JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지(J. Korea Inst. Inf. Commun. Eng.) Vol. 17, No. 10 : 2323~2329, Oct. 2013

실패 전이를 갖는 트리를 이용한 스트리밍 XML 하드웨어 파서

이규희[1] · 한상수[2*]

A Streaming XML Hardware Parser using a Tree with Failure Transition

Kyu-Hee Lee[1] · Sang-soo Han[2*]

[1] Department of Computer and Telecommunications Engineering, Yonsei University, Wonju 220-710, Korea
[2] Department of Energy IT, Gachon University, Seongnam, 461-701, Korea

요 약

웹서비스들은 데이터를 표현하기 위해 XML을 채택하고 있으며, 데이터의 사용을 위해 XML 파서가 필요하다. XML 파서에서 DOM(Document Object Model)은 가장 널리 사용되는 방법이지만, 트리를 생성하는 전처리 작업과 추가 메모리 공간이 요구되기 때문에 제한적 자원을 갖는 시스템들에는 적합하지 않다. 본 논문에서는 시스템의 성능 향상을 위해 전처리가 요구되지 않는 실패 전이를 갖는 스트리밍 XML 트리(StreXTree: Streaming XML Tree) 파서를 제안한다. 이전의 다른 연구들과 비교하여, 제안된 StreXTree 파서는 Search 파서 보다 약 2.39배, RBStreX 파서 보다 약 3.02배 성능이 향상되었다. 게다가, 제안된 StreXTree 파서는 XML의 구조와 문법을 검증하기 위한 Well-Formed 검사를 지원한다.

ABSTRACT

Web-services employ an XML to represent data and an XML parser is needed to use data. The DOM(Document Object Model) is widely used to parse an XML, but it is not suitable for any systems with limited resources because it requires a preprocessing to create the DOM and additional memory space. In this paper, we propose the StreXTree(Streaming XML Tree) with failure transitions and without any preprocessing tasks in order to improve the system performance. Compared to other works, our StreXTree parser achieves 2.39x and 3.02x improvement in system performance in Search and RBStreX, respectively. In addition, our StreXTree parser supports Well-Formed checking to verify the syntax and structure of XML.

키워드 : 실패 전이, XML파서, FPGA, 스트리밍 XML

Key word : Failure Transition, XML Parser, FPGA, Streaming XML

접수일자 : 2013. 08. 31 심사완료일자 : 2013. 09. 30 게재확정일자 : 2013. 10. 11

*** Corresponding Author** Sang-Soo Han(E-mail: sshan@gachon.ac.kr, Tel:+82-31-750-5587)
Department of Energy IT, Gachon University, Seongnam, 461-701, Korea

Open Access http://dx.doi.org/10.6109/jkiice.2013.17.10.2323 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

오늘날 대부분의 웹서비스들은 데이터를 표현하기 위해 구조화된 표준 문서인 XML(eXtensible Markup Language)을 사용하고 있다. 일반적으로 널리 사용되고 있는 SOAP(Simple Object Access Protocol)와 REST (Representational State Transfer)는 XML 기반의 메시지를 전달하는 웹서비스 방식이다[1][2].

XML의 파싱을 위해 제공되는 API들은 이벤트 기반과 트리 기반으로 분류되는데, 대표적으로 SAX(Simple API for XML)와 DOM (Document Object Model)이 여기에 속한다[3][4]. 트리 기반 API는 문서 전체를 메모리에 적재하는 전처리 작업 후 파싱이 수행되어 데이터의 사용과 접근이 용이하지만 제한적 자원을 갖는 시스템들에는 적합하지 않다. 반면에 스트리밍 파서라고도 불리는 이벤트 기반 API는 입력되는 XML 문서를 작은 단위로 저장하여 입력과 동시에 처리하기 때문에 메모리에 적재하는 전처리 과정이 요구되지 않아 제한적 자원을 갖는 시스템들에 널리 사용되고 있다.

파싱이 순차 처리되는 소프트웨어 파서들은 성능의 한계성을 갖는데 이를 해결하기 위해 하드웨어 파서들이 연구되고 있다. 하드웨어 파서들은 높은 성능을 갖는 반면에 하드웨어 구조의 변경이 어렵기 때문에 재구성이 가능한 FPGA(Field Programmable Gate Arrays)의 사용이 일반적이다. 스트리밍 파서는 전체 XML 문서를 저장하지 않기 때문에 이전 데이터의 접근이 어렵고 문서를 순차 처리하여 성능향상에 제약이 있다. 본 논문에서는 스트리밍 XML의 순차 처리 문제점을 해결할 수 있도록 임의 접근이 가능한 트리 기반 스트리밍 XML 하드웨어 파서를 제안한다.

본 논문의 2장에서는 스트리밍 XML 파서들의 관련 연구들을 살펴본다. 3장에서는 제안된 구조의 하드웨어 및 동작을 서술하고, 4장에서는 제안된 파서의 평가와 합성결과를 제시하며, 5장에서 결론을 맺는다.

Ⅱ. 관련연구

XML은 구조화된 문서로서 통상적으로 시작 태그(“<>”)와 종료태그(“</>”)의 엘리먼트 쌍으로 표현되며 의미 데이터의 사용을 위해 문서의 파싱이 요구된다.

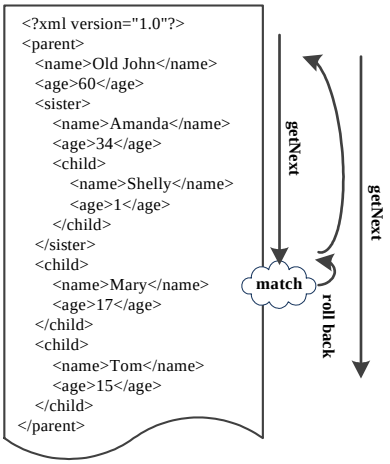

그림 1. RBStreX를 사용하기 위한 시나리오
Fig. 1 A scenario for using RBStreX

DOM 트리는 XML의 모든 정보(엘리먼트 이름, 컨텐트, 부모-형제 노드의 관계)를 파싱하여 메모리에 저장하여 사용한다. DOM 트리는 각 노드에 대한 접근 및 사용이 용이하지만 메모리 요구량은 SAX 파서와 비교하여 비효율적 구조를 갖는다.

[5]의 저자들은 Schema에 대한 유효성 검사와 문서의 구조를 검증하기 위한 Well-Formed 검사를 제외하고 파싱 집중적 하드웨어 파서를 연구하였다. 파싱된 엘리먼트에 대한 재사용이 가능하도록 롤백 기능을 추가한 RBStreX (Roll-Back Streaming XML)를 제시하였고 제공된 명령들은 다음과 같다.

- ***getNext*** : XML을 엘리먼트 단위로 파싱하여 <엘리먼트 종류, 엘리먼트 값>을 반환.
- ***rollBack*** : 검색 성공 시, 파싱된 문서의 최상위 엘리먼트로 이동.

RBStreX 구조의 검증 및 평가를 위해서 그림 1의 시나리오를 사용하였다. 본 논문에서는 이름이 “Mary”인 엘리먼트를 찾는 과정을 **조건검색**이라 하고 “Mary” 부모에 대한 모든 자식들의 나이를 찾는 과정을 **결과검색**이라 표현한다. RBStreX는 조건검색 및 결과검색을 위해, 소프트웨어로 부터 getNext 명령을 제공받아 처리하기 때문에 결과의 판단을 위해 오버헤드가 발생한다.

RBStreX의 오버헤드 단점을 극복하기 위해 [6]의 저자들은 조건검색을 위한 Search 명령을 제안하였다.

Search 구조는 소프트웨어에서 한 번의 명령을 제공하고 파서에서 결과를 판단하도록 구현하였다. RBStreX와 비교하여 조건검색의 오버헤드를 감소되었지만 결과검색에서 getNext를 사용하여 전체시스템의 성능향상은 기대하기 어렵고, Schema 유효성 및 Well-Formed 검사를 지원하지 않는다.

본 논문에서는 [5][6]의 구조에서 발생되는 소프트웨어 오버헤드를 감소시키고 검색의 속도를 향상시킬 수 있는 트리 기반 하드웨어 파서를 제안한다. 또한, XML의 문법과 구조를 검증하는 Well-Formed 검사를 수행할 수 있는 스트리밍 XML 파서를 제안한다.

Ⅲ. 실패 전이를 갖는 XML 트리

본 논문에서는 파싱된 XML 엘리먼트의 재사용 요구에 대한 추가적인 파싱 요구를 해결하기 위해, XML 문서를 트리로 생성하여 접근하는 방식을 채택한다. 본 논문에서 제안하는 구조는 DOM 트리 방식과 유사하지만 메모리 요구량과 검색 속도의 향상을 위해 변형된 트리 형식을 갖는다. 먼저, XML에서 “시작엘리먼트-컨텐트” 엘리먼트 집합을 내부 노드(Internal Node)로서 연결하고 중첩 엘리먼트는 부모-자식의 관계를 갖는 노드로 연결한다. 만일, 자식 노드가 이미 존재하면 자식 노드의 실패 전이 노드로서 연결한다.

예를 들어, 그림 2(a)의 XML DOM 트리가 있을 때, 제안된 구조는 그림 2(b)와 같이 실패 전이를 갖는 트리로 변환된다. 그림 2(b)에서 점선으로 표기된 노드들은 내부 노드들이고 화살표 끝은 실패 전이 노드를 나타낸다. 음영으로 표시된 노드는 컨텐트를 의미한다.

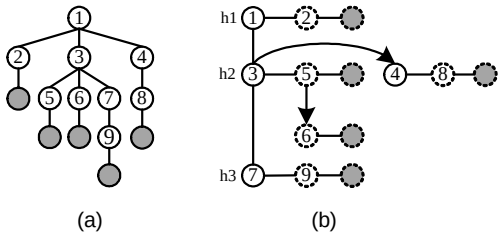

그림 2. DOM과 제안된 StreXTree (a) DOM 트리 (b) 스트리밍 XML 트리
Fig. 2 DOM and proposed StreXTree (a) DOM Tree (b) StreXTree

3.1. 제안된 StreXTree 하드웨어 구조

제안된 스트리밍 XML 트리(StreXTree: Streaming XML Tree) 파서의 하드웨어 구조는 그림 3에 제시하였다. 그림 3의 StreXTree 파서는 Well-Formed 검색기, 상태머신, StreXTree 생성기, 엘리먼트 검색기, 그리고 관련된 메모리들로 구성된다.

Well-Formed 검색기는 계층 구조를 갖는 XML에 대한 문법과 문서 구조를 검증하기 위한 도구로 사용된다. 그림 3의 WF(Well-Formed) 스택은 시작 엘리먼트를 저장하여 종료 엘리먼트가 파싱될 때 마다 스택의 값을 출력하여 비교하고 valid 비트를 출력한다.

StreXTree 생성기는 내부 노드들을 저장하는 IEmem (Internal Element Memory), 중첩 노드들을 저장하는 NEmem(Nested Element Memory), 그리고 제어회로들로 구성된다. 엘리먼트의 종류를 의미하는 3비트(e)는 상태머신으로부터 출력되어 제어회로를 통해 메모리의 엔트리 값들을 갱신할 때에 사용되는데 그림 3에서 *p1*, *p2*는 이전의 두 개의 엘리먼트 종류를 나타낸다. 두 개의 메모리에 대한 엔트리 형식은 그림 3의 좌측 하단에 명시하였다. IEmem 엔트리 형식은 XML 메모리에 저장된 엘리먼트 시작 주소(xdata)와 내부 노드에서의 실패 전이 주소(ifail)로 구성된다. NEmem 엔트리 형식은 XML 메모리에 저장된 엘리먼트 시작 주소(xdata), 현재 노드의 자식 노드에 대한 NEmem의 주소(child), 내부 노드를 가리키는 IEmem의 주소(inode) , 실패 전이 노드를 저장한 주소(nfail)로 구성된다. 메모리 엔트리 형식에서 $p=l+i$, $q=l+2k+i$ 길이를 갖는다. 여기에서 l은 XML메모리의 주소 길이, i는 IEmem의 주소 길이, k는 NEmem의 주소 길이를 나타낸다. 마지막으로, 그림 3에서 elem_searcher는 엘리먼트 검색기로서 그림 4의 구조와 같이 설계되었고, StreXTree를 참조하여 파싱된 XML 엘리먼트들에 대한 재사용에 이용된다.

3.2. StreXTree 생성 알고리즘

그림 3에 명시된 StreXTree는 하나의 엘리먼트가 파싱될 때 마다 상태머신으로부터 출력된 3비트(e)를 참조하여 Algorithm 1과 같은 절차로 생성된다. Algorithm 1에서, 현재 파싱된 엘리먼트의 시작 주소 c는 *buf*와 쉬프트 레지스터로 구성되고, 3비트 엘리먼트 값(e와 $p2$)과 함께 내부 노드를 판별하는데 사용된다.

Algorithm 1은 3비트 엘리먼트 파싱 값(e)에 따라

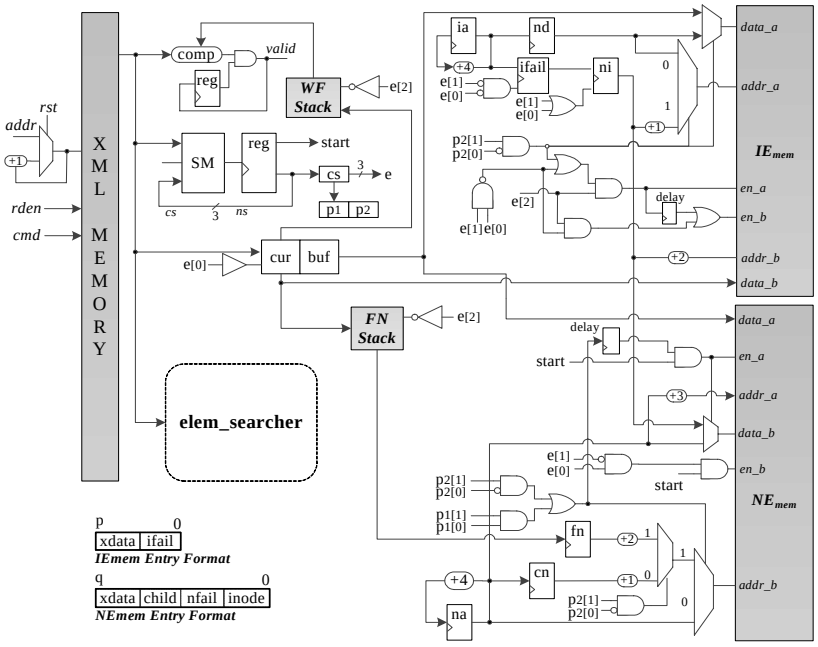

그림 3. StreXTree 파서의 블록도
Fig. 3 Block diagram of StreXTree

IEmem과 NEmem 엔트리를 생성하는데 *e*가 중첩 시작 엘리먼트(“001b”) 일 때, 이전에 파싱된 *p2*에 의해서 현재 처리되는 NEmem의 주소 *na*는 *nfail* 필드 또는 *child* 필드에 저장된다. NEmem 엔트리의 *inode* 필드에는 IEmem에 저장되는 내부 노드의 주소 *ni*를 저장하고 *buf*에 저장된 주소를 *xdata* 필드에 저장한다. 엘리먼트가 컨텐트(“100b”) 일 때, IEmem 엔트리가 생성되는데, 이전의 엘리먼트 *p2*가 종료 엘리먼트이면 IEmem의 *ifail* 필드에 주소 *ia*를 저장하고, IEmem의 *ia*와 *ia*+2에 각각 *buf*와 *c*에 저장된 엘리먼트의 주소를 기록한다. NEmem 메모리에 대한 생성은 3개 필드에 대한 쓰기 동작이 동시에 이루어지는데 FPGA에 내장된 듀얼포트 BlockRAM을 사용하면 다음 엘리먼트가 파싱되는 시간 내에 완료될 수 있다.

3.3. StreXTree 파서의 동작 절차

StreXTree 파서는 소프트웨어로부터 제공되는 명령에 의해 StreXTree 생성과 엘리먼트 검색을 독립적으로 실행하는 구조를 갖는다. 제안된 StreXTree 파서의 동작 절차는 다음과 같다.

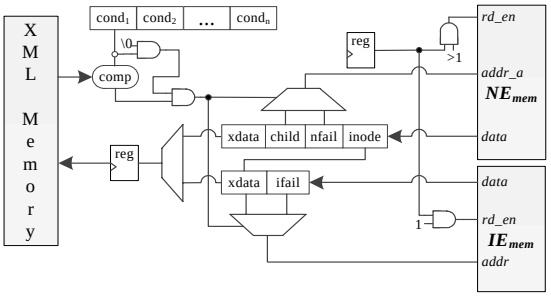

그림 4. 엘리먼트 검색기
Fig. 4 Element Searcher

Step 1: 조건검색을 검색을 위해 search 명령 제공.
Step 2: 검색 성공 후, 그림 1의 XML에서 마지막 엘리리먼트 (</parent>)까지 파싱하여 StreXTree 생성.
Step 3: 결과검색 명령을 제공. 엘리먼트 검색기에서 StreXTree를 이용하여 검색.
Step 4: 결과검색에 사용되는 조건의 개수에 따라 내부 노드 또는 자식 노드로 전이. 남은 검색 조건의 개수가 1보다 크면 NEmem에서 자식 노드로 전이, 1이면 IEmem에서 마지막 조건과 비교 후 검색 결과 인출.

Algorithm 1. Build_streXTree

```
procedure Build_StreXTree(node c, eType e, p1, p2)
  // e: current element, p1 and p2: two previous elements
  when e is 000 do // start
    ifail=ia, buf=c; push wfStack(c), push fnStack(na)
  end when
  when e is 001 do // nested start
    ni=ifail; push wfStack(c), push fnStack(na)
    if p2 is 011 then // nested end
      store na into NEmem[fn+2] as a failure
    else if p2 is 010 or p1 is 011 then // end or nested end
      store na into NEmem[cn+1] as a child
    end if
    store ni to NEmem[naddr+3] as an inode
    cn=na, NEmem[na]=buf, buf=c, na=na+4, nd=na
  end when
  when e is 100 do // content
    if p2 is 010 then store ia to "fail" of IEmem[ni+1] end if
    store buf and c into IEmem, ia=ia+4
  end when
  when e is 010 or 011 do // end or nested end
    pop wfStack, fn= pop fnStack
  end when
end procedure
```

예를 들어, 그림 1에서 검색 조건 “parent-child-age”는 엘리먼트 수가 3이기 때문에 NEmem에서 자식 노드가 선택된다. 이때에 처음 인출된 엘리먼트는 “parent”의 자식 노드 “sister”가 된다. 인출된 노드 “sister”는 다음 검색 조건 “child”와 비교가 실패하고 실패 전이 주소에 저장된 노드 “child”를 다음 클록에 비교한다. 비교가 성공하면 검색 조건에서 “child”를 삭제 하고 검색 조건의 개수가 1(“age”)이 되므로, IEmem에서 “age”에 대한 결과들을 인출한다.

Ⅳ. 평 가

제안된 구조의 성능 평가와 비교를 위해, 그림 1의 XML 문서에 대한 전체 문자 수를 *N*, 파서에서 비교되는 문자 수를 *m*, 그리고 제공된 명령어 수를 *k*라 정의한다. RBStreX 파서는 조건 및 결과검색에서 getNext 명령을 사용하는데, 이는 엘리먼트의 문자 비교 및 결과 판별이 파서와 소프트웨어에서 각각 발생하기 때문에 조건검색에서 2*m*, 결과검색에서 2*N*의 비교를 수행한다. Search 파서는 search 명령을 제공하여 최대 *m*번의 문자를 비교하고 명령어 제공 및 결과 출력에 각 1클록이 소요되며 결과검색은 RBStreX와 동일하다.

표 1에서 제안된 StreXTree 파서는 조건검색에서 [5][6]의 구조 보다 많은 문자 비교를 수행하는데, XML을 제안된 StreXTree로 변환하기 위하여 마지막 엘리먼트까지 파싱하기 때문이다. 그러나, [5][6]에서도 조건검색에서 비교되지 않은 나머지 엘리먼트들은 결과검색에서 비교된다.

표 1. 검색 시간 분석
Table. 1 Search time analysis

파서	조건검색	결과검색	롤백
RBStreX[5]	2*m*+*k*	2*N*+*k*	2
Search[6]	*m*+2	2*N*+*k*	2
StreXTree	*N*+2	$\sum_{i=1}^{d} c_i + (f_d + s_d) + k$	-

표 2. 다른 연구들과의 성능 비교
Table. 2 Comparison with other works

파서	조건검색			결과검색			Well-Formed 검색	P_{total}
	chars	cmd	P_c	chars	cmd	P_r		
RBStreX[5]	154	26	334	235	40	510	×	844
Search[6]	154	1	157	235	40	510	×	667
StreXTree	235	1	237	40	2	42	○	279

표 3. StreXTree 파서의 합성 결과
Table. 3 Synthesis Report of a StreXTree Parser

디바이스	ALUTs	Memory	Fmax
Stratix IV GX 230	76	3,344 bits	358 MHz

제안된 StreXTree 파서의 결과검색 성능을 추정하기 위해 다음과 같은 집합을 정의한다.

$$C=\{c_1,c_2,...,c_h\},\ F=\{f_1,f_2,...,f_h\},\ S=\{s_1,s_2,...,s_h\}$$

*C*는 그림 2(b)에서 부모-자식 관계를 갖는 노드(“1,3,7”)들의 집합을 나타내는데, 같은 트리 깊이에 위치한 노드들의 전체 문자 수를 나타낸다. 마찬가지로, *F*는 실패 전이 노드(“4”), *S*는 내부 노드(“2,5,6,8,9”)들을

의미한다. 여기에서 *h*는 트리의 최대 깊이이며, c_i, f_i, s_i는 트리의 *i* 깊이에 있는 노드들의 전체 문자 수를 나타낸다. 제안된 StreXTree는 최악의 경우 *C*가 최대 깊이를 갖지만, 나머지 *F*와 *S*는 조건의 최종 엘리먼트가 위치한 트리의 깊이에 대해서만 비교되기 때문에 전체 비교 문자 수는 감소한다.

그러므로, XML 문서의 전체 문자 수 *N*과 결과검색에서의 전체 비교 문자 수 *M*은 식(1)과 같이 표현된다. 결과검색 조건이 “parent-child- age” 일 때, 부모-자식 관계를 갖는 노드들의 수를 이용하여 트리의 최대 깊이 *d*가 결정되는데 “age”는 내부 노드로 연결되기 때문에 여기에서 *d*는 2가 된다. 따라서, 식 (1)의 결과를 토대로 *M* < *N*은 만족한다.

$$N=\sum_{i=1}^{h}(c_i+f_i+s_i),\ \ M=\sum_{i=1}^{d}c_i+(f_d+s_d) \qquad (1)$$

표 1의 수식들을 이용하여 하드웨어 파서들의 소비 클록을 표 2에 나타내었다. 표 2에서 chars는 제공된 XML에서 비교되는 전체 문자 수, cmd는 제공되는 명령어 수를 나타낸다. P_{total}은 제안된 파서의 전체 소비 클록을 나타내는데, P_c는 조건검색의 소비클록, P_r은 결과검색의 소비클록을 의미한다. 또한, 제안된 StreXTree 파서는 이전 연구들에서 지원하지 않는 Well-Formed 검사를 지원하도록 설계되었다.

제안된 StreXTree 파서는 Altera의 QuartusII 12.1을 이용하여 StratixIV GX 230 디바이스로 합성하였으며 결과는 표 3과 같다.

Ⅴ. 결 론

스트리밍 XML 파서는 파싱된 XML 문서의 재사용과 검색 성능에 대한 제한점을 갖기 때문에 본 논문에서는 실패 전이를 갖는 변형된 DOM 트리를 사용하여 고성능의 스트리밍 XML 파서를 제안하였다. 표 2에 제시된 소비 클록들을 기반으로 제안된 StreXTree 파서는 Search 구조와 비교하여 약 2.39배, RBStreX 구조와 비교하여 약 3.02배 향상되었다.

또한, 제안된 StreXTree 파서는 기존 연구들에서 제공하지 않는 Well-Formed 검사를 제공한다. DOM 트리와 달리 제안된 구조는 XML의 특정 계층부분에 대한 트리 생성이 가능하기 때문에 제한된 자원을 갖는 시스템들에도 충분히 적용될 수 있는 XML 하드웨어 파서이다.

감사의 글

본 연구는 2013년도 가천대학교(GCU-2013-R281) 지원본부의 지원에 의하여 이루어진 연구로서, 관계부처에 감사드립니다.

REFERENCES

[1] K. Hameseder, S. Fowler, and A. Peterson, “Performance Analysis of Ubiquitous Web Systems for SmartPhones,” Int'l Symposium on PECATS, pp. 84-89, 2011.

[2] F. Belqasmi, J. Singh, S. Y. B. Melhem, and R. H. Glitho, “SOAP-based vs. RESTful Web Services,” Internet Computing, IEEE, Vol. 16, pp. 54-63, 2012.

[3] E. R. Harold, “An Introduction to StAX,” [Internet]. Available: http://www.xml.com/pub/a/2003/01/17/stax.html.

[4] SAX, [Internet]. Available: http://www.saxproject.org.

[5] C. E. Chang, F. M. Asin and A. K. Mustapha, “RBStreX: Hardware XML Parser for Embedded System,” *Internet Technology and Secured Transactions, ICITST'09. International Conference*, pp.1-6, Nov. 2009.

[6] Sae-Woon Kim, Kyu-Hee Lee, and Sang-Kyun Yun, “Implementation of a Hardware-based XML Parser Supporting Search”, *In Proceedings of 33th KIPS Conference*, pp. 41~ 44, 2012.

이규희(Kyu-Hee LEE)

2009년 연세대학교 전산학과 이학석사
현재: 연세대학교 전산학과 박사과정
※관심분야 : FPGA 설계, 임베디드시스템, NIDS

한상수(Sang-Soo Han)

1995년 홍익대학교 전자공학과 공학박사
현재: 가천대학교 교수
※관심분야 : FPGA 설계, 임베디드시스템, 지능 및 퍼지제어