RBStreX: Hardware XML Parser for Embedded System

Chee Er Chang, Faisal Mohd-Yasin and Azhar Kassim Mustapha
Multimedia University, Malaysia
childs_only@yahoo.com, faisal.yasin@mmu.edu.my, azhar@alum.mit.edu

Abstract

The growth of XML usage raises the necessity for embedded XML parser implementation. Embedded systems differed from conventional computing platforms for its limited resources. Among existing XML parsers, SAX and VTD-XML appear to be potential candidates for embedded implementation. The former utilizes minimum resources to perform forward-only XML streaming; while the latter provides random access to XML document at moderate resources. This paper proposes a new idea of Roll-Back Streaming XML (RBStreX) parser for embedded XML parsing. RBStreX is a “pull” type SAX parser that can perform backward referencing through its “roll-back” mechanism. Its resource requirement is only slightly higher than that of SAX and much lower than that of VTD-XML. A simple RBStreX had been designed using VHDL and simulated using Altera Quartus II software.

1. Introduction

The Extensible Markup Language (XML) is one of the lingua franca on Web technologies. It is a simple, standard way to delimit text data. The programmer may create an arbitrary data structure and share it with anyone regardless the programming language and computing platform being used. [1]

The XML parsing is the technique to interpret the data from an XML document and make the data available for the application to process. As the XML related technologies advance, different XML parsers blossom for better processing of XML data. General XML parsers exist in software Application Programming Interface (API) forms, mostly in Java class format, for the ease of interactions with other applications on computing platforms. Existing XML parsers include Simple API for XML (SAX), Document Object Model (DOM), Virtual Token Descriptor for XML (VTD-XML) [2] and lastly, Random Access XML (RAX) [3].

However, as computing technologies go ubiquitous, there raise a need for an XML parser for embedded systems. Differed with conventional computing systems, embedded systems have limited resources.

The attempts to fill in the embedded systems with conventional XML parsers are not the optimum solution. High computation cost of software-based XML parser shall raise the resource usage and lower the performance of embedded systems. The following sections will briefly explain XML and discuss current XML parsing techniques for use of embedded system. This paper then proposes an idea of a new XML parsing technique for embedded systems, Roll-Back Streaming XML (RBStreX) parser. The RBStreX technique is the result of taking experiences of readily available XML parsers and exploiting the capability of direct memory access in embedded system. The main contributions of RBStreX are the process simplicity and the resource minimization as well as the backward referencing capability achieved under these 2 constraints. Note that this paper only provides the method to parse the data contents of XML documents, but ignore some components of XML such as DTD, namespace, etc.

2. XML basics and existing XML parsers overview

This section shall introduce the XML basics and existing XML parsers.

2.1. XML basics

For ease of understanding, this sub-section briefly explains terms used in XML documents. Figure 1 shows a simple example of part of XML data contents. An “element” is XML’s container for content. “Start-tag” and “end-tag” indicate begin and end of an element respectively. The “content” exists between “start-tag” and “end-tag” can be either character data or other elements. There are a few types of simple relationship terms in XML to be used throughout the paper as demonstrated by the following examples:

1. The <example> is “parent element” of <letter> and <number>;
2. the <letter> and <number> are “child elements” of <example>;
3. the <letter> is “sibling element” of <number>, and vice versa;
4. the <example> is “ancestor element” of all other elements.

Copyright © 2009 by the Institute of Electrical and Electronics Engineers, Inc. All rights reserved.
Finally, the XML “depth” refers to the total layers of elements that exist in the XML document. For instance, example in Figure 1 has depth of 3 layers.

2.2. Simple API for XML (SAX)

The most fundamental XML parser is the Simple API for XML (SAX). The SAX parser is an event-based parser. It parses the XML document sequentially over the document, identifies and reports every tag and element as individual event as shown in Figure 2. Besides, it is a streaming parser, which produces output along the parsing process. There is a slightly advance type of SAX parser called “pull” type SAX parser. It only reports the events from the XML document when being requested by the application program. Example of “pull” type SAX parser is the Streaming API for XML (StAX) [4].

The benefits of SAX parser lie with its simplicity, which results in high speed and low resource usage. As a streaming parser, it does not require large buffer space to store the entire parsed data. However, SAX parser only processes XML documents in forward direction. Meanwhile, some XML parsing operations may require “backward referencing”, where it need to access previously parsed information that located at earlier part of the XML document.

2.3. Document Object Model (DOM)

The Document Object Model (DOM) parser is currently the most widely used XML parser. The DOM parser parses the XML document and constructs a corresponding tree structure in its memory as illustrated in Figure 3. The application program can then randomly access any part of the tree, without direction limitation as in SAX parser. The tree structure representation of data enables the application program to easily access the data. The DOM parser is highly object-oriented, thus a very “programming-friendly” parser. The drawbacks of DOM parser are the long parsing time for the parser to construct the tree structure and large memory usage to store the entire tree structure. In the case where the application only accesses to a small part of the XML data, the overhead of tree structure construction becomes notably reluctant.

2.4. Virtual Token Descriptor for XML (VTD-XML)

The Virtual Token Descriptor for XML (VTD-XML) [2] is a XML parsing technique developed by XimpleWare. It is applicable to both software and hardware solutions. Instead of being an API, VTD-XML is a binary format specification. It parses the XML document and creates 64-bit binary format VTD record (token) for each event. Through the list of VTD records, the application program may access to any desired element as if using a DOM parser. Meanwhile, VTD-XML provides higher performance and requires lower resource compared with DOM (VTD-XML only need memory of about 1.3~1.5 times of the original XML document size, compared with DOM’s 5~10 times of that) [2]. However, the VTD-XML has its drawback as its binary VTD record is not object-oriented like DOM.

2.5. Random Access XML (RAX)

The Random Access XML (RAX) [3] is a hardware XML parsing technique implemented by Tarari Inc. (currently acquired by LSI Corporation). It is being employed in the company’s XML parsing accelerator hardware products. The RAX achieves very high throughput using simultaneous XPath processing method. The XPath is a path expression to navigate the XML documents. For simple understanding, it can be considered as same concept as “directories” in our computers. RAX generates XPath for each event in the XML document and run
simultaneous XPath evaluation to retrieve the targeted XML data. The performance of RAX is much higher than all formerly discussed parsers. However, its simultaneous XPath operations shall consume high amount of memory and logic resources.

3. XML parsers in embedded systems

The XML parser implementation in embedded systems differs from conventional computing systems primarily for the limited resources on embedded systems. This section shall examine the existing XML parsing techniques from the point of view of the embedded system.

The SAX and DOM parsers are parsers that are widely used on conventional computing platforms for years. Embedded implementations of SAX and DOM parsers are available in lightweight C and Java codes. The DOM parser is highly object-oriented, thus favored for programming. However, the DOM’s requirement of high resources to construct and store the tree structure representation somehow contradicts to the resources limitation considerations of the embedded system. On the other hand, the SAX parser, as the basic XML parsing technique suits better for embedded systems. The SAX only requires simple operations and little buffer space for XML parsing. Examples of embedded SAX parsers are MinML [5] and NanoXML [6]. Both MinML and NanoXML exist in Java.

The VTD-XML is able to be implemented by either software or hardware implementations. It achieves reasonably fast performance and low resources usage for XML parsing. The VTD-XML parser is not object-oriented. However, it enables random access of XML data, which is a flexible feature to run the application programs. Lastly, the RAX parser is a total-hardware XML parser and provides very high performance. However, the RAX is designed as hardware accelerator for XML parsing instead of lightweight embedded XML parser. It requires high resources and logics for its simultaneous XPath operations.

Table 1. Comparison between XML parsers

<table>
<thead>
<tr>
<th>Parsing style</th>
<th>SAX</th>
<th>DOM</th>
<th>VTD-XML</th>
<th>RAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random accessibility</td>
<td>No.</td>
<td>Yes.</td>
<td>Yes.</td>
<td>Yes.</td>
</tr>
</tbody>
</table>

Table 1 shows the comparison between different XML parsers. Among these parsers, the SAX and VTD-XML appear to be more suitable for embedded XML parser implementation. For basic embedded XML parsing, the SAX is favored for its extreme simplicity and resource-friendly characteristics. Meanwhile, for more advance embedded XML parsing, the VTD-XML is preferred for its reasonably low resource usage, high performance and its capability to provide random access to XML document.

4. Concept of Roll-Back Streaming XML (RBStreX) parser

This paper proposes a new XML parser concept for embedded system, namely Roll-Back Streaming XML (RBStreX) parser. Considering limited resources on embedded system, the RBStreX attempts to optimize the achievable XML parsing features with minimum resources. The implementation requires the condition where direct access to the file location is allowed as in Figure 4.

![Figure 4. RBStreX module](image)

The RBStreX is an evolved version of streaming “pull” type XML parser. Existing streaming parsers such as SAX parse XML document sequentially. They possess high parsing speed, but suffer from inability for backward referencing. As to overcome this limitation of streaming parser, RBStreX introduces “roll-back” method by utilizing fixed-size buffers that hold the memory location of certain elements. The particular element can be any element that is significant towards the currently streamed elements, for example, the parent element. Conventional non-streaming XML parsers such as the DOM and VTD-XML provide random access to XML document, thus backward referencing of data is easily done. The RBStreX on the other hand allows backward referencing by the method to “roll-back” to the previously recorded location of a particular element while XML streaming is in progress. The “roll-back”
The operation is achieved by directly accessing the XML document in memory and setting the address pointer to the location as indicated by the RBStreX's buffer. After "roll-back" to an earlier location, the RBStreX will resume streaming from that location. Repeating "roll-back" allows the parser to return to location of any pre-recorded location of a particular element. Thus, the "roll-back" feature enables the parser to "re-access" any earlier part of XML document.

As mentioned, the selection of element whose location is to be buffered should be a "significant element" with respect to the currently streamed element. By carefully selecting the criteria on defining the "significant element", the resources consumed by RBStreX will be very small, which can be slightly higher than that of a "pull" type SAX parser.

In this RBStreX implementation, such "significant element" is defined to refer as the parent element of the currently streamed element. As the location of parent elements are being recorded along the streaming process that goes through layers of XML elements, the buffer will consist of locations for a set of "direct ancestor" elements with respect to the currently streamed element. The buffer is thereby referred as the "Direct Ancestor" (DA) buffer. Note that during each layer, only 1 element will be recorded, and the record will be overwritten or deleted after this element has been streamed to its end (by detection of the corresponding end-tag). By adding such constraint, the buffer size needed is effectively limited to simply 1 location per layer of the XML document. Figure 5 illustrates an example on how the DA buffer works. When the current streamed element is "Leslie", the array of DA buffer holds the record for locations of all "direct ancestors" of "Leslie", which are "Thomas" (layer 3), "Old John" (layer 2) and "Very old Tom" (layer 1).

5. RBStreX with direct ancestor (DA) buffer implementation

The basic design of RBStreX parser considers only 3 types of events, which are start tag, content and end tag of XML document. Besides, only 2 simple commands are implemented, which are “get_next” and “roll_back”. “Get_next” command returns the next event data and event type; while “roll_back” command brings the parser back to the location of parent element with respect to the currently streamed element. Meanwhile, the DA buffer will always keep track of the location records of direct ancestors with respect to the currently streamed element. As shown in Equation (1), the DA buffer size depends on "max_depth", the maximum depth of target XML document, and "addr_width", the address bus width, which used to support a maximum XML document size.

$$DA\_size\ (-bit) = max\_depth \times addr\_width\ (-bit)$$ (1)

The operation flow of the RBStreX is shown in Figure 6. A simple prototype of RBStreX had been designed in VHDL and tested (Very High Speed Integrated Circuit Hardware Design Language) using Quartus II software from Altera [7]. The design had been verified by the simulation on Quartus II as shown in Figure 7.
During simple data retrieval operations, where backward referencing is unnecessary, it works as a “pull” type SAX parser. Figure 8 shows an example of how an application may utilize RBStreX to perform some less simple data retrieval operations. Consider the scenario where the application program would like to retrieve the information on “the age of every child of Mary’s father”. A SAX parser alone cannot achieve such operation without external buffers at the application program because SAX parser cannot refer to earlier parsed data. On the other hand, RBStreX provides the “roll-back” feature that enables the streaming process goes back to the start of first child element within same parent element. The process of the RBStreX parsing in this example is shown as below:

1. Normal streaming, looking for name of “Mary” (“pull” parser);
2. Found the name of “Mary”;
3. “Roll-back” to the direct ancestor element “parent1”;
4. Continue streaming from element “parent1”, looking for information “age” under “child”; 
5. Found Tommy’s age, “17”;
6. Found Mary’s age, “15”.

At each depth layer, DA buffer only records the location of the parent element with respect to current streaming location. When the streaming proceeds to next element at same depth, the current DA buffer will be loaded with information of new location. Take instance for the previous example in Figure 7, after step 6, the streaming proceeds to another parent element “parent2” that contains “Old Tom” and his children, the DA buffer will replace the location of “parent1” (that contains “Old John” and his children) with the location of “parent2”.

### 6. Comparisons & discussions

Section 3 earlier reviewed that the SAX and VTD-XML as more favorable conventional XML parsers for embedded implementation. Table 2 shows the comparisons between RBStreX, SAX and VTD-XML.

The SAX parser provides high parsing speed and requires very minimum resource requirement, but does not provide backward referencing feature. Meanwhile, the VTD-XML provides random access to XML document by introducing reasonably memory overhead at 1.3–1.5 times of original XML document size. The proposed RBStreX does not provide random access. Instead, it provides “roll-back” mechanism to allow earlier accessed XML data to be “re-accessed”

<table>
<thead>
<tr>
<th>Memory Resource</th>
<th>“pull” SAX</th>
<th>VTD-XML</th>
<th>RBStreX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lowest, only few buffers and logics.</td>
<td>Low, 1.3–1.5 times of original XML document size.</td>
<td>Slightly higher than SAX but lower than VTD-XML, address-bus-width buffer per layer.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Speed</th>
<th>“pull” SAX</th>
<th>VTD-XML</th>
<th>RBStreX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very fast.</td>
<td>Fast.</td>
<td>Variable. (depends on operations complexity)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Backward referencing</th>
<th>“pull” SAX</th>
<th>VTD-XML</th>
<th>RBStreX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not achievable.</td>
<td>Achievable by random access through list of VTD records.</td>
<td>Achievable by “roll-back” to any direct ancestor element of current streamed element.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Random accessibility</th>
<th>“pull” SAX</th>
<th>VTD-XML</th>
<th>RBStreX</th>
</tr>
</thead>
<tbody>
<tr>
<td>No.</td>
<td>Yes.</td>
<td>No.</td>
<td></td>
</tr>
</tbody>
</table>
when backward referencing is necessary. Memory consumption of RBStreX is far less than that of VTD-XML. For instance, to target a XML document with within 1kB size (maximum address bus of 10-bit) and maximum depth of 8-layer, DA buffer with size of merely 80-bit is required. Using VTD-XML, similar size of XML document (regardless maximum depth) shall require 1.3~1.5kB of memory.

The potential drawback of the RBStreX is the variable parsing time, depending on the nature of parsing pattern performed. Each “roll-back” operation restores the parsing location to an earlier point and resumes the parsing from that point. This is a tradeoff between resource usage, operation feature (backward referencing), and performance. The RBStreX is able to deliver XML parsing operations with moderate complexity at reasonably high performance. The worst case scenario for RBStreX performance issue is when the parsing operations require repeated referencing across long distance, for example from deepest layer to first layer. In such cases, the XML document will be streamed over and over again, resulting longer parsing time. However, such worst case scenario is not common and it can be noticed during designing the application operations. As none of the parsers, whether the RBStreX and any of them mentioned earlier, suit the best for all application, an experienced or a careful application programmer shall be able to distinct the needs of the application and thereby determine the suitable parser for the application.

Besides, the RBStreX requires more frequent access to the XML document as compared with conventional parsers. Therefore, it is highly recommended that the RBStreX to be implemented as hardware XML parsing module that can directly access to the memory where the XML document resided (as in Figure 4). Such design can bypass the core processor, thus reduces the workload of the processor on the embedded system. The RBStreX module should be placed physically near to the memory unit as to reduce power usage, especially during worst case scenario, where the RBStreX will repeatedly access the memory unit.

7. Conclusion

The XML parsing is the key technique for development of the Web’s next big thing, the XML technologies. Consequently, embedding XML parsers is nonetheless a necessity to bring the Web towards ubiquitous. However, existing parsers that work well in conventional computing platforms are not fully optimized for embedded systems. Currently the conventional XML parsers that are potential to be embedded are the SAX and VTD-XML.

This paper proposes the idea of the RBStreX as XML parser for embedded system. The RBStreX possesses the simplicity of SAX as well as “roll-back” capability that enables backward referencing of XML data. The RBStreX requires very low resources; only slightly higher resources than that of SAX; which is, at the same time, much lower than that of VTD-XML. The “roll-back” mechanism allows the parser to perform backward referencing that is not achievable by conventional streaming parser. Embedding RBStreX as embedded hardware module that is directly connected to memory unit can reduce the embedded core processor workload and reduce power consumption. Targeting embedded XML parsing with low or moderate complexity, RBStreX shall be more efficient choice over other conventional XML parsers.

8. References


