논문 21-46-09-21 The Journal of Korean Institute of Communications and Information Sciences '21-09 Vol.46 No.09
https://doi.org/10.7840/kics.2021.46.9.1515

제약 및 중첩 반복 연산을 지원하는 정규표현식 매칭 프로세서 구조

서 병 석◆°

Regular Expression Matching Processor Architecture Supporting Restraint and Nested Repetitive Operations

Byung-suk Seo◆°

요 약

정규표현식 패턴 매칭은 네트워크 침입 탐지 시스템(NIDS) 등의 응용 프로그램에서 널리 사용되며 시간 제약이 있어 고성능 처리가 필요한 경우에 하드웨어 기반 패턴 매칭이 사용된다. 패턴이 업데이트될 때마다 재합성 해야 하는 하드웨어 기반 방식의 문제점을 해결하여 프로세서 기반 정규표현식 매칭 프로세서인 ReCPU, SMPU, REMP가 제안되었다. 그러나 이 프로세서 기반 정규표현식 매칭 프로세서들은 정규표현식의 반복 연산을 비효율적으로 처리한다. 본 논문에서는 ReCPU, SMPU의 비효율적인 반복 연산을 개선하기 위해 새로운 명령어 세트를 제시한다. REMP의 명령어 세트를 기반으로 하는 효율적인 반복 연산이 가능한 정규표현식 매칭 프로세서 REMPr을 제안한다. REMPr은 특히 짧은 서브 패턴을 OR의 반복 연산으로 처리하는 비효율적인 방법을 개선하고, 단일 명령어로 처리할 수 있도록 하였다. 또한 다운 카운터와 카운터 스택을 사용하여 제약 및 중첩 반복 연산도 효율적으로 처리하도록 하였다. REMPr은 Verilog로 설계하고 Intel Stratix IV FPGA로 합성하여 동작을 검증하였다.

Key Words : regular expression, regular expression matching processor, special-purpose processor, processor architecture, intrusion detection

ABSTRACT

Regular expression pattern matching is widely used in applications such as network intrusion detection systems (NIDS). Hardware-based pattern matching is used when high-performance processing is required due to time constraints. ReCPU, SMPU, and REMP, which are processor-based regular expression matching processors, have been proposed to solve the problem of the hardware-based method that requires resynthesis whenever a pattern is updated. However, these processor-based regular expression matching processors inefficiently handle repetitive operations of regular expressions. In this paper, we propose a new instruction set to improve the inefficient repetitive operations of ReCPU and SMPU. We propose REMPr, a regular expression matching processor that enables efficient repetitive operations based on the REMP instruction set. REMPr improves the inefficient method of processing a particularly short sub-pattern as a repeat operation OR, and enables processing with a single instruction. In addition, by using a down counter and a counter stack, nested repetitive operations are also efficiently processed. REMPr was described with Verilog and synthesized on Intel Stratix IV FPGA.

◆° First and Corresponding Author : Sangji University, Department of Information Security, seobs@sangji.ac.kr, 정회원
논문번호 : 202107-161-D-RE, Received July 11, 2021; Revised September 2, 2021; Accepted September 3, 2021

Ⅰ. 서 론

문자열 패턴 매칭은 네트워크 침입 탐지 시스템(NIDS), 컴퓨터 바이러스 백신 그리고 DNA 염기서열 분석 등의 애플리케이션에서 계산량이 많은 작업으로 사용되고 있다. 문자열 패턴은 종류와 수가 상대적으로 많아 간결하면서도 효율적으로 표현하기 위해 정규표현식(regular expression)[1]을 많이 사용한다.

정규표현식 매칭은 계산량이 많고 순차적인 처리 때문에 소프트웨어 기반 매칭에서 처리 속도에 제약이 있다. 소프트웨어 기반 정규표현식 매칭의 성능 향상을 위해 하드웨어 기반 솔루션들이 제안되었다[2-4]. 이 솔루션들은 주어진 정규표현식에 대한 하드웨어 기술 언어(hardware description language, HDL)를 사용하여 설계하고 FPGA(Field Programmable Gate Array)에서 구현한다.

그러나, 패턴이 업데이트될 때마다 하드웨어 기술 언어를 사용한 재생성 및 FPGA의 재합성이 필요하다. 이러한 성능저하 작업을 피하기 위해 정규표현식을 일련의 명령어로 나타내어 보통의 프로세서와 비슷한 방식으로 패턴 매칭을 수행하는 정규표현식 매칭 프로세서들이 제안되었으며 ReCPU[5,6]가 대표적인 예이다. 이 정규표현식 매칭 프로세서는 하드웨어를 재합성할 필요가 없는 유연성을 보장한다. 그림 1은 ReCPU의 (a) 명령어 형식과 (b) 비교기 클러스터(comparator clusters)를 보여주고 있다.

ReCPU에서 정규표현식은 명령어 메모리에 저장되는 명령어 시퀀스로 매핑되는데, 정규표현식의 반복 패턴에 비효율적이다. 이런 반복 패턴에 대한 비효율적인 처리를 개선한 프로세서 기반 정규표현식 매칭 프로세서들이 제안되었다. Li et al.[7]는 정규표현식 매칭을 위한 SMPU(String Matching Processing Unit)를 제안했다. SMPU는 LOOP 명령을 사용하여 제약 반복 연산을 구현했지만, *(ab){10}*과 같은 짧은 패턴에 대한 반복 연산을 여러 명령어로 표현하기 때문에 여전히 비효율적이다.

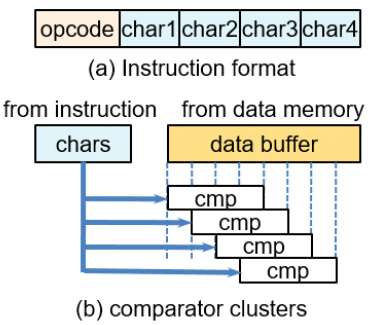

그림 1. ReCPU (a) 명령어 형식 (b) 비교기 클러스터
Fig. 1. ReCPU (a) instruction format (b) comparator clusters

이러한 짧은 패턴에 대한 비효율적 연산은 명령어 세트 구조 때문이다. Ahn et al.[8]은 짧은 반복과 긴 반복 패턴을 위한 별도의 명령어를 두고 반복 연산의 속도를 향상시키는 새로운 명령어 세트를 구현하기 위해 REMP(Regular Expression Matching Processor)를 제안했다. 하지만, REMP도 *{n}, {n,}* 및 *{m, n}* 과 같은 제약 반복 연산을 지원하지 않는다. 효율적인 정규표현식 패턴 매칭을 위해서는 프로세서의 명령어 세트 구조가 매우 중요하다.

위에서 언급한 정규표현식 매칭 프로세서[6-8]는 제약 반복 연산을 RE_1 / RE_2 / ⋯ / RE_n처럼 OR로 변경하여 순차적으로 처리하기 때문에 반복 패턴에 대한 매칭에 효율적이지 않다. 반복 정규표현식 매칭의 병렬처리를 위해 Rana et al.[9]는 ReCPU 코어 세트를 사용하는 재구성이 가능한 구조를 제안했다. 그러나, 이 구조도 Snort[10]와 같은 많은 양의 정규표현식 매칭에는 여전히 비효율적이다.

본 논문에서는 기존의 정규표현식 매칭 프로세서에서 제약 반복 연산의 문제점을 해결할 수 있는 명령어 세트와 명령어 형식을 제시하고 이를 처리하는 프로세서 구조를 제시한다. 이를 통하여 정규표현식의 제약 및 중첩 반복 처리를 효율적으로 수행할 수 있도록 한다.

Ⅱ. 본 론

2.1 명령어 세트

2.1.1 명령어 형식

그림 2와 표 1은 본 논문에서 제안하는 새로운 명령어 형식과 명령어 세트를 보여준다. 그림 2에서 명령어는 4가지 형식을 사용한다. 형식 1은 오퍼랜드(operand)에 최대 4개의 문자(*c4*)를 지정하며, 형식 2는 최대 3개(*c3*)의 문자와 바이트 오프셋 주소(*a1*)를 지정하고, 형식 3은 최대 2개(*c2*)의 문자와 MATCH 명령에 지정하여 사용한다. 형식 4가 본 논문에서 새롭게 추가한 반복 연산을 위한 카운트 정보(*n2, {m,n}*)와 바이트 오프셋 주소를 지정하여 처리하는 명령어 형식이다. 오피코드(opcode) 부분은 명령어(*op*) 4비트와 오퍼랜드의 문자의 개수를 나타내는 사이즈(*sz*) 2비트로 구성되어 있다.

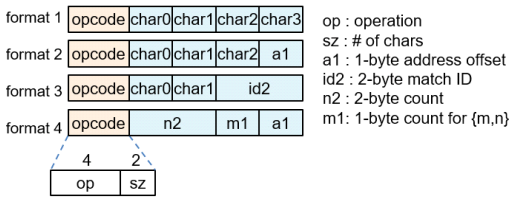

그림 2. 명령어 형식
Fig. 2. Instruction format

2.1.2 명령어 세트

표 1의 명령어 세트는 오피코드(opcode) 내의 명령어(*op*) 4비트로 16개까지 만들 수 있지만, 11개(0~10번)의 명령어로 구성되어 있다.

CMP는 순서대로 처리하는 최대 4개의 문자에 대해 매칭을 수행하며 STAR(*), OPT(?), OR(|)는 각각 (abc)*, (abcd)?, (ab|···)와 같은 짧은 하위 패턴에 대해 빠른 처리를 위한 단일 명령어로 매칭을 수행한다.

ORX, STARX는 단일 명령어로 매칭을 수행할 수 없는 긴 하위 패턴에 대한 |, * 의 시작을, ORS는 OR 블록의 다음 주소를 저장하는 준비단계를 나타낸다. EORX는 긴 하위 패턴의 마지막을 나타낸다.

REPcc(REPEQ, REPGE, REPLE, REPMN)는 하위 패턴에서 제약 반복 연산 *{n}, {n,}, {,n}, {m,n}*의 반복 블록의 다음 주소를 저장하는 준비단계를 나타낸다. 이러한 반복을 준비하는 명령어는 반복된 하위 패턴 이후의 명령어에 대한 8비트 오프셋 주소(*a1*)를 포함한다. EREP는 STARX와 REPcc의 마지막을 나타낸다.

MATCH는 패턴의 마지막 부분을 나타내며 일치하는 경우 16비트 패턴 ID(*id2*)를 보고한다.

표 1. 명령어 세트
Table 1. Instruction set

Op	Mnemonics	Example	Description
0	CMP c4	abcd	compare
1	STAR c4	(abcd)*	short star(*) repeat
2	OPT c4	(abcd)?	short optional(?) repeat
3	OR c4	abcd |	short alternative(OR)
4	ORX c3, a1	abc···|	long OR
5	ORS c3, a1	abc(···|···)	prepare OR
6	STARX c3, a1	abc(···)*	long star(*) repeat
7	REPLE n2, a1	(···){ ,10}	restraint repeat (REPcc)
	REPGE n2, a1	(···){1, }	
	REPEQ n2, a1	(···){10}	
	REPMN m1,n2,a1	(···){2,10}	
8	MATCH c2, id2	…ab (last of patterns)	final match and report
9	NOP		no operation
10	EREP c3	REPcc/STARX ... EREP	end of repeat block
	EORX c3	ORX ... EORX	end of ORX block
~15	reserved	-	-

2.1.3 정규표현식 매칭 프로그램

그림 3은 표 1의 명령어 세트로 작성한 다양한 프로그램을 보여준다. 그림 3(a)는 짧은 하위 패턴에 대한 STAR(*) 연산을 포함하고 있고, 그림 3(b)는 STARX의 예인데, 최대 3개의 문자에 대해 일치하는 단일 명령어로 나타낼 수 없는 긴 하위 패턴에 대한 STAR(*) 연산을 준비한다. 마지막은 EREP로 나타낸다.

그림 3(c)~(e)는 제약 반복을 포함하는 프로그램을 보여준다. REPcc 명령어는 제약 조건 반복에 대한 카운터를 초기화한다. EREP는 반복되는 하위 패턴의 마지막에 사용되며 일치하는 경우 명령어 시퀀스는 스택에서 얻은 하위 패턴의 시작 주소로 돌아가고 카운터가 업데이트된다. 그림 3(g)는 중첩된 반복을 포함하는 패턴에 대한 프로그램을 보여준다.

그림 3(f)는 OR 하위 패턴을 포함하는 패턴 블록 *pq(mn|abcdefgh|x)r*에 대한 프로그램을 보여준다. ORS는 최대 3개의 문자에 대해 일치하는 단일 명령어로 나타낼 수 없는 긴 하위 패턴에 대한 OR(|) 연산을 준비한다. OR의 짧은 하위 패턴은 단일 OR 명령어로 사용되며 일치하지 않으면 OR(|)의 대체 주소가 다음 주소가 된다. ORX는 OR의 긴 하위 패턴에서

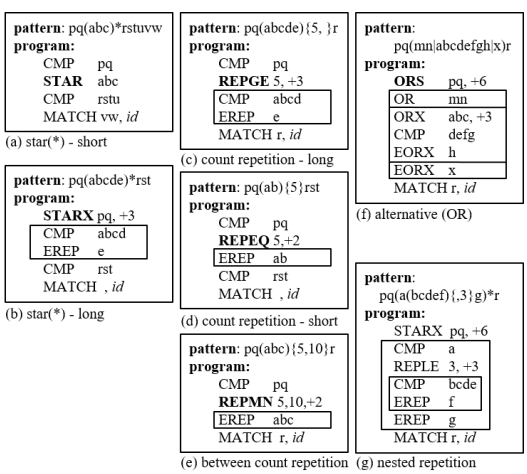

그림 3. 정규표현식 매칭 프로그램의 예
Fig. 3. Example of regular expression matching programs

사용하며 만약 일치에 실패했을 때 이동할 다음 하위 패턴의 오프셋 주소를 포함한다. EORX는 OR의 긴 하위 패턴의 끝이나 마지막 OR 하위 패턴에서 사용된다. EORX 또는 OR에서 일치하게 되면 명령어 시퀀스는 모든 OR 하위 패턴 다음의 명령어 주소로 이동하게 된다.

2.2 제안된 프로세서 구조

2.2.1 프로세서 구조

2.1.2에서 제시된 명령어 세트를 구현하는 프로세서 구조는 그림 4와 같다. 정규표현식 패턴 매칭에 대한 명령어는 명령어 메모리에 저장되어 있고 조사 대상인 데이터는 데이터 메모리에 저장되어 있다.

명령어 메모리에서 명령어를 인출하여 데이터 메모리에서 인출된 데이터와 명령어의 오퍼랜드와 비교기 클러스터에서 매칭을 수행한다. 비교기 클러스터는 그림 1(b)와 유사한 구조이다.

제어 로직은 명령어의 오피코드와 매칭 결과 등에 따라 다음 명령어의 주소와 데이터 주소를 결정한다. 비교기 클러스터가 7바이트의 데이터를 필요로 하기 때문에 두 개의 32비트 데이터(*data0*, *data1*)를 연속해서 읽은 후에 프로그램이 실행된다.

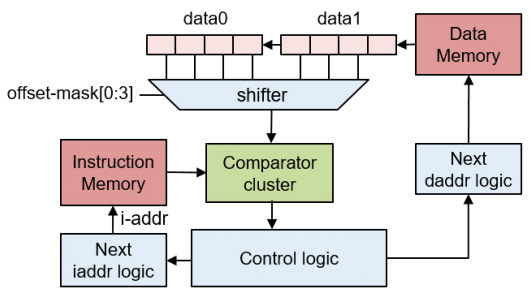

그림 4. 정규표현식 매칭 프로세서의 구조
Fig. 4. Regular expression matching processor architecture

2.2.2 제약 및 중첩된 반복 연산의 구현

그림 5(a)는 제약 및 중첩 반복 연산을 위한 카운터 로직 블록을 보여준다. 다운 카운터(*down counter*)와 보조 카운터(*sN*)를 사용하여 제약 반복 연산을 구현하고 카운터 스택(*counter stack*)을 사용하여 중첩된 반복 연산을 구현한다. REPcc 또는 STARX가 수행되면 카운터 값(*n2*), 카운터 상태(*cstate*), 그리고 보조 카운터의 값이 카운터 스택에 적재(load)된다.

REPGE, REPEQ, 또는 REPLE 명령에서, 카운터 값은 N=n-1로 다운 카운터에 적재(load)되고, 카운터

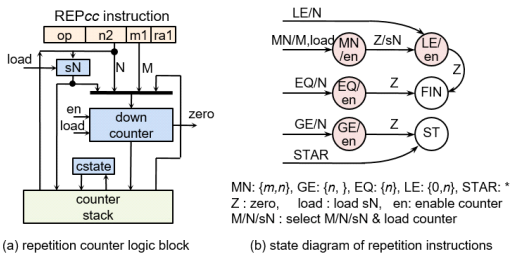

그림 5. 제약 및 중첩 반복 연산을 위한 카운터 로직 블록과 상태 다이어그램
Fig. 5. Counter logic block and state diagram for restraint and nested repetitive operations

상태(*cstate*)는 각각 *GE*, *EQ*, *LE*로 설정된다. REPMN 명령에서는 카운터 값은 M=m-1로 다운 카운터에, 보조 카운터 값은 N=n-m-1로 카운터 스택에 적재된다. 카운터 상태는 *MN*으로 설정된다. STARX 명령은 카운터 상태가 *ST*로 설정된다.

그림 5(b)는 다양한 반복 조건에 따른 상태 변이를 보여준다. EREP 명령이 0이 아닌 카운터 값에 일치하면 카운터가 감소하고 명령어 시퀀스는 스택에 저장된 반복 블록의 시작으로 이동하게 된다. EREP 명령이 제로 카운터 값이 되면, 현재 카운터 상태(*cstate*)에 따라 다음 연산이 결정된다. 카운터 상태가 *MN*, *LE*, 또는 *ST*인 경우 명령어 시퀀스는 반복 블록의 시작으로 이동한다. 카운터 상태가 *MN*인 경우, 보조 카운터(*sN*)를 카운터 스택(*counter stack*)에 적재(load)하고 카운터 상태는 *LE*가 되고, *GE*인 경우 카운터 상태는 *ST*가 된다. 카운터 상태가 *LE*, 또는 *EQ*이면 현재 반복이 완료(*FIN*)되고, 명령어 시퀀스는 스택에 저장된 반복 블록 이후 패턴 주소로 이동하게 된다.

Ⅲ. 실험 및 평가

설계된 정규표현식 매칭 프로세서 구조는 Verilog HDL로 작성되어 Intel Stratix IV FPGA를 타겟 디바이스로 하여 Quartus Prime v18.0 프로그램을 이용하여 합성하였다. 그리고 그림 3의 프로그램 외 여러 종류의 정규표현식 패턴을 대상으로 동작을 확인하였다. 제안된 명령어 세트를 처리하기 위한 컴파일러를 작성하였고 이 프로그램을 사용하여 정규표현식을 처리할 수 있도록 변환하여 검증하였다.

그림 6은 *page=(ab(efghi)*){1,6}vw* 정규표현식에 대한 시뮬레이션을 통해 합성회로의 동작을 확인한 결과이다. 하단의 *match_ok* 신호는 위의 정규표현식에 해당하는 *page=abefghivw* 패턴이 발견되었다는

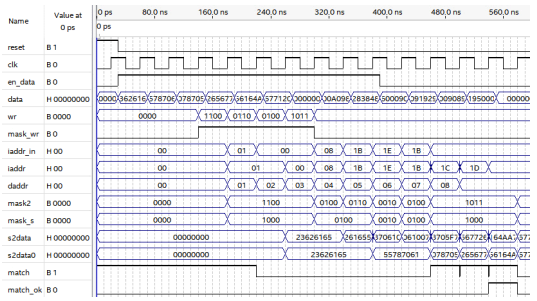

그림 6. 정규표현식 매칭 프로세서 시뮬레이션 타이밍도
Fig. 6. Regular expression matching processor simulation timing diagram

신호로 제약 및 중첩 반복 연산에 대한 동작을 검증하였다.

표 2는 설계된 정규표현식 매칭 프로세서의 Intel Stratix IV(EP4SGX230K40C2) FPGA로의 합성 결과이다. 1,909개의 ALUTs(adaptive look-up tables), 914개의 레지스터(register)가 사용되었고, FPGA 클록의 가능한 최대 속도는 160.28MHz로서 매 클록마다 4바이트(32비트)씩 계속하여 처리하는 경우에 5.13Gbps의 최대 처리 속도를 가질 수 있다. 프로세서의 구조가 상대적으로 복잡해졌기 때문에 제약 및 중첩된 반복 연산 기능이 없는 설계의 최대속도 175.44MHz[8]보다 클록속도가 낮다.

명령어 메모리와 데이터 메모리는 FPGA의 임베디드 메모리 블록을 사용하여 구현하였고, 메모리 크기는 표 2의 합성 보고서에 포함되지 않았으며 필요한 경우 늘릴 수 있다.

표 2. 정규표현식 매칭 프로세서의 FPGA 합성 결과
Table 2. FPGA synthesis results for regular expression matching processor

Device	Logic elements (ALUTs)	Total registers	Freq. max	Speed
REMPr	1,909	914	160.28	~5.13
REMP[8]	312	115	175.44	~5.61

Ⅳ. 결 론

ReCPU, SMPU는 정규표현식을 명령어로 표현하고 이를 처리하는 형태로 정규표현식 패턴 매칭을 수행하는 프로세서로 제안되었지만, 제약 반복 연산을 비효율적으로 수행하는 단점을 가지고 있다. 본 논문에서는 효율적으로 정규표현식의 반복 연산을 지원할 수 있도록 새로운 명령어 세트를 제시하고 이를 구현하는 정규표현식 매칭 프로세서 구조인 REMPr을 제안하였다.

제안된 REMPr 구조는 REMP의 명령어 세트를 기반으로 제안되었으며, 특히 짧은 서브 패턴을 OR의 반복 연산으로 처리하는 비효율적인 방법을 개선하고, 다운 카운터(*down counter*)와 보조 카운터(*sN*)를 사용하여 제약 반복을, 카운터 스택(*counter stack*)을 사용하여 *page=(ab(efghi)*){1,6}vw*와 같은 중첩된 반복 연산을 수행할 수 있도록 설계, 구현하였다.

이러한 기능을 포함하지 않은 기존 정규표현식 프로세서에 비해서 OR 연산은 물론 제약 및 중첩 반복 연산을 단일 명령어로 매우 효율적으로 처리할 수 있다.

향후 Snort[10] 등과 같은 시간 제약이 있는 다중 정규표현식 패턴 매칭을 위해 파이프라인 구조를 사용하여 고성능 처리하는 연구가 필요하다.

References

[1] J. Friedl, *Mastering Regular Expression*, 3rd ed. OReilly Media, 2006.

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

[3] C.-H. Lin, C.-T. Huang, C.-P. Jiang, and S.-C. Chang, “Optimization of regular expression pattern matching circuits on FPGA,” in *Proc conf. Design, automation and test in Europe (DATE’06)*, 2006.

[4] R. Sidhu and V. Prasanna, “Fast regular expression matching using FPGAs,” in *IEEE Symp. Field-Programmable Custom Computing Machines (FCCM’01)*, 2001.

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

[6] I. Bonesana, M. Paolieri, and M. Santambrogio, “An adaptable FPGA-based system for regular expression matching,” in *Proc. Conf. Design, Automation and Test in*

Europe (DATE'08), 2008.

[7] Q. Li, J. Li, J. Wang, B. Zhao, and Y. Qu, "A pipelined processorarchitecture for regular expression string matching," *Microprocessorsand Microsystems*, vol. 36, no. 6, pp. 520-526, Aug. 2012.

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

[9] V. Rana, F. Bruschi, M. Paolieri, D. Sciuto, and M. D. Santambrogio, "On How to Efficiently Implement Regular Expression Matching on FPGA-Based Systems," in *12th IEEE Int. Conf. Embedded and Ubiquitous Computing. IEEE*, 2014.

[10] Snort-Network Intrusion Detection & Prevention System, https://snort.org.

서 병 석 (Byung-suk Seo)

2001년 2월 : 연세대학교 전산학과 학사

2008년 2월 : 연세대학교 전산학과 석사

2021년 8월 : 연세대학교 전산학과 박사

2014년 2월~2020년 2월 : 상지영서대학교 국방정보통신과 교수

2020년 3월~현재 : 상지대학교 정보보안학과 교수

<관심분야> 정보보호시스템, 네트워크보안, 컴퓨터시스템

[ORCID:0000-0002-2771-0112]