プロセッサ, 電子情報通信学会, 1992.

[19] 富田悦次, 横森貴, オートマトン・言語理論, 森北出版, 1992.

[20] 武田圭史, 磯崎宏, ネットワーク侵入検知, ソフトバンク パブリッシング株式会社, 2000.

[21] 特定非営利活動法人日本ネットワークセキュリティ協会, 2012 年度 情報セキュリティ市場調査報告書, `http://www.jnsa.org/result/2013/surv_mrk/2012fymarketresearchreport.pdf`.

[22] TRIPWIRE, INC., `http://www.tripwire.org/`.

[23] K. Wang, S. J. Stolfo, “Anomalous payload-based network intrusion detection,” *Lecture Notes in Computer Science Vol.3224*, pp. 203-222, September 2004.

[24] X. Wang, C. Pan, P. Liu, S. Zhu, “SigFree: a signature-free buffer overflow attack blocker,” *IEEE Transactions on Dependable and Secure Computing, Vol.7, Issue 1*, pp.65-79, March 2010.

[25] P. Porras, D. Schnackenberg, S. Staniford-Chen, M. Stillman, F. Wu, The Common Intrusion Detection Framework Architecture, `http://gost.isi.edu/cidf/drafts/architecture.txt`.

[26] Sourcefire, Inc., SNORT Network Intrusion Detection System, `http://www.snort.org/`.

[27] Jon Allen (JJ), perldoc.perl.org-Official documentation for the Perl programming language, `http://perldoc.perl.org/perlre.html`.

[28] Philip Hazel, PCRE - Perl Compatible Regular Expressions, `http://www.pcre.org/`.

[29] 近藤嘉雪, 定本Cプログラマのためのアルゴリズムとデータ構造, SOFTBANK Publishing, 1998.

[30] J. ホップクロフト, R. モトワニ, J. ウルマン, Information & Computing - 3 オートマトン 言語理論 計算論 I [第 2 版], サイエンス社，2003.

[31] D.E. Knuth, J.H. Morris, V.R. Pratt, “Fast pattern matching in strings,” *SIAM Journal on Computing, Vol.6, No.2*, pp.323-350, June 1977.

[32] R.M. Karp, M.O. Rabin, “Efficient randomized pattern-matching algorithms,” *IBM Journal of Research and Development Vol.31, No.2*, pp.249-260, March 1987.

[33] R.S. Boyer, J.S. Moore, “A fast string searching algorithm,” *Communications of the ACM, Vol.20, No.10*, pp.762-772, October 1977.

[34] J. E. Hopcroft, R. Motwani, J. D. Ullman, Introduction to Automata Theory, *Languages, and Computation (2nd Edition), Addison Wesley*, November 2000.

[35] 丸岡章, 計算理論とオートマトン言語理論：コンピュータの原理を明かす, サイエンス社, 2005.

[36] G. Navarro, M. Raffinot, Flexible pattern matching in strings: practical on-line search algorithms for texts and biological sequences, *Cambridge University Press*, 2002.

[37] K. Thompson, “Programming techniques: Regular expression search algorithm,” *Communications of the ACM, Vol.11, No.6*, pp.419-422, June 1968.

[38] V.M. Glushkov, “The abstract theory of automata,” *Russian Mathematical, Vol.16, No.5*, pp.1-53, October 1961.

[39] 2013 The Bro Project, “The bro network security monitor,” `http://bro.org/`.

[40] 末吉敏則, 天野秀晴, リコンフィギャラブルシステム, オーム社, 2005.

[41] Xilinx Inc., “Partial Reconfiguration User Guide,” `http://japan.xilinx.com/support/documentation/sw_manuals_j/xilinx14_6/ug702.pdf`.

[42] H. Yamamoto, “Regular expression matching algorithms using dual position automata,” *Journal of Combinational Mathematics and Combinatorial Computing, (71), pp.103-125*, November 2009.

[43] 山田菜穂子, 岩井啓輔, 黒川恭一, 天野英晴, “動的部分再構成を利用した切り替え可能な AES S-box 回路の評価,” 信学技報, Vol110, pp.127-132, January 2011.

[44] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner, “Algorithms to accelerate multiple regular expressions matching for deep packet inspection,” *Proc. SIGCOMM’06*, pp.339-350, September 2006.

[45] M. Becchi, P. Crowley, “An improved algorithm to accelerate regular expression evaluation,” *Proc. 3rd ACM/IEEE Symposium on Architecture for networking and communications systems*, pp.145-154, December 2007.

[46] R. Smith, C. Estan, S. Jha, and S. Kong, “Deflating the big bang: fast and scalable deep packet inspection with extended finite automata,” *Proc. SIGCOMM’08*, pp.207-218, August 2008.

[47] K. Wang, T. Qi, Y. Xue, J. Li, “Reorganized and compact DFA for efficient regular expression matching,” *Proc. 2011 IEEE International Conference on Communications*, pp.1-5, June 2011.

[48] R. Smit, C. Estan, S. Jha, “Backtracking algorithmic complexity attacks against a NIDS,” *Proc. 22nd Annual Computer Security Applications Conference*, pp.89-98, December 2006.

[49] G. Vasiliadis, M. Polychronakis, S. Antonatos, E. P. Markatos, S. Ioannidis, "Regular expression matching on graphics hardware for intrusion detection," *Proc. 12th International Symposium on Recent Advances in Intrusion Detection*, pp.265-283, September 2009.

[50] N. Cascarano, P. Rolando, F. Risso, R. Sisto, "iNFAnt: NFA pattern matching on GPGPU devices," *Proc. ACM SIGCOMM Computer Communication Review, Vol. 40 No.5*, pp.21-26, October 2010.

[51] X. Yu, M. Becchi, "Exploiting different automata representation for efficient regular expression matching on GPUs," *Proc. 18th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pp.287-288, February 2013.

[52] J. Bispo, I. Sourdis, J. M. P. Cardoso, S. Vassiliadis, "Regular expression matching for reconfigurable packet inspection," *Proc. 2006 IEEE International Conference on Field Programmable Technology*, pp.119-126, December 2006.

[53] R. Sidhu, V. K. Prasanna, "Fast regular expression matching using FPGAs," *Proc. 2001 IEEE International Symposium on Field-Programmable Custom Computing Machines*, pp.227-238, May 2001.

[54] T. Ganegedara, Y.E. Yang, V. K. Prasanna, "Automation framework for large-scale regular expression matching on FPGA," *Proc. 2010 IEEE International Conference on Field Programmable Logic and Applications*, pp.50-55, September 2010.

[55] Y. SangKyun, L. KyuHee, "Optimization of regular expression pattern matching circuit using at-most two-hot encoding on FPGA," *Proc. 2010 IEEE International Conference on Field Programmable Logic and Applications*, pp.40-43, September 2010.

[56] N. Yamagaki, R. Sidhu, S. Kamiya, "High-speed regular expression matching engine using multi-character NFA," *Proc. 2008 IEEE International Conference on Field Programmable Logic and Applications*, pp.131-136, September 2008.

[57] H. Nakahara, T. Sasao, and M. Matsuura, "A regular expression matching circuit based on a decomposed automaton," *Proc. 7th International Conference on Reconfigurable Computing: Architectures, Tools and Applications*, pp.16-28, March 2011.

[58] C.R. Clark, D.E. Schimmel, “Scalable parallel pattern matching on high-speed networks,” *Proc. 12th IEEE Symposium on Field Programmable Custom Computing Machines*, pp.249-257, April 2004.

[59] T.T. Hieu, T.N. Thinh, T.H. Vu, S. Tomiyama, “Optimization of regular expression processing circuit for NIDS on FPGA,” *Proc. 2011 IEEE Second International Conference on Networking and Computing*, pp.105-112, November 2004.

[60] Christopher R. Clark, David E. Schimmel, “Efficient reconfigurable logic circuits for matching complex network intrusion detection patterns,” *Proc. 2003 IEEE International Conference on Field Programmable Logic and Applications*, pp.956-959, September 2003.

[61] L.H. Long, T.T. Hieu, V.T. Tai, N.H. Hung, T.N. Thinh, D.D. Anh Vu, “Enhanced FPGA-based architecture for regular expression matching in NIDS,” *Proc. 2010 IEEE International Conference on Electrical Engineering/Electronics Computer Telecommunications and Information Technology*, pp.666-670, May 2010.

[62] P. Salvatore, G. Claudio, N. Enrico, T. Simone, B. Giuseppe, “Exploiting dynamic reconfiguration for FPGA based network intrusion detection systems,” *Proc. 2010 IEEE International Conference on Field Programmable Logic and Applications*, pp.10-14, September 2010.

[63] J. Divyasree, H. Rajashekar, Kuruvilla Varghese, “Dynamically reconfigurable regular expression matching architecture,” *Proc. International Conference on Application-Specific Systems, Architectures and Processors*, pp.120-125, July 2008.

[64] Y. Kaneta, S. Yoshizawa, S. Minato, H. Arimura, Y. Miyanaga, “Dynamic reconfigurable bit-parallel architecture for large-scale regular expression matching,” *Proc. 2010 IEEE International Conference on Field Programmable Technology*, pp.21-28, December 2010.

[65] 川中 洋祐, 若林 真一, 永山 忍, “パターン非依存正規表現ストリングマッチングマシンとその FPGA 実装,” 電子情報通信学会論文誌 D, Vol.J92-D, No.12, pp.2159-2167, December 2009.

[66] 若林真一, “制限された正規集合を受理する VLSI 向きパターン・マッチング・マシン,” 電子通信学会オートマトンと言語研究会技術研究報告, AL85-33, 1985.

[67] Y. Wakaba, M. Inagi, S. Wakabayashi, S. Nagayama, ”An extension of systolic regular expression matching hardware for handling iteration of strings using quantifiers,” *Proc. the 16th Workshop on Synthesis and System Integration of Mixed Information Technologies*, pp.412-417, October 2010.

[68] Y. Wakaba, M. Inagi, S. Wakabayashi, S. Nagayama, “An efficient hardware matching engine for regular expression with nested Kleene operators,” *Proc. 2011 IEEE International Conference on Field Programmable Logic and Applications*, pp.157-161, September 2011.

[69] H.T. Kung, Let’s design algorithm for VLSI systems, *Carnegie-Mellon Univ. Comput. Sci. Tech. Rep., CMU-CS-79-151*, 1979.

[70] Y. Han, D. Wood, “The generalization of generalized automata: expression automata,” *International Journal of Foundations of Computer Science, Vol.16 Issue 3*, pp. 499-510, June 2005.

[71] J. Goyvaerts, “Regular-expressions.info”, `http://www.Regular-Expressions.info/`.

[72] J. V. Leeuwen, コンピュータ基礎理論ハンドブック I アルゴリズムと複雑さ, 丸善株式会社,1994.

[73] M.Becchi, P.Crowley, “Extending finite automata to efficiently match Perl compatible regular expressions,” *Proc. the 2008 ACM CoNEXT Conference Article No. 25*, December 2008.

[74] 関山 守, 戸田 賢二, 神徳 徹雄, “大規模 FPGA を用いた規模拡張可能な高速ネットワーク侵入検知システムの開発,” 信学技報, SLDM, Vol.160 (45), pp.1-5, March 2013.