SH-5: The 64-Bit SuperH Architecture

EMBODYING AN EMERGING PHILOSOPHY OF EMBEDDED-CORE DESIGN, THE LATEST SUPERH MICROPROCESSOR PROVIDES A PLATFORM FOR A WIDE RANGE OF MULTIMEDIA APPLICATIONS. ITS SIMD EXTENSIONS PROVIDE THE PARALLELISM REQUIRED FOR EFFICIENT EXECUTION OF THESE APPLICATIONS.

Prasenjit Biswas
Atsushi Hasegawa
Srinivas Mandaville
Hitachi Semiconductor (America)

Mark Debbage
Andy Sturges
STMicroelectronics

Fumio Arakawa
Yasuhiko Saito
Kunio Uchiyama
Hitachi Ltd.

A collaborative effort of Hitachi and STMicroelectronics, the SH-5 is the latest member of the SuperH microprocessor series. Its CPU core is the first implementation of a new instruction set architecture consisting of 32-bit instructions, 64-bit registers, SIMD (single-instruction, multiple-data) instructions for multimedia applications, and a compatibility mode supporting the 16-bit SuperH instruction set.

Embodying an emerging philosophy of embedded-core design, the SH-5 provides a platform for a wide range of applications: set-top cable boxes, digital TV, voice over IP (Internet telephony), network processing, PDAs (personal digital assistants), Internet appliances, in-car information systems, game machines, and so on. A single cost-effective, optimum design that will cater to the requirements of such a wide range of applications is not feasible. So the SH-5 core supports a carefully selected set of functions critical to meeting the performance, power, and code-size requirements of these applications. At the same time, it provides features that ease integration into a system on chip (SOC) that uses application-specific hardware modules to cater to specific requirements.

Overview

The 32-bit SuperH architecture has evolved over the past several years and is implemented in dozens of embedded microprocessor and microcontroller units representing the SH-1, SH-2, SH-3, SH-DSP, and SH-4 generations.[1,2] The 16-bit instruction set architecture's limited operation code space did not allow extension to meet the performance requirements of some multimedia applications. Similarly, the ISA could not support floating-point and DSP instructions in the same implementation because they share the same operation code space.

Code generated for the SuperH 16-bit instruction set has usually been significantly more compact than code for other RISC architectures with 32-bit instruction sets. Typically, RISC architectures with a 32-bit instruction length provide a shorter dynamic-path length, and consequently fewer instructions are necessary to implement an algorithm.[3] Code compactness and lower instruction-cache bandwidth requirements are important in many embedded applications, but in some of these applications, performance is the key and cannot be sacrificed for good code density.

Often, a small fraction of the code dominates an application's performance. A useful rule of thumb is a 90:10 ratio: 90% of the code runs only 10% of the time. If we compile the portion that runs only 10% of the time for high density, we get excellent density for 90% of the code. We can compile the remaining 10% with an instruction set that emphasizes performance, thus yielding excel-

0272-1732/00/$10.00 © 2000 IEEE

lent performance 90% of the time.

To achieve this optimization, as well as backward compatibility with the existing SuperH implementations, we designed the SH-5 with two operating modes: SHmedia and SHcompact.

SHmedia is a new mode we defined especially for the SH-5. Using a clean-slate definition allowed us to develop the architecture for the future without carrying baggage from the past. SHmedia's complete instruction set delivers high performance for integer operations, multimedia and DSP arithmetic operations, and floating-point operations. (See the sidebar for SHmedia's key features.)

SHcompact supports the user-mode instruction set of earlier SuperH RISC devices. It provides user-mode instruction compatibility with software written for previous SuperH generations. Programmers can also use SHcompact for new software to reduce the storage requirements of code that is not especially time critical but forms the bulk of an application.

A simple branch operation provides very fast switching between the two modes, allowing the compiler to provide the optimal balance between application performance and small overall code size. We defined the ABI (application binary interface) and register mapping between the two modes to eliminate the need for register copying from one mode to the other.

Key SHmedia features

- A 64-bit architecture with a large register file that can handle the performance requirements of media-rich applications and network processing. It also provides access to a much larger address space, which will be needed during the lifetime of this architecture.
- Simple 32-bit instruction encoding for fast decoding in high-speed implementations optimized to the demands of modern optimizing compilers and languages.
- A carefully chosen set of SIMD multimedia instructions. Because they operate on the standard register set, these instructions simplify associated software, allow mixing of standard instructions and SIMD instructions, and minimize implementation hardware.
- A removable IEEE-754 floating-point unit that supports high-performance 3D-geometry operations for graphics applications.
- Coexistence of powerful 3D-geometry-oriented floating-point instructions and SIMD multimedia instructions, facilitating simultaneous processing of streams of natural data (for decompression) and synthetic data (for 3D graphics) in future set-top boxes.
- A high-performance branch architecture optimized for both longer scalar pipelines and wide-issue superscalar implementations.
- Software cache management and data prefetching to maximize system bandwidth utilization.
- Future scalability for architecture features such as VLIW, simultaneous multithreading, and application-specific operations.
- Data parallelism available through the 3D-geometry floating-point instructions (parallel floating-point multiply-accumulate operations), supporting efficient DSP solutions for applications such as voice over IP.

SH-5 architecture

In today's applications, efficient support of 64-bit operations is becoming increasingly important. Standard C provides a 64-bit data type, called "long long," and Java mandates a 64-bit, "long" type. For accumulations or high-precision arithmetic, the extra headroom and precision of 64-bit arithmetic are very important. The use of 64-bit registers and arithmetic/logic operations over 64-bit data is important in high-performance network router applications, particularly for header processing of data packets and encryption/decryption operations.

Moreover, the amount of addressable memory is a key parameter of any architecture, and the historical trend is for increasing amounts of addressable memory. Within the next 10 years, some consumer applications will likely require more than 32 bits of addressing. In network applications, addresses of more than 32 bits are becoming common. In the future, networks will have to handle 64-bit-long addresses to meet the upcoming Internet Protocol standard, IPv6.

The SH-5 has a 64-bit architecture with sixty-four 64-bit registers. It provides standard integer operations at a 64-bit width and calculates addresses with 64-bit precision. The SH-5 also supports all operations necessary for efficient execution of existing 32-bit applications.

Figure 1 (next page) shows the SH-5's register organization and the register mapping between the two operating modes. The architecture includes 64-bit general-purpose integer and multimedia (SIMD) registers (R0 to R63), 32-bit floating-point registers (FR0 to FR63), target registers for branch target addresses (TR0 to TR7), and control registers (CR0 to CR63). It also includes a program counter (PC) and a floating-point status-and-control register (FPSCR).

The SHmedia architecture supports 6-bit register operands. A 6-bit field can encode 64 distinct values corresponding to register iden-

tifiers. The general-purpose and floating-point registers are separated into two independent register sets, each containing 64 registers.

As Figure 1 shows, the SHcompact mode uses a set of 32-bit registers, as in earlier SH architectures. The SH-5 uses only a subset of registers to map the 32-bit registers. It uses the lower halves of R19, CR0, and PC to map the T bit (the condition code bit), status register (SR), and program counter of SHcompact mode respectively.

Large register sets

With its large amount of architecturally visible register state, SHmedia is an excellent execution mode for computation-intensive algorithms. Many standard compiler techniques benefit from large register files. Compilers can apply optimizations such as common subexpression elimination, code migration, loop unrolling, function inlining, and instruction scheduling much more aggressively when large numbers of registers are available.[4-6]

Many algorithms using multimedia and floating-point instructions are effectively data pumps, crunching large amounts of data streamed from memory. Large register sets hold many more values at one time, so larger matrices or sets of coefficients can be held in registers without requiring memory accesses.

A large register set typically increases the context-switching overhead because more registers must be saved and restored. In the SH-5, we significantly reduced this overhead by providing a usage bit associated with each eight-register set. Whenever a register in the set is updated, the usage bit is set. The context-switching mechanism considers only register sets for which the usage bit has been set.

Figure 2 shows that SHmedia has only four instruction formats. The key difference among them is the presence and size of an immediate operand. We arranged the instruction fields for maximum regularity to simplify instruction decoding. This simplification reduces implementation complexity, improves clock speed, and reduces silicon area and power.

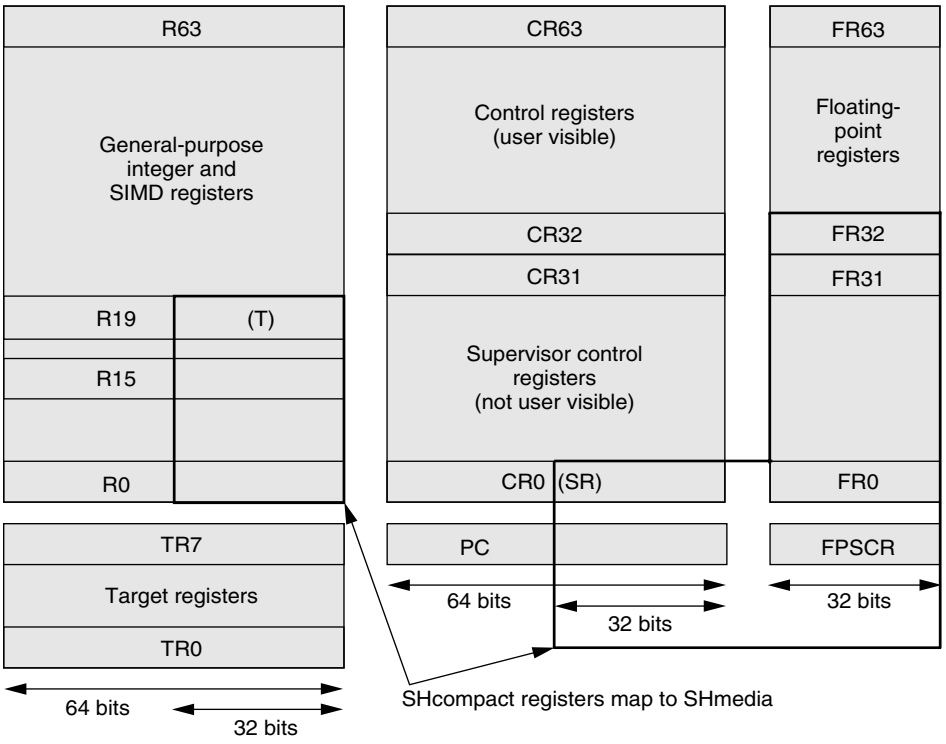


Figure 1. SH-5 register set and mapping.

Opcode	Rm	Ext	Rn	Rd	Reserved
6 bits	6 bits	4 bits	6 bits	6 bits	4 bits
Opcode	Rm	Ext	Immediate	Rd	Reserved
Opcode	Rm	10 bits immediate		Rd	Reserved
Opcode	16 bits immediate			Rd	Reserved

Figure 2. SHmedia instruction formats.

In superscalar implementations, encoding regularity significantly reduces the dependency-checking logic.

SHmedia instruction encoding uses 6 bits of opcode field with a 4-bit extension opcode available in some formats. The SH-5 contains 209 SHmedia instructions and uses less than one third of the available instruction space. Four of the 64 opcode values distinguished by the 6-bit opcode are fully reserved and have no associated instruction formatting. There are also 4 encoding bits reserved for every SHmedia instruction format and every instruction. Future architects can use these to add significant new architecture mechanisms such as predication, speculation,[6] or instruction-bundling information for VLIW (very long instruction word) instructions.[5]

Dynamic mode switching

At any moment, the SH-5's CPU decodes instructions in SHcompact or SHmedia. Executing certain branch instructions changes the mode from one instruction set to the other.[7] Programmers can switch modes dynamically and construct programs from either or both instruction sets. Dynamic mode switching allows application optimization for performance and code density.

The two-mode architecture is a fully optimized solution: SHcompact is optimized for code density, and SHmedia for high performance (including multimedia and 64-bit operation). The separation of code density and high performance ensures that neither is compromised.

Users can effectively and efficiently map software to the dual-mode architecture. Because mode switching takes place at branch points, the mode can be varied at the basic-block level. Typically, mode switching at the procedure level is a convenient programming and compilation model. A mixture of user and automatic control can achieve the split between SHcompact and SHmedia compilation. Optimizing compilers can use heuristics to estimate code performance requirements. The feedback of profile information from actual execution of the application controls mode switching.

By defining a new instruction set, we designed the SH-5 to support digital consumer and net-centered applications demanding much higher performance than typical embedded processors provide. The decision to move up to a 64-bit architecture was critical. It was prompted by the requirement for high performance through parallel manipulation of packed data, as well as the performance required in network applications. The SH-5 uses only a small portion of the available instruction-encoding space of the SHmedia instruction set architecture. The 64-bit infrastructure allows extensions for high-performance network applications requiring efficient processing of addresses and data exceeding 32 bits in width.

Programmers can switch modes dynamically and construct programs from either or both instruction sets.

Split-branch architecture

In most high-performance CPU implementations, deep pipelines are necessary to achieve high clock rates. A typical problem with these pipelines is that they require flushing at branch points. Usually, a pipeline detects a branch and evaluates the condition well into the pipeline. If a branch occurs, the processor must cancel any incorrect instructions that had entered the pipeline and restart the pipeline at the branch target address. With pipelines lengthening to achieve high clock rates, and the large number of branches occurring in common control-intensive code, branch penalties are potentially a very serious performance hazard.[8]

The SHmedia architecture has a unique branch mechanism that uses a cost-effective implementation to achieve a zero branch penalty most of the time. This implementation splits a branch operation into two parts (two separate instructions): prepare-to-branch (prepare target address) and branch (branch to target). Prepare-to-branch loads the target address into one of the eight branch target registers.

The compiler annotates prepare-target (PT)

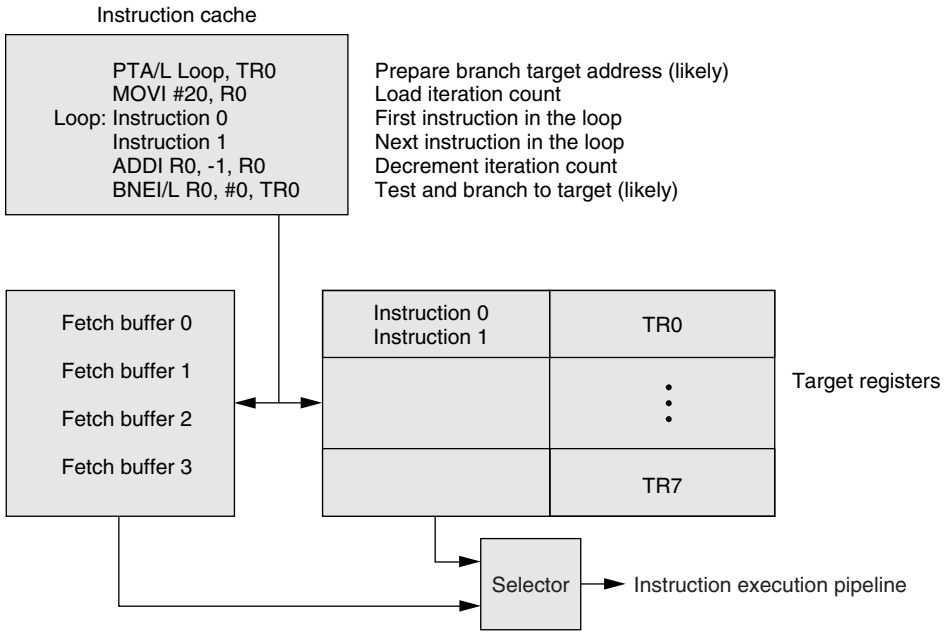


Figure 3. SHmedia's split-branch architecture.

and branch instructions with static prediction information. A Boolean variable, called the likely bit, is encoded within these instructions and serves the following purposes:

- The likely bit in a PT instruction is set if control is likely to flow to the target address. The bit controls whether the SH-5 implementation should attempt to prefetch instructions from that target address out of the instruction cache and into buffers close to the pipeline. Since there could be multiple PT instructions as well as the straight-line path, the SH-5 uses this information to arbitrate the fetch bandwidth between these paths.
- The likely bit in a conditional branch instruction should be set if that branch will likely be taken. This bit controls whether the SH-5 implementation should attempt to execute instructions down the taken or not-taken path. If this static prediction is correct, the branch has zero overhead. If it is a misprediction, there is a penalty while the pipeline is redirected to the other path. On the SH-5, two instructions will have entered the pipeline before misprediction is detected. Using the PT instruction to cause prefetching of branch targets results in only a two-cycle misprediction penalty.

The branch architecture's key property is that it allows low-overhead branching with a modest hardware investment. The SH-5 implementation provides two instruction buffers for each of the eight target registers. Additionally, it provides four instruction-fetch buffers for straight-line fetching. In the branch example shown in Figure 3, the prepare-target-address instruction (PTA) executes only once prior to entering the loop.

The compiler should attempt to schedule prepare-to-branch instructions as early as possible. This allows the processor to prefetch instructions at the branch target so that they are ready to execute when and if needed by the branch instruction. The overall effect is that instructions are prefetched into fetch buffers or target buffers close to the pipeline. Prefetching instructions avoids instruction cache latency, significantly reducing branch penalty costs.

The SH-5 has no hardware-based branch prediction mechanism and relies on software prediction with considerable success. We performed a number of microarchitecture-related performance analyses to determine a pipeline length that minimizes branch penalties. In contrast, desktop or workstation processors often have much longer pipelines and thus higher branch penalties. These processors often have large amounts of silicon dedicated to branch-processing mechanisms such as branch target buffers, hardware branch prediction, additional information in the instruction cache, return address stacks, and speculative instruction execution.[8] They can also achieve low-overhead branching, but the silicon cost is currently inappropriate for the digital consumer market.

The SH-5 instruction set also provides a different set of branch instructions, which fold powerful compare operations into the branch instruction, obviating the need for separate compare instructions.

SIMD instructions

A significant amount of data parallelism exists in most multimedia and DSP applications. SIMD instructions provide the parallelism required for efficient execution of these applications. (Several microprocessors have recently added SIMD extensions to their existing instruction sets.[9-11]) The SH-5's SIMD instructions operate in parallel on multiple pieces of data packed into a single register. For example, as Figure 4 shows, the SIMD instruction MADD.W splits each of its two 64-bit source registers into four 16-bit elements. Then it adds the corresponding elements, producing four 16-bit results, which are packed into the 64-bit result register.

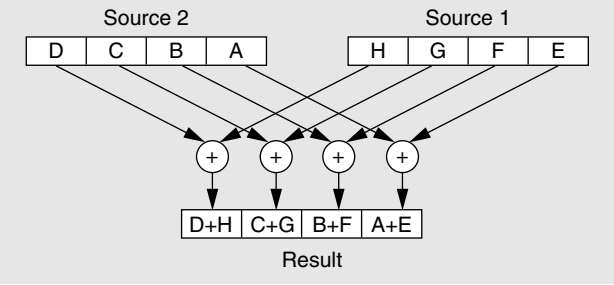


Figure 4. Multimedia addition: the SIMD instruction MADD.W.

Table 1. Summary of SH-5 SIMD multimedia instructions.

Instruction type	Description
Conversions	8-bit, 16-bit, 32-bit → 8-bit, 16-bit, 32-bit
Addition/subtraction	8-bit, 16-bit, 32-bit with/without saturation
Absolute	16-bit, 32-bit
Sum of absolute difference	8-bit
Shifts	16-bit, 32-bit left/right, arithmetic/logical
Comparisons	8-bit, 16-bit, 32-bit equal to/greater than
Full-width multiplies	2 × (2 × 16 bits) → (2 × 32 bits),
	2 × (1 × 32 bits) → (1 × 64 bits)
Multiplies	2 × (4 × 16 bits) → (4 × 16 bits),
	2 × (4 × 16 bits) → (1 × 64 bits)
	Rounding, multiply and add/subtract, MAC (integer, fractional)
Data manipulations	Shuffle, permute, concatenate, extract, bitwise conditional move

The SIMD instructions can perform parallel operations on eight pieces of 8-bit data, four pieces of 16-bit data, or two pieces of 32-bit data. The instructions support standard arithmetic as well as multimedia- and DSP-specific operations. For example, the multimedia unit can sustain a full cross product of two 16-bit vectors with four elements each in every cycle. In other words, it can perform four 16-bit multiplies, three 32-bit adds, and a 64-bit accumulate every cycle. This results in eight arithmetic operations per clock cycle. Thus, at a 400-MHz clock speed, it can achieve 3.2 billion operations per second (3.2 GOPS). The SH-5's SIMD instructions indirectly produce high levels of instruction-level parallelism without the complexity of superscalar implementations.

For video-encoding applications, the SH-5 architecture supports a sum-of-absolute-differences operation that forms the core of MPEG encoding algorithms. The following equation represents the operation:

$$r = r + \sum_{i=1}^{8} |ai - bi|$$

This instruction, when used with 8-bit data elements, effectively performs 24 operations each cycle, delivering 9.6 GOPS at 400 MHz.

Table 1 summarizes the SIMD instructions in the SH-5 instruction set. The instructions include elaborate sets of data-type conversion and data manipulation operations including shuffle, permute, extract, and concatenate. Also included are the SIMD forms of most fundamental arithmetic and logical operations and shifts. The instructions provide multiply-add/subtract and multiply-accumulate (MAC) with 16-bit and 32-bit data types in integer and fractional forms. Saturation operation is available for arithmetic, conversion, and shift operations. Two rounding modes (round toward minus and round toward nearest positive) are available for operations involving fractional data.

Unified register set

The SH-5 uses a single unified register set for both integer and multimedia data types. This approach significantly benefits the software and the implementation. Multimedia codes often reduce packed values to a scalar value—for

Table 2. Finding the inner product of two data streams: sequential code (a); code exploiting SIMD built-in functions (b). SIMD functions are shown in boldface.

(a) Original sequential code	Assembly code	Comment
short *A, *B;	ldx.w rA, ri, ra	16-bit × 1 load
int c;	ldx.w rB, ri, rb	16-bit × 1 load
for (i = 0; i < 128; I++) {	mac ra,rb,rc	16-bit × 1 MAC
c += A[i] * B[i];		
}		
(b) Code using SIMD functions		
long long *A,*B;		
long long a, b, c;		
for (i = 0; i < 32); i++) {	ldx.q rA, ri, ra	16-bit × 4 load
a=*(A + i);	ldx.q rB, ri, rb	16-bit × 4 load
b=*(B + i);	**mmulsum.wq ra, rb, rc**	16-bit × 4 MAC
c= **_vec_16x4mulsum**(a, b, c);		
}		

example, when performing accumulations across vectors. The unified register set ensures that mixed scalar and multimedia programming incurs no overhead. It also ensures that there is no arbitrary division of register state between integer and multimedia types.

High-level languages typically don't provide specialized multimedia data types. Multimedia types are usually mapped onto preexisting integer types of the same size. For example, a 64-bit integer type would represent a 64-bit multimedia type. For parameter-passing conventions, the most natural and efficient method of passing multimedia parameters is the same as that of their analogous integer type. Thus, using a unified general-purpose register set is very effective.

Examples of SIMD codes

C and C++ provide access to the SIMD operations through a simple functional library interface. Each instruction has an equivalent built-in C function (also called an intrinsic) that the programmer can use whenever the SIMD operation is needed. The compiler can convert this function directly to the equivalent SH-5 instruction and also apply its standard set of optimizations. As the integer and SIMD operations operate on a common register file, SIMD values can be simply declared as a 64-bit integer (commonly known as the long long type).

Tables 2 through 4 show codes for some common multimedia functions using SIMD instructions. In each table, section a shows the sequential code, and section b shows the use of the SIMD built-in function. Table 2 shows the basic inner product of two data streams. Section b shows the use of SIMD built-in function _vec_16x4mulsum( ). The C programmer uses the SIMD instruction mmulsum.wq, and the compiler generates code that runs about four times faster after compiler optimizations such as loop unrolling and instruction scheduling.

Table 3 shows the code for finding the maximum value and its location in a data stream. Section b shows SIMD built-in function _vec_16x4cmpgt( ). Using the built-ins, the program generates SIMD instruction mcmpgt.w. After the main loop finishes, some selection operations (a few cycles) are required. The code using the SIMD instructions is about four times faster than the original sequential code.

Table 4 shows the code for a filter for fixed-point fractional data streams. Section b shows the use of SIMD built-in function _vec_16x4mulfxrp( ) and corresponding compiler-generated code using the SIMD instruction mmulfxrp.w. The SIMD code is more than four times faster than the original code.

Performance for DSP applications

Figure 5 (on page 36) shows SH-5 performance for four DSP applications (fixed-point data types). The graph shows the normalized speedup of SH-5 relative to a conventional single-MAC DSP in executing a main loop iteration. Both the SH-5 and the DSP codes are hand optimized. The SH-5 code uses SIMD instructions. Assuming a single-issue SH-5 operating at 400 MHz and a typical high-performance DSP operating at 200 MHz, the graph shows that the SH-5 is about 2.5 to 4.5 times faster.

SH-5 implementation

The first implementation of the SH-5

Table 3. Finding the maximum value and its location in a data stream: sequential code (a); code exploiting SIMD built-in functions (b). SIMD functions are shown in boldface.

(a) Original sequential code	Assembly code	Comment
short *A, max, loc;		
max = A[0]; loc=0;		
	ldx.w rA, ri, ra	16-bit load
for (i = 1; i < 128; I++) {	cmpgt ra, rb, rc	compare
if (A[i] > max) {	cmvne rc, ra, rb	conditional move
max = A[i]; loc = i;	cmvne rc, ri, rd	conditional move
}		
}		

(b) Code using SIMD functions		
long long *A, max, loc, a, b, flag;		
max = *A;		
b = loc = 0x0003000200010000;		
inc = 0x0004000400040004;		
for (i = 1; i < 32; i++) {		
a = *(A + i);	ldx.q rA, ri, ra	16-bit × 4 load
flag = **_vec_16x4cmpgt**(a, max);	**mcmpgt.w ra, rm, rf**	16-bit × 4 compare
b = **_vec_16x4add**(b, inc);	**madd.w rc, rb, rb**	16-bit × 4 add
max = **_vec_condmove** (a, flag,max);	**mcmv ra, rf, rm**	bitwise conditional
loc = **_vec_condmove**(b, flag,loc);	**mcmv rb, rf, rc**	bitwise conditional
}		
.../* reduction code */		

Table 4. A filter for fixed-point fractional data streams: sequential code (a); code exploiting SIMD built-in functions (b). SIMD functions are shown in boldface.

(a) Original sequential code	Assembly code	Comment
short *A, *C, t;	ldx.w rA, ri, ra	
for (i = 0; i < 128; i++) {	ldx.w rC, ri, rc	
C[i] += (t * A[i] + ROUND) >> 15;	mul.l ra, rt, ra	
}	add ra, r1, ra	16-bit × 1 fractional
	shlri ra, 15, ra	rounding
	add ra, rc, rc	
	stx.w rC, ri, rc	

(b) Code using SIMD functions		
long long *A, *C, a, b, c, t;		
for (i = 0; i < 32; i++) {	ldx.q rA, ri, ra	
a = *(A + i);	ldx.q rC, ri, rc	
c = *(C + i);	**mmulfxrp.w ra, rt, rb**	16-bit × 4 fractional
b = **_vec_16x4mulfxrp**(a,t);	**madd.w rb, rc, rc**	mul with rounding
c = **_vec_16x4add**(b, c);	stx.q rC, ri, rc	
*(C + i) = c;		
}		

architecture's RISC core is a 400- to 600-MHz scalar, single-issue design that interfaces to a 200-MHz, split-transaction, pipelined, on-chip SuperHyway bus.[7] The core also includes a powerful debug module. Figure 6 shows a block diagram of the SH-5's first implementation, which includes the core, memory and peripheral interfaces, and a minimal set of peripheral modules.

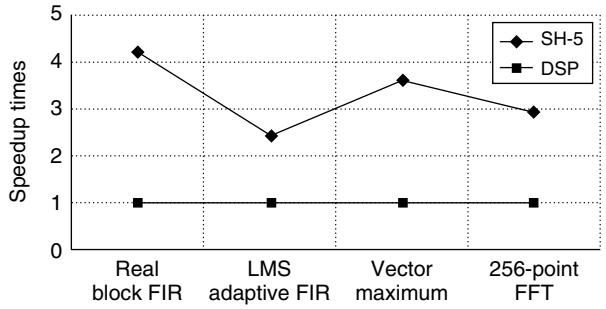
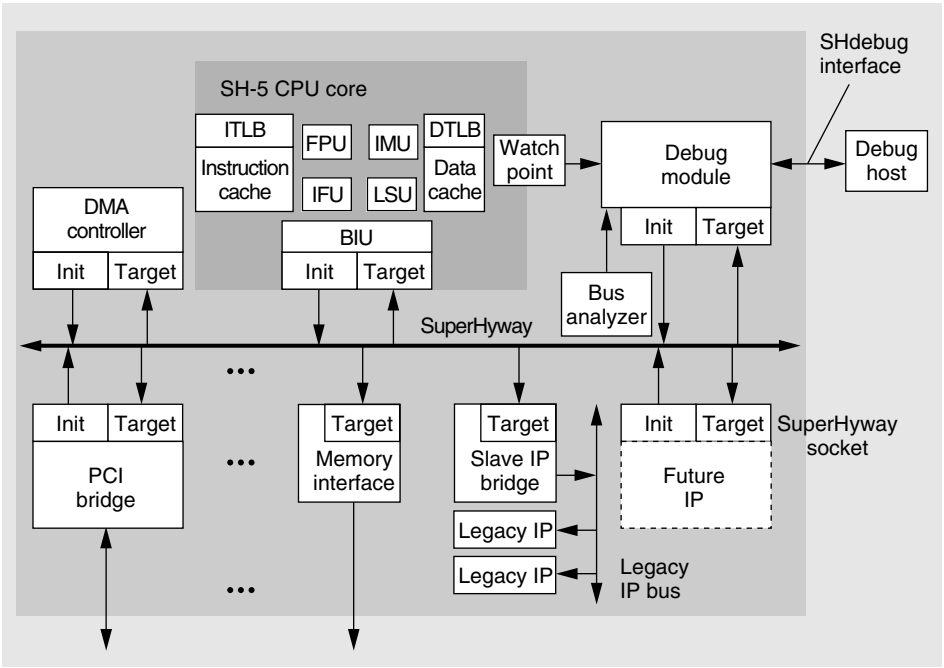


Figure 5. Performance ratio of SH-5 relative to conventional single-MAC DSP. (FIR: finite impulse response; LMS: least mean square; FFT: fast Fourier transform)

The core's salient features are

- a seven-stage pipeline,
- a 64-bit integer/multimedia processing unit,
- a 64-bit floating-point unit (optional),
- 32-Kbyte, four-way set-associative virtual instruction and data caches (64-bit access size and 32-byte line size),
- two separate 64-entry, fully associative TLBs (translation look-aside buffers) for instructions and data (activated only on cache misses), and
- independent bus access for the instruction fetch unit and load store unit, and up to three outstanding transaction requests for the SuperHyway bus.

When a particular algorithm for an application is mature and stable, a hardware (IP) implementation is more cost-effective and efficient in performance and power than a programmable software implementation. Following this principle, SH-5 implementations



BIU	Bus interface unit	Init	Initiator
DMA	Direct memory access	IP	Application-specific hardware module (intellectual property)
DTLB	Data translation look-aside buffer	ITLB	Instruction translation look-aside buffer
FPU	Floating-point unit	LSU	Load store unit
IFU	Instruction fetch unit	PCI	Peripheral component interface
IMU	Integer/multimedia unit		

Figure 6. Organization of the first SH-5 implementation.

will be SOCs consisting of the core and a number of IPs with appropriate hardware/software trade-offs. Integrating these application-specific IPs requires highly efficient and flexible interconnections.

To facilitate efficient SOC integration, the bus interface unit in the SH-5 core connects to the 64-bit, very high bandwidth SuperHyway bus. The bus is compatible with the Virtual Socket Interface (VSI) protocols (http://www.vsi.org). It seamlessly connects to VSI virtual-component libraries. It supports split transactions, and transactions may contain up to 32 bytes of data. In the first SH-5 implementation, the SuperHyway consists of two 64-bit read/write buses. At the 200-MHz peak frequency, the peak bandwidth would be (2 × 64 bits × 200 MHz) or 3.2 Gbytes per second.

Effective support of high-speed, multi-initiator, multitarget data transfer is important for cost-effective SOC implementations. Such data transfer mechanisms also must be flexible to support different configurations.

The SuperHyway can perform two routing jobs at any instant. It can receive data transaction requests from IP-module initiator ports and route one of the requests to the target port of an IP module ready to receive a transaction request. At the same time, it can route a response from a target port. Such a response could be "Read data on read transaction request." The response is automatically directed back to the original initiator port. Since the SuperHyway uses a split-transaction protocol, a response-time constraint and an in-order constraint are unnecessary. Target ports can receive requests and save them in an internal service queue.

Debug support

For SOCs designed around the SH-5, all interconnection will take place via the SuperHyway, making it necessary to provide the capability of accessing the bus for analysis. Traditional external logic analyzers and debug tools cannot be used.

The SH-5 provides an advanced, nonintrusive debugging module, SHdebug, which is transparent to the target application software. SHdebug provides various trigger mechanisms that can invoke break exceptions on the CPU core or retrieve trace data packets via the host debug interface. The CPU includes a watch point controller with 12 watch point channels, through which the CPU's internal activities can be observed by the external debug environment. Chain latches support combinations of watch point conditions for filtering and conditional tracing. The bus analyzer logic shown in Figure 6 is a mechanism for observing SuperHyway transactions. Watch point logic and bus analyzer logic provide a complex combination of trigger conditions, including instruction address, instruction code, operand address, access type, and access size.

Effective support of high-speed, multi-initiator, multitarget data transfer is important for cost-effective SOC implementations.

Through an initiator port on the debug module, the external debug host can access system resources in on-chip IP modules. The debugger can read from or write to all addressable locations. Program or data downloading to the target system flash memory takes place via the debug host interface, the debug module, the SuperHyway, and the memory interface.

The first SH-5 product chip will ship this year. It will include the components shown in Figure 6. Running at 400 MHz in a 0.15-micron CMOS device, it will deliver 714 Dhrystone-1.1 MIPS, 9.6 GOPS, and 2.8 Gflops, with a CPU core power consumption of less than 600 mW. Hitachi and STMicroelectronics will embed the SH-5 CPU core into SOC devices in their SH8000 and ST50 products. MICRO

Acknowledgments

The development of the SH-5's new instruction set architecture and its first imple-

mentation was a team effort, and listing all the people who contributed would be difficult. In particular, however, we thank Sivaram Krishnan for his contributions to the instruction set design.

References

1. A. Hasegawa et al., "SH-3: High Code Density, Low Power," *IEEE Micro*, Vol. 15, No. 6, Nov.-Dec. 1995, pp. 11-19.
2. F. Arakawa et al., "SH-4 RISC Multimedia Processor," *IEEE Micro*, Vol. 18, No. 2, Mar.-Apr. 1998, pp. 26-34.
3. J. Bunda et al., "16-Bit vs. 32-Bit Instructions for Pipelined Microprocessors," *Proc. Int'l Symp. Computer Architecture*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1992, pp. 237-246.
4. J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, 2nd ed., Morgan Kaufmann, San Mateo, Calif., 1996.
5. B.R. Rau and J.A. Fisher, "Instruction Level Parallel Processing: History, Overview and Perspective," *J. Supercomputing*, Vol. 7, No. 1, Jan. 1993, pp. 9-50.
6. W.M. Hwu et al., "Compiler Technology for Future Microprocessors," *Proc. IEEE*, Vol. 83, No. 12, Dec. 1995, pp. 1625-1665.
7. J. Slager and J.M. Rolland, "SH-5: Extending SuperH to 64 Bits," *Proc. Microprocessor Forum*, Cahners MicroDesign Resources, San Jose, Calif., 1999.
8. H.G. Cragon, *Branch Strategy Taxonomy and Performance*, IEEE CS Press, Los Alamitos, Calif., 1991.
9. M. Tremblay et al., "VIS Speeds New Media Processing," *IEEE Micro*, Vol.16, No. 4, July-Aug. 1996, pp. 10-20.
10. A. Peleg and U. Weiser, "MMX Technology Extension to the Intel Architecture," *IEEE Micro*, Vol. 16, No. 4, July-Aug. 1996, pp. 42-50.
11. R. Lee, "Accelerating Multimedia with Enhanced Microprocessors," *IEEE Micro*, Vol. 15, No. 2, Mar.-Apr. 1995, pp. 59-69.

Prasenjit Biswas is a director at Hitachi Semiconductor (America), where he is responsible for microprocessor architecture and functional verification. In previous positions, he was a senior processor architect at Cyrix and a technical researcher at Texas Instruments. He was a Canadian Commonwealth Fellow at the University of Alberta, Canada, and an assistant professor and adjunct associate professor in the Computer Science and Engineering Department of Southern Methodist University. He has published more than 30 technical papers. Biswas holds a BE in electronics and communication from Jadavpur University, Calcutta; an MS in computer science from the University of Alberta; and a PhD in computer and systems sciences from Jawaharlal Nehru University, New Delhi. He is a member of the IEEE and the Computer Society.

Atsushi Hasegawa is a microarchitect and development manager in the Advanced Microprocessor Development Department at Hitachi Semiconductor (America). He is interested in system architecture, microprocessor architecture, and design methodology. Hasegawa received a BS in computer engineering from the University of Electro-Communications, Japan.

Srinivas Mandaville works at Hitachi Semiconductor (America), where he contributed to the development of the SH-5 SIMD instruction set. He is interested in the development of algorithms and systems for multimedia/DSP applications. Prior to joining Hitachi, Mandaville worked for Motorola. He received a PhD from the Indian Institute of Science, Bangalore, India.

Mark Debbage is an architecture manager in the Micro Core Development Division of STMicroelectronics, where he has been a member of the joint SH-5 development team from its inception. He was a major contributor to the SH-5 specification and coauthor of the SH-5 CPU architecture manuals. Debbage earned a BEng in electronics engineering and a PhD in parallel computing from the University of Southampton, England.

Andy Sturges leads the architecture group working on SuperH-based cores at STMicroelectronics. He has worked on a number of processors including the Inmos T800 transputer and the Chameleon multimedia processor. He graduated from the University of Kent at Canterbury, England.

Fumio Arakawa designs microprocessors at the Central Research Laboratory of Hitachi Ltd. in Tokyo. He received a BS and MS degrees in applied physics from the University of Tokyo.

Yasuhiko Saito is a researcher in the Advanced Device Development Department of Hitachi Ltd. His research interests focus on compiler support for high-performance computing and environments for producing high-quality programs. He received an MS in mathematics from Tohoku University.

Kunio Uchiyama works for the Central Research Laboratory, Hitachi Ltd., Tokyo, on design automation, small-scale mainframes, cache memory, and microprocessors. From 1985 to 1986, he was a visiting researcher at Carnegie Mellon University. He received the 1998 Ichimura award, the 1999 R&D100 award, and the 2000 Chief Officer's award of the Japanese Science and Technology Agency. Uchiyama received a BS and an MS in information science from the Tokyo Institute of Technology.

Direct questions and comments about this article to Prasenjit Biswas or Atsushi Hasegawa, Hitachi Semiconductor, 179 E. Tasman Dr., San Jose, CA 95134; prasenjit.biswas@hsa.hitachi.com or atsushi.hasegawa@hsa.hitachi.com.