Compact FPU Design and Embedding in a Ubiquitous Processor for Multimedia Performance Enhancement

Masa-aki Fukase [1], Non-member and **Tomoaki Sato** [2], Member

ABSTRACT

The key to protect huge amount of multimedia data in ubiquitous networks is to introduce safety aware high-performed single VLSI processor systems embedded with cipher process. Thus, we exploited the architecture of a hardware cryptography-embedded multimedia mobile processor named HCgorilla by sophisticatedly unifying up-to-date processor techniques. Although it was provided with carefully selected Java bytecodes and cipher codes, FP (floating point) expression was omitted due to the restriction of hardware resource. Considering recent trend of embedded applications like voice recognition, 3D graphics, and image/vision processing, FP hardware is crucial for further enhancing HCgorillafs Java functions. We focus in this article the development of a compact FPU (Floating point number Processing Unit). A compact FP format specific for HCgorilla is IEEE 754 compatible except the bit width representation of FP data. Prioritizing the latency of FPU, it has only 5 stages. The compact FPU is built in HCgorilla by adding 16 FP arithmetic codes and improving the decode stage of the previous HCgorilla. By using a 0.18-μm standard cell CMOS technology supported by VDEC, we have so far accomplished the logic synthesis and behavior simulation. The 400MHz of clock frequency is justified from delay analysis.

Keywords: Processor, ubiquitous, multimedia, floating point, performance

1. INTRODUCTION

Ever growing ubiquitous environment makes overall demands for user friendly functions, mobility, power consciousness, real time response, and secureness. Although sophisticated functions have been achieved by embedded software, such approach will surely contradict power conscious design. The massive quantity of multimedia information is very difficult for regular techniques to satisfy the overall demands. Thus, drastic improvement is required for embedded system to achieve really promising ubiquitous devices. With respect to this view point, it is expedient to develop a single chip VLSI processor [1-3].

We have so far developed multimedia mobile processor named gorilla [4], hardware cryptography-embedded processor named RAP (random addressing-accelerated processor) [5-6], and a hardware cryptography-embedded multimedia mobile processor named HCgorilla [7]. HCgorilla has potential to cover almost issues described above. It has the parallelism of multicore and multiple pipes to achieve power conscious high-speed performance. Java compatible media pipes are user friendly and cipher pipes execute SIMD (single instruction stream multiple data stream) mode cipher coeds to do streaming cipher.

The stream cipher by HCgorilla is a common key process called RAC (random number-addressing cryptography) [8]. RAC is neither almighty nor universal, and as such, is neither one and only nor exclusive computer security technique. However, RAC is a practical cipher that has ability to keep the temporary security of ubiquitous devices without permanent network infrastructure. One of the most characteristic aspects of RAC is the ad-hoc cipher of ubiquitous devices. The benefit of RAC is due to the direct connection of a built-in random number generator's output and the access line of a data cache. This makes the transfer between a register file and the data cache at random. Thus, HCgorilla encrypts a plaintext block without any execution. This is quite different from other usual ciphers. For example, AES (Advanced Encryption Standard) uses arithmetic, logic, and functional operations.

We applied HCgorilla to image cryptography [9]. The cryptographic streaming of multimedia data was further described in detail [10]. Then, the cryptographic streaming or ubiquitous cryptography, RAC was exploited in cooperation with the continuous improvement of HCgorilla and the development of its software support [8].

As for the Java capability, HCgorilla has been so far provided with carefully selected 61 Java bytecodes. These types are memory and stack access, data type conversion, integer arithmetic and logic operation, branch and jump. Although JVM (Java vir-

Manuscript received on December 27, 2007 ; revised on June 9, 2008.

[1] The author is with Graduate School of Science and Technology, Hirosaki University, Hirosaki 036-8561, Japan Tel.+81-172-39-3630, Fax:+81-172-39-3645, E-mail: slfuka@eit.hirosaki-u.ac.jp.

[2] The author is with C&C Systems Center, Hirosaki University, Hirosaki 036-8561, Japan Tel.+81-172-39-3723, Fax:+81-172-39-3722, E-mail: tsato@cc.hirosaki-u.ac.jp.

tual machine) is provided with FP (floating point) expression, it was omitted in developing HCgorilla. This is mainly due to the restriction of chip area and available circuit scale. Generally, FP hardware occupies large area and consumes much power.

However, FP expressions are crucial for multimedia applications such as voice recognition, 3D graphics, and image/vision processing. Considering an increasing number of embedded applications like cellar phone and PDA, the hardware implementation of FP operations is a matter of urgent subject and crucial. Designing consideration of FPU (floating point number processing unit) for the ubiquitous processor is contemporary to the inherent Java media specification. The lack of FP arithmetic functions deteriorates the utilization of HCgorilla.

We have planed to join HCgorilla with our several works on FPU for multimedia mobile computing [11]. In this article, we describe firstly a FP format specific for HCgorilla. It is IEEE 754 compatible except the bit width representation of FP data [12]. The dominant factor of power and precision is the bit width representation of FP data. Examining the necessary resolution and range of FP arithmetic used for embedded applications, it was pointed out that 9 mantissa bits and 6 to 7 exponent bits are sufficient [13]. Thus, the bit width is fixed to 16 according to the design principle of power conscious high speed with sufficient precision and universality.

We synthesize a compact FPU by using a 0.18-μm standard cell CMOS technology file and CAD tools. The FPU prioritizes latency, and is made as short as possible by optimizing the structure of FP arithmetic pipeline. The resultant 5-stage FPU works at 400MHz according to delay analysis. Then, the FPU is built in HCgorilla by adding 16 FP arithmetic codes and improving the decode stage of the previous HCgorilla. We have so far accomplished the logic synthesis and behavior simulation of HCgorilla.

2. FLOATING POINT NUMBER PROCESSING UNIT

Since our attention is ubiquitous or mobile computing, PC level performance is not always required. Thus, the design principle of the FPU specific for HCgorilla is low power with sufficient precision and universality. The universality is inevitable for ubiquitous environment.

2.1 FP Format

A compact FP format with sufficient precision is exploited so that it may be really useful for ubiquitous devices, especially for HCgorilla. FP representation and computation are inevitable for many mobile/portable electronics devices because they often manipulate human-sensory data such as speech or video imagery digitized by FP formats. Then, human-sensory data is acquired at relatively low bit resolutions, e.g., 4-10 bits of precision. Nevertheless, processing FP programs are easily coded by using popular high resolution FP formats with more bitwidth. Since FP hardwares are very power hungry, it is not always reasonable for low-power electronics devices to use FP standards like IEEE 754 that most desktop microprocessors support by using more than 32bits. Yet, the universality of FP expression is filled with IEEE standard. The IEEE standard governs binary FP arithmetic. It is adopted by JVM. Fig. 1 shows the most universal IEEE 754 standard 32-bit floating point number format.

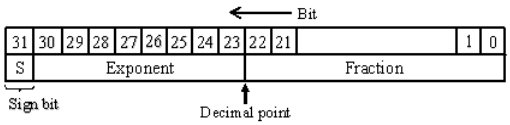

Fig.1: *IEEE 754 Standard 32-bitFfloating Point Number Format.*

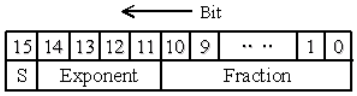

Fig.2: *HCgorilla 16-bit Floating Point Number Format.*

Reducing FP power consumption was explored by minimizing the bitwidth representation [13]. This was justified by the analysis of several signal processing programs such as Alvinn, Bench22, Fast DCT, PCASYS, Sphinx in running multimedia FP data. The relation of program accuracy vs. bitwidth shows 9 mantissa bits and 6 or 7 exponent bits maintain 90% recognition accuracy. This is sufficient precision to keep the reliability of human life. The energy vs. bitwidth plots show that power consumption increases linearly with the operand bitwidth. Thus, up to 66% reduction in power consumption can be achieved in a FP hardware unit by the bitwidth reduction scheme.

With respect to the viewpoint that 16-bitwidth FP expression is sufficient, it is reasonable that HCgorilla has been provided with 16 bits of word width. Thus, a FPU for HCgorilla is also designed to have 16-bit resolution. Fig. 2 shows the FP format of HCgorilla. The mantissa is assigned 11 bits of precision in order to achieve sufficient precision, while the exponent is assigned 4 bits with tolerance for the dynamic range. Table 1 summarizes the interpretation of the format shown in Fig. 2.

2.2 Hardware Structure

In view of mobility and power, the hardware structure of the HCgorillafs FPU is designed as compact as possible. In order to detect invalid computing such

Table 1: *The Set of Interpretation for the HCgorilla Format*

EX, FR	Numerical value	Comments
0 < EX < 15	$(-1)^S \times (1+FR) \times 2^{EX-7}$	Normalized number
EX = 0, FR ≠ 0	$(-1)^S \times FR \times 2^{EX-6}$	Denormalized number
EX = 0, FR = 0	$(-1)^S \times 0$	±0 (signed 0)
EX = 15, FR = 0	$(-1)^S \times \infty$	±infinity
EX = 15, FR ≠ 0	NaN	Not a number

as $0 \times \infty$, $\infty - \infty$, the compatibility with IEEE 754 is important. Thus, following units are used for the FPU.

a. A unit for checking exceptional conditions that are the cases where an exponent is 0000, 1111, and a mantissa is all clear. These cases correspond to over flow and under flow as shown in Table 1.

b. A unit providing the function of gradual underflow shown in Fig. 3.

These units prevent interruption to be caused by invalid computing. Reliable features due to these units are indispensable for the normal function of mobile devices.

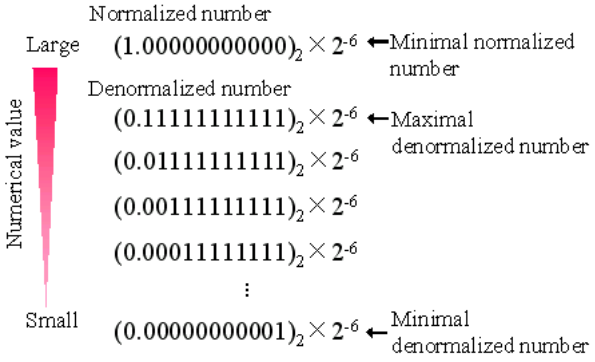

Fig.3: *Gradual Underflow.*

In the hardware implementation of the FP format shown in Fig. 2, we prioritize latency. Since it is a fundamental issue imposed on the FPU, the pipeline structure s optimized. This is achieved by decreasing the pipeline degree and maximum delays.

Fig. 4 shows the pipelined structure of FPU. EXP stage calculates exponents. SHIFT stage has the unit for exceptional process and alignment. EXE stage calculates mantissa. NORMAL stage normalizes the result of EXE and rounded at RND stage. In order to relieve the inherent issue of FPU or to tune the long latency as short as possible, we make following devices.

a. Speculative execution of mantissa calculation: If we did serial execution for comparison, addition, or subtraction in calculating mantissa, EXE stage might take several clocks. Thus, it is parallelized. Since the parallelized EXE stage does redundant calculations, the justice of them is judged and one of them is selected. Thus, the parallel execution is speculative. The parallelization saves the distance of logic depth, but obviously wastes the longitudinal direction. Thus, the tradeoff between delay and area or power is important.

b. Optimizing multiply: By decomposing into partial products, the 12-bit×12-bit multiply takes only 2 clocks at SHIFT and EXE stages.

c. Integration of adjacent units with smaller delays into one stage: Actually, the ROUND unit and the NORMALIZE2 unit are joined at RND stage.

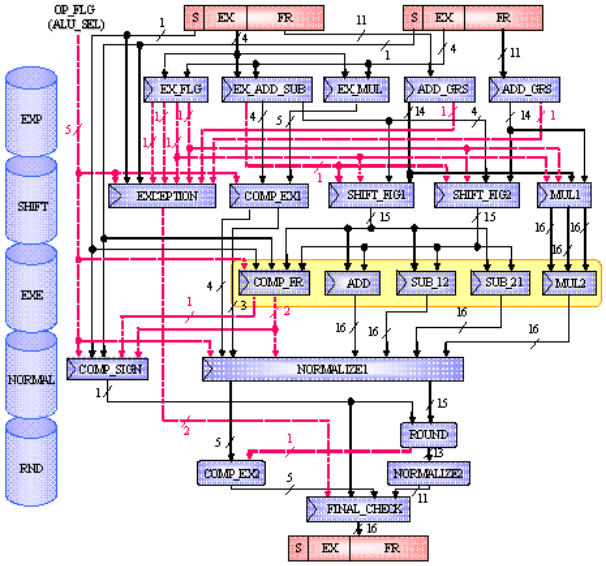

Fig.4: *Organization of the FPU.*

2.3 Logic synthesis

The FPU structure shown in Fig. 4 is synthesized by using a 0.18-μm standard cell CMOS technology file supported by VDEC. Design environments are basically Synopsys and Cadence tools. The designed FPU works at 400MHz as is justified from Table 2 and Fig. 5. Table 2 summarizes the delay analysis of the FPU. The critical path exists in the NORMALIZE1 unit. From this, we derive Fig. 5 that overviews the timing condition of the FPU and the previous HCgorilla. The condition allows normal operation. It is clear from this that the FPU covers the operation possible region of HCgorilla. Thus, we fix the clock frequency at 400MHz.

Fig. 6 shows the simulation of the FPU working at 400MHz. The simulation result exactly verifies that 5-stage pipelining, multiplication, exceptional conditions, and gradual underflow are regularly treated.

Table 2: *Delay analysis of the FPU*

Stages	Units	Max Delay [ns]
EX	ADD_GRS1	0.91
	ADD_GRS2	0.91
	EX_FLG	0.54
	EX_ADD_SUB	1.89
	EX_MUL	1.29
SHIFT	EXCEPTION	2.15
	COMP_EX1	1.45
	SHIFT_FIG1	2.30
	SHIFT_FIG2	2.27
	MUL1	**2.31**
EXE	COMP_FR	1.89
	ADD	2.31
	SUB_12	2.29
	SUB_21	2.32
	MUL2	2.32
NORMAL	COMP_SIGN	1.58
	NORMALIZE1	**2.37**
RND	ROUND	0.80
	NORMALIZE2	0.19
	COMP_EX2	0.78
	FINAL_CHECK	0.48

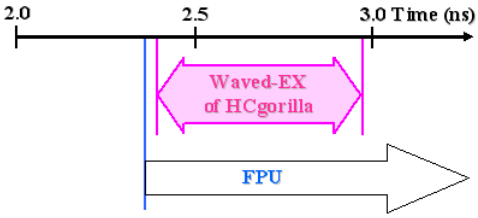

Fig.5: *Clock Cycle of the FPU.*

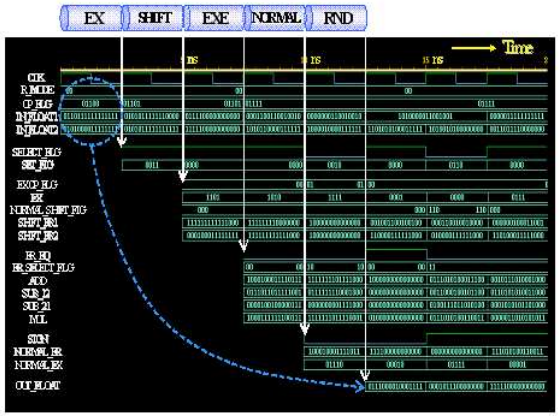

Fig.6: *Simulation of the the FPU.*

3. HCGORILLA

The new version of HCgorilla that builds in the 5-stage FPU is named HCgorilla.4.

3.1 Opcode Set

We have carefully selected 16 FP arithmetic codes from 26 float type Java bytecodes in order to achieve a compact opcode set. The remaining 10 codes will be unfolded by a software support like API (application program interface). Although sophisticated demands for multimedia processing tend to complicate the opcode set and increase its size, compactness is also important in order to lighten the burden on mobile hardware. This is solved by simplifying the code format and reducing the size or the number of opcodes.

Table 3 shows the code set of HCgorilla.4 composed of 79 codes. 16 FP arithmetic codes are added to previous opcode set composed of 63 codes. The decode unit of HCgorilla.4 has been improved so that it covers the control signals of the FPU too.

The reason why we do not say instruction set but code set is because the concept of the machine instruction is vague as shown in Table 4. While a so-called instruction is formed by an opcode and an operand, they are grammatically independent in case of HCgorilla and JVM.

Table 3: *Code set of HCgorilla.4*

Mnemonic	Binary	Dependent operand (byte)	iconst_0	0x03	0	astore_1	0x4c	0	fconst_0	0x0b	0
			iconst_1	0x04	0	astore_2	0x4d	0	fconst_1	0x0c	0
			iconst_2	0x05	0	astore_3	0x4e	0	fconst_2	0x0d	0
nsw	0xf0	0	iconst_3	0x06	0	pop	0x57	0	fload	0x17	1
nlw	0xf1	0	iconst_4	0x07	0	pop2	0x58	0	fload_0	0x22	0
nop	0x00	0	iconst_5	0x08	0	dup	0x59	0	fload_1	0x23	0
bipush	0x10	1	sipush	0x11	2	dup_x1	0x5a	0	fload_2	0x24	0
iload	0x15	1	aload	0x19	1	dup_x2	0x5b	0	fload_3	0x25	0
istore	0x36	1	iload_0	0x1a	0	dup2	0x5c	0	fstore	0x38	1
iadd	0x60	0	iload_1	0x1b	0	dup2_x1	0x5d	0	fstore_0	0x43	0
isub	0x64	0	iload_2	0x1c	0	dup_x2	0x5e	0	fstore_1	0x44	0
ineg	0x74	0	iload_3	0x1d	0	imul	0x68	0	fstore_2	0x45	0
ishl	0x78	0	aload_0	0x2a	0	irushr	0x7c	0	fstore_3	0x46	0
ishr	0x7a	0	aload_1	0x2b	0	ifilt	0x9b	2	fadd	0x62	0
iand	0x7e	0	aload_2	0x2c	0	ifge	0x9c	2	fsub	0x66	0
ior	0x80	0	aload_3	0x2d	0	ifgt	0x9d	2	fmul	0x6a	0
ixor	0x82	0	astore	0x3a	1	ifle	0x9e	2			
ifeq	0x99	2	istore_0	0x3b	0	if_icmpeq	0x9f	2			
ifne	0x9a	2	istore_1	0x3c	0	if_icmpne	0xa0	2			
if_icmplt	0xa1	2	istore_2	0x3d	0	if_icmpge	0xa2	2			
goto	0xa7	2	istore_3	0x3e	0	if_icmpgt	0xa3	2			
iconst_ml	0x02	0	astore_0	0x4b	0	if_icmple	0xa4	2			

Table 4: *Code of HCgorilla VS. JVM*

Architecture	Assembler code		Executable code
HCgorilla	Opcode	Media code (Java byte code)	1 byte
		Cipher code	
	Operand		1 byte
JVM	Java byte code		1 byte
	Operand		0-8 bytes

3.2 Architecture

Fig. 7 shows the structure of HCgorilla.4. It has double cores, each of which has 5 pipes. Java compatible media pipes are 2 integer pipes and 2 FP number pipes. The remaining is a cipher pipe. The integer pipe has 2-waved execute stage, while FPU takes 5-clock latency. Thus, the scheduling is necessary for opcodes corresponding to these media pipes.

The parallelism of multiple pipes is LIW (long instruction word). LIW is not so broad like VLIW (very

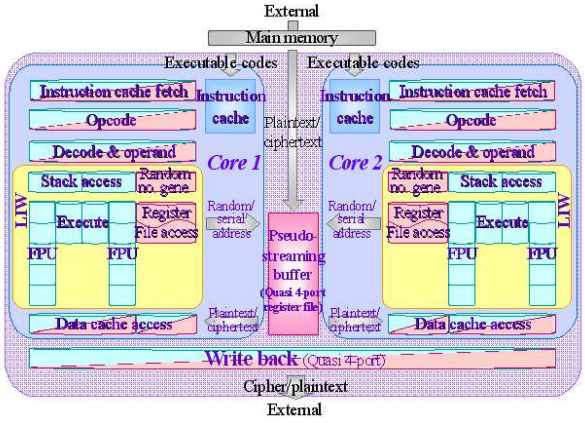

Fig.7: *Structure of HCgorilla.4.*

long instruction word). Yet, LIW is effective to practically enhance multimedia communication. In order to cover LIW in conjunction with Java native codes, microprogramming technique is useful. Then, the key technique of the hardware parallelism is superscalar-like IFU (instruction fetch unit). This is a wired logic detecting the length of each instruction and distributing it to appropriate pipes. TLP (thread level parallelism) and ILP (instruction level parallelism) are on the back of instruction folding by API and LIW compiler. Table 5 summarizes architectures and processor derivatives related to HCgorilla.

Table 5: *Processors Related to HCgorilla*

Name	ISA		Parallelism				Hardware cipher	Chip	Process	Clock (MHz)	Current status
	No. of instructions	Format	No. of cores	Pipelining							
				Regular		Waved					
				Deg.	Number						
gorilla.1	16	JVM	2	8	2 media pipes	2-wave EX	Not available	gorilla035	0.35μm	240	4.9mm chip
gorilla.2				7		Not available		gorilla035v2		200	Synthesis
RAP	17	RISC	1	5	1 cipher pipe		SISD	Not available	FPGA	45	FPGA
HCgorilla.1	18	JVM	2	8	2 cipher-embedded media pipes			HCgorilla035	0.35μm	150	4.9mm chip
HCgorilla.2	63			7	3 (2 media pipes and a cipher pipe)	2-wave EX	SIMD	HCgorilla018	0.18μm	400	2.8mm chip
								HCgorilla018v2			Synthesis
								HCgorilla018v3			5.9mm chip
HCgorilla.4	79				5 (4 media pipes and a cipher pipe)						7.5mm synthesis

3.3 Simulation of HCgorilla.4

A test bench is prepared for the behavioral verification of HCgorilla.4. The test bench simulates parallel execution in full of media pipes and cipher pipes as is illustrated in Fig. 8. Here, we show the behavior of the one of double cores for the sake of illustration. Core 1 simultaneously executes the SIMD mode random store code rsw and the two FP arithmetic operations (fmal, fadd). The encryption within core 1 works for 32 1-byte graduation elements stored in the register file. On the other hand, the arithmetic operations by using Java instructions work for stacks.

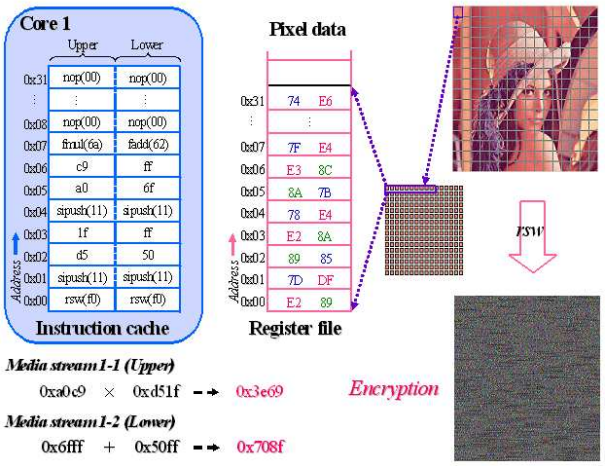

Fig.8: *Test program.*

Fig. 9 shows a result of HCgorilla.4 simulation in running the test bench. The pipelined structure attached in the above corresponds to 400-MHz clock. At $t < 0$, FP data are already pushed in the stacks of media pipes. Similarly, pixel data are buffered in the register file. The simulation result exactly verifies the parallel execution of the SIMD mode encryption for the FP data by the cipher code rsw and the floating point arithmetic for the pixel data by $fmul$ and $fadd$.

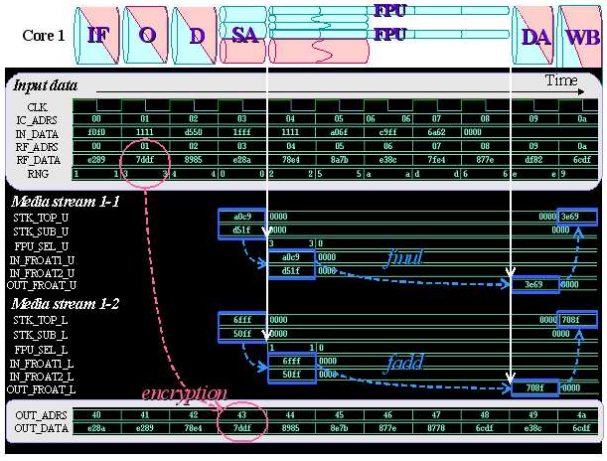

Fig.9: *Simulation of HCgorilla.4.*

Fig. 10 shows one of most typical applications of HCgorilla that simultaneously handle audio, text, image, moving picture, etc. The secret of these data is guaranteed by RAC. The HCgorilla-embedded cellar phone makes the most of HCgorillafs ad-hoc ability. The bidirectional capability provided by the double cores is preferable to many of ubiquitous applications, especially communication applications. In the bidirectional real time communication, each of two cores is dedicated to unidirectional data flow. High

efficiency due to utmost parallelism achieved by the double core and six pipelines is very suited to time-consuming ubiquitous applications.

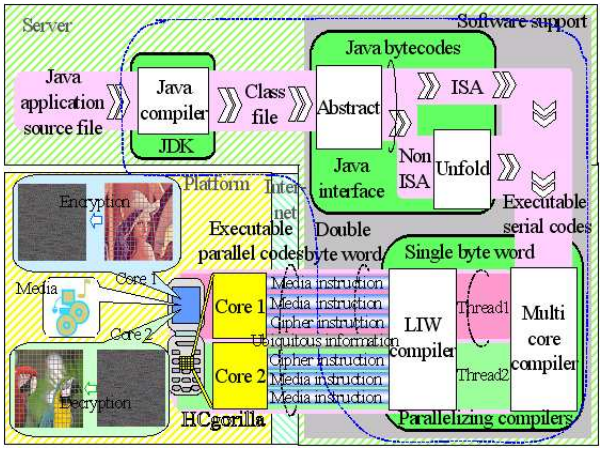

Fig.10: *Application of HCgorilla.*

4. CONCLUDING REMARKS

We have exploited a 16-bit FP format specific for HCgorilla. It is IEEE 754 compatible except the bit width representation of FP data. This is fixed to achieve a compact FPU with sufficient precision. The compact FPU has been designed so that the latency is as short as possible by optimizing the FP arithmetic pipeline structure. According to delay analysis by using a 0.18-μm CMOS standard cell technology file supported by VDEC, the resultant 5-stage FPU works at 400MHz. Then, the FPU is built in HCgorilla.4 by adding 16 arithmetic codes carefully selected from Java bytecodes, and improving the decode stage of the previous HCgorilla. We have so far accomplished the logic synthesis and behavior simulation of HCgorilla.4.

The next step of our study is as follows.

a. The optimization of chip design. It is vital for the ubiquitous performance of HCgorilla, because FP multipliers are generally expensive components in a processorfs power budget.

b. Chip implementation.

c. Chip evaluation like the tradeoff between delay, area, and power of the speculative FPU.

d. Development of software supports in running HCgorilla: Scheduling for integer and FP number operations may be covered by a LIW compiler [7] or multifunctional pipelining. Since multifunctional wave-pipelines are released from awkward instruction scheduling without the slowdown of clock speed, the arrangement makes it possible to realize wide-range dynamic ILP at a rate higher than regular superscalar architecture [14].

5. ACKNOWLEDGMENT

This work is supported in part by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc. and Cadence Design Systems, Inc.

References

[1] A. Jerraya, H. Tenhunen, and W. Wolf, "Multiprocessor Systems-on-Chips," *Computer Magazine,* Vol. 38, No. 7, pp. 36-40, Jul. 2005.

[2] L. Garber, "Company Makes First Multipurpose Programmable Mobile Chip," *Computer Magazine,* Vol. 38, No. 10, p24, Oct. 2005.

[3] M. Takahashi, "The development of a multimedia SoC for cellular phone applications," *The Journal of the IEICE,* Vol. 84, No. 11, pp.790-795, Nov. 2001.

[4] M. Fukase, Y. Nakamura, R. Akaoka, and T. Sato, "Development of a Multimedia Mobile Processor,h *Proc. of ISCIT 2004,* pp. 672-677, Oct. 2004.

[5] M. Fukase and T. Sato, "Power Conscious Endeavor in Processors to Speed Up Random Sampling," *Proc. of SCI2003,* Vol. V, pp. 111-116, Jul. 2003.

[6] M. Fukase, H. Takeda, R. Tenma, K. Noda, Y. Sato, R. Sato, and T. Sato, "Development of a Multimedia Stream Cipher Engine," *Proc. of ISPACS 2006,* pp. 562-565, Dec. 2006.

[7] M. Fukase, H. Takeda, and T. Sato, "Hardware/Software Co-Design of a Secure Ubiquitous System," *Computer Intelligence and Security,* Springer Berlin/Heidelberg, LNCS Vol. 4456/2007, pp. 385-395, Sept. 2007.

[8] M. Fukase and T. Sato, "Innovative Ubiquitous Cryptography and Sophisticated Implementation," *Proc. of ISCIT2006,* Oct. 2006.

[9] M. Fukase, Y. Sato, and T. Sato, "Design of a Hardware Security-Embedded Multimedia Mobile Processor," *Proc. of ISCIT2004,* pp. 362-367, Oct. 2004.

[10] M. Fukase, R. Akaoka, L. Lei, C. T. Shu, and T. Sato, "Hardware Cryptography for Ubiquitous Computing," *Proc. of ISCIT2005,* vol. 1, pp. 462-465, Oct. 2005.

[11] P. Khondkar, M. Fukase, and T. Nakamura, "Designing a Floating-Point Unit for High-Level Language Integrated Media Processor," *Proc. of ICCIT,* pp. 500-504, Dec. 2002.

[12] M. Fukase, K. Noda, H. Takeda, and T. Sato, "Multimedia Performance of a Ubiquitous Processor," *Proc. of ISCIT2007,* pp. 1464-1469, Oct. 2007.

[13] J. Y. F. Tong, D. Nagle, and R. A. Rutenbar "Reducing Power by Optimizing the Necessary Precision/Range of Floating-Point Arithmetic," *IEEE Trans. on VLSI Syst.,* Vol. 8, No. 3, pp. 273-286, Jun. 2000.

[14] M. Fukase and T. Sato, "Exploiting Design and Testing Methods of High-Speed Power Conscious Wave-Pipelines," *Proc. of NASA2007,* pp. 5.1.1-5.1.6, Jun. 2007.

Masa-aki Fukase received the B.S., M.S., and Dr. of Eng. Degrees in Electronics Engineering from Tohoku University in 1973, 1975, and 1978, respectively. He was Research staff member from 1978 to 1979 at The Semiconductor Research Institute of the Semiconductor Research Foundation. He was Assistant Professor from 1979 to 1991, and Associate Professor from 1991 to 1994 at the Integrated Circuits Engineering Laboratory of the Research Institute of Electrical Communication, Tohoku University. He has been Professor of computer engineering since 1995 at the Faculty of Science and Technology, Hirosaki University. He has been the Director of the Hirosaki University C&C Systems Center since 2004. He is also Representative of Hirosaki University R&DC of Next Generation IT Technologies since 2008. His current research activities are mainly concerned with the design, chip implementation, and application of power conscious highly performable VLSI processors.

Tomoaki Sato received the B.S. and M.S. degrees from Hirosaki University, Japan, in 1996 and 1998 respectively, and the Ph.D. degree from Tohoku University, Japan, in 2001. From 2001 to 2005, he was an Assistant Professor of Sapporo Gakuin University, Japan. Since 2005, he has been an Associate Professor of Hirosaki University. His research interests include VLSI Design, Computer Hardware, and Computer and Network Security.