Design and VLSI implementation of a Java System-on-Chip

Xuming Lu, Desheng Yang, Hongzhou Tan
School of Information Science and Technology, Sun Yat-sen University
Guangzhou, China
E-mail: luxuming@189.cn

***Abstract*—A new architecture of Java System-on-Chip (SoC) is proposed for executing Java bytecodes directly. This Java SoC integrates a Java core, a Floating Point Unit (FPU), a Direct Memory Access (DMA) controller, and some other popular peripheral controllers on a single chip. All modules are interconnected with Advanced Microcontroller Bus Architecture (AMBA) standard buses. IO reuse is adopted to reduce the number of chip pins. The proposed SoC is implemented in very large scale integration (VLSI) of 130nm CMOS Logic process. After being taped-out and packaged, the chip is tested with a verification board. The performance is evaluated by Dhrystone and Whetstone benchmarks. The running results show that the proposed SoC has a better performance compared with VP6000 and JOP3.**

Keywords-Java SoC; AMBA; VLSI; benchmark score

I. INTRODUCTION

Java technology has been developing rapidly since it was released in 1995, due to its great characteristics of versatility, efficiency, platform portability, and security. Platform portability of Java is achieved by compiling the Java code to an intermediate representation called Java bytecode, instead of directly to platform-specific machine code. Java bytecode instructions are analogous to machine code, but are intended to be interpreted by a unique virtual machine for the host hardware. Therefore, Java virtual machine (JVM) connects high-level Java programs and underlying hardware.

Despite the advantages mentioned above, programs written in Java have a reputation for being slower and requiring more memory than those written in C. This is partly due to the software-implemented JVM, since it has low execution efficiency or requires relative large memory resources. It will be a serious problem when implemented in hardware-restricted embedded devices such as cell phones or TV set-top boxes. The performance may not be satisfying as software-implemented JVM is such a burden for embedded devices.

To deal with the execution efficiency problem, three approaches may be summarized: dedication, translation, and acceleration. Processors, such as PicoJava II [1] and JOP3 [2], belong to dedicated Java processor. Processors of this kind take bytecodes as their native instructions and execute them directly. In the translation approach, a small hardware unit is added between the instruction fetch and decoding units of a general-purpose processor core to convert most of the bytecodes into native instructions at run-time. Examples using this method are ARM Jazelle [3] and JA108 [4]. Coprocessors, such as the AU-J2000 [5] and REALJava [6], execute Java bytecodes so that the system can execute the application programs written by both Java and other programming languages supported by the host processor. Such coprocessors provide good support to Java without affecting the compatibility of the host general-purpose processors, but chip area and power consumption increase significantly, which are critical factors in embedded devices.

This paper proposes a 32-bit AMBA based Java SoC named JM8BC013A with a Java core, a Floating Point Unit (FPU), a Direct Memory Access (DMA) controller and some other popular peripheral controllers integrated. The paper focuses on the design and implementation as well as the performance of the Java SoC, which shall be called an updated version of Java SoC VP6000 [7]. The running results show that the proposed Java SoC work better compared with VP6000 and JOP3.

This paper is organized as follow. Section II presents the design of the proposed Java SoC. Section III is devoted to the implementation of the SoC, including the FPGA and VLSI implementations. Section IV describes the performance results and the comparison with other related processors. Finally, in Section V, conclusions are derived.

II. SYSTEM DESIGN

JM8BC013A is designed to be dedicated to Java applications. As a Java SoC, it is based on a Dual-AHB AMBA bus, with a modified Java core called JOP, FPU, DMA and some other peripheral controllers attached. The architecture of JM8BC013A is shown in Fig. 1.

A. Dual-AHB AMBA Bus Architecture

As shown in Fig. 1, there exist two sets of Advanced High-performance Buses (AHB) and one Advanced Peripheral Bus (APB), which is called Dual-AHB AMBA bus architecture. The AMBA AHB is for high-performance, high clock frequency system modules whereas APB is optimized for minimal power consumption and reduced interface complexity to support peripheral functions. The AHB acts as the high-performance system backbone bus [8].

Published by Atlantis Press, Paris, France.

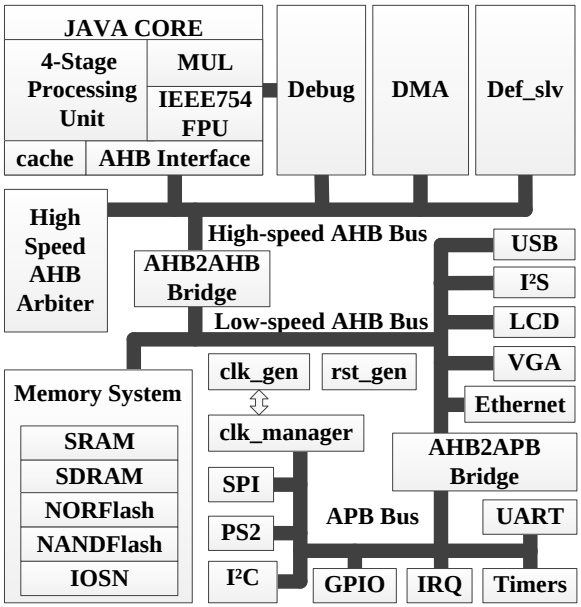


Figure 1. Architecture of Java SoC JM8BC013A

The proposed Dual-AHB AMBA bus architecture divides AHB into two layers, a high-speed and low-speed. On the high-speed AHB bus, three masters and two slaves are connected along with the high-speed AHB arbiter. The three masters are Java core, Debug module and DMA controller respectively. The two slaves are AHB2AHB bridge and the default slave module. On the low-speed AHB bus, the AHB2AHB bridge is the only one master. Six peripheral controllers that require relative high speed or high data throughput, i.e., USB, I^2S, LCD, VGA, Ethernet, and memory system, are connected as slaves. Besides, the AHB2APB bridge is connected to the low-speed AHB bus as a slave. For the APB bus, the only master is the AHB2APB bridge and the eight slaves are UART, Timer, Interrupt request (IRQ), general purpose input/output (GPIO), I^2C, PS2, SPI, and clk_manager.

With the Dual-AHB AMBA bus architecture, modules on the two layers of AHB can operate at different clock frequencies. The ratio of the frequencies can be configured dynamically with the corresponding register in the clk_manager module. With a proper ratio, total dynamic power consumption can be reduced without lowering the whole performance of the chip.

B. *Bus Access Priority*

As can be seen, there are three masters connected on the high-speed AHB bus. Since only one master can access the bus at a time, it comes up with a bus access priority problem.

Java core is a stack based fully pipelined architecture with single cycle execution of microcode instructions. This reduced instruction set computer (RISC) Java processing core uses a four-stage pipeline to accomplish the execution.

Debug module is used to initialize the inner RAMs which are supposed to contain the instruction set and other runtime related information. It loads all the microcodes, jump table addresses, variables, etc., from external flash into the chip [9]. Then the Java core runs properly after initialization [10].

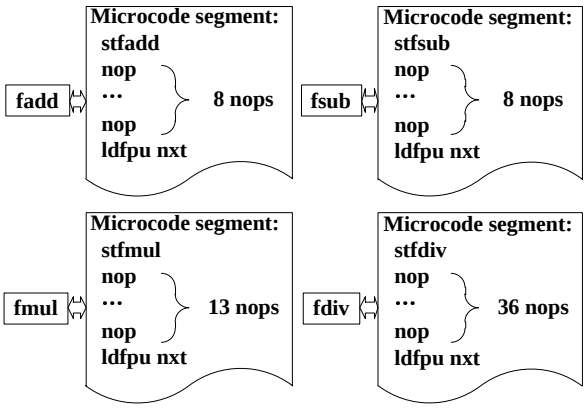


Figure 2. Mapping relationship between bytecodes and microcodes

DMA controller is designed to accomplish the mass data transmission between any two modules. Therefore, it is a master in accessing the high-speed AHB bus.

As described above, the debug module should obtain the highest bus access priority. After the initialization, the access for the bus is released from the debug module. During runtime, Java core is the main core taking charge of all hardware modules. Therefore, it has the second highest priority. The lowest priority belongs to DMA controller, which ensures that Java core can access the AHB bus even in a transmission task.

For the low-speed AHB bus and the APB bus, the AHB2AHB bridge and the AHB2APB bridge are the only masters, respectively. Thus, no access priority scheme is required for these buses.

C. *Floating Point Unit*

To support floating-point arithmetic operations in hardware, a 32-bit floating point unit (FPU) is implemented. It supports addition, subtraction, multiplication, and division.

As a hardware accelerator, the communication between Java core and FPU should be as direct as possible to ensure a higher efficiency. Thus the FPU is integrated into the extension module directly, instead of mounting on the AMBA bus as a peripheral module, as shown in Fig. 1.

The microcodes stfadd (0x0E), stfsub (0x06), stfmul (0x07), and stfdiv (0x1A) are introduced for FPU to read parameters and start a calculation. The microcode ldfpu (0xE6) controls FPU to return operation results back to the top register of the stack.

With these microcodes, the four single-precision floating point type bytecodes, namely fadd, fsub, fmul, and fdiv, can be implemented with microcode segments, instead of Java methods. The mapping relationship is shown in Fig. 2.

D. *IO Reuse*

With a number of peripherals integrated, JM8BC013A is competent for various kinds of embedded applications. However, it also leads to a large number of chip pins, which will greatly increase the complexity and cost for packaging

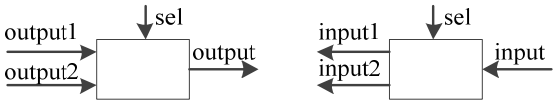


Figure 3. IO reuse scheme

the chip. Therefore, IO reuse method is adopted to reduce the number of chip pins, as shown in Fig. 3.

For output reuse, output1 and output2 are the output pins before reuse, and output is the output pin after reuse. output=output1 when sel=0, output=output2 when sel=1. Similarly,for input reuse, input1=input when sel=0, input2=input when sel=1.

A special register named io_reg should be configured to select which peripheral controller uses the reused IO pads.

By adopting IO reuse, 101 external pins have been saved. There are finally 297 external pins, among which 216 are functional pins and the other 81 are power pins.

E. Address Mapping

In the AMBA based system, a united address mapping method for memory system and peripheral controllers is proposed. The address mapping of AHB bus is shown in TABLE I.

Since the AHB2APB bridge is the only master on APB bus, all visits from AHB to APB must go through this bridge. Thus the addresses of modules on APB bus should be within 0x80000000-0x8FFFFFFF.

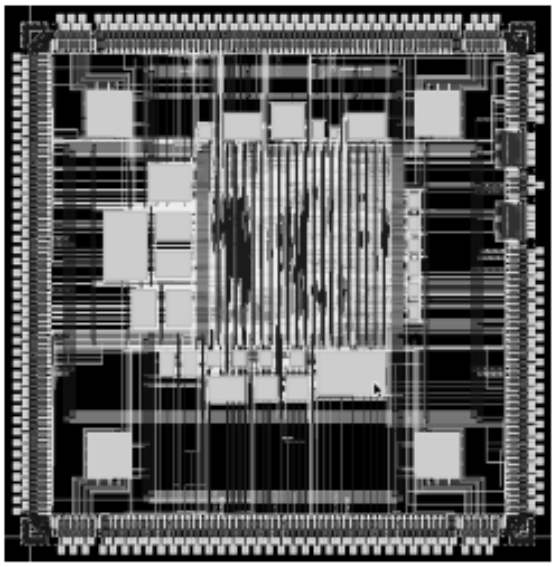
F. Low-Power Design

In order to reduce the power consumption, the operating frequencies of modules are designed to be configurable. Some modules allow a standby operating mode when idle. This is realized by lowering its operating frequency or shutting down the clocks, such as Debug module. The module clk_manager in Fig. 1 is used to store the configuration information, and the module clk_gen generates clocks as configured in clk_manager.

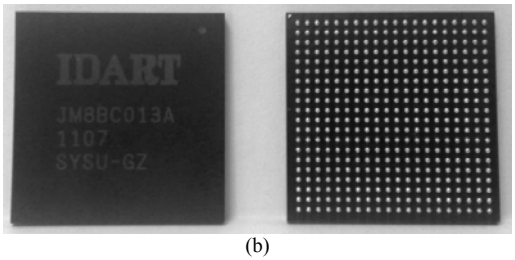
Although the Dual-AHB AMBA bus architecture makes it possible to set different frequencies for each bus, some modules mounted on the low-speed AHB bus may operate at a lower frequency due to their data throughputs. For example, the LCD module operates at 100MHz, whereas the USB module operates at 48MHz and Ethernet module at 25MHz. Therefore, these modules operate at different frequencies, and have asynchronous interfaces for mounting on AHB bus.

TABLE I. AHB SLAVES ADDRESS MAPPING TABLE

Address Range	Mapping Module
0x00000000-0x7FFFFFFF	Memory System
0x80000000-0x8FFFFFFF	AHB2APB Bridge
0x90000000-0x9FFFFFFF	LCD
0xA0000000-0xAFFFFFFF	VGA
0xB0000000-0xBFFFFFFF	USB
0xC0000000-0xCFFFFFFF	I^2S
0xD0000000-0xDFFFFFFF	Ethernet
0xE0000000-0xEFFFFFFF	DMA
0xF0000000-0xFFFFFFFF	Default_slv



(a)



(b)

Figure 4. JM8BC013A: (a) final layout (b) packaged chip

III. IMPLEMENTATION

The proposed Java SoC is coded in hardware description language (HDL). The prototype is verified by simulation and FPGA implementation. Besides, the implementation in very large scale integration (VLSI) technology has been taped out.

A. VLSI Implementation

The design has been implemented on a single VLSI chip using 130nm 1-poly 6-metal layer CMOS technology, which is SMIC (Semiconductor Manufacturing International Corporation) 130nm standard cell library. The results of implementation are listed in TABLE II. The dies are packaged into CABGA-400, whose body size is 17mm×17mm×1.2mm. Fig. 4 presents the final layout of JM8BC013A and the packaged chip.

TABLE II. RESULTS OF VLSI IMPLEMENTATION

Technology	SMIC 130nm
Gates	59.3K
Area	4.8mm × 4.8mm
Maximum clock frequency	330 MHz
Supply voltage	3.3V & 1.2V
Total dynamic power	147 mW

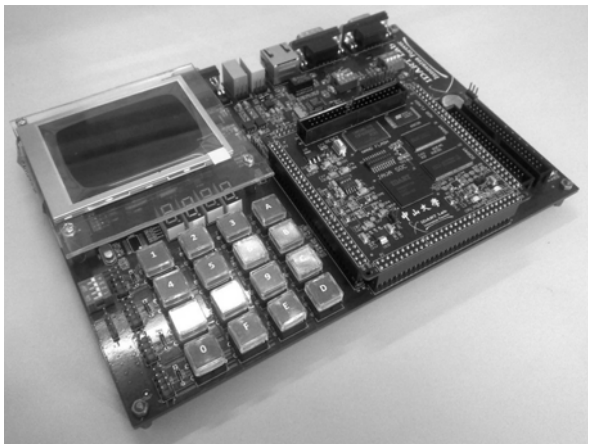


Figure 5. Verification platform of JM8BC013A

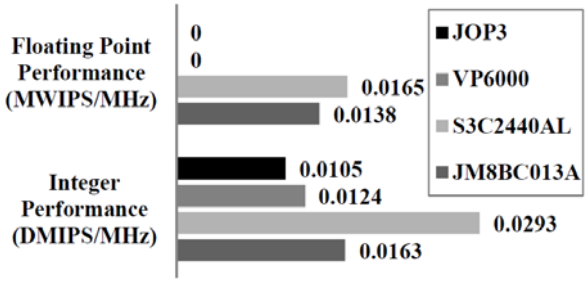


Figure 6. Comparison of Java performance

B. *Verification Platform*

A hardware verification platform is designed to verify the final SoC. The platform consists of a core board and a mother board. The core board contains the minimum system of the Java SoC, whereas the mother board has other modules to verify the peripheral interfaces of the chip. The verification platform is shown in Fig. 5.

IV. PERFORMANCE

The performance of a processor can be evaluated by running Dhrystone and Whetstone benchmarks. Since the operating frequency of a processor has a great impact on its benchmark score, the units DMIPS/MHz and MWIPS/MHz are introduced.

The integer performance score of proposed Java SoC is 4.076 DMIPS when running a Java version Dhrystone benchmark at 250MHz. Therefore, the Dhrystone benchmark score of JM8BC013A is 0.0163 DMIPS/MHz. As the floating-point performance, the Whetstone benchmark score is 0.0138 MWIPS/MHz.

The performance of JM8BC013A is compared with JOP3, VP6000 and S3C2440AL by running the same Java benchmarks. It is worth mentioning here that S3C2440AL is running a JVM called CrE-Me on WinCe5.0 operating system. The comparison results are shown in Fig. 6.

It can be seen that the integer performance of JM8BC013A is 31.5% higher than VP6000's and 55.2% higher than JOP3's. JM8BC013A supports floating-point operations whereas VP6000 and JOP3 do not. JM8BC013A achieves about 83.7% of the Java floating-point performance of S3C2440AL.

V. CONCLUSIONS

A new architecture of a Java SoC has been proposed for real-time embedded systems. It is based on a Dual-AHB AMBA bus, with a Java core, an FPU, a DMA, and some other peripheral controllers integrated. The proposed design has been implemented in VLSI, and the chip has been successfully taped out and packaged. Both integer and floating-point performance have been measured by running Java benchmarks. The benchmark scores of JM8BC013A show a better performance in comparison with previous design VP6000 and JOP3.

For future research, more optimized Java core in terms of the number of pipeline and multicore architecture are expected. Also, some other common functional modules, such as graphics acceleration unit, are expected to be integrated into the SoC.

REFERENCES

[1] H. McGhan and M. O'Connor, "PicoJava: a direct execution engine for Java bytecode," Computer, vol.31, no.10, Oct. 1998, pp. 22-30, doi: 10.1109/2.722273.

[2] M. Schoberl, "JOP: A Java Optimized Processor for Embedded Real-Time Systems," Phd dissertation, http://www.jopdesign.com, 2005.

[3] ARM Ltd Std., "Jazelle Technology for Java Application," 2001.

[4] NAZOMI Communications Inc., "JA108˜ Multimedia Application Processor (Product Brief)," 2003.

[5] Aurora VLSI Inc., "AU-J2000: Super High Performance Java Processor Core (Data Sheet)," 2000.

[6] Z. Liang, J. Plosila and K.Sere, "Asynchronous Java accelerator for embedded Java virtual machine," Proc, IEEE Symp. Emerging Technologies: Frontiers of Mobile and Wireless Communication, IEEE Press, vol. 1, MayJune 2004, pp. 253- 256, doi: 10.1109/CASSET.2004.1322968.

[7] W. W. Dai, H. N. Wang and H. Z. Tan, "AMBA Bus-Based Java System-on-Chip," Circuits and Systems for Communications (ICCSC 2008), May 2008, pp. 604607, doi: 10.1109/ICCSC.2008.134.

[8] ARM Ltd Std., "AMBA™ Specification Rev2.0," May 1999.

[9] Z. R. Chen and H. Z. Tan, "Logic Structure of Programmable Instructions," Journal of Electronics (CHINA), vol. 26 Sep. 2009, pp. 711714, doi: 10.1007/s11767-009-0022-6.

[10] Z. R. Chen and H. Z. Tan, "Dynamic instruction set load-in method for Java SoC," IP 08 Conference, 2008, Available: http://www.design-reuse.com/articles/20091/java-processor-soc.html.