Implementing Dynamic Web Page Interactions with a Java Processor Core on FPGA

Xiang Shen, Xu Wang, Yongxin Zhu, Tian Huang, Xue Kong
School of Microelectronics
Shanghai Jiao Tong University
Shanghai, China
shenxiang@ic.sjtu.edu.cn, wang2002xu@gmail.com, zhuyongxin@sjtu.edu.cn, ian_malcolm@sjtu.edu.cn, kongxue448@163.com

Abstract- In the era of cloud computing, web servers as the major channel in cloud computing need to be redesigned to meet performance and power constraints. Considerable efforts have been invested in web server distribution and web caching strategies, but very few efforts have been paid to improve hardware-favored web services. In this paper, we implement a Field Programmable Gate Array (FPGA) based embedded web server that can process simple dynamic web page in hardware. In our design, a light weight web server Lightweight IP (LwIP) is adopted as the basic implementation of TCP/IP networking stack. More importantly, we implement a Java Optimized Processor (JOP) as the hard core processor to directly support calculation required by dynamic web page interactions. The JOP processor is tightly coupled with the LwIP running on a soft core *MicroBlaze* to accelerate dynamic web page processing. We implement dynamic web pages to process matrix multiplication requests which are actually handled either by JOP or *MicroBlaze* as our testing case. In our experiment, the execution time by JOP is about one tenth of the time taken by *MicroBlaze*. As probably the first dynamic web page processing based on JOP in cloud computing domain, our implementation would be an interesting milestone to further improve efficiency of cloud computing.

Keywords- Dynamic web page; JOP; LwIP; FPGA;JVM

I. INTRODUCTION

The exponential growth of the Internet, coupled with the increasing popularity of dynamically generated content on the World Wide Web, has created the need for more and faster Web servers serving a soaring number of Internet users [1]. Web pages, the basic elements of website, are closer to people's lives and need to meet higher demands. Traditional static web pages cannot perfectly meet people's demands, and dynamic web pages which are based on database and change their content or layout with the users, have been growing very rapidly. Web servers need to store numerous data submitted by users and give a quick response to clients. As people's living standards continue to rise, web-enabled systems are applied more and more widely, and embedded web servers come into being. However, in most cases embedded web servers have to be able to receive requests and commands from a remote Hypertext Transfer Protocol (HTTP) web client and also have to be able to send the control signals and management interface of the web-enabled device in the form of Hypertext Markup Language (HTML) pages. The computation, storage and resources in embedded web servers are restricted [2]. To meet these conflicting requirements, many researchers are working toward implementing a functionally minimized Web server on silicon—WebChip [3], or even FPGA [4]. Also, embedded web server should be a system on a chip that is capable of serving dynamic web pages [5].

At present, major dynamic web page technologies are Active Server Page (ASP), Java Server Pages (JSP) and Hypertext Preprocessor (PHP). Compared with the other two technologies, JSP technology has a greater advantage. A JSP page contains standard markup language elements, such as HTML tags. However, a JSP page also contains special JSP elements that allow the server to insert dynamic content in the page. JSP elements can be used for a variety of purposes, such as retrieving information from a database or registering users. When a user asks for a JSP page, the server executes the JSP elements, merges the results with the static parts of the page, and sends the dynamically composed page back to the browser [6]. To process JSP pages, a JSP container, responsible for intercepting requests for JSP pages, is needed by the server. The container first turns the JSP page into a servlet, a piece of code that adds new functionality to a server, and then converts all JSP elements to Java code that implements the corresponding dynamic behavior [6]. Next, the container compiles the servlet class to generate Java bytecode which is executed by Java Virtual Machine (JVM) [7, 8].

According to above introduction of JSP processing, a JSP container which is often implemented as a servlet that is configured to handle all requests for JSP pages, is a complex and nearly impossible implementation in hardware systems. Fortunately, JSP that embeds special active elements into HTML page, separates static content and dynamic content to achieve the separation of content and presentation. These active elements which are actually componentized Java programs that the server executes when a user requests the page look similar to HTML elements. Therefore, we propose a FPGA based embedded web server to realize dynamic web page by HTML page with active elements and without JSP elements. The dynamic requests from users are resolved by *MicroBlaze* [9] and implemented by Java Optimized Processor (JOP) [10, 11, 12] with Java programs.

JOP, an implementation of the Java virtual machine targeted for small embedded systems in hardware, is intended for

applications in embedded real-time systems. The primary technology for JOP in hardware is implemented on FPGA [13]. Compared with software based JVM on a standard PC, JOP is much faster than an interpreting JVM on a standard processor for an embedded system [14]. Since embedded applications are multi-threaded systems, the performance can easily be enhanced using a multi-processor solution. The authors of [15, 16] propose a Chip-MultiProcessor (CMP) architecture composed of multiple JOP cores and a shared memory and compare the performance and size of JopCMP with a complex Java processor. Chip multiprocessing design is an emerging trend for embedded systems.

The author of [17] describes an embedded system example design of a Web server running on the *MicroBlaze* soft processor with the Embedded Development Kit (EDK). Recently, Lightweight IP (lwIP) [18], an open source TCP/IP networking stack for embedded systems, is implemented on Xilinx XUPV5_LX110T board. The application example describes how to add networking capability to an embedded system. LwIP can be utilized to develop the application of web server.

In this paper, we propose an embedded web server architecture composed of JOP processor core and LwIP which is the basic implementation of TCP/IP networking stack, and implement it on FPGA. Dynamic web page will store lots of data submitted by user, so this system will often read data from memory and write data to memory. The problem of limited computation and storage resources in embedded web server is really difficult to solve. Therefore, we focus on the operations to memory and the computation capability of the embedded web server system. We choose matrix multiplication which is compute-intensive with low synchronization overhead as our application. The application poses a challenge to computation and storage. We implement a dynamic web page to submit matrix multiplication request which is actually handled either by JOP or *Microblaze* as our testing case. In our experiment, the execution time by JOP is about one tenth of the time taken by *MicroBlaze*.

The rest of the paper is structured as follows. Section II presents related work and delivers an insight into LwIP and JOP. In Section III, we describe design and implementation of dynamic web page processing on FPGA, including system architecture of our embedded web server, micro-architecture of SW/HW partitions inside FPGA and dynamic web page processing. Section IV presents the experimental setup for our system. In section V, we show the results of our experiment and make a simple analysis. Finally, Section VI concludes the paper and provides guidelines for future work.

II. RELATED TECHNICAL COMPONENTS

Up to date, there are many Java processors implemented in hardware, such as picoJava， aJile, Moon, Lightfoot, Cjip, Komodo and FemtoJava [14]. Compared with above embedded Java processors, JOP is the smallest realization of hardware JVM on an FPGA and also has the highest clock frequency. LwIP provides support for the Internet Protocol (IP) and Transmission Control Protocol (TCP) [19], which are needed in our embedded web server. In the following parts, a brief overview of the architecture of JOP and LwIP is given.

A. JOP

This part gives an overview of JOP architecture. Fig.1 [20] shows JOP's major function units. A typical configuration of JOP contains the processor core (CPU), a memory interface and I/O interface. The I/O interface, memory interface and scratchpad memory are connected to memory controller which is the interface of processor core via SimpCon bus [21] by a multiplexer. SimpCon is an interconnection interface standard. Scratchpad memory is used with a mapping to a Real-Time Specification for Java (RTSJ) style scoped memory [22].

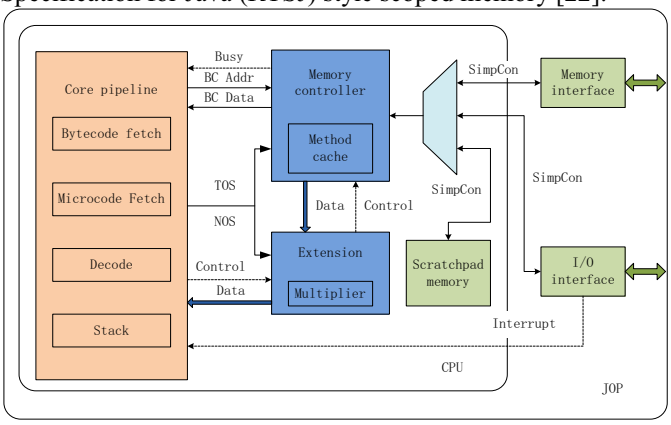

Fig.1 Block diagram of JOP

The processor core contains three pipeline stages of microcode fetch, decode and execute (stack) as well as an additional translation stage of bytecode fetch, memory controller and multiplier. The ports of core pipeline to the other modules are Top-Of-Stack (TOS), Next-Of-Stack (NOS), input to the stack (Data), bytecode cache address and data, and some control signals. The memory controller with method cache [23] implements the simple memory load and store operations, and the field and array access bytecodes. The memory interface provides a connection between the main memory and the memory controller. The extension module controls data read and write. The busy signal is used to synchronize the processor core with the memory unit.

The I/O interface contains peripheral devices, such as the system timer, timer interrupt for real-time thread scheduling, a serial interface and application-specific devices. Read and write to and from this module are controlled by the memory controller.

The division of the processor into those modules greatly simplifies the adaptation of JOP for different application domains or hardware platforms. Porting JOP to a new FPGA board usually results in changes in the memory interface alone.

B. LwIP

Fig.2 shows the system architecture of LwIP. The processor of the system is soft core *MicroBlaze* processor which is connected with a Multi-Port Memory Controller (MPMC) and the other required peripherals (timer, Ethernet and GPIO) on the Processor Local Bus (PLB) [24] interface bus. Interrupts

from both the timer and the Ethernet are required, so the timer and the Ethernet use the interrupt controller which is connected to PLB interface bus. IIC is used to collect local time information from EEPROM. *MicroBlaze* Debug Module (MDM) and UART are necessary to debug and monitor the network status.

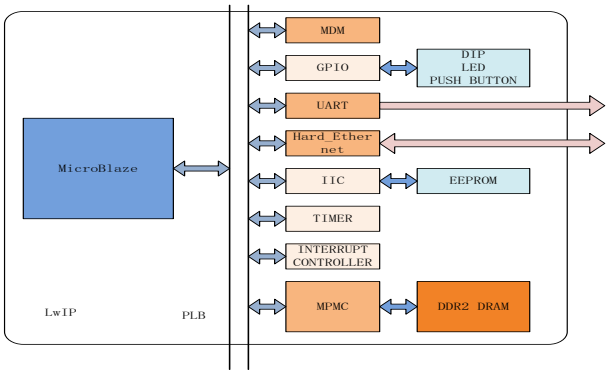

Fig.2 System architecture of LwIP

The Web server based-on LwIP is used to control or monitor an embedded platform via a browser. The embedded web server implements only a subset of the HTTP 1.1 protocol. The http thread first reads the request, identifies if the request is a GET or a POST operation, and then performs the appropriate operation. For a GET request, the thread looks for a specific file in the memory file system. If the specific file is present, the file is returned to the Web browser initiating the request. If the file is not available, a HTTP 404 error code is sent back to the browser. LwIP provides an easy way to add TCP/IP-based networking capability to an embedded system.

III. Design and Implementation of Dynamic Web Page Processing

In this section, we will present our system architecture and micro-architecture as well as dynamic web page processing in detail. Our design idea is reflected in the part of the system architecture and the micro-architecture is the implementation of our embedded web server on FPGA. The third part introduces dynamic web page processing, which is the ultimate goal of our present study.

A. System Architecture

The overall system architecture of our embedded web server is shown in Fig.3. *MicroBlaze* soft core processor in LwIP is the main processor and connects the collaborative Java processor JOP by a share memory. The share memory, a dual-port block ram, connects LwIP by PLB interface bus with IP Interface (IPIF) [25] which is designed to provide user with a quick to implement and highly adaptable interface between PLB bus and a user IP core, as well as JOP by SimpCon bus with memory interface.

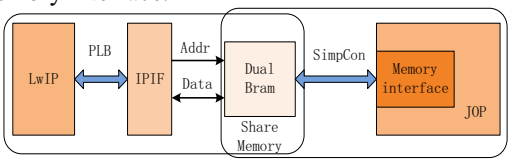

Fig.3 System architecture of our embedded web server

JOP is open source under the GNU General Public License, and complete VHDL source and tools in Java are available for downloading from website. With complete VHDL source and EDK tool, it is not difficult to package all VHDL source into IP and connect the IP to PLB bus. Before this, we need to change the memory for bram and ensure that the modified system can work properly.

To make the entire web server work, the problems of clock and address map have to be considered. The clock in LwIP is 125MHz while the clock in JOP is 100 MHz. It is easy to lower the whole system clock to 100MHz except the part of Ethernet because the Ethernet must work at 125MHz in the LwIP based system. The share memory is allocated 64KB in EDK with the address of 0x5000000 to 0x5000ffff, and in JOP the address is 0x0000 to 0xffff. In hardware, we connect 16-bit address in JOP with low 16-bit address of 32-bit address in PLB to complete the address map.

B. Micro-architecture of SW/HW partitions inside FPGA

Our system hardware/software partitions are shown in Fig.4, as the micro-architecture. *MicroBlaze*, the soft core, is connected to hardware modules by PLB bus. Two UARTs and one Ethernet, the main peripherals in our system, are used to debug and show our results. One UART is the I/O device of JOP and used to download the *.jop* file which makes up the linked Java application that runs on JOP, and show the debug information about JOP. The other UART is directly connected with PLB and used to debug the software for LwIP. By Ethernet, the main application of our system, we can realize dynamic web page with the HTTP protocol and TCP/IP protocol provided by LwIP. MDM module is used to debug the applications running on *MicroBlaze*, download file of memory file system (*.mfs*) used for web server applications which is generated by web page files and download software application file. DRAM is used to store the file of memory file system from the address of 0x92000000 which is set by the developers in EDK. The remaining peripherals of timer and interrupt controller (INTC) are also essential.

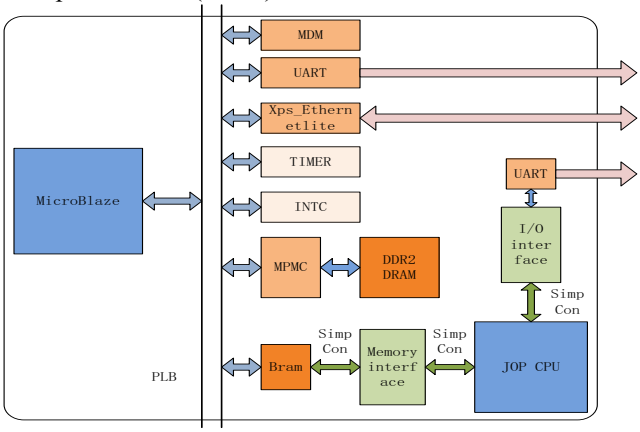

Fig.4 Micro-architecture of SW/HW partitions

C. Dynamic Web Page Processing

Fig.5 shows the dynamic web page processing. We call the hardware and software processing our web pages Dynamic Hypertext Markup Language (DHTML) Container which processes the request from client and sends response to the

client. When the container receives the request from client, the DHTML file which is transported from client by TCP/IP protocol, is resolved by *MicroBlaze*. Then we can get the dynamic content that should be processed and stored. Next, the dynamic content is written to the share memory. We set up a flag in memory for Java program running on JOP and when the flag is enabled by *MicroBlaze*, the Java program begins to be executed. Similarly, we set up another flag in memory for LwIP and the flag is enabled by Java program in JOP when the results are obtained and written back to bram in JOP. At this time, the right results are in share memory and can be visited by *MicroBlaze*. Then the results will be inserted into web page by *MicroBlaze* and new DHTML file (new web page) is generated. In the end, new web page is transported to client. One operation to dynamic web page is completed. This is the whole dynamic web page processing.

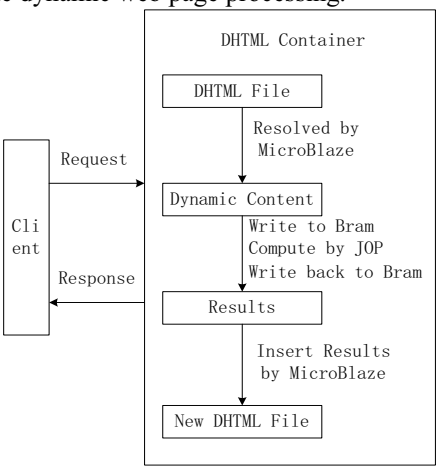

Fig.5 Dynamic web page processing

IV. EXPERIMENTAL SETUP

The platform of our system is XUPV5_LX110T prototyping board supported by Xilinx. The design tool we used is Xilinx ISE Design Suite 12.2, including ISE and EDK development tools. Firstly, we transplant the open source into board XUPV5_LX110T with ISE tool, and make sure that we can run Java program properly. And then, we package all VHDL code into IP and connect the IP to PLB bus with EDK tool. Next, we unify the clock frequencies and deal with the problem of two UARTs in hardware. Fortunately, there are two sets of UART interface. Only one can be connected with serial line and we connect the other by interconnection lines. Besides, we should make a right interconnection of ports in EDK.

After connecting power line, Ethernet, USB-JTAG programming cable and two UARTs with board and PC, we first download *.bit* file into board with ISE iMPACT tool and then download *.jop* file with download command by UART. Next, in MDM debug interface we download *.mfs* file into address 0x92000000 of DDR and *.elf* file into bram by specific commands in command line mode. Open hyperterminal and browser, enter command *run* in MDM debug interface, input the Uniform Resource Locator (URL) in browser, and then we can see some information in hyperterminal. Input the values of matrices in web page and click the button *compute*, and several seconds later we can get the results displaying in web page.

V. EXPERIMENTAL RESULTS

In this section, we first present our functional results and then we show and compare the slice logic utilization of FPGA between the project without JOP and the project with JOP, and execution time spanning from writing dynamic content to bram to writing back results to bram after processed by JOP. We evaluate the system with the benchmark of matrix multiplication that gives some idea about the performance of a compute-intensive application.

A. Functional Results

Fig.6 shows the functional results of our system implemented on FPGA.

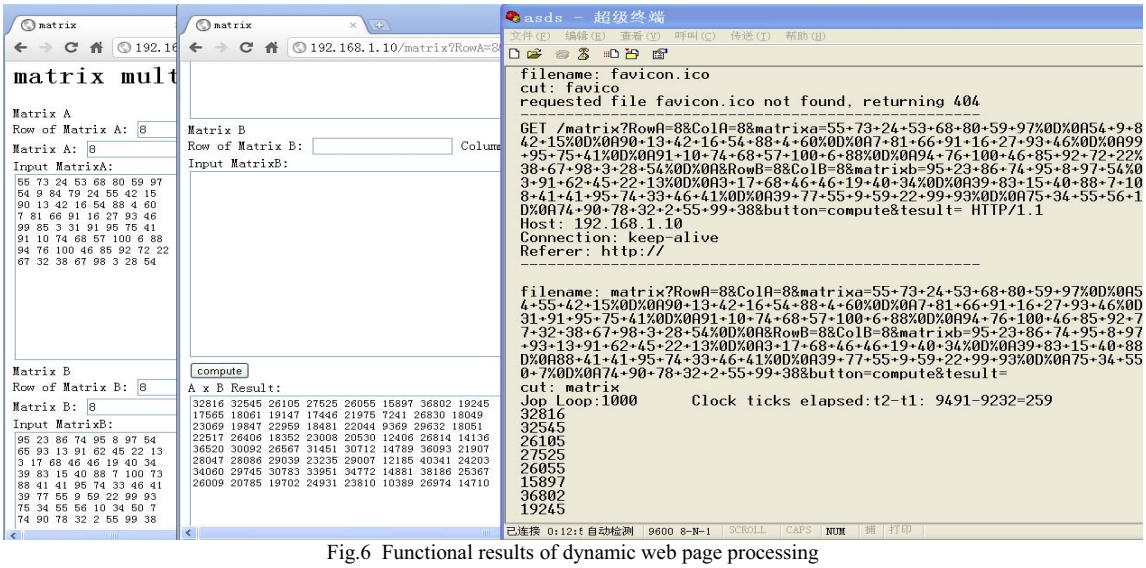

Fig.6 Functional results of dynamic web page processing

There are three parts in this figure. The leftmost part of the figure is the home page which shows a matrix input page, including the row and column input boxes of the two matrices, text boxes of the two matrices, a submit button and result text box. We can see the two 8x8 matrices whose values are keyed by user in the figure. The middle part of the figure is the response page which shows the results obtained by computing in JOP. The right part of the figure is the debug information in hyperterminal and we can see the partial request content sending by TCP/IP protocol. Between the two dotted lines is the requested dynamic content which is another display form for our request in the leftmost part of the figure and they are absolutely identical. Below the second dotted line is some debug information. We can see the number of clock ticks which is reflected the execution time for JOP processor and partial results of matrix multiplication.

B. Device Utilization

As a very small realization of hardware JVM on FPGA, JOP just uses a little logic resources of FPGA. Table I shows the slice logic utilization in our research on board XUPV5_LX110T. We compare the project without JOP to the project with JOP from three aspects: slice registers, slice LUTs and BlockRAM/FIFO. The rate in the second row and third row is used logic compared with available logic in the whole FPGA. We can see that the number of BlockRAM/FIFO increases a lot because we use the bram as the share memory and store dynamic content from client in the bram. And the other two aspects of slice registers and slice LUTs increase not much, which can be seen from the additional rates of 13.4% and 22.3%. Based on the logic utilization, we can extend our web server to the Java multiprocessor system, and multiprocessing will give more performance improvement.

C. Execution Time

In this part, we simply evaluate the performance of our embedded web server. The most important performance of the web page application is response time and the most part of the response time is execution time in compute-intensive application. So we evaluate the execution time instead of response time. We cannot directly get the execution time, but Xilkernel provides software timer functionality for time relating processing. *xget_clock_ticks()* returns the number of kernel ticks elapsed since the kernel was started and the parameter of *sleep ()* is the number of milliseconds to sleep. With those two functions we can get the approximate execution time.

Fig.7 shows the clock ticks for matrix multiplication running for 1000 loops and speedup of JOP compared to *MicroBlaze*. By the functions of *sleep ()* and *xget_clock_ticks()* we can know that one clock tick is equal to 10 milliseconds. From the figure we can see that for the 8x8 matrix multiplication, the execution time by JOP is 259 clock ticks for 1000 loops (2.59 ms for one loop) while the time taken by *MicroBlaze* is 2780 clock ticks (27.8 ms for one loop) and the speedup is 9.7 by simple mathematical operation. In this experiment, we just test up to 8x8 matrix because of the limitation of GET method in HTTP protocol. The minimum speedup is 9.5 while the maximum speedup is 11.0 and the average speedup is 9.9. Fortunately, the results can illustrate that our system is meaningful and obtains a relatively large speedup.

TABLE I SLICE LOGIC UTILIZATION

Slice Logic	Slice Registers		Slice LUTs		BlockRAM/FIFO	
	Number	Rate	Number	Rate	Number	Rate
Without JOP	13,314	19%	12,325	17%	39	26%
With JOP	15,099	21%	15,077	22%	100	67%
Additional	1,785	13.4%	2,752	22.3%	61	156.4%

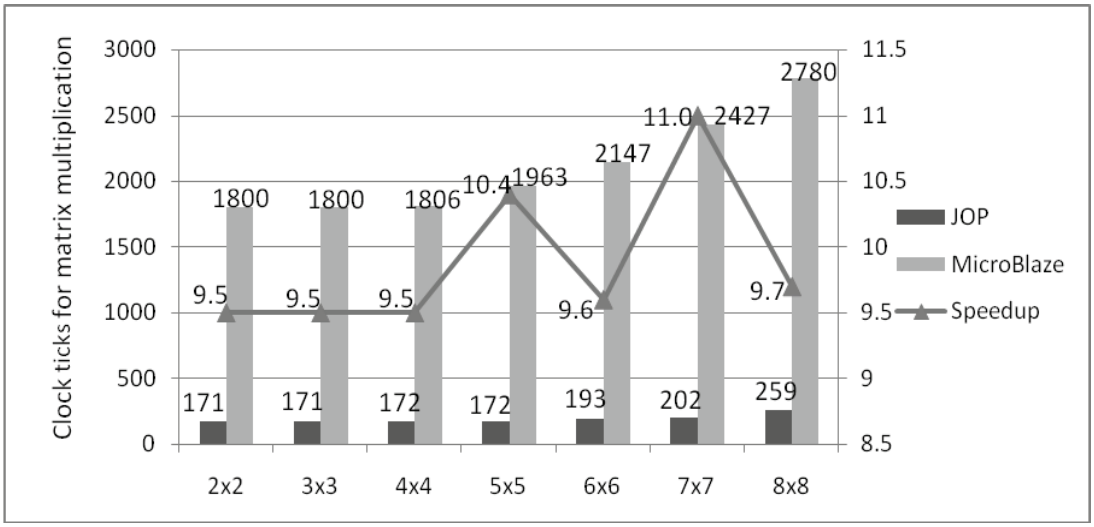

Fig.7 Performance comparison

VI. CONCLUSION AND FUTURE WORK

We propose the architecture of a dynamic web page processing system and implement it on FPGA containing a JOP Java processor. As far as we know, our work would be the first implementation of dynamic web page on FPGA with hardware implementation of JVM. Matrix multiplication is chosen as the benchmark to evaluate our implementation. As the experimental results, our embedded web server is shown that an average speedup of 9.9 is achieved over a software based system. Although our results are not perfect in terms of completeness of functionalities, our design and implementation experience may serve as a useful reference to the peer researchers and developers.

In the future, in order to go beyond the functionality limitation of GET method in HTTP protocol, we will further implement POST method in our web server. Extension to designs containing multiple JOP processors within one FPGA chip will be also our new milestone since multiprocessing design is an emerging trend for embedded systems and Java multiprocessor system-on-chip has been implemented on FPGA.

ACKNOWLEDGMENT

This paper is partially sponsored by the National High-Technology Research and Development Program of China (863 Program) (No. 2009AA012201) and the Shanghai International Science and Technology Collaboration Program (09540701900).

REFERENCES

[1] T. Schroeder, S. Goddard, B. Ramamurthy, "Scalable Web server clustering technologies," IEEE Network, May-June 2000, pp. 38-45.

[2] D. Raskovic, V. Revuri, D. Giessel, A.Milenkovic, "Embedded Web Server for Wireless Sensor Networks," In 41st Southeastern Symposium on System Theory, Tullahoma, TN, USA, March 2009.

[3] J. Riihijarvi, P. Mahonen, M. J. Saaranen, J. Roivainen, J. P.Soininen, "Providing network connectivity for small appliances: a functionally minimized embedded Web server," IEEE Communication, Mag, vol. 39, no. 10, pp. 74–79, Oct 2001.

[4] H. Fallside, M. John, S. Smith, "Internet connected FPGAs," In 2000 IEEE Symposium Field-Programmable Custom Computing Machines, Napa Valley, CA, Apr. 17–19, 2000, pp. 289–290.

[5] I. Klimchynski, "Extensible embedded web server architecture for Internet-based data acquisition and control," Sensors Journal, IEEE, vol. 6, pp. 804–811, June 2006.

[6] O'Reilly, JavaServer Pages, 3rd Edition, ISBN: 0-596-00563-6, December 2003.

[7] T. Lindholm, F. Yellin, The Java Virtual Machine Specification, 2nd Edition, Reading, MA, USA: Addison-Wesley, 1999.

[8] B.Vcnners, Inside the Java Virtual Machine, Second Edition, The McGraw-Hill Companies, ISBN: 0-07-135093-4, 1999.

[9] Xilinx, MicroBlaze Processor Reference Guide, Embedded Development Kit EDK 13.1, http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/mb_ref_guide.pdf.

[10] M. Schoeberl, "Jop: A java optimized processor for embedded real-time systems," Ph.D. dissertation, Vienna University of Technology, 2005.

[11] M. Schoeberl, "A Java processor architecture for embedded real-time systems," Journal of Systems Architecture, vol. doi:10.1016/j.sysarc, 2007.06.001, 2007.

[12] M. Schoeberl, "A time predictable Java processor," In Proceedings of the Design, Automation and Test in Europe Conference (DATE 2006), Munich, Germany, March 2006, pp. 800–805.

[13] M. Schoeberl, "Using a Java Optimized Processor in a Real World Application," In Proceedings of the First Workshop on Intelligent Solutions in Embedded Systems, WISES 2003, Vienna, Austria, June 2003.

[14] M.Schoeberl , "Evaluation of a Java processor," Tagungsband Austrochip 2005, Vienna, Austria, October 2005, pp.127–134.

[15] C. Pitter, M. Schoeberl, "A real-time Java chip-multiprocessor." ACM Trans. Embed. Compute. Syst., 10(1):9:1–34, 2010.

[16] C. Pitter, M. Schoeberl, "Performance evaluation of a Java chip-multiprocessor," In Proceedings of the 3rd IEEE Symposium on Industrial Embedded Systems, SIES 2008, Jun, 2008

[17] R. Armstrong, M. Muggli, M. Ouellette, S. Thammanur, Embedded System Example: Web Server Design Using MicroBlaze Soft Processor, http://www.xilinx.com/support/documentation/application_notes/xapp433.pdf.

[18] S. MacMahon, N. Zang, A. Sarangi, LightWeight IP (lwIP) Application Examples, http://www.xilinx.com/support/documentation/application_notes/xapp1026.pdf.

[19] Xilinx, EDK OS and Libraries Reference Guide, Embedded Development Kit EDK 6.2i, http://www.xilinx.com/ise/embedded/edk_libs_ref_guide.pdf.

[20] M. Schoeberl, JOP Reference Handbook: Building Embedded Systems with a Java Processor,CreateSpace, ISBN:978-1438239699, August 2009.

[21] M. Schoeberl, "SimpCon-a simple and efficient SoC interconnect," In Proceedings of the 15th Austrian Workhop on Microelectronics, Austrochip 2007, Graz, Austria, October 2007.

[22] A. Wellings, M. Schoeberl, "Thread-local scope caching for real-time Java," In Proceedings of the 12th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2009), IEEE Computer Society, Tokyo, Japan, 2009.

[23] M. Schoeberl, "A time predictable instruction cache for a Java processor," In On the Move to Meaningful Internet Systems 2004: Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES 2004), ser.LNCS, vol. 3292. Agia Napa, Cyprus: Springer, October 2004, pp.371–382.

[24] Xilinx, LogiCORE IP Processor Local Bus (PLB) v4.6 (v1.05a), Product Specification, http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf.

[25] Xilinx, PLB IPIF (v2.02a), Embedded Development Kit EDK 12.2, http://www.xilinx.com/support/documentation/ip_documentation/plb_ipif.pdf