Introducing the FR500 Embedded Microprocessor

Developed for digital consumer products, the FR500 microprocessor issues four instructions simultaneously and can be configured in a small-scale circuit, making it possible to implement a low-cost, high-performance system.

Atsuhiro Suga
Fujitsu Laboratories Limited

Kunihiko Matsunami
Fujitsu Limited

Because conventional RISC processors have insufficient processing power to support the continuing development of digital consumer products, we need a new high-performance processor for multimedia applications. Processing multimedia video images requires more than 10 times the currently available performance. At Fujitsu, we provide this higher performance in software to attain a high degree of flexibility.

We developed the FR500 microprocessor with a novel embedded VLIW (very long instruction word) architecture for use in such digital consumer products.[1] The FR500 is the first product in the FR-V line, Fujitsu's generic name for VLIW architecture microprocessors. The FR-V line offers the flexibility to develop new products optimized for a wide variety of digital consumer products.

Here, we describe the FR-V architecture, which includes our variable-length VLIW and instruction set architectures, speculative execution control, and conditional execution control. We also evaluate its performance.

An embedded VLIW processor

Performance requirements are soaring for embedded processors, whose demand in multimedia processing is rising now more than ever. Some DSP and media processors satisfy this by means of eight-way VLIW architectures.[2-6] However, for embedded processors, less code, low power, and small dies are mandatory. To satisfy these requirements, we developed an embedded processor using a four-way VLIW architecture. The architecture can be characterized as follows:

- parallel execution by variable-length VLIW,
- 32-bit-length instruction format with 64-entry register files,
- general CPU function in combination with a media-processing function for enhancing multimedia-processing ability, and
- instructions for global scheduling.

Instruction set

As Figure 1 shows, the instruction set consists mainly of integer, floating-point, and media instructions for global scheduling. Each instruction is 32-bits wide. The FR500 simultaneously executes four instructions, packed in a single 128-bit-wide VLIW packet. Floating-point instructions support single-precision floating-point operations, augmenting performance in media-oriented applications such as graphics and voice recognition. Some are single-instruction, multiple-data stream (SIMD) instructions for simultaneous operations.

Media instructions, an instruction set for multimedia extensions, are 16-bit fixed-point operations. They are also SIMD instructions that perform two or four operations simultaneously, augmenting digital processing perfor-

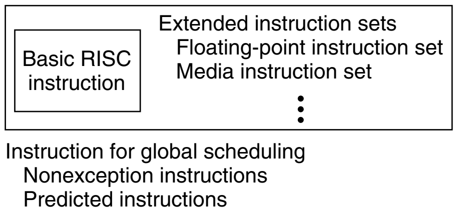
0272-1732/00/$10.00 © 2000 IEEE

mance such as voice and image processing. Moreover, to augment instruction-level parallelism (ILP), we prepared predicated and nonexception instructions for global scheduling.

Variable-length VLIW instructions

Each instruction provided in the FR-V architecture is executed in parallel using VLIW architecture methods. To specify the VLIW boundary, all instructions include a packing flag for suppressing NOP instructions.

The FR500 four-way VLIW architecture can issue up to four 32-bit-long instructions simultaneously. As shown in Figure 2, the FR500 can issue up to two integer, or I, instructions and up to two floating-point operation or media-processing instructions. Also, a branch instruction or logical operation instruction for the predicate register—that is, B instructions—can be issued to any of the four slots, contributing to improved simultaneously executable instructions.



Basic RISC instruction
Extended instruction sets
Floating-point instruction set
Media instruction set
Instruction for global scheduling
Nonexception instructions
Predicted instructions

Figure 1. FR-V instruction sets.

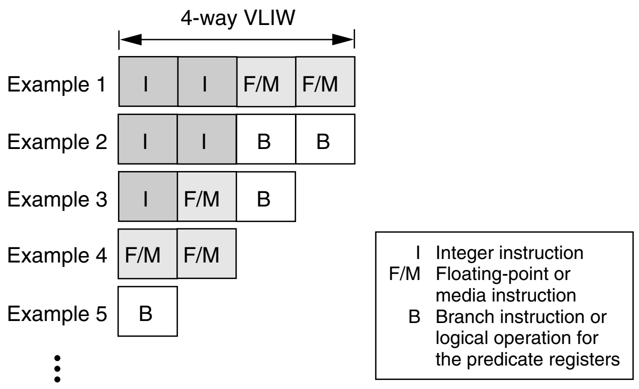


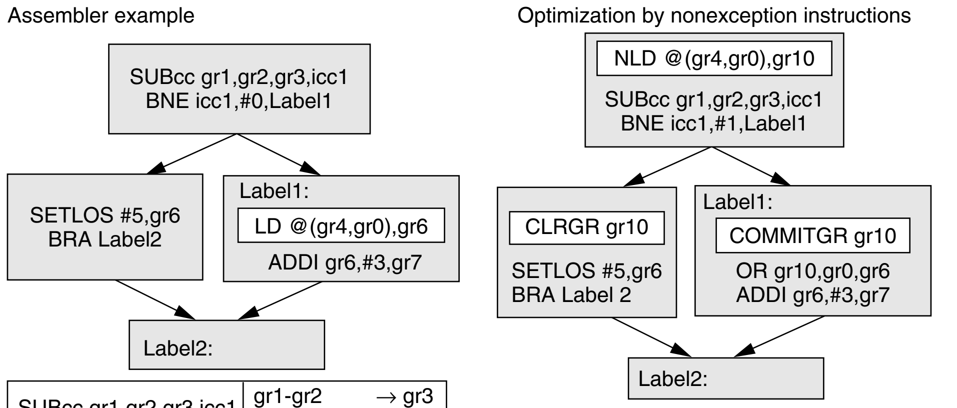
Figure 2. Variable-length VLIW instructions.

Speculative execution control

Speculative execution refers to the execution of an instruction before it is known that its execution is required.[7] To perform speculative execution control, the FR-V architecture provides nonexception instructions that suspend execution of exception processing when an exception condition is raised. The raised exception condition can be handled at a later time or canceled completely under software control. This makes it possible to move the exception-causing instruction across a branch instruction.

In the right program of Figure 3, the nonexception load (NLD) instruction to substitute the value for gr6 is set away from the ADDI instruction referring to gr6. This decreases the influence of the memory delay. An exception that is caused by an NLD instruction is suspended, a COMMIT instruction raises it, and a CLEAR instruction cancels it.

By using the NLD instruction, beyond a



SUBcc gr1,gr2,gr3,icc1	gr1-gr2 → gr3 Condition → icc1
BNE icc1,#0,Label1	conditional branch
SETLOS #5,gr6	5 → gr6
BRA Label2	branch always
LD @(gr4,gr0),gr6	@(gr4+0) → gr6
ADDI gr6,#3,gr7	Gr6+3 → gr7

NLD @(gr4,gr0),gr10	@ (gr4+0) → gr10 Suspend exception on gr10
CLRGR gr10	Clear exception on gr10
COMMITGR gr10	Raise exception on gr10
OR gr10,gr0,gr6	Gr10 or 0 → gr10

Figure 3. Speculative execution control example. GR: general-purpose register; icc: conditional code register.

branch instruction or a loop, the FR500 compiler can control speculative execution. This optimization improves the ILP and reduces memory delays.

Conditional execution control

Predicated execution instructions are useful in improving the ILP.[8] However, in cases where conditions are nested within nested IF clauses, recent predicated execution mechanisms require many predicate registers. In our architecture, a predicated instruction is executed when, and only when, one of three conditions—true, false, or undefined—stored in the predicate register, is equivalent to either the true or the false specified in the instruction field.

Furthermore, the FR500 supports two logical operations, AND and ANDN, between the predicate registers. Figure 4 shows these operations along with their truth tables. Figure 5 shows sample coding. By accommodating these operations, we observed a 30% code size reduction in the sample coding.

As a part of our chip evaluation, we observed the utility of predicated instructions using the ADPCM program. (Adaptive Differential Pulse Code Modulation is one of the simplest and oldest forms of audio coding.) In comparing the overall execution time of the program with and without the predicated instructions, we found a 1.7 performance improvement in terms of execution time.

FR500 hardware

Figure 6 shows the block diagram of our chip. The execution units of the floating-point and media units (the FM unit in the figure) have two and four arithmetic units respectively; each consists of two-stage pipelines. Thus there are a total of four floating-point execution units and eight media execution units. The GR and FR register files are register sets for integer and floating-point/media operations. Both are 32-bits wide with a depth of 64. As shown in Figure 7, the peak performance at 266 MHz is 1,064 Mflops and 4,256 MOPS, which is four times the frequency.

The cache unit executes two 8-byte-long load instructions simultaneously. The non-blocking technique reduces miss penalties. The data cache also has write-through and copy-back modes that can be selected by the user to optimize the data cache according to the application. In addition, the data cache has a memory mode, enabling it to function as internal main memory and hiding the external transfer latency via external DMA.

Evaluation

To verify the parallelization ability of the

andcc		2nd arg		
		T	F	U
1st arg	T	T	F	U
	F	U	U	U
	U	U	U	U

andcc		2nd arg		
		T	F	U
1st arg	T	U	U	U
	F	T	F	U
	U	U	U	U

Figure 4. Logical operation between predicate registers. T = true; F = false; U = undefined.

Sample code

```
IF    ( A=B  )
  IF (C=D)
              Z=X + Y
  } else
              Z=X − Y
     }
}   else {
  IF (E=F)    Z=X*Y
              } else
               Z= X/Y
     }
  }
}
```

Conventional predicate code

```
subcc A,B,cr0
subcc C,D cr1
subcc E,F cr2
cicc EQ cr0
cicc EQ cr2
cicc EQ cr1
notcr  cr0 cr3
notcr  cr1 cr4
notcr  cr2 cr5
andcr  cr0 cr1 cr6       7 instructions
andcr  cr0 cr4 cr7       10 registers
andcr  cr3 cr2 cr8
andcr  cr3 cr5 cr9
cdiv X,Y,Z cr9 F
cmul X Y Z cr8 T
csub X Y Z cr7 F
cadd X Y Z cr6 T
```

3-value predicate code

```
subcc A,B,cr0
subcc C,D cr1
subcc E,F cr2
cicc EQ cr0
cicc EQ cr2
cicc EQ cr1
andcr  cr0 cr1            2 instructions
andncr  cr0 cr2           3 registers
cdiv X,Y,Z cr2  F
cmul X Y Z cr2 T
csub X Y Z cr1  F
cadd X Y Z cr1 T
```

Figure 5. Three-value predicate mechanism.

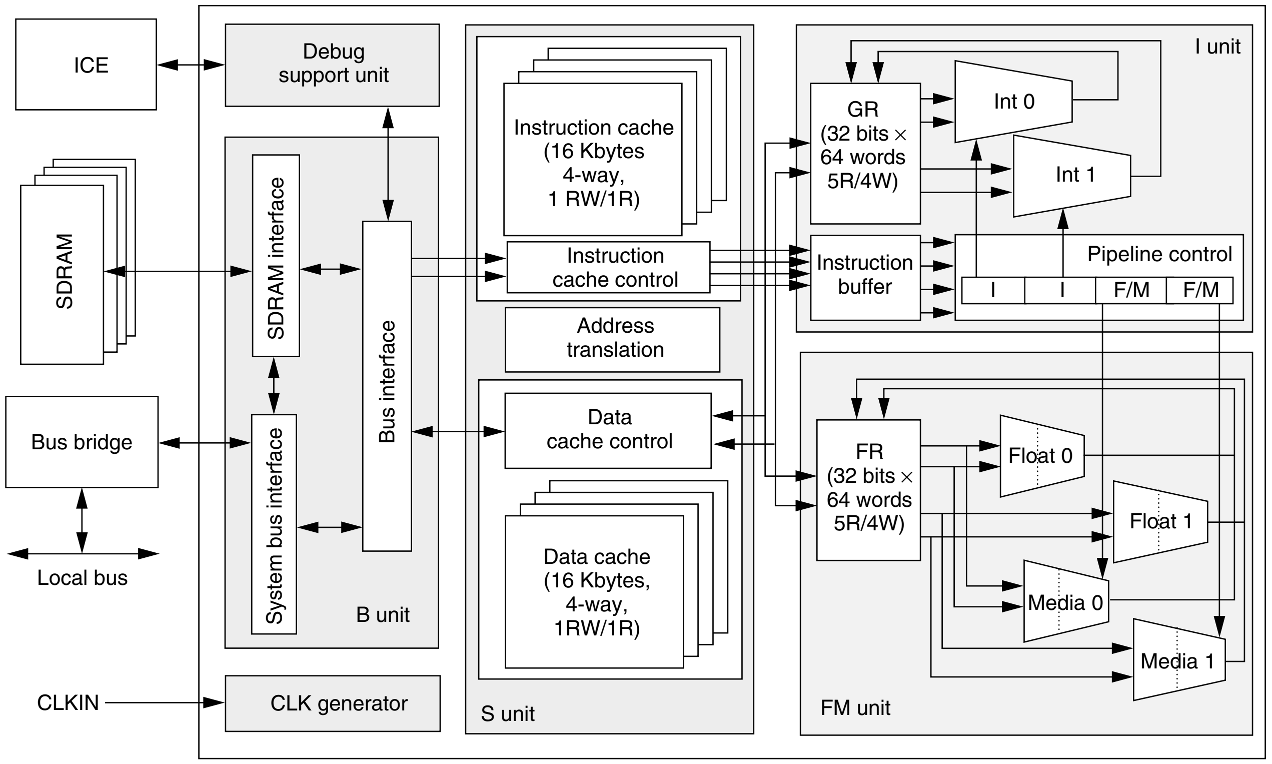


Figure 6. FR500 block diagram.

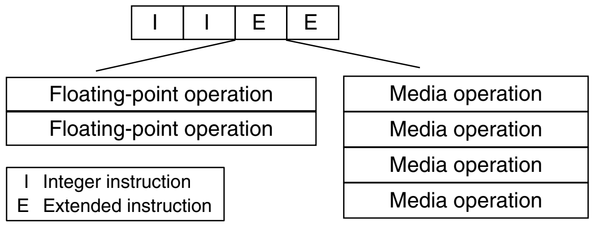


Figure 7. FR500 peak performance: 1,064 Mflops at 266 MHz and 4,256 MOPS at 266 MHz, respectively.

FR500 compiler, we evaluated the parallelism, the delays caused by register conflicts, and the execution cycle count in some benchmark programs by using our Cycle Accurate Simulator, which was developed as an engine of our simulator debugger. This simulator can report executed instructions, executed cycles, and penalty cycles.

Figure 8 (next page) shows the ILP change. We used four options to compare the optimization effects on three benchmark programs. Program 1 is a fast Fourier transform (FFT) program that is composed of only integer instructions. Program 2 is an FFT program composed with single-precision floating-point and integer instructions. Program 3 is the Dhrystone 1.1 program. Option 1 (no schedule) performs traditional and general optimizations but not parallelization. Option 2 (local schedule) schedules only instructions in basic blocks. Option 3 (trace schedule) schedules traces. Option 4 (trace schedule + predication) performs Option 3 and conditional execution control.

The execution results show that the following relationship is established: local schedule <trace schedule <trace schedule + predication. No effects of conditional execution control are seen in Program 2 because the program structure is simple; there is no pattern in which an instruction can be replaced with the predicated instruction.

In Program 1 (consisting of integer-only operations), the ILP is 1.75, meaning that nearly two instructions are always executed concurrently. In the FR500 when executing an integer-only-operation program, two I instructions and two B instructions can be executed concurrently. When B instructions are excluded, the ILP is 1.58. This means that instruc-

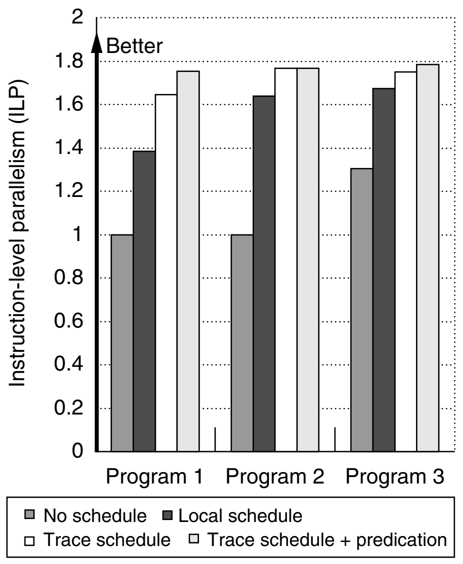


Figure 8. ILP change.

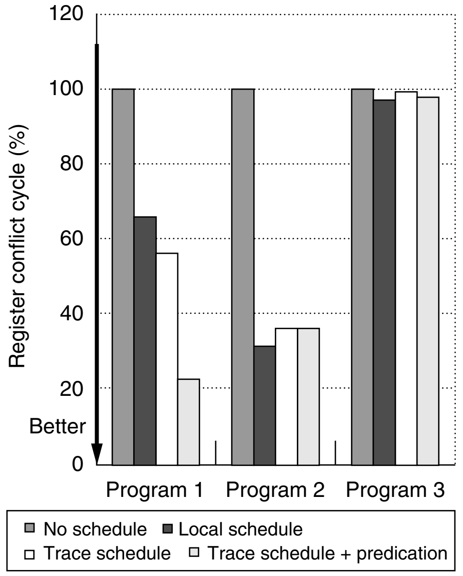


Figure 9. Delay cycle count change.

tions can be supplied to the I unit at 79% of capacity, which is a very high operating ratio.

Figure 9 shows the change in delays caused by register conflicts. In this figure, the delay cycle occurring at execution of each program is normalized with the no schedule value defined as 1. For no schedule, the same optimization as in other cases is performed, except that no scheduling is performed. Consequently, no schedule = 1 is used as an index of the scheduling capability. In Program 1, the effects of trace scheduling and conditional execution control eliminate 77.2% of the delays caused by register conflicts. Eliminating delays by scheduling improves pipeline throughput, which improves execution speed. See the "Trace scheduling" box .

Figure 10 shows the change in the execu-

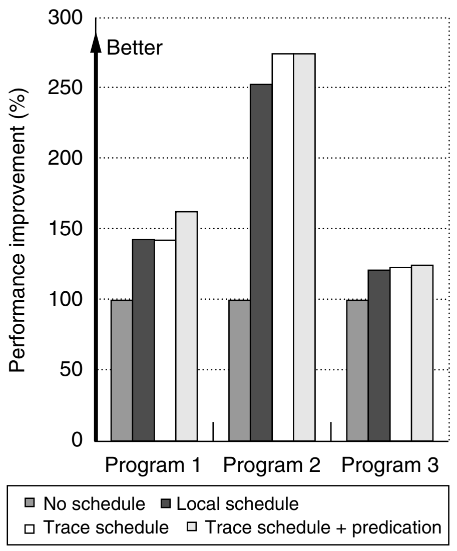


Figure 10. Execution cycle count change.

Trace scheduling

Very long instruction word architectures require fine-grained parallelism. By using trace-scheduling technology,[1] a compiler finds such instruction-level parallelism and generates code suitable to the VLIW CPU.

Since many conditional branches occur in a program, execution counts of each basic block differ. It focuses on the basic block with the highest execution count in the program. The trace scheduler traces basic blocks with a high branch probability and a high execution count to find the group of basic blocks (herein called trace) with the greatest possibility of being executed. Then scheduling is performed for instructions included in that trace.

A more wide-ranging instruction can be scheduled by scheduling traces composed of two or more basic blocks. This means that a greater count of ILPs can be found.

Reference

1. J.R. Ellis, *Bulldog: A Compiler for VLIW Architectures,* ISBN 0-262-05034-X, MIT Press, Cambridge, Mass., London, 1986.

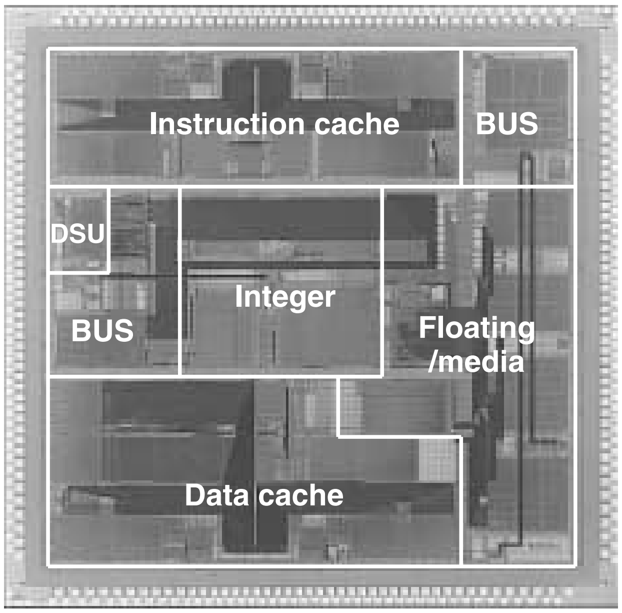


Figure 11. FR500 die photo.

Table 1. FR500 chip specifications.

Item	Description
Process technology	0.18-micron, 5-metal-layer CMOS
Issues	4 VLIW (2 integer, 2 floating, or 2 media)
Frequency	266 MHz
Peak performance	532 MIPS + 1,064 Mflops, 4,256 MOPS
Register file	GR: (5R/4W) 32 bits × 64 words
	FR: (5R/4W) 32 bits × 64 words
Caches	Instruction: (1RW/1R) 16 Kbyte, 4-way set associative
	Data: (1RW/1R) 16 Kbyte, 4-way set associative
Bus interfaces	SDRAM: 133 MHz, 1 Gbyte/s (maximum)
	System: 133 MHz, 1 Gbyte/s (maximum)
No. of transistors	Logic: 3.2 million, RAM; 3.5 million
	Total: 6.7 million
Chip size	7.5 mm × 7.5 mm
Package type	Plastic BGA352
Power dissipation	2.0 watts at 1.8 volts

tion cycle count. The execution cycle count in each option is 1, and the figure shows the approximate improvement in the cycle count compared to no schedule. Program 2 achieves 2.74 times the performance.

We designed the FR500 embedded general-purpose microprocessor using the VLIW architecture to provide a good balance of performance, flexibility, and power consumption. We fabricated the 0.18-micron, five-metal-layer CMOS processor on a 7.5-mm × 7.5-mm die. Figure 11 displays the FR500 die photo, and Table 1 lists the chip specifications.

According to our performance evaluation, our compiler improved performance. We plan to adopt a technique to give feedback on dynamically collected information (such as profile information and extracting traces with a high execution count) to improve the performance of programs using trace scheduling.

The importance of a good optimizing compiler for VLIW embedded processors cannot be stressed enough. The FR500 microprocessor is well suited to meet the demanding needs of digital consumer products. MICRO

Acknowledgments

We thank all FR-V project members and parties who were involved with the chip and software developments.

References

1. A. Suga et al., "A 4-Way VLIW Embedded Multimedia Processor," *2000 Digest of Tech. Papers Int'l Solid-State Circuits Conf. (ISSCC),* IEEE Press, Piscataway, N.J., Feb. 2000, pp. 240-241.
2. J. O'Donnell, "MAP1000A: A 5W, 230-MHz VLIW Mediaprocessor," Hot Chips Symp., Aug. 1999; contact the authors for copies.
3. J. Choquette, "Genesis Microprocessor," Hot Chips X Symp., Aug. 1998; contact the authors for copies.
4. G. Slavenburg et al., "TM-1300 High-Speed, Low-Cost, Enhanced PCI, VLIW Media Processor," Hot Chips Symp., Aug. 1999; contact the authors for copies.
5. O. Nishii et al., "A 200MHz 1.2W 1.4GFLOPS Microprocessor With Graphic Operation Unit," *ISSCC Digest of Tech. Papers,* IEEE Press, Feb. 1998, pp. 288-289.
6. D. Ditzel, "Transmeta's Crusoe: A Low-Power X86-Compatible Microprocessor Built With Software," *Proc. Cool Chips III Symp.,* Apr. 2000, Kikai-Shinko-Kaikan, Tokyo, pp. 1-30; contact the authors for copies.
7. W. Wen-mei et al., "The Superblock: An Effective Technique for VLIW and Superscalar Compilation," *J. Supercomputing,* Vol. 7, 1993, pp. 229-248.
8. S.A Mahlje et al., "A Comparison of Full and

Partial Predicated Execution Support for ILP Processors," *Proc. 22th Int'l Symp. Computer Architecture,* IEEE CS Press, Los Alamitos, Calif., June 1995, pp.138-150.

Atsuhiro Suga is a senior researcher in the System LSI Development Laboratories at Fujitsu Laboratories Limited in Kawasaki, Japan. His research interests include microprocessors and architectures for digital consumer products. Suga received ME degrees from Yokohama National University.

Kunihiko Matsunami is a senior engineer at Fujitsu Limited, where he develops compilers for embedded Gmicro family/FR family/FFMC family/FR-V family microprocessors. His current interests are in performance issues on C++ specification. Matsunami received a BE degree in information engineering from the University of the Ryukyus.

Direct questions about this article to Atsuhiro Suga, System LSI Development Laboratories, Fujitsu Laboratories Ltd. 1-1, Kamiodanaka 4-chome, Nakahara-ku, Kawasaki 211-8588, Japan; asuga@flab.fujitsu.co.jp.