ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 42, No. 10, pp. 1280-1285, 2015. 10
http://dx.doi.org/10.5626/JOK.2015.42.10.1280

문자클래스 매칭을 지원하는 정규표현식 매칭 프로세서 구조

(Regular Expression Matching Processor Architecture Supporting Character Class Matching)

윤 상 균 †
(SangKyun Yun)

요 약 고속 정규표현식 매칭을 수행하기 위한 여러 종류의 정규표현식 매칭 하드웨어 구조가 연구되었다. 특히 프로그램과 같이 패턴의 갱신이 쉽도록 범용 프로세서와 유사한 방식으로 정규표현식 매칭을 수행하는 ReCPU와 SMPU와 같은 정규표현식 프로세서가 연구되었다. 그렇지만 기존의 정규표현식 프로세서들은 문자클래스 매칭을 위한 별도의 기능을 제공하지 않아서 문자클래스 처리에 비효율적이다. 본 논문에서는 문자클래스 매칭을 지원하는 정규표현식 매칭 프로세서의 명령어 집합을 제시하고, 이에 대한 프로세서 구조를 설계 구현한다. 제시된 프로세서는 문자클래스, 문자 범위와 부정 문자클래스 처리 기능을 포함하고 있어서 문자클래스 매칭을 매우 효율적으로 처리할 수 있다.

키워드: 정규표현식 매칭, 문자 클래스 매칭, 특화 프로세서 구조, 침입탐지

Abstract Many hardware-based regular expression matching architectures are proposed for high performance matching. In particular, regular expression processors such as ReCPU and SMPU perform pattern matching in a similar approach to that used in general purpose processors, which provide the flexibility when updating patterns. However, these processors are inefficient in performing class matching since they do not provide character class matching capabilities. This paper proposes an instruction set and architecture of a regular expression matching processor, which can support character class matching. The proposed processor can efficiently perform character class matching since it includes character class, character range, and negated character class matching capabilities.

Keywords: regular expression matching, character class matching, custom processor architecture, intrusion detection

1. 서 론

문자열 매칭은 네트워크 침입탐지 시스템, 프로토콜 파서 등의 응용분야에서 널리 사용되며, 문자열 패턴을 나타내는 데에 많은 문자열 패턴들을 간결하게 표현할 수 있는 정규표현식이 널리 사용되고 있다. 전통적인 소프트웨어 기반의 정규표현식 매칭은 처리 속도에 제약이 있기 때문에 고성능 처리 속도를 갖는 정규표현식 매칭을 위한 하드웨어 기반 구조들이 여러 연구자들에 의해서 제시되었다.

하드웨어 기반 구조들은 정규표현식 패턴을 갱신할 수 있도록 정규표현식에 대한 DFA(Deterministic Finite Automata) 또는 NFA(Nondeterministic FA)를 재구성이 가능한 FPGA를 사용하여 구현하는 방식들이 주로 제안되었다[1-3]. 그렇지만 이러한 방식의 설계는 패턴

· 이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임(과제번호 2011-0025467)

† 종신회원 : 연세대학교 컴퓨터정보통신공학부 교수
skyun@yonsei.ac.kr

논문접수 : 2015년 5월 18일
(Received 18 May 2015)
논문수정 : 2015년 8월 10일
(Revised 10 August 2015)
심사완료 : 2015년 8월 12일
(Accepted 12 August 2015)

이 갱신될 때마다 하드웨어 기술언어(HDL)로 기술된 설계를 다시 생성하고 FPGA를 재합성을 해야 하는 단점이 있다.

이러한 방식의 단점을 해결하기 위하여 하드웨어의 재합성이 필요 없는 구조들이 제안되었다[4-9]. 특히 정규표현식을 일련의 명령어로 변환하여 정규표현식 매칭을 보통의 프로세서와 유사한 방식으로 수행하는 정규표현식 프로세서에 대한 연구들이 수행되었으며 ReCPU[4-6]가 정규표현식 프로세서의 대표적인 예이다. 이 방법은 처리 속도는 특정 패턴용으로 설계된 하드웨어 기반의 구현보다는 느리지만 정규표현식 패턴을 보통의 프로그램처럼 쉽게 변경할 수 있는 장점이 있다.

ReCPU에서는 정규표현식을 일련의 명령어들로 변환하여 명령어 메모리에 저장하고, 명령어를 순서대로 실행한다. ReCPU는 패턴의 문자열을 명령어의 오퍼랜드로 나타내어 명령어를 실행할 때 입력 문자열과의 패턴 매칭을 수행하도록 구현되어 있으며, 한 번에 여러 바이트 매칭을 수행하도록 설계되었다. ReCPU는 괄호로 둘러싸인 (abcd)*와 같은 반복패턴이나 (abcd)|(efg)와 같은 OR 연산을 중첩된 괄호의 처리를 위하여 call-return 방식으로 수행하는 특징을 갖고 있는데 이 방식은 처리 속도가 다소 비효율적인 문제가 있다.

ReCPU의 단점을 개선한 정규표현식 프로세서로 SMPU와 REMP가 제시되었다. SMPU[7]는 ReCPU와 유사한 구조를 갖고 있지만 매칭이 실패된 경우의 처리 속도를 개선하기 위하여 이중 exit 구조를 제시했으며 반복 횟수의 범위를 지정하는 가변 반복 명령어도 구현하였다. REMP[8]는 반복패턴을 효율적으로 처리하기 위하여 함수 호출 기법을 사용하지 않는 방식을 채택하였고 짧은 패턴에 대한 반복 명령어를 별도로 제공하였다.

정규표현식에서는 패턴을 나타내는 데 문자 뿐만 아니라 [acef]와 같이 문자들의 집합을 나타내는 문자클래스도 사용할 수 있다. 그렇지만 기존의 정규표현식 프로세서들은 문자클래스에 대한 지원 기능을 별도로 포함하고 있지 않기 때문에 문자클래스를 a|c|e|f와 같이 문자들에 대한 연속적인 OR 연산으로 표현하여 여러 개의 명령어를 사용해야 하므로 비효율적이다.

본 논문에서는 이러한 문제점을 해결하기 위하여 문자클래스 매칭을 지원할 수 있는 명령어 형식 및 명령어와 이를 처리하는 정규표현식 프로세서의 구조를 제시한다.

2. 관련 연구

ReCPU[4,5]는 프로세서 방식으로 설계된 최초의 정규표현식 매칭 프로세서로서 ·(연결), |(OR), *(0번 이상 반복), +(1번이상 반복) 및 괄호 연산자와 같은 기본적인 정규표현식 연산자들을 처리할 수 있도록 설계되었다.

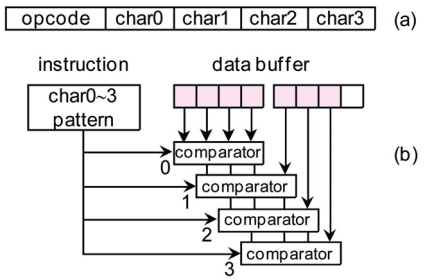

그림 1 ReCPU (a) 명령어 형식 (b) 비교기 클러스터
Fig. 1 ReCPU (a) instruction format (b) comparator cluster

ReCPU에서는 정규표현식 연산자와 문자 패턴을 그림 1(a)와 같이 opcode와 오퍼랜드로 형태로 나타내었으며 연속된 여러 개의 문자들을 오퍼랜드로 지정하여 여러 바이트의 데이터를 동시에 비교할 수 있도록 되어 있다. 그리고 그림 1(b)와 같이 오퍼랜드의 문자 패턴을 한 문자씩 시프트된 위치의 데이터 문자열과 동시에 비교하는 비교기 클러스터가 구성되어서 임의의 위치에서 매칭을 시작할 수 있도록 하였다. SMPU와 REMP도 이와 유사한 비교기 구조를 갖고 있다. abcd와 같이 한 명령어에 포함할 수 있는 길이의 문자열 패턴에 대한 매칭은 한 번에 처리할 수 있으며 더 긴 문자열 패턴은 연속된 명령어를 사용하여 처리한다.

ReCPU는 (abcde)*와 같은 반복이나 (abc)|(bcd)와 같은 OR 연산을 call-return 방식으로 구현한다. "("에 대응되는 명령어가 입력되면 abcde)*에 대한 명령어 시퀀스의 시작 주소를 call하며 ")*"에 대응되는 명령어가 입력되면 return동작을 수행하며 ")"다음의 메타문자에 따라서 적절한 동작을 함께 수행한다. SMPU[7]도 괄호에 대해서는 이와 유사한 방식으로 처리를 한다. 그렇지만 이와 같은 방식에서의 빈번한 함수 호출은 성능 저하의 요인이 된다. REMP[8]는 정규표현식 패턴을 괄호가 필요없는 전위 표기법으로 변환한 후 명령어로 표현하여 괄호에 대한 명령어가 필요없도록 하여 함수 호출의 오버헤드를 없앴다. 그리고 짧은 패턴에 대한 반복 명령어를 별도로 제공하여 짧은 패턴에 대한 반복을 특히 효율적으로 수행하도록 하였다.

정규표현식은 패턴을 나타내는 데 문자 뿐만 아니라 문자클래스도 사용할 수 있다. 정규표현식에서 문자클래스는 [abcde]와 같은 개별적인 문자들의 집합이나 [a-f]와 같은 문자 범위로 나타내거나 [a-fpqr]와 같은 이들의 조합으로 나타낸다. 그리고 포함되는 문자들 대신에 제외되는 문자들을 나타내기 위하여 ^를 함께 사용하여 [^abcd]와 같은 부정(negated) 문자클래스를 사용한다.

그렇지만 기존의 정규표현식 프로세서는 문자클래스에 대한 지원 기능을 포함하지 않고 있어서 이에 대한 처리를 OR 연산을 연속적으로 사용하여 수행해야 하므로 비효율적이다. 그러므로 정규표현식 프로세서에서 이러한 문자클래스를 효율적으로 처리할 수 있는 기능을 제공하는 것이 바람직하다.

3. 문자클래스 매칭 지원 프로세서 구조

3.1 명령어 집합

본 논문에서 제안하는 정규표현식 프로세서 REMPc는 본 저자가 제안한 REMP 명령어 집합[8]을 기반으로 하여 문자클래스 매칭 기능을 지원하도록 명령어 형식을 수정하고 범위 비교를 수행하는 명령어를 추가하였다.

문자클래스를 지원하기 위하여 그림 2와 같은 명령어 형식을 사용한다. 명령어는 최대 4문자의 오퍼랜드를 포함하며 유효한 오퍼랜드 문자 수는 sz+1개이다. 명령어의 오퍼랜드는 플래그 비트 C와 N의 값에 따라서 연속적인 문자열 패턴을 나타내거나 문자클래스에 속하는 문자 집합을 나타낸다. 앞의 경우에는 오퍼랜드를 같은 수의 연속적인 입력 문자열과 비교하여 (n대n 비교라고 부름) 모두 같으면 매칭 성공이 되며, 뒤의 경우에는 오퍼랜드의 문자들을 한 개의 입력 문자와 각각 비교하여 (n대1 비교라고 부름) 같은 것이 존재하면 매칭 성공이 된다.

명령어의 비교 동작은 표 1과 같이 플래그 값에 따라서 정해진다. C와 N이 모두 0일 때에 n대n 비교를 수행하고, 나머지 경우는 문자클래스 매칭을 위해 n대1 비교를 수행한다. N이 1일 때에는 비교기의 결과를 반대로 하여 부정 문자클래스 매칭에 사용된다. 매칭 성공이

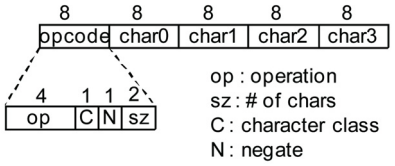

그림 2 명령어 형식
Fig. 2 Instruction format

표 1 오퍼랜드 유형과 비교 동작
Table 1 Operand type and compare operation

C	N	operand syntax	operation	
0	0	abcd	n-to-n compare	
0	1	[^abcd]!	n-to-1 compare (for character class matching)	inverse matching maintain current data
1	0	[abcd]		
1	1	[^abcd]		inverse matching

표 2 REMPc 명령어 집합
Table 2 Instruction set of REMPc

mnemoic	operation	format
CMP	simple compare	CMP c4
STAR STARX	simple * extended *	STAR c4 STAX c3, ra1
PLUSX	extended +	PLUSX c3, ra1
OPT OPTX	simple ? extended ?	OPT c4 OPTX c3, ra1
ORS OR ORX	save OR exit address simple | extended |	ORS c3, ra1 OR c4 ORX c3, ra1
E_REP E_OR	end of repeat block end of OR block	E_REP c3 E_OR c3
MATCH	match, report ID	MATCH c2, n2
CMPR ORR	compare with range simpe | with range	CMPR r2c1 ORR r2c1

c4, c3 : at most four or three characters, respectively
ra1 : one byte relative address, n2 : two byte number
r2c1 : start/end chars of a range and at most one char

면 데이터는 기본적으로 비교 데이터 다음으로 이동하지만 C=0, N=1인 경우에 부정 문자클래스 매칭이 성공이더라도 현재 데이터를 다음 명령어에서도 사용하도록 그대로 유지한다.

정규표현식 프로세서 REMPc의 명령어 집합은 표 2와 같다. REMP의 명령어 집합을 기본으로 하고 있으며 그림 2의 오퍼랜드 형식으로 나타낼 수 없는 범위 문자클래스 매칭을 위하여 범위 문자클래스 매칭을 수행하는 명령어인 CMPR과 ORR을 추가하였다. CMP는 순서대로 처리하는 문자열 매칭을 수행하며 STAR와 OPT는 각각 짧은 문자열에 대한 *와 ? 연산을 수행한다. 그리고 STARX, PLUSX, OPTX는 각각 한 명령어로 나타낼 수 없는 긴 서브패턴에 대한 *, +, ? 연산의 시작을 나타내며, E_REP는 긴 반복 서브패턴의 마지막을 나타낸다. 서브패턴들의 OR 연산은 ORS 명령어를 사용하여 전체 OR 블록의 다음 주소를 저장하는 것으로 시작된다. 짧은 OR 서브패턴은 OR 명령어로 나타내고, 긴 OR 서브패턴은 ORX 명령어로 시작하며, E_OR 명령어로 끝낸다. OR나 ORX 명령어가 실패하면 다음의 OR 서브패턴으로 이동하며, OR나 E_OR가 성공하면 ORS가 저장한 주소로 분기한다. 마지막 OR 서브패턴은 다음 서브패턴이 없으므로 OR나 ORX를 사용하지 않으며 E_OR 명령어로 끝낸다. MATCH는 매칭에 성공하면 패턴이 최종적으로 매칭된 것으로서 패턴 ID를 보고한다. REMP와 REMPc에서는 ReCPU와 SMPU에서와 달리 명령어들의 패턴 순서와 정규표현식의 문자열 순서가 일치한다. 다음은 두 개의 정규표현식을 REMPc 프로그램으로 나타낸 예이다. 분기주소는 +3과 같이 상

대주소를 사용하며, 사각형으로 표시한 것은 서브패턴을 나타낸다.

1) RE: abcde(pq(bc)*r)+s

```
CMP abcd
CMP e
PLUSX pq, +3
STAR bc
E_REP r
MATCH s, 1234
```

2) RE: (abc|pqrst|vwxyz)d

```
ORS +6
OR abc
ORX pqr, +2
E_OR st
CMP vwxy
E_OR z
MATCH d, 5678
```

3.2 문자클래스 처리

REMPc의 명령어는 표 1과 같이 플래그 비트 C와 N 값에 따라서 4개까지의 문자들로 구성된 문자클래스에 대한 매칭을 수행할 수 있다. 다음은 이를 사용하는 예이며 문자클래스 오퍼랜드는 CMP 명령어 뿐만 아니라 다른 명령어에서도 사용할 수 있다.

3) RE: [abcd]

```
CMP [abcd]
```

4) RE: [^abc]

```
CMP [^abc]
```

CMPR 명령어는 범위 비교를 수행하는 명령어로서 오퍼랜드의 첫째와 둘째 문자가 범위의 시작과 마지막 문자로 사용되며 추가로 문자클래스에 한 문자를 더 포함할 수 있다. 범위를 포함한 문자클래스는 다음과 같이 CMPR 명령어로 나타낼 수 있다.

5) RE: [a-fp]

```
CMPR [a-fp]
```

6) RE: [^a-f]

```
CMPR [^a-f]
```

하나의 명령어로 나타낼 수 없는 긴 문자클래스는 여러 개의 짧은 문자클래스들의 OR 연산으로 나타낼 수 있다. 나누어진 짧은 문자클래스들 중 하나에 속하게 되면 매칭 성공이 된다. 다음은 연속적인 OR 연산을 사용하여 나타낸 긴 문자클래스에 대한 프로그램의 예이다.

7) RE: [a-fpqrsxy] = [a-f] | [pqrs] | [xy]

```
ORS +4
ORR [a-f]
OR [pqrs]
E_OR [xy]
```

OR 연산에서 범위 비교 기능을 수행하고자 할 때는 위의 예에서와 같이 OR 대신에 ORR 명령어를 사용한다.

하나의 명령어로 나타낼 수 없는 부정 문자클래스는 여러 개의 짧은 부정 문자클래스 매칭 결과를 결합하여 나타낼 수 있다. 입력 문자가 여러 개의 짧은 부정 문자클래스들에 모두 속하게 되면 부정 문자클래스 매칭은 성공이 된다. 연속되는 짧은 부정 문자클래스 매칭이 같은 입력문자에 대해서 계속하여 수행해야 하므로, 마지막이 아닌 부정 문자클래스 매칭 명령어는 매칭 성공이더라도 데이터를 이동하지 않고 현재 데이터를 유지해야 한다. 이를 위하여 표 1의 두 번째 오퍼랜드 유형(C=0, N=1)이 사용되며 [^abcd]!와 같이 !를 함께 표기한다. 다음은 긴 부정 문자클래스에 대한 프로그램의 예이다.

8) RE: [^abcdp-rxy] = [^abcd] & [^p-r] & [^xy]

```
CMP [^abcd]!
CMPR [^p-r]!
CMP [^xy]
```

제안된 REMPc의 명령어 집합은 이와 같이 범위 비교와 부정 문자클래스를 포함한 모든 유형의 문자클래스 매칭을 처리할 수 있다.

3.3 프로세서 구조

REMPc의 명령어 집합을 구현하는 프로세서의 구조는 그림 3과 같으며 REMP의 구조를 바탕으로 REMPc의 수정된 오퍼랜드 형식과 명령어를 지원하도록 하였다. 정규표현식 패턴에 대한 명령어는 명령어 메모리에 저장되어 있고 검사 대상인 데이터는 데이터 메모리에 저장되어 있다. pc 레지스터와 addr 레지스터는 각각 인출되어야 하는 명령어 메모리와 데이터 메모리의 주소들을 저장하며 패턴매칭이 시작될 때에 패턴의 시작주소(I_start)와 데이터의 시작주소(D_start)로 초기화된다. 명령어 메모리에서 명령어를 인출하여 데이터 메모리에서 인출된 데이터와 명령어의 오퍼랜드와 비교기 클러스터에서 매칭을 수행한다.

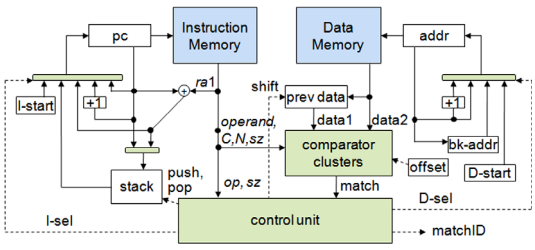

그림 3 REMPc의 프로세서 구조

Fig. 3 Processor architecture of REMPc

비교기 클러스터는 그림 1(b)와 유사하게 4개의 비교기로 구성되어 있지만 각 비교기는 문자클래스 매칭을 지원하기 위하여 그림 4와 같은 구조를 갖는다. 비교기는 4-4 비교기, 4-1 비교기, 범위 비교기로 구성되며 두 플래그 비트 C와 N과 range 신호에 따라서 비교기 출력을 선택하여 매칭 출력으로 내보낸다. range 신호는 명령어가 범위비교 명령어 CMPR 또는 ORR일 때에 1이 된다.

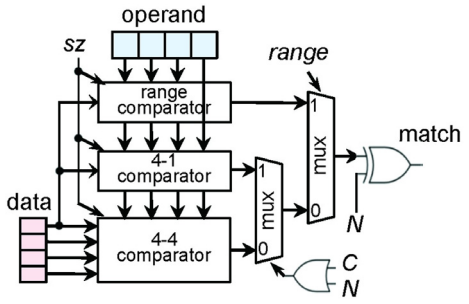

그림 4 비교기 구조
Fig. 4 Comparator architecture

제어 장치는 명령어의 opcode와 비교기의 매칭 결과 등에 따라서 다음의 명령어 주소와 데이터 주소를 결정한다. 비교기 클러스터가 7바이트의 데이터를 필요로 하기 때문에 두 개의 32비트 데이터를 연속하여 읽은 후에 프로그램의 실행이 시작된다. 먼저 읽은 데이터는 prev data 레지스터에 저장된다. 첫 번째 명령어에서는 모든 위치에서 데이터 매칭을 수행하는 멀티 클러스터 모드로 동작하여 4개의 비교기 모두에서 비교를 하고 매칭되는 것이 있으면 다음 명령어로 진행을 한다. 다음 명령어 부터는 단일 클러스터 모드로 동작하여 이전 명령어에서 매칭된 비교기 위치와 오퍼랜드 크기에 의해서 정해진 오프셋에 해당하는 하나의 비교기의 결과만 사용한다.

명령어 주소 pc의 다음 값은 명령어 opcode와 매칭 결과에 따라서 다음의 5개 값 중에서 선택된다: (1) 다음 주소 (pc+1) (2) 현재 주소 (pc) (3) 분기 주소 (pc+ra1) (4) 스택 저장 주소 (5) 시작 주소(I_start). 데이터 주소 addr의 다음 값은 다음의 4개의 값 중에서 선택된다. (1) 다음 주소 (addr+1) (2) 현재 주소 (addr) (3) 재시작 주소(bk_addr) (4) 시작 주소 (D_start). addr가 다음 주소로 증가하는 경우에 현재 데이터는 prev data 레지스터에 저장된다.

멀티 클러스터 모드에서 매칭이 실패했을 때에 addr를 다음 주소로 이동하여 다음 데이터에 대해서 다시 매칭을 시도한다. 매칭이 성공하면 비교기는 단일 클러스터 모드가 되고 사용할 비교기 위치를 나타내는 2비트 오프셋은 이전에 매칭된 비교기 위치와 sz+1의 합으로 정해지며, 이 덧셈에서 오버플로가 발생하면 addr가 1이 증가되어 다음 주소로 설정된다. 멀티 클러스터 모드에서 매칭이 성공일 때에 매칭이 시작되는 데이터의 다음 주소인 addr를 bk_addr로 저장하여 패턴매칭이 실패할 경우에 패턴매칭을 재시작하는 데이터 주소로 사용한다. 단일 클러스터 모드에서 매칭이 성공하면, 2비트 오프셋은 오퍼랜드 크기만큼 증가되고 다음의 addr는 멀티 클러스터 모드에서와 마찬가지로 오버플로 여부에 따라 결정된다.

4. 평가

3절에서 제시한 구조의 REMPc 프로세서는 Verilog HDL로 작성되어 Altera Stratix IV GX FPGA를 타겟 디바이스로 하여 Quartus II v13.0 프로그램을 사용하여 합성하였다. 프로그램과 데이터는 FPGA 내부의 블록 RAM에 초기값으로 지정하여 구현하였으며 여러 종류의 문자클래스를 포함한 정규표현식에 대해서 동작을 확인하여 설계를 검증하였다.

표 3은 기존의 정규표현식 프로세서와 REMPc의 문자클래스 매칭을 구현하는 방법을 비교한 것이다. 기존의 프로세서는 문자클래스를 집합에 속한 개별 문자들의 OR 연산을 이용하여 구현해야 하며 문자 범위와 부정 문자클래스도 모두 보통의 문자클래스로 변환한 후 OR 연산을 이용하여 구현해야 한다. 그러므로 문자클래스에 속한 문자 개수의 명령어가 필요하며 특히 문자 범위와 부정 문자클래스 처리에 매우 비효율적이다. 이에 비해 REMPc는 한 번에 4개까지의 문자클래스를 처리할 수 있으며, 문자 범위와 부정 문자클래스 등 모든 유형의 문자클래스 처리 기능이 포함되어 있으므로 문자클래스를 효율적으로 처리할 수 있다. 예를 들어서 기존 프로세서들은 [a-z]를 26개의 문자들의 OR로 나타내어야 하며, [^abc]는 253개의 문자들의 OR로 나타내어야 하지만 REMPc는 두 가지 모두 하나의 명령어로 처리할 수 있다.

표 4는 설계된 REMPc의 FPGA합성 결과이다. 클럭의 가능한 최대 속도는 166.42 MHz로서 매 클럭마다 4바이트씩 계속하여 처리하는 경우에 5.32Gbps의 최대 처리속도를 가질 수 있다. 비교기가 상대적으로 복잡해

표 3 문자클래스 매칭의 구현
Table 3 Implementation of character class matching

Type	REMPc	ReCPU, SMPU, REMP
Character Class	○	OR of individual characters
Character Range	○	OR of all characters in the range
Negated Char. Class	○	OR of all characters except specified characters

표 4 REMPc에 대한 FPGA 합성 결과
Table 4 FPGA synthesis results for REMPc

Device	ALUTs	Registers	Freq	Speed
EP4SGX230 KF40C2	341	158	166.42 MHz	~5.32 Gbps
	341 ALU/Reg pairs			

졌기 때문에 문자클래스 매칭의 지원기능이 없는 설계의 최대 클록속도 175.44 MHz[8]보다 클록속도가 낮다.

5. 결 론

최근에 정규표현식을 명령어 형태로 나타낸 후에 프로세서 방식으로 처리하는 ReCPU, SMPU와 같은 정규표현식 프로세서가 제시되었다. 그렇지만 기존의 정규표현식 프로세서는 문자클래스에 대한 지원 기능을 포함하지 않는다. 본 논문에서는 이러한 문제점을 해결하기 위하여 문자클래스 매칭의 지원 기능이 있는 정규표현식 프로세서의 명령어 집합을 제시하고 이에 대한 프로세서를 설계, 구현하였다.

제안된 REMPc는 단순한 문자클래스 뿐 만 아니라, 범위 비교, 부정 문자클래스의 처리 기능을 포함하고 있고 여러 개의 명령어를 사용하여 긴 문자클래스를 처리하는 방법도 제시하고 있다. 이러한 기능을 포함하지 않은 기존 정규표현식 프로세서에 비해서 문자클래스 매칭을 매우 효율적으로 처리할 수 있다.

References

[1] R. Sidhu and V. Prasanna, "Fast regular expression matching using FPGAs," *IEEE Symp. Field Prog. Custom Computing Machines (FCCM'01)*, 2001.

[2] C.-H. Lin, C.-T. Huang, C.-P. Jiang, and S.-C. Chang, "Optimization of regular expression pattern matching circuits on FPGA," *Proc Conf. Design, Automation and Test in Europe (DATE'06)*, 2006.

[3] J. C. Bispo, I. Sourdis, J. M. Cardoso, and S. Vassiliadis, "Regular expression matching for reconfigurable packet inspection," *IEEE Int. Conf. Field Programmable Technology (FPT'06)*, 2006.

[4] M. Paolieri, I. Bonesana, M. Santambrogio, "ReCPU: a parallel and pipelined architecture for regular expression matching," *Proc. IFIP Int. Conf. VLSI-SoC*, 2007.

[5] I. Bonesana, M. Paolieri, and M. Santambrogio, "An adaptable FPGA-based system for regular expression matching," *Proc. Conf. Design, Automation and Test in Europe (DATE'08)*, 2008.

[6] F. Bruschi, M. Paolieri, and V. Rana, "A reconfigurable system based on a parallel and pipelined solution for regular expression matching," *Int. Conf. Field Programmable Logic and Appl. (FPL'10)*, 2010.

[7] Q Li, J. Li, J. Wang, B. Zhao, and Y. Qu, "A pipelined processor architecture for regular expression string matching," Microprocessors and Microsystems, Vol. 36, No. 6, pp. 520-526, Aug. 2012.

[8] B. Ahn, K.H Lee, and S.K. Yun, "Regular expression matching processor supporting efficient repetitions," *Journal of KIISE : Computing Practices and Letters*, Vol. 19, No. 11, pp. 553-558, Nov. 2013. (in Korean)

[9] T. Arumugam, S. Sezer, D. Burns, and V. Vasu, "High performance custom regular expression processing core," *Irish Signal and System Conf. (ISSC'11)*, 2011.

윤 상 균

1984년 서울대학교 전자공학과(학사). 1986년 KAIST 전기및전자공학과(석사). 1995년 KAIST 전기및전자공학과(박사). 1984년~1990년 현대전자(주). 1992년~2001년 서원대학교 전자계산학과 교수. 1998년~1999년 University of Michigan, Ann Arbor, Visiting Scholar. 2001년~현재 연세대학교 컴퓨터정보통신공학부 교수. 관심분야는 컴퓨터 및 네트워크 시스템. 임베디드시스템, NIDS