情報処理学会論文誌 Vol.53 No.4 1401–1409 (Apr. 2012)

ハードウェアアクセラレータを用いた組込み機器向けWebブラウザの動作性能改善

井口 慎也[1,a)] 中越 洋[1] 原田 諭[1] 望月 義則[1] 宗像 尚郎[2] 竹内 弘道[2]

受付日 2011年7月30日, 採録日 2012年1月13日

概要：組込み機器の多様化が進む中，Web アプリケーションが注目されている．Web アプリケーションは，1 度記載すれば改造することなく，Web ブラウザを搭載した様々な端末上で動作する．しかし C++ 等で開発されたアプリケーションより動作が遅くリソースの消費量が多い．そこで，Web ブラウザの実行時の処理時間と消費電力を同時に削減する Web ブラウザのハードウェアアクセラレーション方式を検討した．組込み機器向け Web ブラウザとして普及している Web ブラウザ Webkit の解析結果に基づき，HTML 字句解析処理のハードウェアアクセラレータ化が有効であると仮説を立てた．ここでハードウェアアクセラレータが CPU とメモリを共有する必要があるため，CPU とのメモリアクセス競合を低減させる方式も検討した．そして仮説検証用の試作機を開発し有効性を検証した．結果，Web ブラウザ全体の処理時間を，ソフトウェア処理のみの場合と比較し約 88%に短縮し，さらに字句解析処理については，消費電力を約 1/200 に削減できる見通しを得た．

キーワード：Web ブラウザ，ハードウェアアクセラレータ，Webkit，組込み機器，XML

Improving Performance of Web Browser for Embedded Devices with Hardware Accelerator

Shinya Iguchi[1,a)] Hiroshi Nakagoe[1] Satoru Harada[1] Yoshinori Mochizuki[1]
Hisao Munakata[2] Hiromichi Takeuchi[2]

Received: July 30, 2011, Accepted: January 13, 2012

Abstract: During increasingly diverse of embedded systems, web application is focused to develop one write multi-use application for various embedded systems. But, web application has worse performance and more consuming resources than native application such as developing C++. We examine the web browser accelerator to reduce processing time and power consumption on embedded systems during executing web browser. As a result of web browser architecture analysis, we assume HTML tokenizer hardware accelerator is efficiency to develop it. Hardware accelerator must share the memory with CPU. So we consider reducing the memory access collision between CPU and accelerator. We develop the prototype to verify these inventions efficiency. As a result, we have got the prospect which the overall web browser processing time reduced about 88% against full software approach, and the power consumption of HTML tokenized process reduced about 1/200.

Keywords: Web browser, hardware accelerator, embedded, Webkit, XML

1. はじめに

ネットワーク連携型の携帯端末の普及にともない，インターネット上に存在する様々なリソースを活用したサービスを携帯端末上で提供することが可能となった．

[1] 株式会社日立製作所
Hitachi Ltd., Yokohama, Kanagawa 244–0817, Japan
[2] 株式会社ルネサスソリューションズ
Renesas Solutions Corp., Chiyoda, Tokyo 100–0004, Japan
[a)] shinya.iguchi.uj@hitachi.com

他方，携帯端末も，携帯電話，タブレット端末，NetBook等用途別に多様化しており，複数の端末を状況別に使い分ける携帯端末利用者が登場した．そして，携帯端末とPCを併用してインターネットに接続する利用者は，国内で6,000万人を超えたとの統計がある[1]．

サービス提供側は提供するサービスの利用機会を増やすため，サービスの利用を想定する複数種別の端末（マルチデバイス）それぞれに最適化したGUI（Graphical User Interface）を持つアプリケーションを提供している．

結果，端末別にアプリケーションを開発する手間が増大したため，マルチデバイス対応で共通的にアプリケーションを開発し，改造することなく複数の端末上で実行できるアプリケーション実行環境実現へのニーズが高まっている．この解決策として携帯端末開発企業とアプリケーション開発企業は，ハードウェア/ソフトウェア環境の標準化に着目している．特にWeb技術は，

- PC（Personal Computer）から携帯端末に至るまでWebブラウザの搭載率が高い，
- HTML（Hyper Text Markup Language），JavaScriptといった標準化された技術で開発が可能，
- 開発したコンテンツを，Webブラウザを搭載した異なる装置間で移植および流用が容易，
- 20年近く利用されている技術のため，デザイナと開発者が多数存在する，

といった特長を持つため，Web技術で開発したアプリケーション（Webアプリケーション）の携帯端末への搭載について関心が高まっている．そして，Webアプリケーション実行用に特化した軽量なOS環境である，WebOS[2]，ChromeOS[3]等が登場している．

しかし，Webアプリケーションは，アプリケーションのソースコードをWebブラウザ上で逐次解析して実行するため，事前にコンパイルされて配布されるC++等で開発されたアプリケーションより動作負荷が高い．さらに，携帯端末は電池で長時間駆動させる必要があり，消費電力を低減する目的から，CPU及システムバスの動作周波数に制限がある．しかし，携帯端末はさらなる高機能化と高性能化が求められるため，消費電力効率の大幅な改善を目的として，描画処理と動画処理の高速化と低消費電力化の両立を目的としたハードウェアアクセラレータの搭載が進んでいる．

ここで携帯端末へWebブラウザを搭載した場合の動作効率改善について考える．

ブラウザの処理は，

- HTML解析と構造木（DOMツリー）生成
- 画面の描画
- JavaScriptの実行

に大別できる．ここで，PCにおいてマルチコアの活用によるJavaScript処理の高速化，GPGPU[4]を活用した描画高速化処理等が実現しつつある状況をふまえると，画面の描画とJavaScriptの実行に関しては，携帯端末においても同等なアプローチが普及すると推察する．しかし，HTML解析と画面表示処理手順の構築については，CPUの処理性能の向上以外の手段が見当たらない．

そこで本稿では，WebブラウザのHTML解析とDOM（Document Object Model）ツリーと呼ばれる構造木の生成処理の一部をハードウェアアクセラレータで実現することにより，CPUの性能向上では達成が困難な消費電力効率を実現できる組込みシステムのアーキテクチャを考案し，試作した結果を述べる．

2. Webブラウザと処理改善方法

2.1 Webブラウザのアーキテクチャ

はじめに，既存Webブラウザのアーキテクチャを解析し性能評価を実施した．

現在，多様なWebブラウザが存在するが，PC，組込み端末の両方で採用されている主流5ブラウザ[5]の基本機能を実現するブラウザエンジンは，Trident[6]，Gecko[7]，Opera[8]，Webkit[9]の4種類が存在する．

本稿では，PCから組込み機器向けに普及しているブラウザエンジンWebkitを解析対象に選んだ．Webkitは，PC用Webブラウザ Safari，Google Chromeから，携帯電話プラットフォームであるAndroid，iOSに至るまで幅広く利用されており，オープンソースであるため内部解析が可能である．なおWebkitはKDEプロジェクト[10]で開発が進められていたHTMLレンダリングエンジンKHTMLとJavaScript処理系KJSをベースに，Apple社がSafariブラウザ開発のためにモジュール化し2005年にオープンソースとして公開したモジュールである．

ソースコードライセンスは修正BSD[11]/LGPL[12]混在である．小規模で高速に動作するため組込みが容易であるといわれており，以下の2大モジュールとこれらのモジュールが利用する各種ライブラリから構成される．

- WebCore：HTMLを解析し，その構造に基づいた描画等の処理を実行する．
- JavaScriptCore：JavaScriptを実行する．WebCoreから呼び出される．

Webkitの基本構成を示す（**図1**）．またWebkitがHTMLファイルを読み込んでから処理完了までの処理の流れを示す（**図2**）．

2.2 Webkitの処理時間比率

Webkitが標準的なWebページを表示した場合の各機能の処理時間比率の計測を実施した．ベンチマークには，文字，画像，表，スクリプト等が含まれており，ベンチマークサイトとして広く用いられているStandard Page Load Test[13]の表示データ一式をダウンロードして利用した．

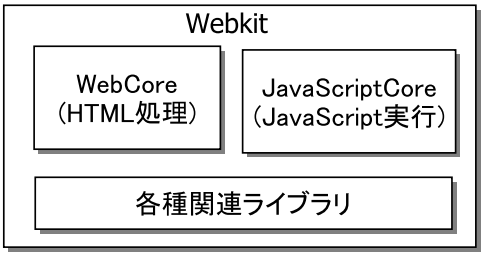


図 1 Webkit の基本構成
Fig. 1 Basic archtecture of Webkit.

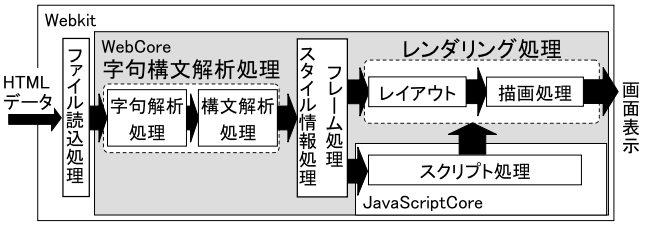


図 2 HTML 解析処理の流れ
Fig. 2 HTML processing flow.

表 1 処理時間比率計測に用いた PC の仕様
Table 1 Specification of PC for evaluation.

項目	仕様
プロセッサ (動作周波数)	Celeon(1.8GHz)
搭載メモリ	512MByte
画面サイズ(ブラウザ表示領域)	800x600(800x541)
OS(ディストリビューション名)	Linux(Fedora Core 7)

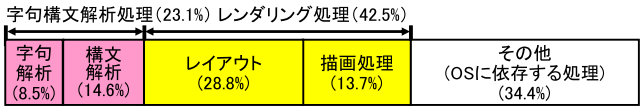


図 3 Webkit のモジュール別処理時間比率
Fig. 3 Processing time rate of Webkit modules.

ダウンロードしたファイルサイズは 146 KByte である．ベンチマーク用 PC (**表 1**) に Linux 環境を構築して Webkit をインストールし，oProfile [14] を有効化した状態で，Webkit に Standard Page Load Test を表示させ Webkit 内の各モジュールの処理時間比率を計測した（**図 3**）．

結果，Webkit では，レンダリング処理の実行時間が最も長く全体の約 42.5%，次いでスクリプト処理が約 33.5%，字句構文解析処理が約 23.1%であり，これら 3 処理が実行時間の大半を占めていることが分かった．

2.3 Web ブラウザの実装方式の比較

Web ブラウザの性能解析を実施し，字句解析部分をアクセラレータ化した場合のブラウザ全体への効果を見積もった．さらに，CPU とアクセラレータのメモリアクセス競合を低減させるため，アクセラレータからのメモリアクセスを低減させる方法を考案し，試作により効果を検証した．結果について述べる．

Web ブラウザを組込み機器上で実装する方法は，大別すると単一 CPU を用いたソフトウェア実装，マルチコア実装，そしてハードウェアアクセラレータ実装の 3 種類が考えられる．これらの実装方法に対する定性的な仮説を立て比較表にした（**表 2**）．組込み Web ブラウザの性能改善で要求される処理速度，消費電力，そして機能の変更容易性の 3 観点で比較している．機能の変更容易性は，Web ブラウザがネット連携の主要技術であり，つねにセキュリティホールの対策等のソフトウェアアップデートが必須となっている現状に鑑み追加した．そして，ソフトウェア実装を基準として，マルチコア実装とハードウェアアクセラレータ実装方式の優劣を定性的に，○は同等，◎は優れている，△と記号で表現した．

表 2 実装方法の特長比較
Table 2 Comparison of implementation.

比較項目	実装方法		
	ソフトウェアのみで実装		ハードウェア アクセラレータを利用
	シングルコア (既存実装)	マルチコア	
処理速度が速い	-	◎	◎
消費電力が少ない	-	△	◎
機能の変更が容易	-	△	△

この評価結果について述べる．マルチコア実装は，複数の CPU コアを利用するため，並列処理が可能であれば処理性能は高い．しかし CPU コア数分，消費電力が増加する．また，機能の変更を実施するには，ソフトウェア変更時，複数の CPU コアの連携を考慮する必要があり，単一 CPU による処理実行と比較して変更工数がかかると推察した．次にハードウェアアクセラレータ実装の場合，処理の性質にもよるが専用のハードウェアを開発することにより処理時間，消費電力ともに最も削減できる，ただし 1 度開発したハードウェアの改変は困難である場合が多いと推察した．しかし，処理時間と消費電力の両者を削減できる方法はハードウェアアクセラレータ方式以外に想定できなかった．そこで本稿では，この方式において，Web ブラウザで必要となる最低限の機能の変更容易性を満たせる実装手法を確立可能であるか検討した．

2.4 性能改善方法と関連研究

前述のとおり，レンダリング処理とスクリプト処理に関しては，ハードウェアアクセラレータの適用もふまえた性能向上への取り組みが活発化しているが，字句構文解析処理に関しては見当たらない．また，Webkit も含めオープンソースブラウザは進化が速い．一方，ハードウェアアクセラレータは 1 度端末に搭載するとアーキテクチャの変更が困難である．さらに，2012 年度に HTML5 勧告が World Wide Web Consortium から発表されるため，HTML4 からの仕様変更にともなう修正が数年にわたって大幅に入ると予測される．これらを考慮すると，Webkit の字句構文解析処理において，実装が単体で閉じやすく，他のライブ

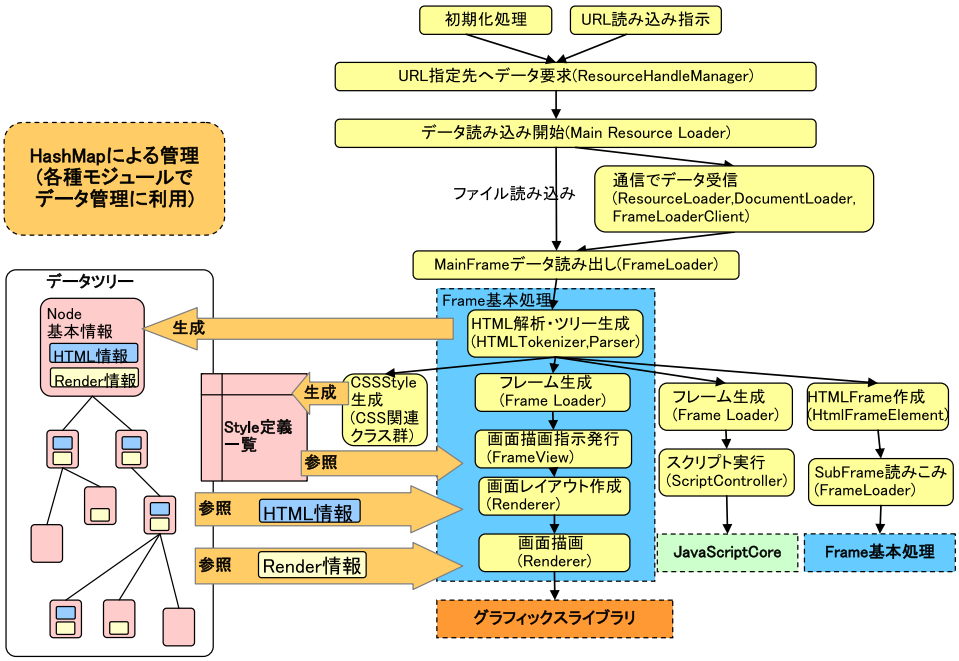


図 4 Webkit の詳細アーキテクチャ
Fig. 4 Detail architecture of Webkit.

ラリを極力参照せず，アーキテクチャが安定している個所がハードウェア化に適しているといえる．そこで Webkit を詳細に解析し該当箇所の有無を調査した（図 4）．

この解析結果において，注目すべきは，XML の DOM ツリーに相当するレンダリング用データツリーの各ノードに，HTMLTag と呼び出すべき Render オブジェクト情報が含まれている点である．これらの情報は，Webkit 内のレンダリング部およびスクリプト処理系と密接に連携している．さらに，構文解析部がオブジェクト情報を Webkit 内の Hash Map テーブルから取得して生成する．したがって，構文解析部は Webkit 全般の構造と密接に連携しているため，今後も頻繁に変化すると判断し，字句解析系のみハードウェア化することが妥当であると仮説を立てた．

次に，HTML は XML と構文が同等であるため，XML 処理用ハードウェアアクセラレーション技術が適用可能であると仮定し，関連研究を調査した．高速処理手法としては，既存ハードウェアアクセラレータ実現手法を組み合わせて，Well-formed check までを，1 byte/cycle の処理性能で実行する高速 XML アクセラレータのアーキテクチャが提案されている [15]．この方式では，XML テキスト文字列を XML アクセラレータに入力すると，字句解析後，Well-formed check と DOM ツリー生成を同時に実行することで処理速度を向上させる．

また，多段 XML の処理に，多重ステートマシンを適用することで，XML データを 2 Byte/cycle の速度で処理するハードウェア XML アクセラレータ Skeleton-CAM-Based XML Parsing（SCBXP）も提唱されている [16]．そして，XML スキーマ検証の過程できる情報を構文解析にも流用し，解析効率を向上させる方式も提案されている [17]．この文献では，XSD [18] を解析し，それに対応した XML 処理を高速に実行する Parser 処理系を生成する方法とその評価 [19] について述べられている．

そして，事前に複数タイプの XML 解析用有限オートマトン（DFA）を生成し，解析対象の XML が入力された時点で，一定の塊にそれらを分割し，投機的に複数の DFA を適用して解析をすすめ，正確に解析できた結果を抽出する方式 [20] も提唱されている．

他方，メモリ使用量の削減に関しては，BART-FSM を用いた XML アクセラレータを構築し，FPGA 上での評価結果を提示している事例がある [21]．BART-FSM は，優先度つきルールに基づき，状態を決定する状態遷移方式で，つねに 1 サイクルでの遷移処理が完了，ルールベースのためプログラムベースよりメモリ使用量が少ない，カスタマイズがしやすいという特徴を持つ．そして SAX [22] を用いることで，DOM ツリーの生成を不要としメモリ使用量を削減することができるが，一般的にはデータの先頭からの逐次アクセスのみ可能である．それに対して，RBStreX parser と呼ばれる XML データアクセス方法が提唱されており，この方式は SAX よりメモリ使用量がわずかに多いだけで，Roll-Back アクセスを可能とする [23]．この方法では，SAX に “Direct Ancestor”（DA）buffer と呼ばれるスタック型バッファを設け，直系の親エレメントをスタックし参照可能することで，Roll-back を実現する．しかし上記調査の結果，

- 字句解析部のみ切り出しハードウェア化した構成での検討はない，

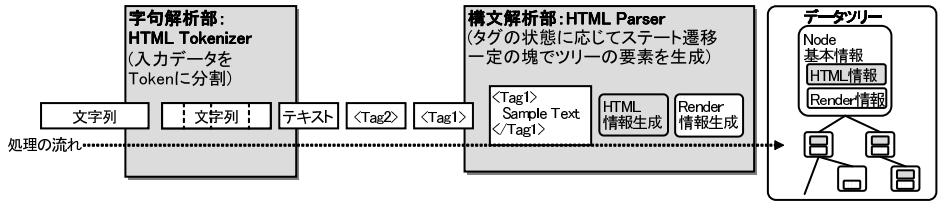


図 5 字句構文解析処理の流れ
Fig. 5 Processing flow of HTML tokenizing and parsing.

- XML データ保存メモリをアクセラレータが占有することが前提で，CPU とのアクセス競合については検討されていない，

ということが判明した．

3. 字句解析処理のハードウェア化と評価システムの構成

字句解析処理のハードウェアアクセラレータ化方法と，それを搭載した Web ブラウザ搭載端末の評価を実施するため，Webkit の字句解析処理と評価用システムアーキテクチャを検討した．

3.1 字句構文解析部の構造

Webkit が入力されたテキストデータから Token を抽出，データツリーを生成する処理に着目して，ソースコードを解析し，字句構文解析部の構造を解析した（**図 5**）．

字句構文解析部は，字句解析部と構文解析部とから構成され，字句解析部でテキストを入力順に 1 文字ずつチェックし，“<”，“>” を区切り記号として Token に分割し，Token が 1 つ完成するごとに，構文解析部へ Token が送信される．構文解析部は，Token を保持し，同名のタグが閉じたタイミングで，Tag の種類ごとに対応するオブジェクト情報を HashMap テーブルから取得してリンクしデータツリーを生成する．さらに，字句構文解析部は，HTML4 特有の記述方法までステートマシンで検出し，Well-formed ではない HTML ファイルでも処理できる機能が搭載されていることが判明した．たとえば，
 といった HTML では正しいが XML の文法上は well-formed ではないタグが存在しても，1 階層上位のタグの終了タグを検出すると，このタグを強制的に解釈しでデータツリーを生成するという例外処理機能を実装している．ここで，字句解析処理は，入力データをバイト単位で検査し，対応するステートに遷移する．しかし HTML の場合は，遷移すべき状態が非常に多く，ソフトウェアで実装した場合，1 回の状態遷移ごとに少なくとも入力データが，アルファベット，数字，特殊記号，どれに相当するか識別するため，数十以上の判定処理を実施する必要があると判明した．

一方，ハードウェアで同等な処理を実装した場合，多量の比較回路を用いることで入力データを並列で比較し同時

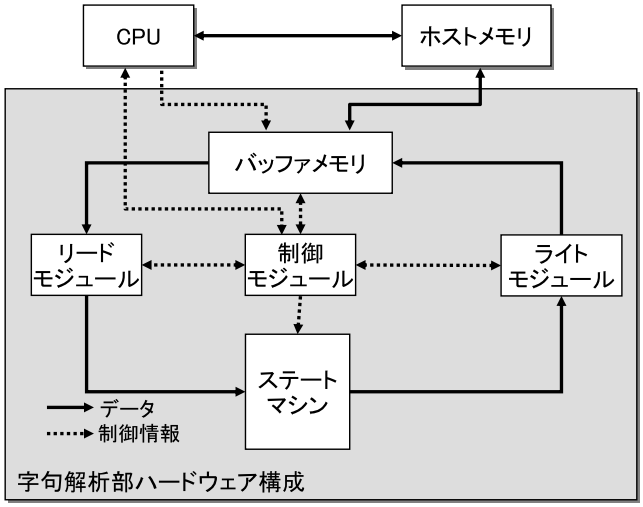


図 6 字句解析ハードウェアアクセラレータの構成
Fig. 6 Architecture of hardware tokenizer.

に処理することが可能となるため，遷移状態の数によらずに，数クロックで入力データの判定および状態遷移処理が可能となり，高速が図れると判断した．そこで本稿では，この状態遷移に対応したステートマシンを搭載した字句解析ハードウェアアクセラレータ（以下，字句解析アクセラレータと略す）を設計し性能検証する方針に定めた．しかし，Webkit の字句解析処理のソースコードを詳細に解析した結果，前述の例外処理を実現するため字句解析部と構文文解析部が密接に関係を保っている箇所があり，単に入力テキストをトークンに分割するだけでなく，例外処理を含む構文解析部の処理の一部も取り込んだハードウェアアクセラレータを設計する必要があると判明した．

字句解析アクセラレータの構成を説明する（**図 6**）．字句解析アクセラレータは，制御モジュール，リードモジュール，ライトモジュール，ステートマシンから構成される．字句解析アクセラレータでは，ステートマシンがコアとなるモジュールである．ステートマシンは，トークンの検出や要素名や属性名といったトークンの種類の判別，“title” や “style” 等といった特定の要素名の検出，ならびに，“"” や “&” といった文字実態参照の検出を行う．字句解析処理を実行するにあたり，CPU は解析対象の文章データ，ならびに文章データの格納先アドレスといった字句解析処理に必要となる情報を RAM に格納し，字句解析ハードウェアに対して解析処理の実行を要求す

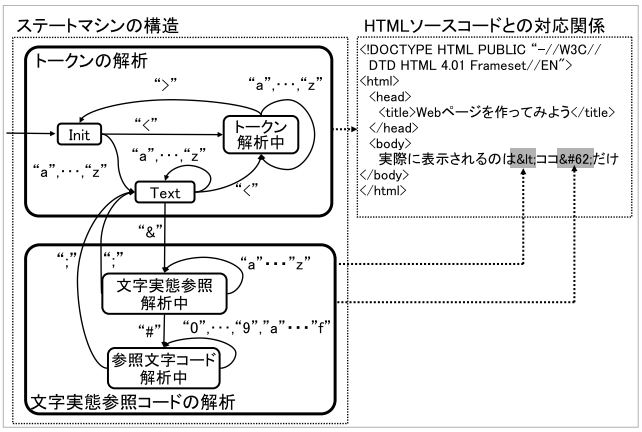


図 7 字句解析アクセラレータのステートマシンの構造
Fig. 7 State diagram of hardware tokenizer.

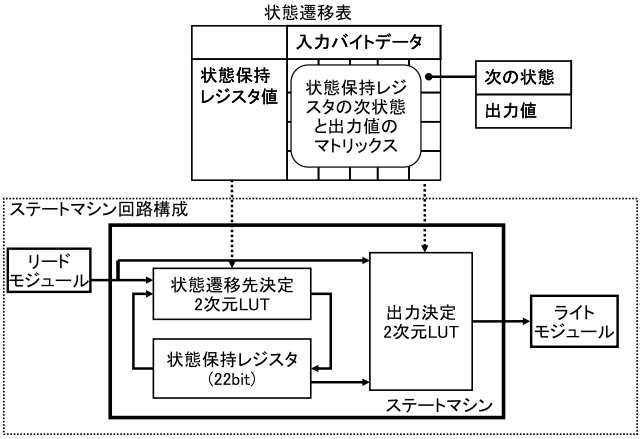


図 8 ステートマシンのハードウェア構成
Fig. 8 Architecture of state machine.

る．字句解析ハードウェアは実行要求を受信すると，リードモジュールを利用して RAM に格納された文章データを読み出し，1 Byte ずつに分割しながらステートマシンに入力する．ステートマシンが入力データを解析し内部状態を遷移させた結果，トークンが生成される場合は，ライトモジュールにトークンの書き出しを要求する．要求を受信したライトモジュールはレジスタの値の中から必要な情報を RAM に書き込む．すべての文章データに対する解析処理が完了すると，CPU に完了通知を送信する．続いて，今回実装したステートマシンの詳細について述べる．ステートマシンは大別して下記，2 種類の状態遷移から構成される（図 7）．

- HTML トークンの解析処理：HTML に含まれる，タグ，本文等を解析する．2054 の状態を持つ．
- 文字実態参照コードの解析：トークン解析中に出現した，“¥"”，¥¥“¥&¥#123;” 等の実態参照コードを文字に変換する．176 の状態を持つ．

ここで述べた状態遷移を実装したステートマシンのハードウェア構成について説明する（図 8）．22 bit の記憶領域を持つ状態保持レジスタが，現在の状態を保持する．リードモジュールから入力データがバイト単位で入力されると，状態保持レジスタの現在値と入力データの値に基づき，状態遷移先決定 2 次元 LUT（Look Up Table）が参照され次の状態遷移レジスタの値が決定する．一方，入力データと，現在の状態保持レジスタの値に基づき出力決定 2 次元 LUT を参照して出力値を決定し，ライトモジュールへデータを送信する．

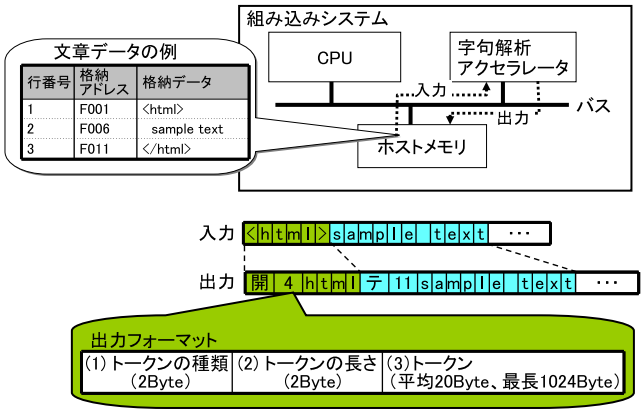


図 9 字句解析系が出力する Token 長
Fig. 9 Token length of tokenizer output.

HTML トークンの解析処理と文字実態参照コードの解析の状態遷移から，入力バイトデータを横軸，現在の状態（状態保持レジスタの値）を縦軸とした状態遷移表を作成し，これを 2 次元の LUT に変換してハードウェアに実装した．これら 2 次元 LUT は，入力データと状態保持レジスタの値を入力とし，状態保持レジスタの次の状態，もしくはライトモジュールへ出力する値を生成する組合せ回路で構成されており，HTML トークンの解析処理と文字実態参照コードの状態遷移先決定用の比較処理を各々 1 クロックで完了する．したがって，約 2,000 存在するステートを探索して次の遷移先ステートを決定する処理を 1〜2 クロックで実現することが可能となり，ソフトウェアで同規模のステート遷移を実現した場合と比較して数百〜数千倍の処理時間短縮が可能となる．

3.2 メモリアクセス低減方法

従来の提案方式では，XML データ保存メモリをハードウェアアクセラレータが占有することが前提で，CPU とのアクセス競合については検討されていない．しかし，組込み機器では，一般的に実装面積と消費電力を極力低減させるため，CPU とハードウェアアクセラレータがメモリを共有する．ここで，Webkit の字句解析部と構文解析部のソフトウェア実装に忠実なハードウェアを構築することを仮定すると，字句解析部が生成した解析結果をすべてメモリへ転送することになる．HTML では，長文テキストが多数含まれ，たとえば Standard Page load Test においては，トークン長が，平均 20 Byte，最長 1,024 Byte となる

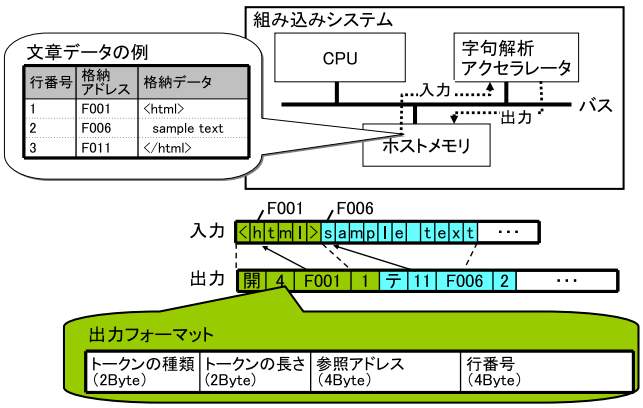


図 10 Token の解析元データ参照先出力方式

Fig. 10 The method of original data referenced address output.

(図 9).

そこで，アクセラレータの字句解析部と，CPU の構文解析部によるメモリアクセス競合を低減させる方法を検討した．字句解析系の本来の役割は，トークン種別の検出であるため，トークン内に含まれるテキスト情報自体は構文解析部側で不要の場合もある．そこで，字句解析部からの出力として，トークン自体でなく，その入力元データの参照先アドレスを返す方式を採用した．結果，1 トークンあたりの転送データ量をつねに 12 バイトに短縮できた（図 10）．

4. 性能評価

4.1 評価システムの構成と計測結果

実装結果の評価システムを説明する（図 11，表 3）．本構成では，SH-MobileR2 [24] 搭載組込み機器であるアルゴシステム社の Algo Smart Panel AP-3300-D1 [25] のローカル I/F へ FPGA（Field Programmable Gate Array）である XILINX 社 Virtex-5 XC5VLX220 [26] 搭載ボードを接続し検証環境を構築した．字句解析アクセラレータは Verilog で記述し，論理合成ツール Xilinx 社 ISE（ver.11.1）を用いて論理合成した結果を FPGA へ搭載した．ここで今回準備した評価環境では，FPGA から機器内のホストメモリへ直接アクセスできないため，FPGA が搭載するバッファメモリを字句解析アクセラレータとのデータ授受に利用した．

この環境で Standard Page Load Test を実行し処理時間を計測した．計測は，Webkit を起動した状態で，Webkit がページ読み込みを開始してから表示完了までの時間を，以下の 2 種類の方法に対して実施した．

- Webkit のソフトウェア処理
- 字句解析処理をアクセラレータで実行した場合

字句解析部の処理時間計測結果を（表 4）に示す．ここで，ソフトウェア処理については，Webkit へ字句解析部のデータ入出処理で計測を実行するソフトウェア Timer 機能を組み込み測定した．また，字句解析アクセラレータに関

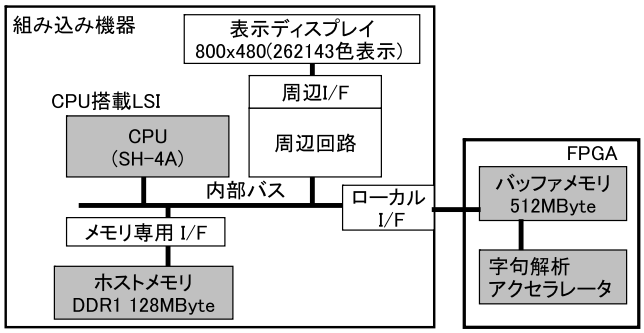


図 11 評価システム

Fig. 11 Evaluation system.

表 3 評価システム構成要素の動作周波数

Table 3 Frequency specs of each I/F on evaluation system.

項目	動作周波数 (MHz)
CPU	400
内部バス/メモリ専用 I/F	133.4
ローカル I/F	66.7
FPGA	133.4

表 4 字句解析処理時間と比較

Table 4 Comparison of tokenize processing time.

項目		値(実測値)
Webkit ソフトウェア字句解析時間(msec) …(a)		1846.6
字句解析アクセラレータ処理時間(msec)	アクセラレータ本体処理 …(b)	1.2
	ホストメモリ-バッファメモリ間データ転送 …(c)	16.1
	合計(b)+(c) … (d)	17.3
実測値処理時間比(d)/(a)		0.94%

しては，FPGA 上にハードウェアカウンタを実装し，データ処理中のクロックをカウントし算出した．

結果，本アクセラレータを用いることで，Webkit でソフトウェア実装されている字句解析処理部と比較し，字句解析処理実行時間を 0.94%に短縮できるとの結果を得た．

4.2 Web ページ表示処理全体の処理時間の短縮

次に，Web ページ表示全体処理時間を比較する．比較に際し Webkit のページ表示処理時間を計測するため，HTML の読み込みを開始してから Web ページ表示完了までの時間を，Standard Page Load Test に組み込まれている JavaScript で記述されたタイマの表示内容を参照した．ただし，計測では，Webkit を 15 秒程度動作させる．結果，計測を開始するタイミングによっては，OS のバックグラウンド処理等の影響を受け，数百 ms の測定誤差が生じた．そこで，この誤差の影響を極力抑えるため，3 回，同一の測定を実施し，その平均値で Webkit のページ表示処理時間を算出した．

そして，算出誤差が少ない Webkit ソフトウェア字句解析時間 (a) と字句解析アクセラレータ処理時間 (d) を用いて，字句解析アクセラレータ利用時の Webkit ページ表示時間の推測値を算出した．この結果を（表 5）に示す．結果，Web ページの表示処理時間を 88%に短縮できる見込み

表 5 ページ表示処理全体の処理時間と比較
Table 5 Comparison of total processing time.

項目		値
Webkit ページ表示処理時間(msec)	ソフトウェア処理のみの実測値…(e)	15262
	字句解析アクセラレータ適用時の推測値 (e)-(a)+(d) …(f)}	13433
処理時間比 (f)/(e)		88.0%

表 6 消費電力計算用の既定パラメータ群
Table 6 Knowing parameters to estimate power.

項目		値
CPU コア：SH-4A 理論値(カタログ値)	消費電流試算パラメータ (mA（typ.）/MHz) …(g)	0.6
	コア動作周波数(MHz) …(h)	400
	バス動作周波数(MHz) …(i)	133.4
	動作電圧(V) …(j)	1.2
実装想定ゲートアレイ：CMOS-12M 理論値	消費電力試算パラメータ (nW/MHz/gate) …(k)	21.6
	動作周波数(MHz) …(l)	133.4
	動作電圧(V) …(m)	1.5

表 7 試作回路のゲート数試算
Table 7 Estimation of gate number.

項目	値
FPGA：Virtex-5 XC5VLX220 論理合成結果 LUT 数 …(n)	5971
LUT 1 個当たりのゲート数論理値 …(o)[27]	12
ゲート数試算結果 …(n)×(o) …(t)	71652

表 8 消費電力と消費電力量の試算結果
Table 8 Estimation of power and power consumption.

項目			値
消費電力 (mW)	CPU コア消費電力 (g)×(h)×(j) …(u)		288.0
	字句解析アクセラレータ消費電力 (k)×(l)×(t) …(v)		206.5
消費電力量 (mWsec)	字句解析処理単体	ソフトウェア処理 (u)×(a)…(w)	531.8
		字句解析アクセラレータ (v)×((b)+(c)/2)[i]…(y)	1.91
		消費電力量比 (y)/(w)	0.4%
	ページ表示処理全体	ソフトウェア処理 (e)×(u) …(k)	4395.5
		字句解析アクセラレータ利用 (u)×((e)-(a))+(y) …(l)	3865.5
		消費電力量比 (l)/(k)	87.9%

i 字句解析アクセラレータのローカル I/F 周波数が 2 倍(133.4MHz)と仮定したときの処理時間推測値

を得た．

4.3 消費電力の考察

本アクセラレータの回路規模と試作機の評価データに基づき，組込み機器用システム LSI にモジュールとして組み込んだ場合の消費電力量を試算した．ここで，比較の前提をできるだけ単純にするため，ルネサスエレクトロニクス社のゲートアレイ上に実装することを想定した [28]．試算に用いる既定のパラメータを**表 6** に，FPGA 上に合成した回路規模から試算に用いるゲート数を計算した結果を**表 7** に示す．以上のデータに基づき，消費電力および処理実行時の消費電力量を計算した（**表 8**）．結果，字句解析処理実行時の消費電力量を 0.4%，すなわち約 1/200 に，そしてページ表示処理全体の消費電力を 87.9%に削減できる見通しを得た．

5. まとめ

本稿では，組込み機器向けに普及が進むオープンソースのブラウザエンジン Webkit の性能解析を実施し，組込み機器上へ Web ブラウザを実装する際に，実行速度と消費電力の両方を同時に改善する方式を検討した．検討の結果，字句解析処理部をハードウェアアクセラレータ化することが有効であるとの仮説を立てた．そして，ハードウェアアクセラレータと CPU が同一ホストメモリを利用することによるアクセス競合を低減させるため，ハードウェアアクセラレータからは，解析結果であるトークン種別とそのトークン生成に利用した入力元データへの参照先アドレスを返す方式を考案した．結果，データ長によらず，つねに 12 バイトの情報を出力する方式を実現した．本提案方式は，既存関連研究では提案されていない．

本提案方式を検証する試作機を作成し性能を評価した結果，Web ブラウザ全体の処理時間を，ソフトウェアのみの処理と比較し，88%に短縮，字句解析処理については消費電力量を 1/200 に削減できる見通しを得た．

参考文献

[1] 総務省：平成 21 年「通信利用動向調査」(2009)，入手先 ⟨http://www.soumu.go.jp/johotsusintokei/statistics/data/100427_1.pdf⟩.

[2] Hewlett-Packard Development Company: Why webOS? – HP webOS Developer Center, available from ⟨https://developer.palm.com/content/showcase/why_webos.html⟩.

[3] The Chromium Projects: The Chromium Projects, available from ⟨http://www.chromium.org/chromium-os⟩.

[4] 玉置亮太：スパコンからスマフォまで高速，省エネで新用途を開拓，日経コンピュータ，pp.80–84, 日経 BP 社 (2011).

[5] アイティメディア株式会社：IE，欧州で首位を Firefox に譲る—StatCounter の Web ブラウザ調査，入手先 ⟨http://www.itmedia.co.jp/enterprise/articles/1101/05/news014.html⟩.

[6] Microsoft Corporation: Internet Explorer Architecture, available from ⟨http://msdn.microsoft.com/en-us/library/aa741312(v=vs.85).aspx⟩.

[7] Mozilla Foundation: Gecko – MDC Doc Center, available from ⟨https://developer.mozilla.org/ja/Gecko⟩.

[8] Opera Software ASA：Web ブラウザ Opera，入手先 ⟨http://www.opera.com/⟩.

[9] The WebKit Open Source Project: The WebKit Open Source Project, available from ⟨http://www.webkit.org/⟩.

[10] KDE(R) Community: KDE – Experience Freedom!, available from ⟨http://www.kde.org/⟩.

[11] Open Source Initiative: Open Source Initiative OSI – The BSD. License:Licensing, available from ⟨http://www.opensource.org/licenses/bsd-license.php⟩.

[12] Free Software Foundation, Inc.: GNU LESSER GENERAL PUBLIC LICENSE Version 3, available from ⟨http://www.gnu.org/licenses/lgpl-3.0.txt⟩.

[13] nonTROPPO.org: Standard Page Load Test, available

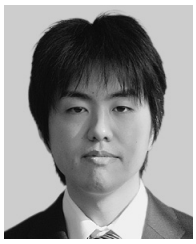
from ⟨http://nontroppo.org/timer/⟩.
[14] Levon, J.: OProfile – A System Profiler for Linux (News), available from ⟨http://oprofile.¥¥sourceforge.net/⟩.
[15] Dai Z., Ni, N. and Zhu, J.: A 1 Cycle-Per-Byte XML Parsing Accelerator, *FPGA'10*, February 21-23, Monterey, California, USA. (2010), available from ⟨http://oprofile.sourceforge.net/news/⟩.
[16] El-Hassan, F. and Ionescu, D.: SCBXP: An efficient hardware-based XML parsing technique, *5th Southern Conference on Programmable Logic, 2009 SPL*, pp.45–50, IEEE (2009).
[17] Chiu, K. and Lu, W.: A Compiler-Based Approach to Schema-Specific XML Parsing (2004).
[18] Weblio：XSD とは — .NET Framework 用語 Weblio 辞書，入手先 ⟨http://www.weblio.jp/content/XSD⟩.
[19] Kostoulas, M.G., Matsa, M., Mendelsohn, N., Perkins, E. and Heifets, A.: XML screamer: An integrated approach to high performance XML parsing, validation and deserialization, *15th International World Wide Web Conference*, pp.930–102, ACM Press (2006).
[20] Zhang, Y., Pan, Y. and Chiu, K.: *Speculative p-DFAs for Parallel XML Parsing*, pp.388–397, IEEE (2009).
[21] Van Lunteren, J., Engbersen, T., Bostian, J., Carey, B. and Larsson, C.: XML Accelerator Engine, *The 1st International Workshop on High Performance XML Processing* (2004).
[22] XML 用語事典，SAX (The Simple API for XML)，入手先 ⟨http://www.atmarkit.co.jp/aig/01xml/sax.html⟩.
[23] Chang, C.E., Mohd-Yasin, F. and Mustapha, A.K.: RB-StreX: Hardware XML parser for embedded system, *International Conference for Internet Technology and Secured Transactions*, *ICITST 2009*, pp.1–6, IEEE (2010).
[24] Renesas Electronics Corporation: SH-MobileR2, available from ⟨http://japan.renesas.com/products/mpumcu/sh_mobile/sh_mobile_r/sh_mobile_r2/sh_mobile_r2.jsp⟩.
[25] ALGO System：AP-3300【AlgoSmartPanel（アルゴスマートパネル)】，入手先 ⟨http://www.algosystem.co.jp/front/bin/ptdetail.phtml?Part=AP3300&Category=5286⟩.
[26] Xilinx Inc.：Virtex-5 FPGA ユーザーガイド，入手先 ⟨http://japan.xilinx.com/support/documentation/virtex-5_user_guides.htm⟩.
[27] Renesas Electronics Corporation：動作速度向上を目指しての FPGA からゲートアレイへの置き換え例，p.12 (2008).
[28] Renesas Electronics Corporation：CMOS-12M シリーズ，入手先 ⟨http://www2.renesas.com/gatearray/ja/12m/⟩.

井口 慎也 （正会員）

日立製作所．平成 10 年神戸大学工学部自然科学研究科情報知能工学専攻博士前期課程修了．同年（株）日立製作所に入社．横浜研究所にて，大量多種多様な非構造データからの情報報抽出とその応用に関する研究に従事．

中越 洋

日立製作所．平成 15 年九州大学大学院電気電子工学科卒業．平成 17 年早稲田大学大学院情報生産システム研究科博士前期課程修了．同年（株）日立製作所入社．同社横浜研究所においてシステム LSI 開発，運用管理システムの研究開発に従事．



原田 諭

日立製作所．平成 17 年大阪大学大学院工学研究科通信工学専攻博士前期課程修了．平成 20 年（株）日立製作所に入社．横浜研究所にて，大量のデータを高速に解析・処理するハードウェアの研究開発に従事．

望月 義則

日立製作所．平成 16 年横浜国立大学大学院工学府物理情報工学専攻博士前期課程修了．同年（株）日立製作所に入社．横浜研究所にて，制御システム向けネットワークに関する研究に従事．

宗像 尚郎

ルネサスソリューションズ．（株）ルネサスソリューションズにて SuperH CPU および ARM コア内蔵 SoC 向けの Linux の実装，デバイスドライバ開発，および組込み機器への実装における性能最適化支援業務に従事．あわせてオープンソース開発コミュニティ活動を支援．

竹内 弘道

ルネサスソリューションズ．平成 4 年（株）日立マイクロデバイス（現ルネサスエレクトロニクス販売（株））入社．論理設計・SoC 開発に従事し，現在は（株）ルネサスソリューションズにて，Linux Kernel，Android 関連の開発業務に従事．