1 Low Power Multicore Processors for Embedded Systems

FUMIO ARAKAWA

1.1 MULTICORE CHIP WITH HIGHLY EFFICIENT CORES

A multicore chip is one of the most promising approaches to achieve high performance. Formerly, frequency scaling was the best approach. However, the scaling has hit the power wall, and frequency enhancement is slowing down. Further, the performance of a single processor core is proportional to the square root of its area, known as Pollack's rule [1], and the power is roughly proportional to the area. This means lower performance processors can achieve higher power efficiency. Therefore, we should make use of the multicore chip with relatively low performance processors.

The power wall is not a problem only for high-end server systems. Embedded systems also face this problem for further performance improvements [2]. MIPS is the abbreviation of million instructions per second, and a popular integer-performance measure of embedded processors. The same performance processors should take the same time for the same program, but the original MIPS varies, reflecting the number of instructions executed for a program. Therefore, the performance of a Dhrystone benchmark relative to that of a VAX 11/780 minicomputer is broadly used [3, 4]. This is because it achieved 1 MIPS, and the relative performance value is called VAX MIPS or DMIPS, or simply MIPS. Then GIPS (giga-instructions per second) is used instead of the MIPS to represent higher performance.

Figure 1.1 roughly illustrates the power budgets of chips for various application categories. The horizontal and vertical axes represent performance (DGIPS) and efficiency (DGIPS/W) in logarithmic scale, respectively. The oblique lines represent constant power (W) lines and constant product lines

Embedded Systems: Hardware, Design, and Implementation, First Edition.
Edited by Krzysztof Iniewski.

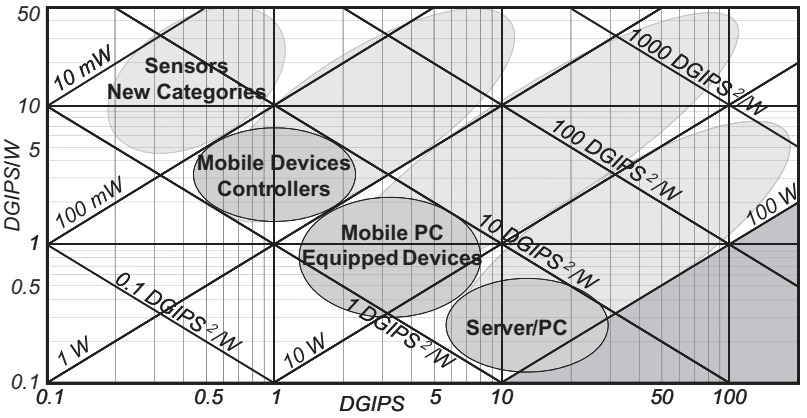

FIGURE 1.1. Power budgets of chips for various application categories.

of the power–performance ratio and the power ($DGIPS^2/W$). The product roughly indicates the attained degree of the design. There is a trade-off relationship between the power efficiency and the performance. The power of chips in the server/personal computer (PC) category is limited at around 100 W, and the chips above the 100-W oblique line must be used. Similarly, the chips roughly above the 10- or 1-W oblique line must be used for equipped-devices/mobile PCs, or controllers/mobile devices, respectively. Further, some sensors must use the chips above the 0.1-W oblique line, and new categories may grow from this region. Consequently, we must develop high $DGIPS^2/W$ chips to achieve high performance under the power limitations.

Figure 1.2 maps various processors on a graph, whose horizontal and vertical axes respectively represent operating frequency (MHz) and power–frequency ratio (MHz/W) in logarithmic scale. Figure 1.2 uses MHz or GHz instead of the DGIPS of Figure 1.1. This is because few DGIPS of the server/PC processors are disclosed. Some power values include leak current, whereas the others do not; some are under the worst conditions while the others are not. Although the MHz value does not directly represent the performance, and the power measurement conditions are not identical, they roughly represent the order of performance and power. The triangles and circles represent embedded and server/PC processors, respectively. The dark gray, light gray, and white plots represent the periods up to 1998, after 2003, and in between, respectively. The GHz^2/W improved roughly 10 times from 1998 to 2003, but only three times from 2003 to 2008. The enhancement of single cores is apparently slowing down. Instead, the processor chips now typically adopt a multicore architecture.

Figure 1.3 summarizes the multicore chips presented at the International Solid-State Circuit Conference (ISSCC) from 2005 to 2008. All the processor chips presented at ISSCC since 2005 have been multicore ones. The axes are

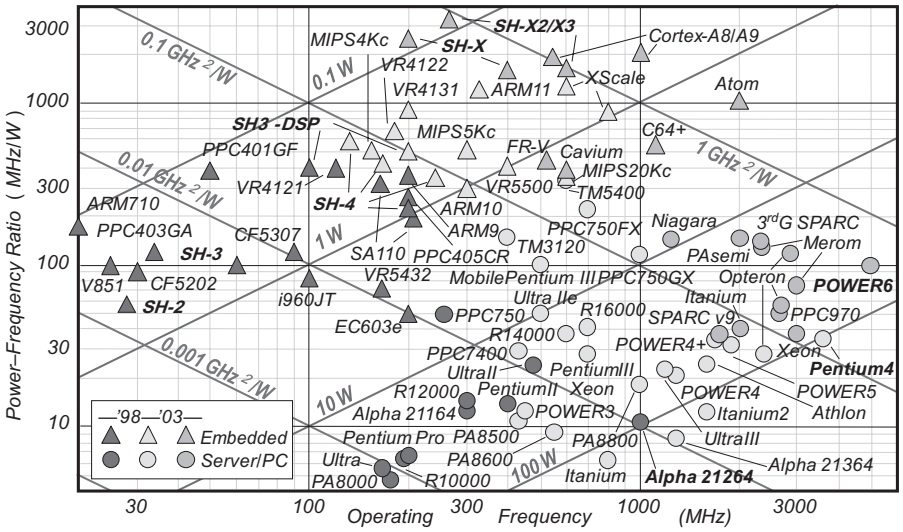

FIGURE 1.2. Performance and efficiency of various processors.

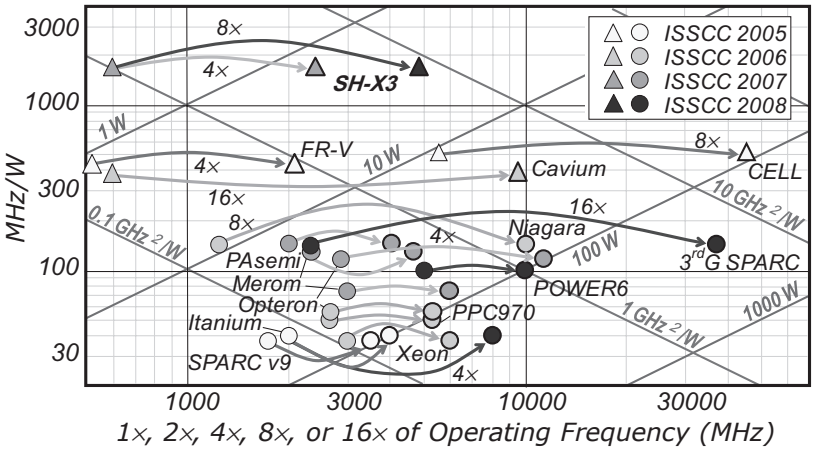

FIGURE 1.3. Some multicore chips presented at ISSCC.

similar to those of Figure 1.2, although the horizontal axis reflects the number of cores. Each plot at the start and end points of an arrow represent single core and multicore, respectively.

The performance of multicore chips has continued to improve, which has compensated for the slowdown in the performance gains of single cores in both the embedded and server/PC processor categories. There are two types

of muticore chips. One type integrates multiple-chip functions into a single chip, resulting in a multicore SoC. This integration type has been popular for more than 10 years. Cell phone SoCs have integrated various types of hardware intellectual properties (HW-IPs), which were formerly integrated into multiple chips. For example, an SH-Mobile G1 integrated the function of both the application and baseband processor chips [5], followed by SH-Mobile G2 [6] and G3 [7, 8], which enhanced both the application and baseband functionalities and performance. The other type has increased number of cores to meet the requirements of performance and functionality enhancement. The RP-1, RP-2 and RP-X are the prototype SoCs, and an SH2A-DUAL [9] and an SH-Navi3 [10] are the multicore products of this enhancement type. The transition from single core chips to multicore ones seems to have been successful on the hardware side, and various multicore products are already on the market. However, various issues still need to be addressed for future multicore systems.

The first issue concerns memories and interconnects. Flat memory and interconnect structures are the best for software, but hardly possible in terms of hardware. Therefore, some hierarchical structures are necessary. The power of on-chip interconnects for communications and data transfers degrade power efficiency, and a more effective process must be established. Maintaining the external input/output (I/O) performance per core is more difficult than increasing the number of cores, because the number of pins per transistors decreases for finer processes. Therefore, a breakthrough is needed in order to maintain the I/O performance.

The second issue concerns runtime environments. The performance scalability was supported by the operating frequency in single core systems, but it should be supported by the number of cores in multicore systems. Therefore, the number of cores must be invisible or virtualized with small overhead when using a runtime environment. A multicore system will integrate different subsystems called domains. The domain separation improves system reliability by preventing interference between domains. On the other hand, the well-controlled domain interoperation results in an efficient integrated system.

The third issue relates to the software development environments. Multicore systems will not be efficient unless the software can extract application parallelism and utilize parallel hardware resources. We have already accumulated a huge amount of legacy software for single cores. Some legacy software can successfully be ported, especially for the integration type of multicore SoCs, like the SH-Mobile G series. However, it is more difficult with the enhancement type. We must make a single program that runs on multicore, or distribute functions now running on a single core to multicore. Therefore, we must improve the portability of legacy software to the multicore systems. Developing new highly parallel software is another issue. An application or parallelization specialist could do this, although it might be necessary to have specialists in both areas. Further, we need a paradigm shift in the development, for example, a higher level of abstraction, new parallel languages, and assistant tools for effective parallelization.

1.2 SUPERH™ RISC ENGINE FAMILY (SH) PROCESSOR CORES

As mentioned above, a multicore chip is one of the most promising approaches to realize high efficiency, which is the key factor to achieve high performance under some fixed power and cost budgets. Therefore, embedded systems are employing multicore architecture more and more. The multicore is good for multiplying single-core performance with maintaining the core efficiency, but does not enhance the efficiency of the core itself. Therefore, we must use highly efficient cores. SuperH™ (Renesas Electronics, Tokyo) reduced instruction set computer (RISC) engine family (SH) processor cores are highly efficient typical embedded central processing unit (CPU) cores for both single- and multicore chips.

1.2.1 History of SH Processor Cores

Since the beginning of the microprocessor history, a processor especially for PC/servers had continuously advanced its performance while maintaining a price range from hundreds to thousands of dollars [11, 12]. On the other hand, a single-chip microcontroller had continuously reduced its price, resulting in the range from dozens of cents to several dollars with maintaining its performance, and had been equipped to various products [13]. As a result, there was a situation of no demand on the processor of the middle price range from tens to hundreds of dollars.

However, with the introduction of the home game console in the late 1980s and the digitization of the home electronic appliances from the 1990s, there occurred the demands to a processor suitable for multimedia processing in this price range. Instead of seeking high performance, such a processor has attached great importance to high efficiency. For example, the performance is 1/10 of a processor for PCs, but the price is 1/100, or the performance equals to a processor for PCs for the important function of the product, but the price is 1/10. The improvement of area efficiency has become the important issue in such a processor.

In the late 1990s, a high performance processor consumed too high power for mobile devices, such as cellular phones and digital cameras, and the demand was increasing on the processor with higher performance and lower power for multimedia processing. Therefore, the improvement of the power efficiency became the important issues. Furthermore, when the 2000s begins, more functions were integrated by further finer processes, but on the other hand, the increase of the initial and development costs became a serious problem. As a result, the flexible specification and the cost reduction came to be important issues. In addition, the finer processes suffered from the more leakage current.

Under the above background, embedded processors were introduced to meet the requirements, and have improved the area, power, and development efficiencies. The SH processor cores are one of such highly efficient CPU cores.

The first SH processor was developed based on SuperH architecture as one of embedded processors in 1993. Then the SH processors have been developed

as a processor with suitable performance for multimedia processing and area-and-power efficiency. In general, performance improvement causes degradation of the efficiency as Pollack's rule indicates [1]. However, we can find ways to improve both performance and efficiency. Although individually each method is a small improvement, overall it can still make a difference.

The first-generation product, SH-1, was manufactured using a 0.8-μm process, operated at 20 MHz, and achieved performance of 16 MIPS in 500 mW. It was a high performance single-chip microcontroller, and integrated a read-only memory (ROM), a random access memory (RAM), a direct memory access controller (DMAC), and an interrupt controller.

The second-generation product, SH-2, was manufactured using the same 0.8-μm process as the SH-1 in 1994 [14]. It operated at 28.5 MHz, and achieved performance of 25 MIPS in 500 mW by optimization on the redesign from the SH-1. The SH-2 integrated a cache memory and an SDRAM controller instead of the ROM and the RAM of the SH-1. It was designed for the systems using external memories. The integrated SDRAM controller did not popular at that time, but enabled to eliminate an external circuitry, and contributed to system cost reduction. In addition, the SH-2 integrated a 32-bit multiplier and a divider to accelerate multimedia processing. And it was equipped to a home game console, which was one of the most popular digital appliances. The SH-2 extend the application field of the SH processors to the digital appliances with multimedia processing.

The third-generation product SH-3 was manufactured using a 0.5-μm process in 1995 [15]. It operated at 60 MHz, and achieved performance of 60 MIPS in 500 mW. Its power efficiency was improved for a mobile device. For example, the clock power was reduced by dividing the chip into plural clock regions and operating each region with the most suitable clock frequency. In addition, the SH-3 integrated a memory management unit (MMU) for such devices as a personal organizer and a handheld PC. The MMU is necessary for a general-purpose operating system (OS) that enables various application programs to run on the system.

The fourth-generation product, SH-4, was manufactured using a 0.25-μm process in 1997 [16–18]. It operated at 200 MHz, and achieved performance of 360 MIPS in 900 mW. The SH-4 was ported to a 0.18-μm process, and its power efficiency was further improved. The power efficiency and the product of performance and the efficiency reached to 400 MIPS/W and 0.14 GIPS^2/W, respectively, which were among the best values at that time. The product roughly indicates the attained degree of the design, because there is a trade-off relationship between performance and efficiency.

The fifth-generation processor, SH-5, was developed with a newly defined instruction set architecture (ISA) in 2001 [19–21], and an SH-4A, the advanced version of the SH-4, was also developed with keeping the ISA compatibility in 2003. The compatibility was important, and the SH-4A was used for various products. The SH-5 and the SH-4A were developed as a CPU core connected to other various HW-IPs on the same chip with a SuperHyway

standard internal bus. This approach was available using the fine process of 0.13 μm, and enabled to integrate more functions on a chip, such as a video codec, 3D graphics, and global positioning systems (GPS).

An SH-X, the first generation of the SH-4A processor core series, achieved a performance of 720 MIPS with 250 mW using a 0.13-μm process [22–26]. The power efficiency and the product of performance and the efficiency reached to 2,880 MIPS/W and 2.1 GIPS^2/W, respectively, which were among the best values at that time. The low power version achieved performance of 360 MIPS and power efficiency of 4,500 MIPS/W [27–29].

An SH-X2, the second-generation core, achieved 1,440 MIPS using a 90-nm process, and the low power version achieved power efficiency of 6,000 MIPS/W in 2005 [30–32]. Then it was integrated on product chips [5–8].

An SH-X3, the third-generation core, supported multicore features for both SMP and AMP [33, 34]. It was developed using a 90-nm generic process in 2006, and achieved 600 MHz and 1,080 MIPS with 360 mW, resulting in 3,000 MIPS/W and 3.2 GIPS^2/W. The first prototype chip of the SH-X3 was a RP-1 that integrated four SH-X3 cores [35–38], and the second one was a RP-2 that integrated eight SH-X3 cores [39–41]. Then, it was ported to a 65-nm low power process, and used for product chips [10].

An SH-X4, the latest fourth-generation core, was developed using a 45-nm low power process in 2009, and achieved 648 MHz and 1,717 MIPS with 106 mW, resulting in 16,240 MIPS/W and 28 GIPS^2/W [42–44].

1.2.2 Highly Efficient ISA

Since the beginning of the RISC architecture, all the RISC processor had adopted a 32-bit fixed-length ISA. However, such a RISC ISA causes larger code size than a conventional complex instruction set computer (CISC) ISA, and requires larger capacity of program memories including an instruction cache. On the other hand, a CISC ISA has been variable length to define the instructions of various complexities from simple to complicated ones. The variable length is good for realizing the compact code sizes, but requires complex decoding, and is not suitable for parallel decoding of plural instructions for the superscalar issue.

SH architecture with the 16-bit fixed-length ISA was defined in such a situation to achieve compact code sizes and simple decoding. The 16-bit fixed-length ISA was spread to other processor ISAs, such as ARM Thumb and MIPS16.

As always, there should be pros and cons of the selection, and there are some drawbacks of the 16-bit fixed-length ISA, which are the restriction of the number of operands and the short literal length in the code. For example, an instruction of a binary operation modifies one of its operand, and an extra data transfer instruction is necessary if the original value of the modified operand must be kept. A literal load instruction is necessary to utilize a longer literal than that in an instruction. Further, there is an instruction using an

implicitly defined register, which contributes to increase the number of operand with no extra operand field, but requires special treatment to identify it, and spoils orthogonal characteristics of the register number decoding. Therefore, careful implementation is necessary to treat such special features.

1.2.3 Asymmetric In-Order Dual-Issue Superscalar Architecture

Since a conventional superscalar processor gave priority to performance, the superscalar architecture was considered to be inefficient, and scalar architecture was still popular for embedded processors. However, this is not always true. Since the SH-4 design, SH processors have adopted the superscalar architecture by selecting an appropriate microarchitecture with considering efficiency seriously for an embedded processor.

The asymmetric in-order dual-issue superscalar architecture is the base microarchitecture of the SH processors. This is because it is difficult for a general-purpose program to utilize the simultaneous issue of more than two instructions effectively; a performance enhancement is not enough to compensate the hardware increase for the out-of-order issue, and symmetric superscalar issue requires resource duplications. Then, the selected architecture can maintain the efficiency of the conventional scalar issue one by avoiding the above inefficient choices.

The asymmetric superscalar architecture is sensitive to instruction categorizing, because the same category instruction cannot be issued simultaneously. For example, if we categorize all floating-point instructions in the same category, we can reduce the number of floating-point register ports, but cannot issue both floating-point instructions of arithmetic and load/store/transfer operations at a time. This degrades the performance. Therefore, the categorizing requires careful trade-off consideration between performance and hardware cost.

First of all, both the integer and load/store instructions are used most frequently, and categorized to different groups of integer (INT) and load/store (LS), respectively. This categorization requires address calculation unit in addition to the conventional arithmetic logical unit (ALU). Branch instructions are about one-fifth of a program on average. However, it is difficult to use the ALU or the address calculation unit to implement the early-stage branch, which calculates the branch addresses at one-stage earlier than the other type of operations. Therefore, the branch instruction is categorized in another group of branch (BR) with a branch address calculation unit. Even a RISC processor has a special instruction that cannot fit to the superscalar issue. For example, some instruction changes a processor state, and is categorized to a group of nonsuperscalar (NS), because most of instructions cannot be issued with it.

The 16-bit fixed-length ISA frequently uses an instruction to transfer a literal or register value to a register. Therefore, the transfer instruction is categorized to the BO group to be executable on both integer and load/store

(INT and LS) pipelines, which were originally for the INT and LS groups. Then the transfer instruction can be issued with no resource conflict. A usual program cannot utilize all the instruction issue slots of conventional RISC architecture that has three operand instructions and uses transfer instructions less frequently. Extra transfer instructions of the 16-bit fixed-length ISA can be inserted easily with no resource conflict to the issue slots that would be empty for a conventional RISC.

The floating-point load/store/transfer and arithmetic instructions are categorized to the LS group and a floating-point execution (FE) group, respectively. This categorization increases the number of the ports of the floating-point register file. However, the performance enhancement deserves the increase. The floating-point transfer instructions are not categorized to the BO group. This is because neither the INT nor FE group fit to the instruction. The INT pipeline cannot use the floating-point register file, and the FE pipeline is too complicated to treat the simple transfer operation. Further, the transfer instruction is often issued with a FE group instruction, and the categorization to other than the FE group is enough condition for the performance.

The SH ISA supports floating-point sign negation and absolute value (FNEG and FABS) instructions. Although these instructions seem to fit the FE group, they are categorized to the LS group. Their operations are simple enough to execute at the LS pipeline, and the combination of another arithmetic instruction becomes a useful operation. For example, the FNEG and floating-point multiply–accumulate (FMAC) instructions became a multiply-and-subtract operation.

Table 1.1 summarizes the instruction categories for asymmetric superscalar architecture. Table 1.2 shows the ability of simultaneous issue of two instructions. As an asymmetric superscalar processor, each pipeline for the INT, LS, BR, or FE group is one, and the simultaneous issue is limited to a pair of different group instructions, except for a pair of the BO group instructions, which can be issued simultaneously using both the INT and LS pipelines. An NS group instruction cannot be issued with another instruction.

1.3 SH-X: A HIGHLY EFFICIENT CPU CORE

The SH-X has enhanced its performance by adopting superpipeline architecture to the base micro-architecture of the asymmetric in-order dual-issue superscalar architecture. The operating frequency would be limited by an applied process without fundamental change of the architecture or microarchitecture. Although conventional superpipeline architecture was thought inefficient as was the conventional superscalar architecture before applying to the SH-4, the SH-X core enhanced the operating frequency with maintaining the high efficiency.

TABLE 1.1. Instruction Categories for Asymmetric Superscalar Architecture

INT	FE
ADD; ADDC; ADDV; SUB; SUBC; SUBV; MUL; MULU; MULS; DMULU; DMULS; DIV0U; DIV0S; DIV1; CMP; NEG; NEGC; NOT; DT; MOVT; CLRT; SETT; CLRMAC; CLRS; SETS; TST Rm, Rn; TST imm, R0; AND Rm, Rn; AND imm, R0; OR Rm, Rn; OR imm, R0; XOR Rm, Rn; XOR imm, R0; ROTL; ROTR; ROTCL; ROTCR; SHAL; SHAR; SHAD; SHLD; SHLL; SHLL2; SHLL8; SHLL16; SHLR; SHLR2; SHLR8; SHLR16; EXTU; EXTS; SWAP; XTRCT	FADD; FSUB; FMUL; FDIV; FSQRT; FCMP; FLOAT; FTRC; FCNVSD; FCNVDS; FMAC; FIPR; FTRV; FSRRA; FSCA; FRCHG; FSCHG; FPCHG
	BO
	MOV imm, Rn; MOV Rm, Rn; NOP
	BR
	BRA; BSR; BRAF; BSRF; BT; BF; BT/S; BF/S; JMP; JSR; RTS
	NS
LS	AND imm, @(R0,GBR); OR imm, @(R0,GBR); XOR imm, @(R0,GBR); TST imm, @(R0,GBR); MAC; SYNCO; MOVLI; MOVCO; LDC (SR/SGR/DBR); STC (SR); RTE; LDTLB; ICBI; PREFI; TAS; TRAPA; SLEEP
MOV (load/store); MOVA; MOVCA; FMOV; FLDI0; FLDI1; FABS; FNEG; FLDS; FSTS; LDS; STS; LDC (except SR/SGR/DBR); STC (except SR); OCBI; OCBP; OCBWB; PREF	

TABLE 1.2. Simultaneous Issue of Instructions

		Second Instruction Category					
		BO	INT	LS	BR	FE	NS
First Instruction Category	BO	✓	✓	✓	✓	✓	
	INT	✓		✓	✓	✓	
	LS	✓	✓		✓	✓	
	BR	✓	✓	✓		✓	
	FE	✓	✓	✓	✓		
	NS						

TABLE 1.3. Microarchitecture Selections of SH-X

	Selections	Other Candidates	Merits
Pipeline stages	7	5, 6, 8, 10, 15, 20	1.4 times frequency enhancement
Branch acceleration	Out-of-order issue	BTB, branch with plural instructions	Compatibility, small area, for low frequency branch
Branch prediction	Dynamic (BHT, global history)	Static (fixed direction, hint bit in instruction)	
Latency concealing	Delayed execution, store buffers	Out-of-order issue	Simple, small

1.3.1 Microarchitecture Selections

The SH-X has seven-stage superpipeline to maintain the efficiency among various numbers of stages applied to various processors up to highly superpipelined 20 stages [45]. The conventional seven-stage pipeline degraded the cycle performance compared with the five-stage one that is popular for efficient embedded processors. Therefore, appropriate methods were chosen to enhance and recover the cycle performance with the careful trade-off judgment of performance and efficiency. Table 1.3 summarizes the selection result of the microarchitecture.

An out-of-order issue is the popular method used by a high-end processor to enhance the cycle performance. However, it requires much hardware and is too inefficient especially for general-purpose register handling. The SH-X adopts an in-order issue except branch instructions using no general-purpose register.

The branch penalty is the serious problem for the superpipeline architecture. The SH-X adopts a branch prediction and an out-of-order branch issue, but does not adopt a more expensive way with a branch target buffer (BTB) and an incompatible way with plural instructions. The branch prediction is categorized to static and dynamic ones, and the static ones require the architecture change to insert the static prediction result to the instruction. Therefore, the SH-X adopts a dynamic one with a branch history table (BHT) and a global history.

The load/store latencies are also a serious problem, and the out-of-order issue is effective to hide the latencies, but too inefficient to adopt as mentioned above. The SH-X adopts a delayed execution and a store buffer as more efficient methods.

The selected methods are effective to reduce the pipeline hazard caused by the superpipeline architecture, but not effective to avoid a long-cycle stall caused by a cache miss for an external memory access. Such a stall could be avoided by an out-of-order architecture with large-scale buffers, but is not a serious problem for embedded systems.

1.3.2 Improved Superpipeline Structure

Figure 1.4 illustrates a conventional seven-stage superpipeline structure. The seven stages consist of 1st and 2nd instruction fetch (I1 and I2) stages and an instruction decoding (ID) stage for all the pipelines, 1st to 4th execution (E1, E2, E3, and E4) stages for the INT, LS, and FE pipelines. The FE pipeline has nine stages with two extra execution stages of E5 and E6.

A conventional seven-stage pipeline has less performance than a five-stage one by 20%. This means the performance gain of the superpipeline architecture is only $1.4 \times 0.8 = 1.12$ times, which would not compensate the hardware increase. The branch and load-use-conflict penalties increase by the increase of the instruction-fetch and data-load cycles, respectively. They are the main reason of the 20% performance degradation.

Figure 1.5 illustrates the seven-stage superpipeline structure of the SH-X with delayed execution, store buffer, out-of-order branch, and flexible forwarding. Compared with the conventional pipeline shown in Figure 1.4, the INT pipeline starts its execution one-cycle later at the E2 stage, a store data is buffered to the store buffer at the E4 stage and stored to the data cache at the E5 stage, the data transfer of the floating-point unit (FPU) supports

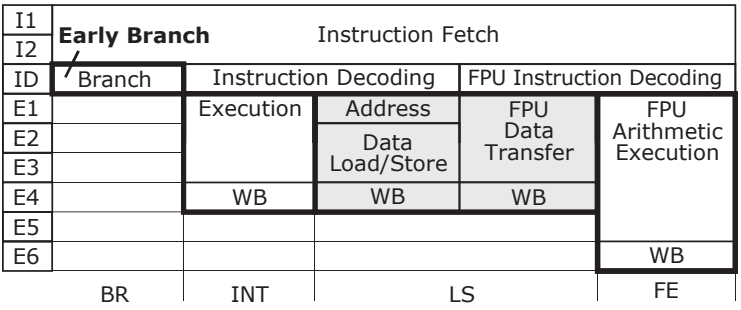

FIGURE 1.4. Conventional seven-stage superpipeline structure.

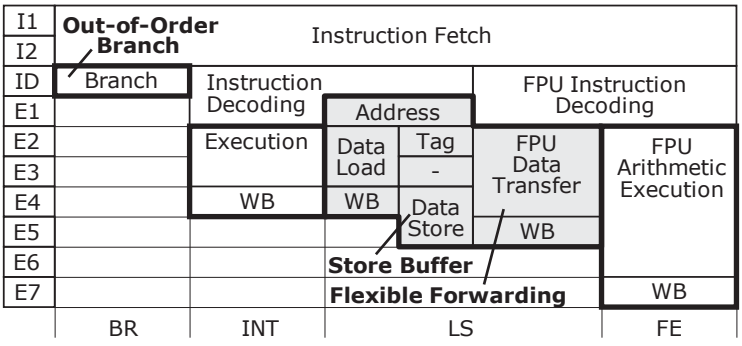

FIGURE 1.5. Seven-stage superpipeline structure of SH-X.

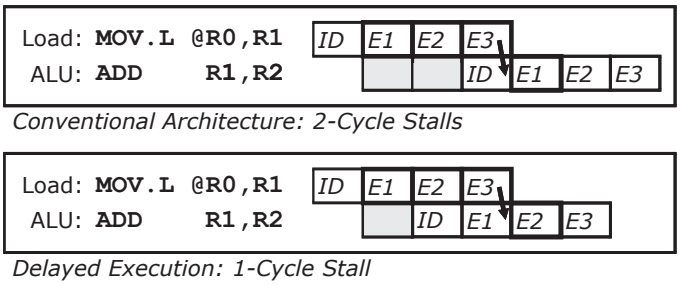

FIGURE 1.6. Load-use conflict reduction by delayed execution.

flexible forwarding. The BR pipeline starts at the ID stage, but is not synchronized to the other pipelines for an out-of-order branch issue.

The delayed execution is effective to reduce the load-use conflict as Figure 1.6 illustrates. It also lengthens the decoding stages into two except for the address calculation, and relaxes the decoding time. With the conventional architecture shown in Figure 1.4, a load instruction, MOV.L, set ups an R0 value at the ID stage, calculates a load address at the E1 stage, loads a data from the data cache at the E2 and E3 stages, and the load data is available at the end of the E3 stage. An ALU instruction, ADD, setups R1 and R2 values at the ID stage, adds the values at the E1 stage. Then the load data is forwarded from the E3 stage to the ID stage, and the pipeline stalls two cycles. With the delayed execution, the load instruction execution is the same, and the add instruction setups R1 and R2 values at E1 stage, adds the values at the E2 stage. Then the load data is forwarded from the E3 stage to the E1 stage, and the pipeline stalls only one cycle. This is the same cycle as those of conventional five-stage pipeline structures.

As illustrated in Figure 1.5, a store instruction performs an address calculation, TLB (translation lookaside buffer) and cache-tag accesses, a store-data latch, and a data store to the cache at the E1, E2, E4, and E5 stages, respectively, whereas a load instruction performs a cache access at the E2 stage. This means the three-stage gap of the cache access timing between the E2 and the E5 stages of a load and a store. However, a load and a store use the same port of the cache. Therefore, a load instruction gets the priority to a store instruction if the access is conflicted, and the store instruction must wait the timing with no conflict. In the N-stage gap case, N entries are necessary for the store buffer to treat the worst case, which is a sequence of N consecutive store issues followed by N consecutive load issues, and the SH-X implemented three entries.

1.3.3 Branch Prediction and Out-of-Order Branch Issue

Figure 1.7 illustrates a branch execution sequence of the SH-X before branch acceleration with a program sequence consisting of compare, conditional-branch, delay-slot, and branch-target instructions.

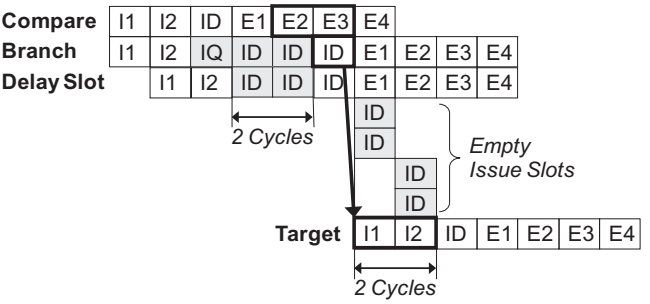

FIGURE 1.7. Branch execution sequence before branch acceleration.

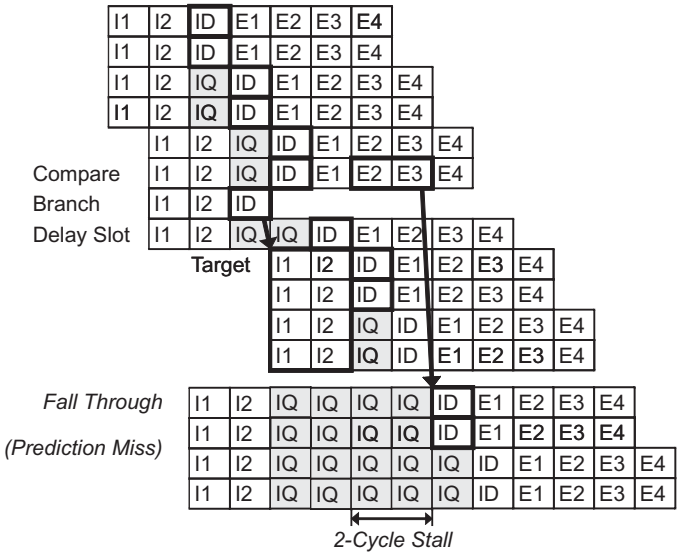

FIGURE 1.8. Branch execution sequence of SH-X.

The conditional-branch and delay-slot instructions are issued three cycles after the compare instruction issue, and the branch-target instruction is issued three cycles after the branch issue. The compare operation starts at the E2 stage by the delayed execution, and the result is available at the middle of the E3 stage. Then the conditional-branch instruction checks the result at the latter half of the ID stage, and generates the target address at the same ID stage, followed by the I1 and I2 stages of the target instruction. As a result, eight empty issue slots or four stall cycles are caused as illustrated. This means only one-third of the issue slots are used for the sequence.

Figure 1.8 illustrates the execution sequence of the SH-X after branch acceleration. The branch operation can start with no pipeline stall by a branch prediction, which predicts the branch direction that the branch is taken or not

taken. However, this is not early enough to make the empty issue slots zero. Therefore, the SH-X adopted an out-of-order issue to the branches using no general-purpose register.

The SH-X fetches four instructions per cycle, and issues two instructions at most. Therefore, Instructions are buffered in an instruction queue (IQ) as illustrated. A branch instruction is searched from the IQ or an instruction-cache output at the I2 stage and provided to the ID stage of the branch pipeline for the out-of-order issue earlier than the other instructions provided to the ID stage in order. Then the conditional branch instruction is issued right after it is fetched while the preceding instructions are in the IQ, and the issue becomes early enough to make the empty issue slots zero. As a result, the target instruction is fetched and decoded at the ID stage right after the delay-slot instruction. This means no branch penalty occurs in the sequence when the preceding or delay-slot instructions stay two or more cycles in the IQ.

The compare result is available at the E3 stage, and the prediction is checked if it is hit or miss. In the miss case, the instruction of the correct flow is decoded at the ID stage right after the E3 stage, and two-cycle stall occurs. If the correct flow is not held in the IQ, the miss-prediction recovery starts from the I1 stage, and takes two more cycles.

Historically, the dynamic branch prediction method started from a BHT with 1-bit history per entry, which recorded a branch direction of taken or not for the last time, and predicted the same branch direction. Then, a BHT with 2-bit history per entry became popular, and the four direction states of strongly taken, weekly taken, weekly not taken, and strongly not taken were used for the prediction to reflect the history of several times. There were several types of the state transitions, including a simple up-down transition. Since each entry held only one or two bits, it is too expensive to attach a tag consisting of a part of the branch-instruction address, which was usually about 20 bits for a 32-bit addressing. Therefore, we could increase the number of entries about ten or twenty times without the tag. Although the different branch instructions could not be distinguished without the tag and there occurred a false hit, the merit of the entry increase exceeded the demerit of the false hit. A global history method was also popular for the prediction, and usually used with the 2-bit/entry BHT.

The SH-X stalled only two cycles for the prediction miss, and the performance was not so sensitive to the hit ratio. Further, the one-bit method required a state change only for a prediction miss, and it could be done during the stall. Therefore, the SH-X adopted a dynamic branch prediction method with a 4K-entry 1-bit/entry BHT and a global history. The size was much smaller than the instruction and data caches of 32 kB each.

1.3.4 Low Power Technologies

The SH-X achieved excellent power efficiency by using various low-power technologies. Among them, hierarchical clock gating and pointer-controlled

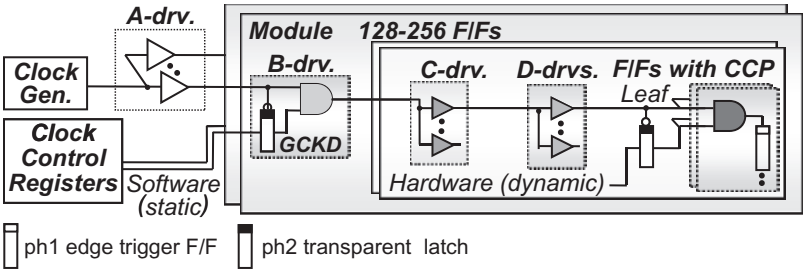

FIGURE 1.9. Conventional clock-gating method. CCP, control clock pin; GCKD, gated clock driver cell.

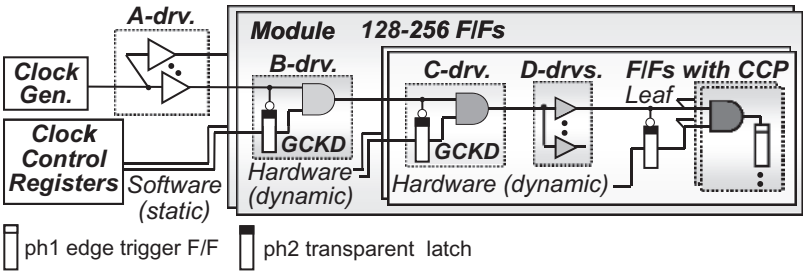

FIGURE 1.10. Clock-gating method of SH-X. CCP, control clock pin; GCKD, gated clock driver cell.

pipeline are explained in this section. Figure 1.9 illustrates a conventional clock-gating method. In this example, the clock tree has four levels with A-, B-, C-, and D-drivers. The A-driver receives the clock from the clock generator, and distributes the clock to each module in the processor. Then, the B-driver of each module receives the clock and distributes it to various submodules, including 128–256 flip-flops (F/Fs). The B-driver gates the clock with the signal from the clock control register, whose value is statically written by software to stop and start the modules. Next, the C- and D-drivers distribute the clock hierarchically to the leaf F/Fs with a Control Clock Pin (CCP). The leaf F/Fs are gated by hardware with the CCP to avoid activating them unnecessarily. However, the clock tree in the module is always active while the module is activated by software.

Figure 1.10 illustrates the clock-gating method of the SH-X. In addition to the clock gating at the B-driver, the C-drivers gate the clock with the signals dynamically generated by hardware to reduce the clock tree activity. As a result, the clock power is 30% less than that of the conventional method.

The superpipeline architecture improved operating frequency, but increased number of F/Fs and power. Therefore, one of the key design considerations

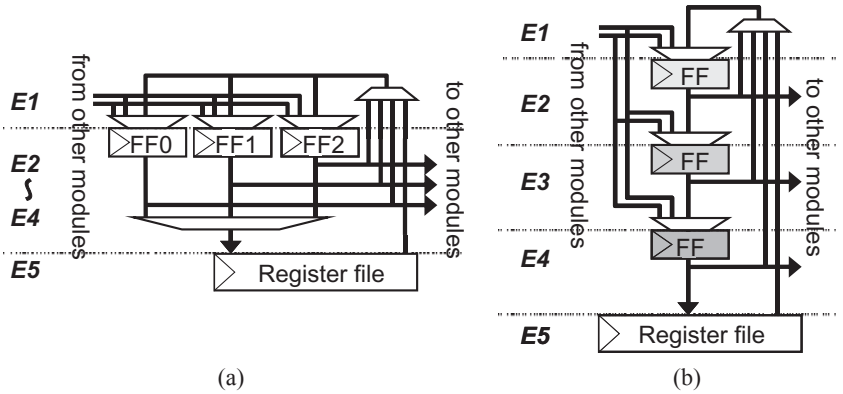

FIGURE 1.11. F/Fs of (a) pointer-controlled and (b) conventional pipelines.

TABLE 1.4. Relationship of F/Fs and Pipeline Stages

Pointer	FF0	FF1	FF2
0	E2	E4	E3
1	E3	E2	E4
2	E4	E3	E2

was to reduce the activity ratio of the F/Fs. To address this issue, a pointer-controlled pipeline was developed. It realizes a pseudo pipeline operation with a pointer control. As shown in Figure 1.11a, three pipeline F/Fs are connected in parallel, and the pointer is used to show which F/Fs correspond to which stages. Then, only one set of F/Fs is updated in the pointer-controlled pipeline, while all pipeline F/Fs are updated every cycle in the conventional pipeline, as shown in Figure 1.11b.

Table 1.4 shows the relationship between F/Fs FF0-FF2 and pipeline stages E2-E4 for each pointer value. For example, when the pointer indexes zero, the FF0 holds an input value at E2 and keeps it for three cycles as E2, E3, and E4 latches until the pointer indexes zero again and the FF0 holds a new input value. This method is good for a short latency operation in a long pipeline. The power of pipeline F/Fs decreases to 1/3 for transfer instructions, and decreases by an average of 25% as measured using Dhrystone 2.1.

1.3.5 Performance and Efficiency Evaluations

The SH-X performance was measured using the Dhrystone 2.1 benchmark, as well as those of the SH-3 and the SH-4. The Dhrystone is a popular benchmark

for evaluating integer performance of embedded processors. It is small enough to fit all the program and data into the caches, and to use at the beginning of the processor development. Therefore, only the processor core architecture can be evaluated without the influence from the system level architecture, and the evaluation result can be fed back to the architecture design. On the contrary, the system level performance cannot be measured considering cache miss rates, external memory access throughput and latencies, and so on. The evaluation result includes compiler performance because the Dhrystone benchmark is described in C language.

Figure 1.12 shows the evaluated result of the cycle performance, architectural performance, and actual performance. Starting from the SH-3, five major enhancements were adopted to construct the SH-4 microarchitecture. The SH-3 achieved 1.0 MIPS/MHz when it was released, and the SH-4 compiler enhanced its performance to 1.1. The cycle performance of the SH-4 was enhanced to 1.81 MIPS/MHz by Harvard architecture, superscalar architecture, adding BO group, early-stage branch, and zero-cycle MOV operation. The SH-4 enhanced the cycle performance by 1.65 times form the SH-3, excluding the compiler contribution. The SH-3 was a 60-MHz processor in a 0.5-μm process, and estimated to be a 133-MHz processor in a 0.25-μm process. The SH-4 achieved 200 MHz in the same 0.25-μm process. Therefore, SH-4 enhanced the frequency by 1.5 times form the SH-3. As a result, the

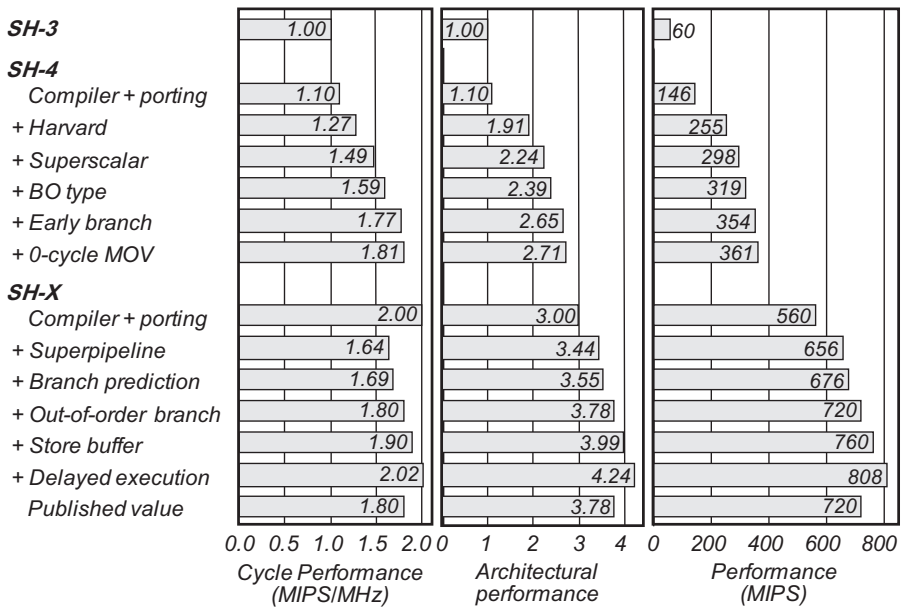

FIGURE 1.12. Performance improvement of SH-4 and SH-X.

architectural performance of the SH-4 is 1.65 × 1.5 = 2.47 times as high as that of the SH-3.

With adopting a conventional seven-stage superpipeline, the performance was decreased by 18% to 1.47 MIPS/MHz. Branch prediction, out-of-order branch issue, store buffer and delayed execution of the SH-X improve the cycle performance by 23%, and recover the 1.8 MIPS/MHz. Since 1.4 times high operating frequency was achieved by the superpipeline architecture, the architectural performance of the SH-X was also 1.4 times as high as that of the SH-4. The actual performance of the SH-X was 720 MIPS at 400 MHz in a 0.13-μm process, and improved by two times from the SH-4 in a 0.25-μm process.

Figures 1.13 and 1.14 show the area and power efficiency improvements, respectively. The upper three graphs of both the figures show architectural performance, relative area/power, and architectural area–/power–performance ratio. The lower three graphs show actual performance, area/power, and area–/power–performance ratio.

The area of the SH-X core was 1.8 mm^2 in a 0.13-μm process, and the area of the SH-4 was estimated as 1.3 mm^2 if it was ported to a 0.13-μm process. Therefore, the relative area of the SH-X was 1.4 times as much as that of the SH-4, and 2.26 times as much as the SH-3. Then, the architectural area efficiency of the SH-X was nearly equal to that of the SH-4, and 1.53 times as high as the SH-3. The actual area efficiency of the SH-X reached 400 MIPS/mm^2, which was 8.5 times as high as the 74 MIPS/ mm^2 of the SH-4.

SH-4 was estimated to achieve 200 MHz, 360 MIPS with 140 mW at 1.15 V, and 280 MHz, 504 MIPS with 240 mW at 1.25 V. The power efficiencies were 2,500 and 2,100 MIPS/W, respectively. On the other hand, SH-X achieved 200 MHz, 360 MIPS with 80 mW at 1.0 V, and 400 MHz, 720 MIPS with

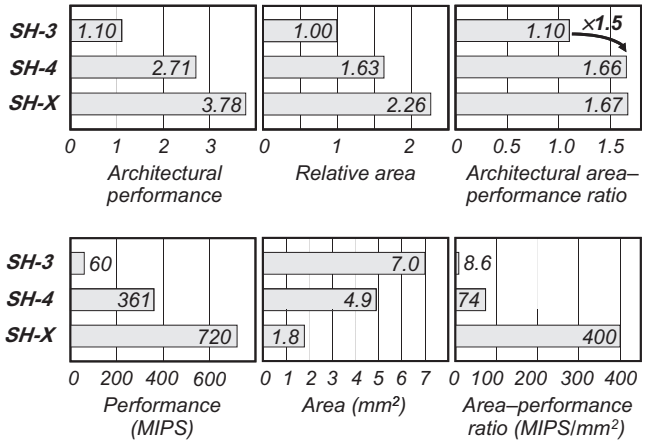

FIGURE 1.13. Area efficiency improvement of SH-4 and SH-X.

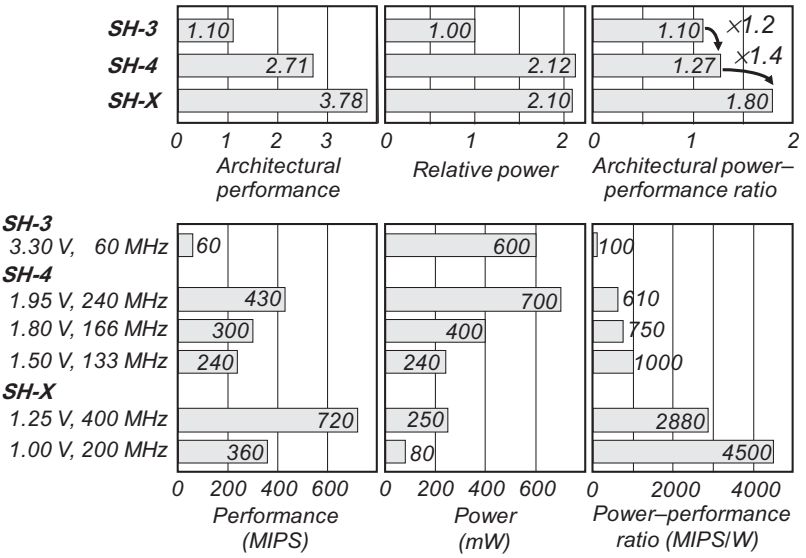

FIGURE 1.14. Power efficiency improvement of SH-4 and SH-X.

250 mW at 1.25 V. The power efficiencies were 4,500 and 2,880 MIPS/W, respectively. As a result, the power efficiency of the SH-X improved by 1.8 times from that of the SH-4 at the same frequency of 200 MHz, and by 1.4 times at the same supply voltage with enhancing the performance by 1.4 times. These were architectural improvements, and actual improvements were multiplied by the process porting.

1.4 SH-X FPU: A HIGHLY EFFICIENT FPU

The floating-point architecture and microarchitecture of the SH processors achieve high multimedia performance and efficiency. An FPU of the SH processor is highly parallel with keeping the efficiency for embedded systems in order to compensate the insufficient parallelism of the dual-issue superscalar architecture for highly parallel applications like 3D graphics.

In late 1990s, it became difficult to support higher resolution and advanced features of the 3D graphics. It was especially difficult to avoid overflow and underflow of fixed-point data with small dynamic range, and there was a demand to use floating-point data. Since it was easy to implement a four-way parallel operation with fixed-point data, equivalent performance had to be realized to change the data type to the floating-point format at reasonable costs.

Since an FPU was about three times as large as a fixed-point unit, and a four-way SMID required four times as large a datapath, it was too expensive

to integrate a four-way SMID FPU. The latency of the floating-point operations was long, and required more number of registers than the fixed-point operations. Therefore, efficient parallelization and latency-reduction methods had to be developed.

1.4.1 FPU Architecture of SH Processors

Sixteen is the limit of the number of registers directly specified by the 16-bit fixed-length ISA, but the SH FPU architecture defines 32 registers as two banks of 16 registers. The two banks are front and back banks, named FR0-FR15 and XF0-XF15, respectively, and they are switched by changing a control bit FPSCR.FR in a floating-point status and control register (FPSCR). Most of instructions use only the front bank, but some instructions use both the front and back banks. The front bank registers are used as eight pairs or four length-4 vectors as well as 16 registers, and the back bank registers are used as eight pairs or a four-by-four matrix. They are defined as follows:

$$\mathrm{DR}n = (\mathrm{FR}n, \mathrm{FR}[n+1])(n\colon 0, 2, 4, 6, 8, 10, 12, 14),$$

$$\mathrm{FV0} = \begin{pmatrix} FR0 \\ FR1 \\ FR2 \\ FR3 \end{pmatrix}, \mathrm{FV4} = \begin{pmatrix} FR4 \\ FR5 \\ FR6 \\ FR7 \end{pmatrix}, \mathrm{FV8} = \begin{pmatrix} FR8 \\ FR9 \\ FR10 \\ FR11 \end{pmatrix}, \mathrm{FV12} = \begin{pmatrix} FR12 \\ FR13 \\ FR14 \\ FR15 \end{pmatrix},$$

$$\mathrm{XD}n = (\mathrm{XF}n, \mathrm{XF}[n+1])(n\colon 0, 2, 4, 6, 8, 10, 12, 14),$$

$$\mathrm{XMTRX} = \begin{pmatrix} XF0 & XF4 & XF8 & XF12 \\ XF1 & XF5 & XF9 & XF13 \\ XF2 & XF6 & XF10 & XF14 \\ XF3 & XF7 & XF11 & XF15 \end{pmatrix}.$$

Since an ordinary SIMD architecture of an FPU is too expensive for an embedded processor as described above, another parallelism is applied to the SH processors. The large hardware of an FPU is for a mantissa alignment before the calculation and normalization and rounding after the calculation. Further, a popular FPU instruction, FMAC, requires three read and one write ports. The consecutive FMAC operations are a popular sequence to accumulate plural products. For example, an inner product of two length-4 vectors is one of such sequences, and popular in a 3D graphics program. Therefore, a floating-point inner-product instruction (FIPR) is defined to accelerate the sequence with smaller hardware than that for the SIMD. It uses the two of four length-4 vectors as input operand, and modifies the last register of one of the input vectors to store the result. The defining formula is as follows:

$$\mathrm{FR}[n+3] = \mathrm{FV}m \cdot \mathrm{FV}n(m, n\colon 0, 4, 8, 12).$$

This modifying-type definition is similar to the other instructions. However, for a length-3 vector operation, which is also popular, you can get the result without destroying the inputs, by setting one element of the input vectors to zero.

The FIPR produces only one result, which is one-fourth of a four-way SIMD, and can save the normalization and rounding hardware. It requires eight input and one output registers, which are less than the 12 input and four output registers for a four-way SIMD FMAC. Further, the FIPR takes much shorter time than the equivalent sequence of one FMUL and three FMACs, and requires small number of registers to sustain the peak performance. As a result, the hardware is about half of the four-way SIMD.

The rounding rule of the conventional floating-point operations is strictly defined by an American National Standards Institute/Institute of Electrical and Electronics Engineers (ANSI/IEEE) 754 floating-point standard. The rule is to keep accurate values before rounding. However, each instruction performs the rounding, and the accumulated rounding error sometimes becomes very serious. Therefore, a program must avoid such a serious rounding error without relying to hardware if necessary. The sequence of one FMUL and three FMACs can also cause a serious rounding error. For example, the following formula results in zero if we add the terms in the order of the formula by FADD instructions:

$$1.0\times 2^{127} + 1.FFFFFE\times 2^{102} + 1.FFFFFE\times 2^{102} - 1.0\times 2^{127}.$$

However, the exact value is $1.\text{FFFFFE} \times 2^{103}$, and the error is $1.\text{FFFFFE} \times 2^{103}$ for the formula, which causes the worst error of 2^{-23} times of the maximum term. We can get the exact value if we change the operation order properly. The floating-point standard defines the rule of each operation, but does not define the result of the formula, and either of the result is fine for the conformance. Since the FIPR operation is not defined by the standard, we defined its maximum error as "2^{E-25} + rounding error of result" to make it better than or equal to the average and worst-case errors of the equivalent sequence that conforms the standard, where E is the maximum exponent of the four products.

A length-4 vector transformation is also popular operation of a 3D graphics, and a floating-point transform vector instruction (FTRV) is defined. It requires 20 registers to specify the operands in a modification type definition. Therefore, the defining formula is as follows, using a four-by-four matrix of all the back bank registers, XMTRX, and one of the four front-bank vector registers, FV0-FV3:

$$\text{FV}n = \text{XMTRX}\cdot\text{FV}n(n\text{: } 0, 4, 8, 12).$$

Since a 3D object consists of a lot of polygons expressed by the length-4 vectors, and one XMTRX is applied to a lot of the vectors of a 3D object, the

XMTRX is not so often changed, and is suitable for using the back bank. The FTRV operation is implemented as four inner-product operations by dividing the XMTRX into four vectors properly, and its maximum error is the same as the FIPR.

The newly defined FIPR and FTRV can enhance the performance, but data transfer ability becomes a bottleneck to realize the enhancement. Therefore, a pair load/store/transfer mode is defined to double the data move ability. In the pair mode, floating-point move instructions (FMOVs) treat 32 front- and back-bank floating-point registers as 16 pairs, and directly access all the pairs without the bank switch controlled by the FPSCR.FR bit. The mode switch between the pair and normal modes is controlled by a move-size bit FPSCR. SZ in the FPSCR.

The 3D graphics requires high performance but uses only a single precision. On the other hand, a double precision format is popular for server/PC market, and would eases a PC application porting to a handheld PC. Although the performance requirement is not so high as the 3D graphics, software emulation is too slow compared with hardware implementation. Therefore, the SH architecture has single- and double-precision modes, which are controlled by a precision bit FPSCR.PR of the FPSCR. Further, a floating-point register-bank, move-size, and precision change instructions (FPCRG, FSCHG, and FRCHG) were defined for fast changes of the modes defined above. This definition can save the small code space of the 16-bit fixed length ISA. Some conversion operations between the precisions are necessary, but not fit to the mode separation. Therefore, SH architecture defines two conversion instructions in the double-precision mode. An FCNVSD converts a single-precision data to a double-precision one, and an FCNVDS converts vice versa. In the double-precision mode, eight pairs of the front-bank registers are used for double-precision data, and one 32-bit register, FPUL, is used for a single-precision or integer data, mainly for the conversion.

The FDIV and floating-point square-root instruction (FSQRT) are long latency instructions, and could cause serious performance degradations. The long latencies are mainly from the strict operation definitions by the ANSI/IEEE 754 floating-point standard. We have to keep accurate value before rounding. However, there is another way if we allow proper inaccuracies.

A floating-point square-root reciprocal approximate (FSRRA) is defined as an elementary function instruction to replace the FDIV, FSQRT, or their combination. Then we do not need to use the long latency instructions. 3D graphics applications especially require a lot of reciprocal and square-root reciprocal values, and the FSRRA is highly effective. Further, 3D graphics require less accuracy, and the single-precision without strict rounding is enough accuracy. The maximum error of the FSRRA is $\pm 2^{E-21}$, where E is the exponent value of an FSRRA result. The FSRRA definition is as follows:

$$\mathrm{FR}n = \frac{1}{\sqrt{\mathrm{FR}n}}.$$

A floating-point sine and cosine approximate (FSCA) is defined as another popular elementary function instruction. Once the FSRRA is introduced, extra hardware is not so large for the FSCA. The most poplar definition of the trigonometric function is to use radian for the angular unit. However, the period of the radian is 2π, and cannot be expressed by a simple binary number. Therefore, the FSCA uses fixed-point number of rotations as the angular expression. The number consists of 16-bit integer and 16-bit fraction parts. Then the integer part is not necessary to calculate the sine and cosine values by their periodicity, and the 16-bit fraction part can express enough resolution of 360/65,536 = 0.0055°. The angular source operand is set to a CPU-FPU communication register FPUL because the angular value is a fixed-point number. The maximum error of the FSCA is $\pm 2^{-22}$, which is an absolute value and not related to the result value. Then the FSCA definition is as follows:

$$\mathrm{FR}n = \sin(2\pi \cdot \mathrm{FPUL}),\ \mathrm{FR}[n+1] = \cos(2\pi \cdot \mathrm{FPUL}).$$

1.4.2 Implementation of SH-X FPU

Table 1.5 shows the pitches and latencies of the FE-category instructions of the SH-3E, SH-4, and SH-X. As for the SH-X, the simple single-precision instructions of FADD, FSUB, FLOAT, and FTRC, have three-cycle latencies. Both single- and double-precision FCMPs have two-cycle latencies. Other single-precision instructions of FMUL, FMAC, and FIPR, and the double-precision instructions, except FMUL, FCMP, FDIV, and FSQRT, have five-cycle latencies. All the above instructions have one-cycle pitches.

The FTRV consists of four FIPR like operations resulting in four-cycle pitch and eight-cycle latency. The FDIV and FSQRT are out-of-order completion instructions having two-cycle pitches for the first and last cycles to initiate a special resource operation and to perform postprocesses of the result. Their pitches of the special resource expressed in the parentheses are about halves of the mantissa widths, and the latencies are four cycles more than the special-resource pitches. The FSRRA has one-cycle pitch, three-cycle pitch of the special resource, and five-cycle latency. The FSCA has three-cycle pitch, five-cycle pitch of the special resource, and seven-cycle latency. The double-precision FMUL has three-cycle pitch and seven-cycle latency.

Multiply–accumulate (MAC) is one of the most frequent operations in intensive computing applications. The use of four-way SIMD can achieve the same throughput as the FIPR, but the latency is longer and the register file has to be larger. Figure 1.15 illustrates an example of the differences according to the pitches and latencies of the FE-category SH-X instructions shown in Table 1.5. In this example, each box shows an operation issue slot. Since FMUL and FMAC have five-cycle latencies, we must issue 20 independent

TABLE 1.5. Pitch/Latency of FE-Category Instructions

Single Precision	SH-3E	SH-4	SH-X
FADD FR*m*, FR*n*	1/2	1/3	1/3
FSUB FR*m*, FR*n*	1/2	1/3	1/3
FMUL FR*m*, FR*n*	1/2	1/3	1/5
FDIV FR*m*, FR*n*	13/14	2 (10) /12	2 (13) /17
FSQRT FR*n*	13/14	2 (9) /11	2 (13) /17
FCMP/EQ FR*m*, FR*n*	1/1	1/2	1/2
FCMP/GT FR*m*, FR*n*	1/1	1/2	1/2
FLOAT FPUL, FR*n*	1/2	1/3	1/3
FTRC FR*m*, FPUL	1/2	1/3	1/3
FMAC FR0, FR*m*, FR*n*	1/2	1/3	1/5
FIPR FV*m*, FV*n*, FR*n* + 3	–	1/4	1/5
FTRV XMTRX, FV*n*	–	4/7	4/8
FSRRA FR*n*	–	–	1 (3) /5
FSCA FPUL, DR*n*	–	–	3 (5) /7
Double Precision	–	SH-4	SH-X
FADD DR*m*, DR*n*	–	6/8	1/5
FSUB DR*m*, DR*n*	–	6/8	1/5
FMUL DR*m*, DR*n*	–	6/8	3/7
FDIV DR*m*, DR*n*	–	5 (23) /25	2 (28) /32
FSQRT DR*m*, DR*n*	–	5 (22) /24	2 (28) /32
FCMP/EQ DR*m*,DR*n*	–	2/2	1/2
FCMP/EQ DR*m*,DR*n*	–	2/2	1/2
FLOAT DR*n*	–	2/4	1/5
FTRC DR*m*, FPUL	–	2/4	1/5
FCNVSD FPUL, FR*n*	–	2/4	1/5
FCNVDS DR*m*, FPUL	–	2/4	1/5

operations for peak throughput in the case of four-way SIMD. The result is available 20 cycles after the FMUL issue. On the other hand, five independent operations are enough to get the peak throughput of a program using FIPRs. Therefore, FIPR requires one-quarter of the program's parallelism and registers.

Figure 1.16 compares the pitch and latency of an FSRRA and the equivalent sequence of an FSQRT and an FDIV according to Table 1.5. Each of the FSQRT and FDIV occupies 2 and 13 cycles of the MAIN FPU and special resources, respectively, and takes 17 cycles to get the result, and the result is available 34 cycles after the issue of the FSQRT. In contrast, the pitch and latency of the FSRRA are one and five cycles that are only one-quarter and approximately one-fifth of those of the equivalent sequences,

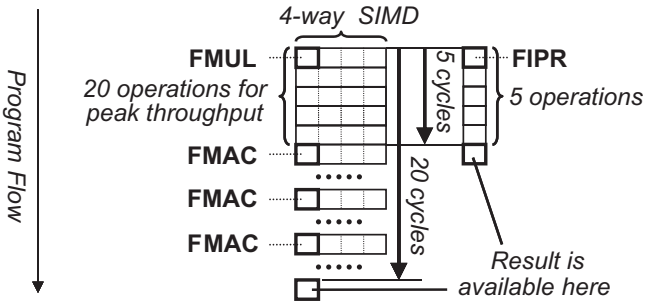

FIGURE 1.15. Four-way SIMD versus FIPR.

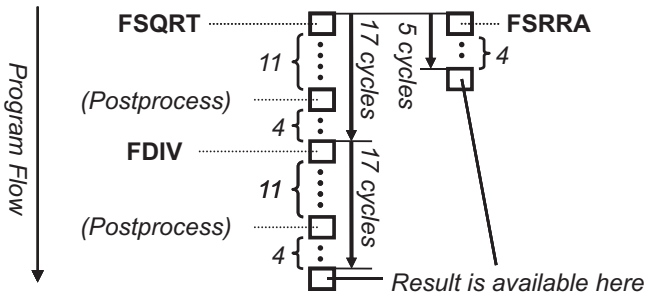

FIGURE 1.16. FSRRA versus equivalent sequence of FSQRT and FDIV.

respectively. The FSRRA is much faster using a similar amount of the hardware resource.

The FSRRA can compute a reciprocal as shown in Figure 1.17. The FDIV occupies 2 and 13 cycles of the MAIN FPU and special resources, respectively, and takes 17 cycles to get the result. On the other hand, the FSRRA and FMUL sequence occupies 2 and 3 cycles of the MAIN FPU and special resources, respectively, and takes 10 cycles to get the result. Therefore, the FSRRA and FMUL sequence is better than using the FDIV if an application does not require a result conforming to the IEEE standard, and 3D graphics is one of such applications.

Figure 1.18 illustrates the FPU arithmetic execution pipeline. With the delayed execution architecture, the register-operand read and forwarding are done at the E1 stage, and the arithmetic operation starts at E2. The short arithmetic pipeline treats three-cycle latency instructions. All the arithmetic pipelines share one register write port to reduce the number of ports. There are four forwarding source points to provide the specified latencies for any

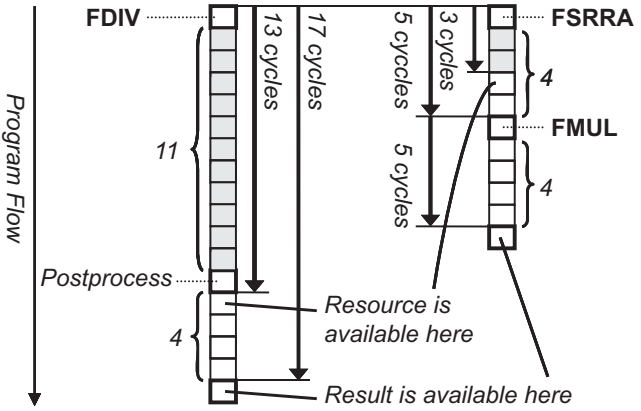

FIGURE 1.17. FDIV versus equivalent sequence of FSRRA and FMUL.

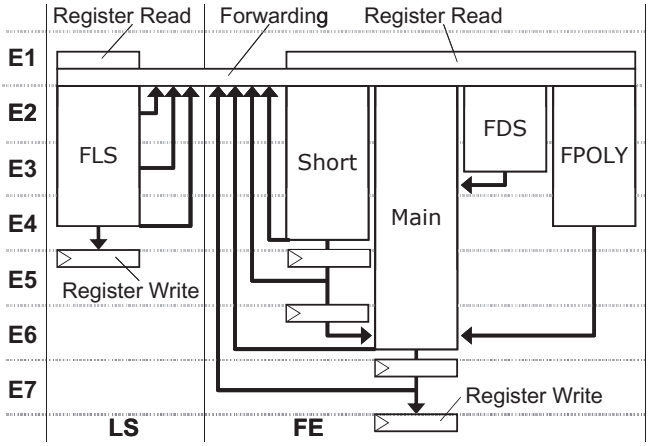

FIGURE 1.18. Arithmetic execution pipeline of SH-X FPU.

cycle distance of the define-and-use instructions. The FDS pipeline is occupied by 13/28 cycles to execute a single/double FDIV or FSQRT, and these instructions cannot be issued frequently. The FPOLY pipeline is three-cycles long and is occupied three or five times to execute an FSRRA or FSCA instruction. Therefore, the third E4 stage and E6 stage of the main pipeline are synchronized for the FSRRA, and the FPOLY pipeline output merges with the main pipeline at this point. The FSCA produce two outputs, and the first output is produced at the same timing of the FSRRA, and the second one is produced two-cycle later, and the main pipeline is occupied for three cycles, although

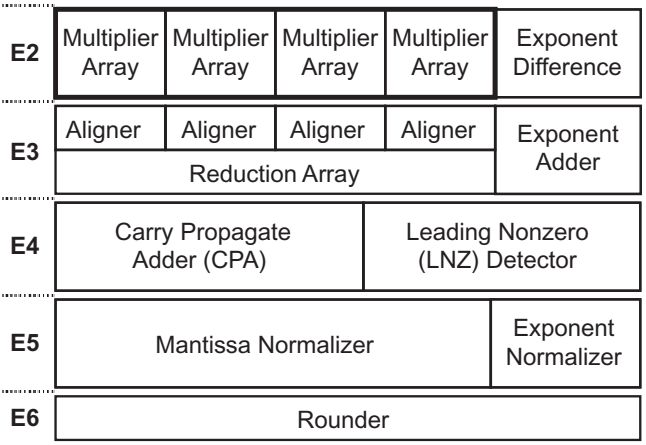

FIGURE 1.19. Main pipeline of SH-X FPU.

the second cycle is not used. The FSRRA and FSCA are implemented by calculating the cubic polynomials of the properly divided periods. The width of the third-order term is 8 bits, which adds only a small area overhead, while enhancing accuracy and reducing latency.

Figure 1.19 illustrates the structure of the main FPU pipeline. There are four single-precision multiplier arrays at E2 to execute FIPR and FTRV and to emulate double-precision multiplication. Their total area is less than that of a double-precision multiplier array. The calculation of exponent differences is also done at E2 for alignment operations by four aligners at E3. The four aligners align eight terms consisting of four sets of sum and carry pairs of four products generated by the four multiplier arrays, and a reduction array reduces the aligned eight terms to two at E3. The exponent value before normalization is also calculated by an exponent adder at E3. A carry propagate adder (CPA) adds two terms from the reduction array, and a leading nonzero (LNZ) detector searches the LNZ position of the absolute value of the CPA result from the two CPA inputs precisely and with the same speed as the CPA at E4. Therefore, the result of the CPA can be normalized immediately after the CPA operation with no correction of position errors, which is often necessary when using a conventional 1-bit error LNZ detector. Mantissa and exponent normalizers normalize the CPA and exponent-adder outputs at E5 controlled by the LNZ detector output. Finally, the rounder rounds the normalized results into the ANSI/IEEE 754 format. The extra hardware required for the special FPU instructions of the FIPR, FTRV, FSRRA and FSCA is about 30% of the original FPU hardware, and the FPU area is about 10–20% of the processor core depending on the size of the first and second on-chip memories. Therefore, the extra hardware is about 3–6% of the processor core.

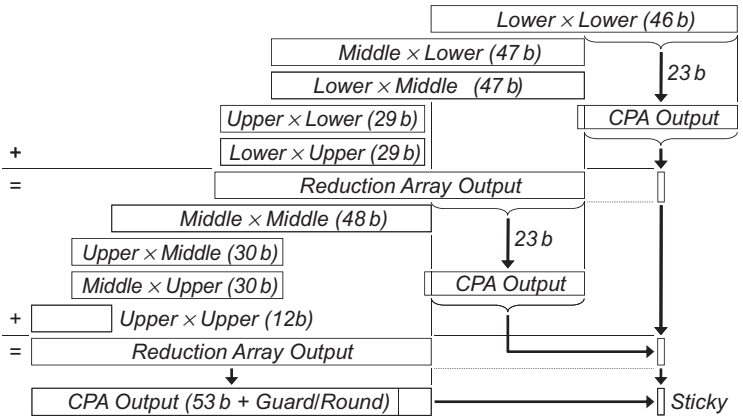

FIGURE 1.20. Double-precision FMUL emulation by four multipliers.

The SH-X FPU can use four 24-by-24 multipliers for the double-precision FMUL emulation. Since the double-precision mantissa width is more than twice of the single-precision one, we have to divide a multiplication into nine parts. Then we need three cycles to emulate the nine partial multiplications by four multipliers. Figure 1.20 illustrates the flow of the emulation. At the first step, a lower-by-lower product is produced, and its lower 23 bits are added by the CPA. Then the CPA output is ORed to generate a sticky bit. At the second step, four products of middle-by-lower, lower-by-middle, upper-by-lower, and lower-by-upper are produced and accumulated to the lower-by-lower product by the reduction array, and its lower 23 bits are also used to generate a sticky bit. At the third step, the remaining four products of middle-by-middle, upper-by-middle, middle-by-upper, and upper-by-upper are produced and accumulated to the already accumulated intermediate values. Then, the CPA adds the sum and carry of the final product, and 53-bit result and guard/round/sticky bits are produced. The accumulated terms of the second and third steps are 10 because each product consists of sum and carry, but the bitwise position of some terms are not overlapped. Therefore, the eight-term reduction array is enough to accumulate them.

1.4.3 Performance Evaluations with 3D Graphics Benchmark

The floating-point architecture was evaluated by a simple 3D graphics benchmark shown in Figure 1.21. It consists of coordinate transformations, perspective transformations, and intensity calculations of a parallel beam of light in Cartesian coordinates. A 3D-object surface is divided into triangular polygons to be treated by the 3D graphics. The perspective transformation assumes a flat screen expressed as $z = 1$. A strip model is used, which is a 3D object expression method to reduce the number of vertex vectors. In the model, each

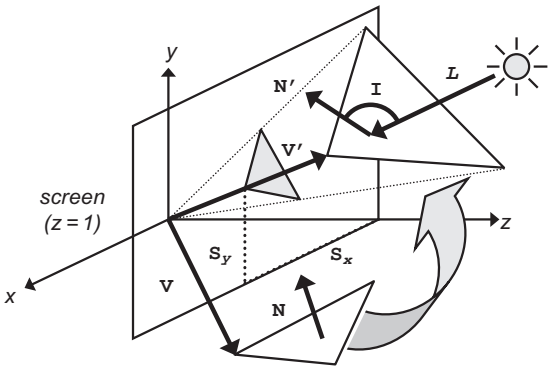

FIGURE 1.21. Simple 3D graphics benchmark.

triangle has three vertexes, but each vertex is shared by three triangles, and the number of vertex per triangle is one. The benchmark is expressed as follows, where T represents a transformation matrix, V and N represent vertex and normal vectors of a triangle before the coordinate transformations, respectively, N′ and V′ represent the ones after the transformations, respectively, S_x and S_y represent x and y coordinates of the projection of V′, respectively, L represents a vector of the parallel beam of light, I represents a intensity of a triangle surface, and V″ is an intermediate value of the coordinate transformations.

$$V'' = TV,\ V' = \frac{V''}{V''_w},\ S_x = \frac{V'_x}{V'_z},\ S_y = \frac{V'_y}{V'_z},\ N' = TN,\ I = \frac{(L, N')}{\sqrt{(N', N')}},$$

$$T = \begin{pmatrix} T_{xx} & T_{xy} & T_{xz} & T_{xw} \\ T_{yx} & T_{yy} & T_{yz} & T_{yw} \\ T_{zx} & T_{zy} & T_{zz} & T_{zw} \\ T_{wx} & T_{wy} & T_{wz} & T_{ww} \end{pmatrix},\ V = \begin{pmatrix} V_x \\ V_y \\ V_z \\ 1 \end{pmatrix},\ V' = \begin{pmatrix} V'_x \\ V'_y \\ V'_z \\ 1 \end{pmatrix},\ V'' = \begin{pmatrix} V''_x \\ V''_y \\ V''_z \\ V''_w \end{pmatrix},$$

$$N = \begin{pmatrix} N_x \\ N_y \\ N_z \\ 0 \end{pmatrix},\ N' = \begin{pmatrix} N'_x \\ N'_y \\ N'_z \\ 0 \end{pmatrix},\ L = \begin{pmatrix} L_x \\ L_y \\ L_z \\ 0 \end{pmatrix}.$$

The coordinate and perspective transformations require 7 FMULs, 12 FMACs, and 2 FDIVs without FTRV, FIPR, and FSRRA, and 1 FTRV, 5 FMULs, and 2 FSRRAs with them. The intensity calculation requires 7 FMULs, 12 FMACs, 1 FSQRT, and 1 FDIV without them, and 1 FTRV, 2 FIPRs, 1 FSRRA, and 1 FMUL with them.

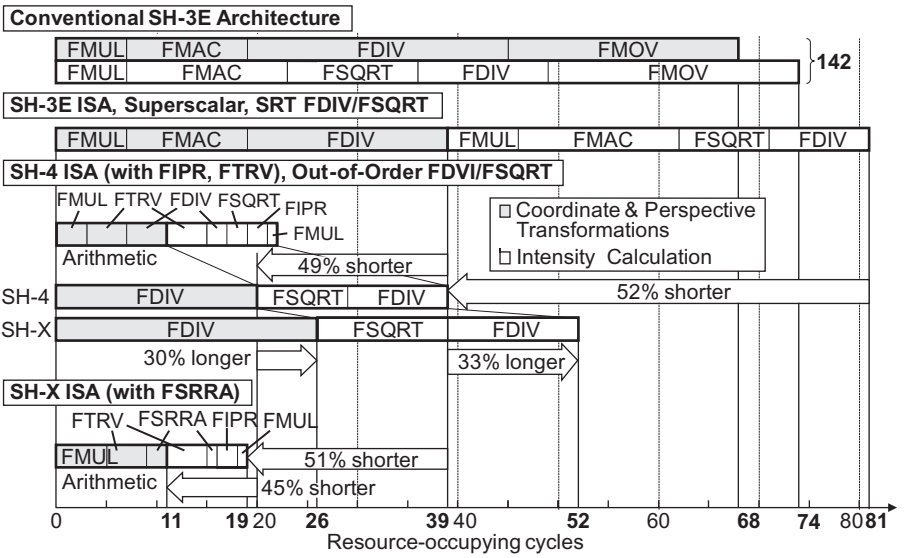

FIGURE 1.22. Resource occupying cycles for a 3D benchmark.

Figure 1.22 shows the resource-occupying cycles of the SH-3E, SH-4, and SH-X. After program optimization, no register conflict occurs, and performance is restricted only by the floating-point resource-occupying cycles. The gray areas of the graph represent the cycles of the coordinate and perspective transformations.

The Conventional SH-3E architecture takes 68 cycles for coordinate and perspective transformations, and 142 cycles when intensity is also calculated. Applying superscalar architecture and SRT method for FDIV/FSQRT with keeping the SH-3E ISA, they become 39 and 81 cycles, respectively. The SH-4 architecture having the FIPR/FTRV and the out-of-order FDIV/FSQRT makes them 20 and 39 cycles, respectively. The performance is good, but only the FDIV/FSQRT resource is busy in this case. Further, applying the super-pipline architecture with keeping the SH-4 ISA, they become 26 and 52 cycles, respectively. Although the operating frequency grows higher by the super-pipline architecture, the cycle performance degradation is serious, and almost no performance gain is achieved. In the SH-X ISA case with the FSRRA, they become 11 and 19 cycles, respectively. Clearly, the FSRRA solves the long pitch problem of the FDIV/FSQRT.

Since we emphasized the importance of the efficiency, we evaluated the area and power efficiencies. Figure 1.23 shows the area efficiencies. The upper half shows architectural performance, relative area, and architectural area–performance ratio to compare the area efficiencies with no process porting effect. According to the above cycles, the relative cycle performance of the coordinate and perspective transformations of the SH-4 and SH-X to the

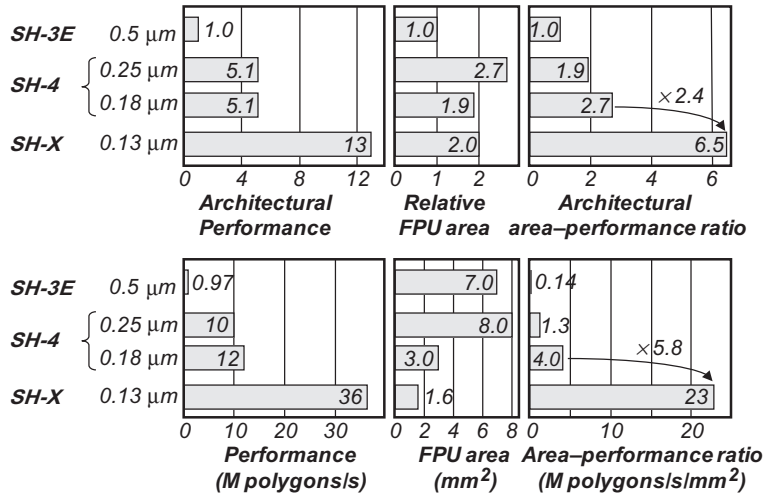

FIGURE 1.23. Area efficiencies of SH-3E, SH-4, and SH-X.

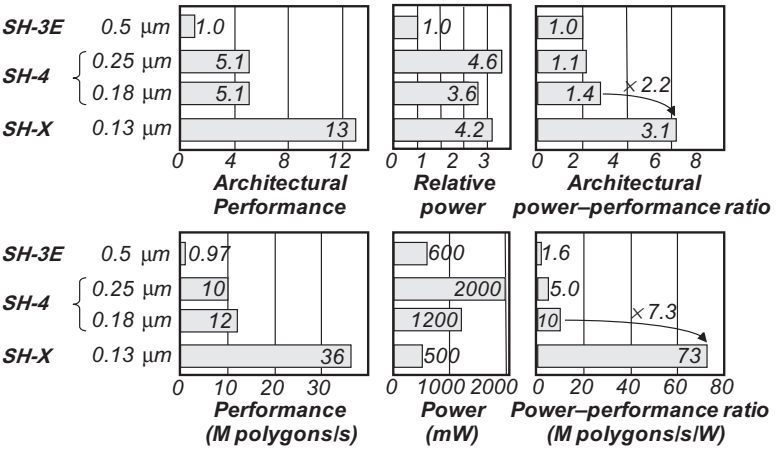

FIGURE 1.24. Power efficiencies of SH-3E, SH-4, and SH-X.

SH-3E are 68/20 = 3.4 and 68/11 = 6.2, respectively. As explained in Section 1.3.5, the relative frequency of the SH-4 and SH-X are 1.5 and 2.1, respectively. Then the architectural performance of the SH-4 and SH-X are 3.4 × 1.5 = 5.1 and 6.2 × 2.1 = 13, respectively. Although the relative areas increased, the performance improvements are much higher, and the efficiency is greatly enhanced. The lower half shows real performance, area, and area–performance ratio. The frequencies of the SH-3E, SH-4 in 0.25- and 0.18-μm and SH-X are 66, 200, 240, and 400 MHz, respectively. The efficiency is further enhanced using the finer process. Similarly, the power efficiency is also enhanced greatly, as shown in Figure 1.24.

1.5 SH-X2: FREQUENCY AND EFFICIENCY ENHANCED CORE

An SH-X2 was developed as the second-generation core, and achieved performance of 1,440 MIPS at 800 MHz using a 90-nm process. The low power version achieved the power efficiency of 6,000 MIPS/W. The performance and efficiency are greatly enhanced from the SH-X by both the architecture and micro-architecture tuning and the process porting.

1.5.1 Frequency Enhancement

According to the SH-X analyzing, the ID stage was the most critical timing part, and the branch acceleration successfully reduced the branch penalty. Therefore, we added the third instruction fetch stage (I3) to the SH-X2 pipeline to relax the ID stage timing. The cycle performance degradation was negligible small by the successful branch architecture, and the SH-X2 achieved the same cycle performance of 1.8 MIPS/MHz as the SH-X.

Figure 1.25 illustrates the pipeline structure of the SH-X2. The I3 stage was added, and performs branch search and instruction predecoding. Then the ID stage timing was relaxed, and the achievable frequency increased.

Another critical timing path was in first-level (L1) memory access logic. SH-X had L1 memories of a local memory and I- and D-caches, and the local memory was unified for both instruction and data accesses. Since all the memories could not be placed closely, a memory separation for instruction and data was good to relax the critical timing path. Therefore, the SH-X2 separated the unified L1 local memory of the SH-X into instruction and data local memories (ILRAM and OLRAM). With the other various timing tuning, the SH-X2 achieved 800 MHz using a 90-nm generic process from the SH-X's 400 MHz using a 130-nm process. The improvement was far higher than the process porting effect.

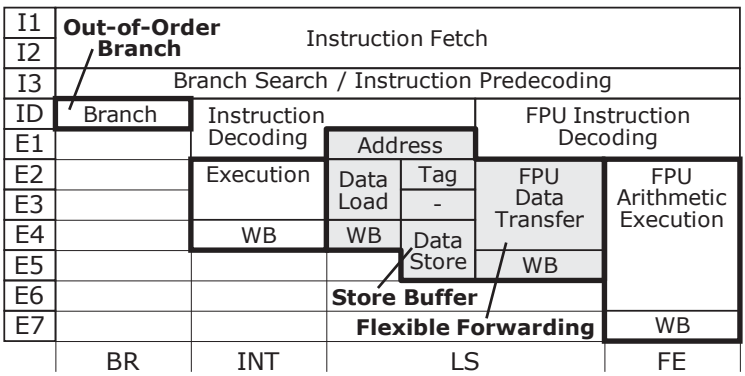

FIGURE 1.25. Eight-stage superpipeline structure of SH-X2.

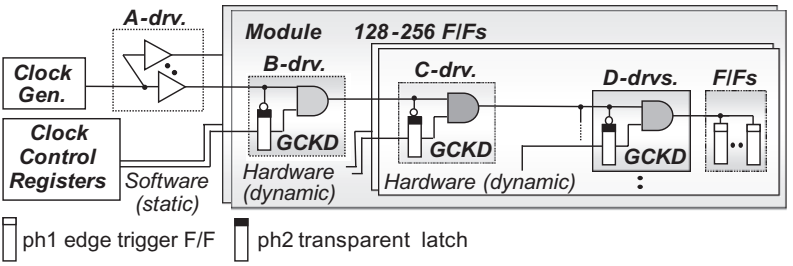

FIGURE 1.26. Clock-gating method of SH-X2. GCKD, gated clock driver cell.

1.5.2 Low Power Technologies

The SH-X2 enhanced the low power technologies from that of the SH-X. Figure 1.26 shows the clock gating method of the SH-X2. The D-drivers also gate the clock with the signals dynamically generated by hardware, and the leaf F/Fs requires no CCP. As a result, the clock tree and total powers are 14 and 10% lower, respectively, than in the SH-X method.

The SH-X2 adopted a way prediction method to the instruction cache. The SH-X2 aggressively fetched the instructions using branch prediction and early branch techniques to compensate branch penalty caused by long pipeline. The power consumption of the instruction cache reached 17% of the SH-X2, and the 64% of the instruction cache power was consumed by data arrays. The way prediction misses were less than 1% in most cases, and was 0% for the Dhrystone 2.1. Then, the 56% of the array access was eliminated by the prediction for the Dhrystone. As a result, the instruction cache power was reduced by 33%, and the SH-X2 power was reduced by 5.5%.

1.6 SH-X3: MULTICORE ARCHITECTURE EXTENSION

Continuously, the SH cores has achieved high efficiency as described above. The SH-X3 core is the third generation of the SH-4A processor core series to achieve higher performance with keeping the high-efficiency maintained in all the SH core series. The multicore architecture is the next approach for the series.

1.6.1 SH-X3 Core Specifications

Table 1.6 shows the specifications of an SH-X3 core designed based on the SH-X2 core. The most of the specifications are the same as that of the SH-X2 core as the successor of it. In addition to such succeeded specifications,

TABLE 1.6. SH-X3 Processor Core Specifications

ISA	SuperH 16-Bit Encoded ISA
Pipeline structure	Dual-issue superscalar 8-stage pipeline
Operating frequency	600 MHz (90-nm generic CMOS process)
Performance	
Dhrystone 2.1	1080 MIPS
FPU (Peak)	4.2/0.6 GFLOPS (single/double)
Caches	8–64 KB I/D each
Local memories	
1st/2nd level	4–128 KB I/D each/128 KB to 1 MB
Power/power efficiency	360 mW/3,000 MIPS/W
Multiprocessor support	
SMP support	Coherency for data caches (up to 4 cores)
AMP support	DTU for local memories
Interrupt	Interrupt distribution and Inter-processor interrupt
Low power modes	Light sleep, sleep, and resume standby
Power management	Operating frequency and low power mode can be different for each core.

the core supports both symmetric and asymmetric multiprocessor (SMP and AMP) features with interrupt distribution and interprocessor interrupt, in corporate with an interrupt controller of such SoCs as RP-1 and RP-2. Each core of the cluster can be set to one of the SMP and AMP modes individually.

It also supports three low power modes of light sleep, sleep, and resume standby. The new light-sleep mode is to respond to a snoop request from the SNC while the core is inactive. In this mode, the data cache is active for the snoop operation, but the other modules are inactive.

In a chip multiprocessor, the core loads are not equal, and each SH-X3 core can operate at a different operating frequency and in a different low power mode to minimize the power consumption for the load. The core can support the SMP features even such heterogeneous operation modes of the cores.

1.6.2 Symmetric and Asymmetric Multiprocessor Support

The four SH-X3 cores constitute a cluster sharing an SNC and a DBG to support symmetric multiprocessor (SMP) and multicore debug features. The SNC has a duplicated address array (DAA) of data caches of all the four cores, and is connected to the cores by a dedicated snoop bus separated from the SuperHyway to avoid both deadlock and interference by some cache coherency protocol operations. The DAA minimizes the number of data cache accesses of the cores for the snoop operations, resulting in the minimum coherency maintenance overhead.

The supported SMP data cache coherency protocols are standard MESI (modified, exclusive, shared, invalid) and ESI modes for copyback and write-through modes, respectively. The copyback and MESI modes are good for performance, and the write-through and ESI modes are suitable to control some accelerators that cannot control the data cache of the SH-X3 cores properly.

The SH-X3 outputs one of the following snoop requests of the cache line to the SNC with the line address and write-back data, if any:

1. Invalidate request for write and shared case.
2. Fill-data request for read and cache-miss case.
3. Fill-data and invalidate request for write and cache-miss case.
4. Write-back request to replace a dirty line.

The SNC transfers a request other than a write-back one to proper cores by checking its DAA, and the requested core processes the requests.

The on-chip RAMs and the data transfer among the various memories are the key features for the AMP support. The use of on-chip RAM makes it possible to control the data access latency, which cannot be controlled well in systems with on-chip caches. Therefore, each core integrates L1 instruction and data RAMs, and a second-level (L2) unified RAM. The RAMs are globally addressed to transfer data to/from the other globally addressed memories. Then, application software can place data in proper timing and location. The SH-X3 integrates a data transfer unit (DTU) to accelerate the data transfer to/from the other modules.

1.6.3 Core Snoop Sequence Optimization

Each core should operate at the proper frequency for its load, but in some cases of the SMP operation, a low frequency core can cause a long stall of a high frequency core. We optimized the cache snoop sequences for the SMP mode to minimize such stalls. Table 1.7 summarizes the coherency overhead cycles. These cycles vary according to various conditions; the table indicates a typical case.

Figure 1.27a,b show examples of core snoop sequences before and after the optimization. The case shown is a "write access to a shared line," which is the third case in the table. The operating frequencies of core #0, #1, and #2 are 600, 150, and 600 MHz, respectively. Initially, all the data caches of the cores hold a common cache line, and all the cache-line states are "shared."

Sequence (a) is as follows:

1. *Core Snoop Request*: Core #0 stores data in the cache, changes the stored-line state from "Shared" to "Modified," and sends a "Core Snoop Request" of the store address to the SNC.

TABLE 1.7. Coherency Overhead Cycles

Access Type	Cache Line State: Accessed Core	Cache Line State: Snooped Core	Overhead (SCLK Cycles): Snooped Core: 600 MHz, Not Optimized	Overhead (SCLK Cycles): Snooped Core: 600 MHz, Optimized	Overhead (SCLK Cycles): Snooped Core: 150 MHz, Not Optimized	Overhead (SCLK Cycles): Snooped Core: 150 MHz, Optimized
Read	S, E, M	–	0	0	0	0
Write	E, M	–	0	0	0	0
	S	S	10	**4**	19	**4**
Read or Write	Miss	Miss	5	5	5	5
		S	10	**5**	19	**5**
		E	10	10	19	19
		M	13	13	22	22

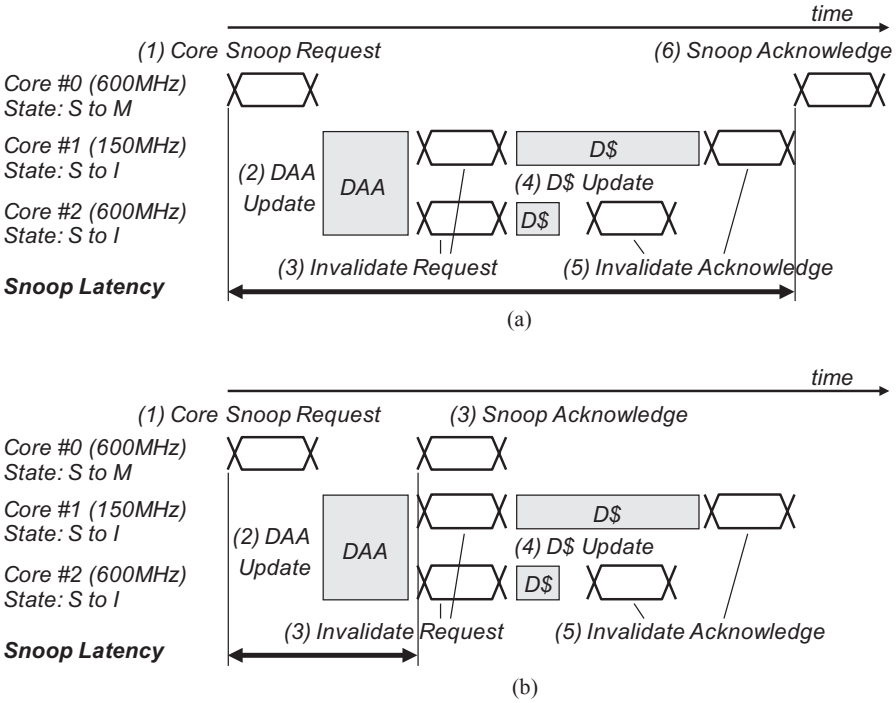

FIGURE 1.27. Core snoop sequences (a) before and (b) after optimization.

2. *DAA Update*: The SNC searches the DAA of all the cores, and changes the states of the hit lines from "Shared" to "Modified" for core #0 and "Invalid" for cores #1 and #2. The SNC runs at SCLK frequency (300 MHz).
3. *Invalidate Request*: The SNC sends "Invalidate Request" to cores #1 and #2.
4. *Data Cache Update*: Cores #1 and #2 change the states of the corresponding cache lines from "Shared" to "Invalid." The processing time depends on each core's ICLK.
5. *Invalidate Acknowledge*: Cores #1 and #2 return "Invalidate Acknowledge" to the SNC.
6. *Snoop Acknowledge*: The SNC returns "Snoop Acknowledge" to core #0.

As shown in Figure 1.27a, the return from core #1 is late due to its low frequency, resulting in long snoop latency.

Sequence (b) is as follows by the optimization:

1. Core Snoop Request
2. DAA Update
3. Snoop Acknowledge and Invalidate Request
4. Data Cache Update
5. Invalidate Acknowledge

The "Snoop Acknowledge" is moved from the 6th to the 3rd step by eliminating the wait of the "Invalidate Acknowledge," and the late response of the slow core does not affect the operation of the fast core. In the optimized sequence, the SNC is busy for some cycles after the "Snoop Acknowledge," and the next "Core Snoop Request" must wait if the SNC is still busy. However, this is rare for ordinary programs.

The sequence of another case, a "read miss and hit to another core's modified line," which is the last case in the table, is as follows:

1. *Core Snoop Request*: A data read of core #0 misses its cache and sends a "Core Snoop Request" of the access address to the SNC.
2. *DAA Update*: The SNC searches the DAA of all the cores, and changes the states of the hit lines from "Modified" to "Shared."
3. *Data Transfer Request*: The SNC sends a "Data Transfer Request" to the core of the hit line for the cache fill data of core #0.
4. *Data Cache Update*: The requested core reads the requested data and changes the states of the corresponding line of the DAA to "Shared." The processing time depends on each core's ICLK.

5. *Data Transfer Response and Write-Back Request*: The requested core returns the requested data and requests a write back to the SNC.
6. *Snoop Acknowledge and Write-Back Request*: The SNC returns "Snoop Acknowledge" to core #0 with the fill data, and requests a write-back of the returned data to the main memory.
7. *Data Cache Update 2*: Core #0 completes the "Read" operation by replacing a cache line with the fill data.

In this case, core #0 must wait for the fill data, and the early "Snoop Acknowledge" is impossible.

1.6.4 Dynamic Power Management

Each core can operate at different CPU clock (ICLK) frequencies and can stop individually while other processors are running with a short switching time in order to achieve both the maximum processing performance and the minimum operating power for various applications. A data cache coherency is maintained during operations at different frequencies, including frequencies lower than the on-chip system bus clock (SCLK). The following four schemes make it possible to change each ICLK frequency individually while maintaining data cache coherency.

1. Each core has its own clock divider for an individual clock frequency change.
2. A handshake protocol is executed before the frequency change to avoid conflicts in bus access, while keeping the other cores running.
3. Each core supports various ICLK frequency ratios to SCLK, including a lower frequency than that of SCLK.
4. Each core has a light-sleep mode to stop its ICLK while maintaining data cache coherency.

The global ICLK and the SCLK that run up to 600 and 300 MHz, respectively, are generated by a global clock pulse generator (GCPG) and distributed to each core. Both the global ICLK and SCLK are programmable by setting the frequency control register in the GCPG. Each local ICLK is generated from the global ICLK by the clock divider of each core. The local CPG (LCPG) of a core executes a handshake sequence dynamically when the frequency control register of the LCPG is changed, so that it can keep the other cores running and can maintain coherency in data transfers of the core. The previous approach assumed a low frequency in a clock frequency change, and it stopped all the cores when a frequency was changed. The core supports "light-sleep mode" to stop its ICLK except for its data cache in order to maintain the data cache coherency. This mode is effective for reducing the power of an SMP system.

1.6.5 RP-1 Prototype Chip

The RP-1 is the first multicore chip with four SH-X3 CPU cores. It supports both symmetric and asymmetric multiprocessor (SMP and AMP) features for embedded applications. The SMP and AMP modes can be mixed to construct a hybrid system of the SMP and AMP. Each core can operate at different frequencies and can stop individually with maintaining its data-cache coherency while the other processors are running in order to achieve both the maximum processing performance and the minimum operating power for various applications.

1.6.5.1 RP-1 Specifications Table 1.8 summarizes the RP-1 specifications. It was fabricated as a prototype chip using a 90-nm CMOS process to accelerate the research and development of various embedded multicore systems. The RP-1 achieved a total of 4,320 MIPS at 600 MHz by the four SH-X3 cores measured using the Dhrystone 2.1 benchmark. The RP-1 integrates four SH-X3 cores with a snoop controller (SNC) to maintain the data-cache coherency among the cores, DDR2-SDRAM and SRAM memory interfaces, a PCI-Express interface, some HW-IPs for various types of processing, and some peripheral modules. The HW-IPs include a DMA controller, a display unit, and accelerators. Each SH-X3 core includes a 32-kB four-way set-associative instruction and data caches, an 8-kB instruction local RAM (ILRAM), a 16-kB operand local RAM (OLRAM), and a 128-kB unified RAM (URAM).

Figure 1.28 illustrates a block diagram of the RP-1. The four SH-X3 cores, a snoop controller (SNC), and a debug module (DBG) constitute a cluster.

TABLE 1.8. RP-1 Specifications

Process technology	90-nm, 8-layer Cu, triple-Vth, CMOS
Chip size/power	97.6 mm^2 (9.88 mm × 9.88 mm)/3 W (typical, 1.0 V)
Supply voltage/clock frequency	1.0 V (internal), 1.8/3.3 V(I/O)/600 MHz
SH-X3 core	
Size	2.60 mm × 2.80 mm
I/D-cache	32-kB four-way set-associative (each)
ILRAM/OLRAM/URAM	8/16/128 KB (unified)
Snoop controller (SNC)	Duplicated Address Array (DAA) of four D-caches
Centralized shared memory (CSM)	128 kB
External interfaces	DDR2-SDRAM, SRAM, PCI-Express
Performance	
CPU	4,320 MIPS (Dhrystone 2.1, 4 core total)
FPU	16.8 GFLOPS (peak, 4 core total)
Package	FCBGA 554 pin, 29 × 29 mm

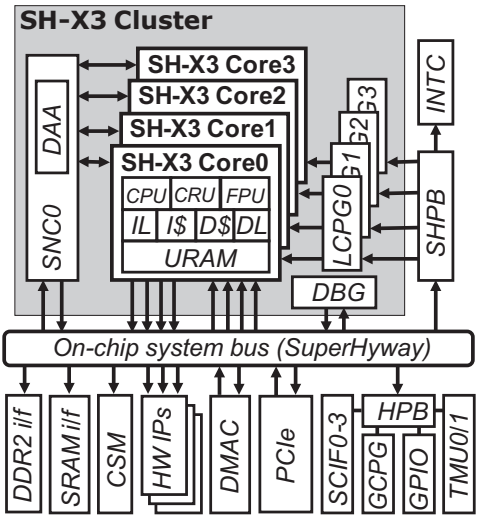

FIGURE 1.28. Block diagram of RP-1. SNC, snoop controller (cntl.); DAA, duplicated address array; CRU, cache RAM control unit; I$/D$, instruction (inst.)/data cache; IL/DL, Inst./data local memory; URAM, unified RAM; DBG, debug module; GCPG/LCPG, global/local CPG; INTC, interrupt cntl.; SHPB,HPB, peripheral bus bridge; CSM, centralized shared memory; DMAC, direct memory access cntl.; PCIe, PCIexpress interface (i/f); SCIF, serial communication i/f; GPIO, general purpose IO; TMU, timer unit.

The HW-IPs are connected to an on-chip system bus (SuperHyway). The arrows to/from the SuperHyway indicate connections from/to initiator/target ports, respectively.

1.6.5.2 Chip Integration and Evaluations Figure 1.29 shows the chip micrograph of the RP-1. The chip was integrated in two steps to minimize the design period of the physical integration, and successfully fabricated: (1) First, a single core was laid out as a hard macro and completed timing closure of the core, and (2) the whole chip was laid out with instancing the core four times.

We evaluated the processing performance and power reduction in parallel processing on the RP-1. Figure 1.30 plots the time required to execute the SPLASH-2 suite [46] depending on the number of threads on an SMP Linux system. The RP-1 reduced the processing time to 50.5–52.6% and 27.1–36.9% with two and four threads, respectively. The time should be 50 and 25% for ideal performance scalability. The major overhead was synchronization and snoop time. The SNC improved cache coherency performance, and the performance overhead by snoop transactions was reduced to up to 0.1% when SPLASH-2 was executed.

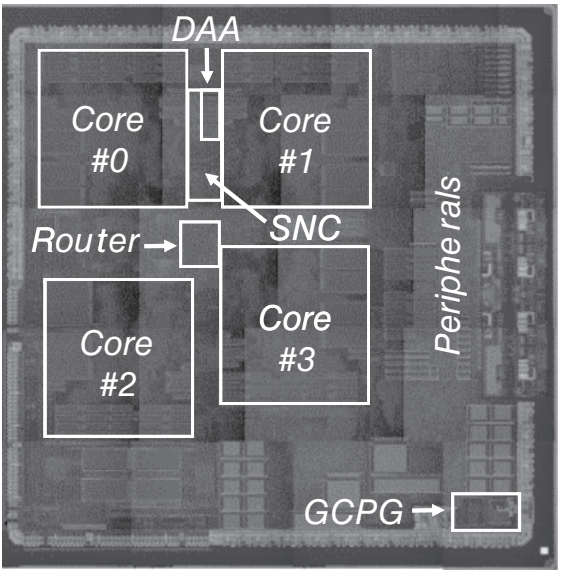

FIGURE 1.29. Chip micrograph of RP-1.

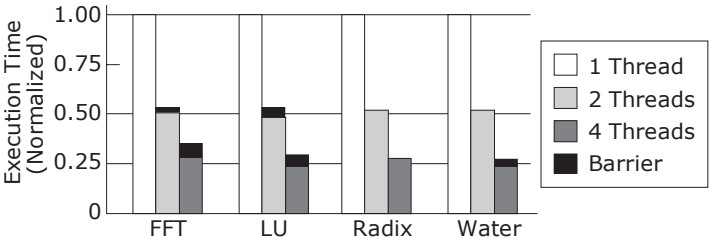

FIGURE 1.30. Execution time of SPLASH-2 suite.

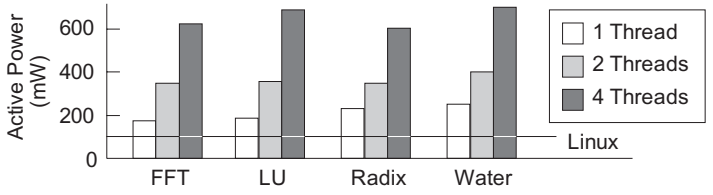

FIGURE 1.31. Active power of SPLASH-2 suite.

Figure 1.31 shows the power consumption of the SPLASH-2 suite. The suite ran at 600 MHz and at 1.0 V. The average power consumption of one, two, and four threads was 251, 396, and 675 mW, respectively. This included 104 mW of active power for the idle tasks of SMP Linux. The results of the performance and power evaluation showed that the power efficiency was maintained or enhanced when the number of threads increased.

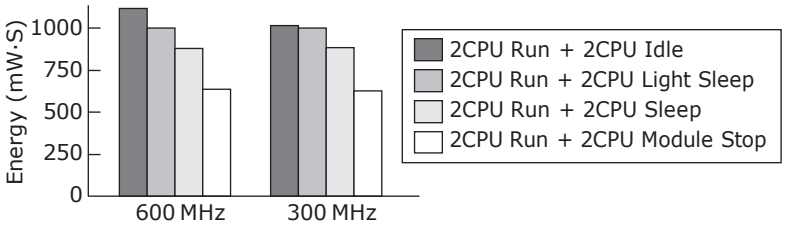

FIGURE 1.32. Energy consumption with low power modes.

Figure 1.32 shows the energy consumption with low power modes. These modes were implemented to save power when fewer threads were running than available on CPU cores. As a benchmark, two threads of fast Fourier transform (FFT) were running on two CPU cores, and two CPU cores were idle. The energy consumed in the light sleep, sleep, and module stop modes at 600 MHz was 4.5, 22.3, and 44.0% lower than in the normal mode, respectively, although these modes took some time to stop and start the CPU core and to save and return the cache. The execution time increased by 79.5% at 300 MHz, but the power consumption decreased, and the required energy decreased by 5.2%.

1.6.6 RP-2 Prototype Chip

The RP-2 is a prototype multicore chip with eight SH-X3 CPU cores. It was fabricated in a 90-nm CMOS process that was the same process used for the RP-1. The RP-2 achieved a total of 8,640 MIPS at 600 MHz by the eight SH-X3 cores measured with the Dhrystone 2.1 benchmark. Because it is difficult to lay out the eight cores close to each other, we did not select a tightly coupled cluster of eight cores. Instead, the RP-2 consists of two clusters of four cores, and the cache coherency is maintained in each cluster. Therefore, the intercluster cache coherency must be maintained by software if necessary.

1.6.6.1 RP-2 Specifications Table 1.9 summarizes the RP-2 specifications. The RP-2 integrates eight SH-X3 cores as two clusters of four cores, DDR2-SDRAM and SRAM memory interfaces, DMA controllers, and some peripheral modules. Figure 1.33 illustrates a block diagram of the RP-2. The arrows to/from the SuperHyway indicate connections from/to initiator/target ports, respectively.

1.6.6.2 Power Domain and Partial Power Off Power-efficient chip design for embedded applications requires several independent power domains where the power of unused domains can be turned off. The power domains were initially introduced to an SoC for mobile phones [5], which defined 20 hierarchical power domains. In contrast, high performance multicore chips use leaky low-Vt transistors for CPU cores, and reducing the leakage power of such cores is the primary goal.

TABLE 1.9. RP-2 Specifications

Process technology	90-nm, 8-layer Cu, triple-Vth, CMOS
Chip size/power	104.8 mm^2/2.8 W (typical, 1.0 V, Dhrystone 2.1)
Supply voltage/clock frequency	1.0 V (internal), 1.8/3.3 V(I/O)/600 MHz
SH-X3 core	
Size	6.6 mm^2 (3.36 × 1.96 mm)
I/D-cache	16-kB four-way set-associative (each)
ILRAM/OLRAM/URAM	8/32/64 kB (unified)
CSM/external interfaces	128 kB/DDR2-SDRAM, SRAM
Performance	
CPU	8,640 MIPS (Dhrystone 2.1, 8 core total)
FPU	33.6 GFLOPS (peak, 8 core total)

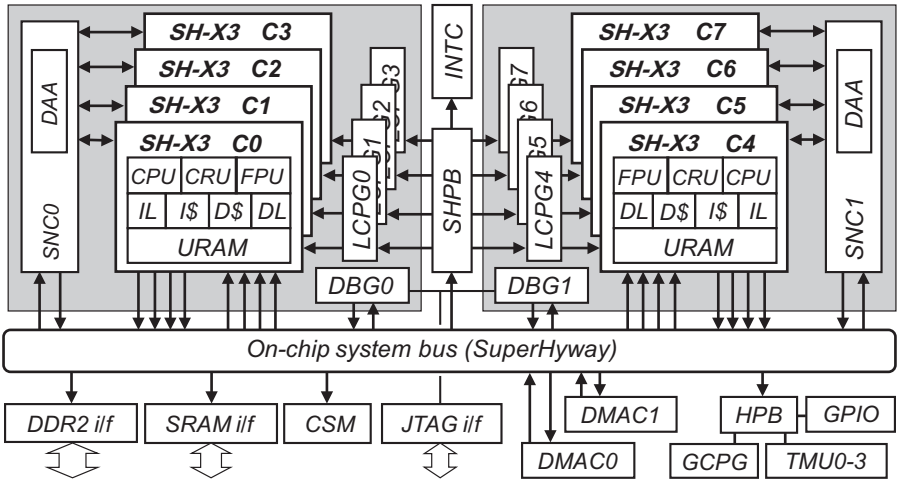

FIGURE 1.33. Block diagram of RP-2.

The RP-2 was developed for target use in power-efficient high performance embedded applications. Sixteen power domains were defined so that they can be independently powered off. A resume-standby mode was also defined for fast resume operation, and the power levels of the CPU and the URAM of a core are off and on, respectively. Each processor core can operate at a different frequency or even dynamically stop the clock to maintain processing performance while reducing the average operating power consumption.

Figure 1.34 illustrates the power domain structure of eight CPU cores with eight URAMs. Each core is allocated to a separate power domain so that

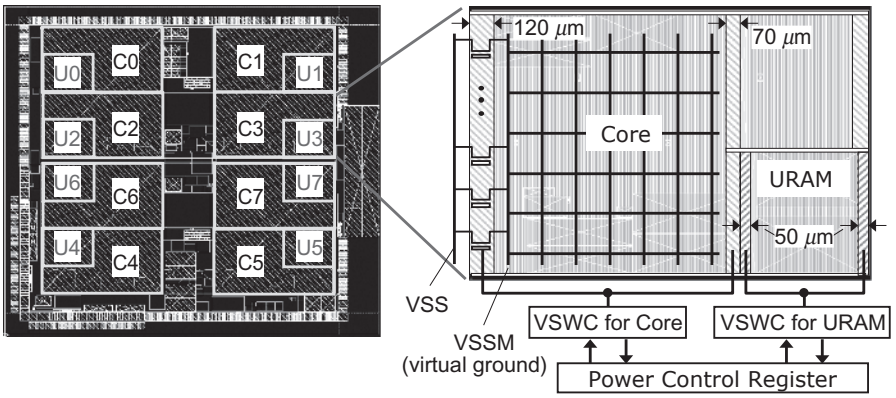

FIGURE 1.34. Power domain structure of eight CPU cores with eight URAMs.

TABLE 1.10. Power Modes of CPU Cores

CPU Power Modes	Normal	Light Sleep	Sleep	Resume	Power Off
Clock for CPU and URAM	On	Off	Off	Off	Off
Clock for I/D Cache	On	On	Off	Off	Off
Power supply for CPU	On	On	On	Off	Off
Power supply for URAM	On	On	On	On	Off
Leakage current (mA)[a]	162	162	162	22	0

[a] Measured at room temperature at 1.0 V, eight-core total.

the power supply can be cut off while unused. Two power domains (Cn and Un, for *n* ranging from 0 to 7) are assigned to each core, where Un is allocated only for URAM. By keeping the power of Un on, the CPU status is saved to URAM before the Cn power is turned off, and restored from URAM after Cn power is turned on. This shortens the restart time compared with a power-off mode in which both Cn and Un are powered off together. Each power domain is surrounded by power switches and controlled by a power switch controller (VSWC).

Table 1.10 summarizes the power modes of each CPU. Light sleep mode is suitable for dynamic power saving while cache coherency is maintained. In sleep mode, almost all clocks for the CPU core are stopped. In resume standby mode, the leakage current for eight cores is reduced to 22 from 162 mA in sleep mode, and leakage power is reduced by 86%.

1.6.6.3 Synchronization Support Hardware The RP-2 has barrier registers to support CPU core synchronization for multiprocessor systems. Software can

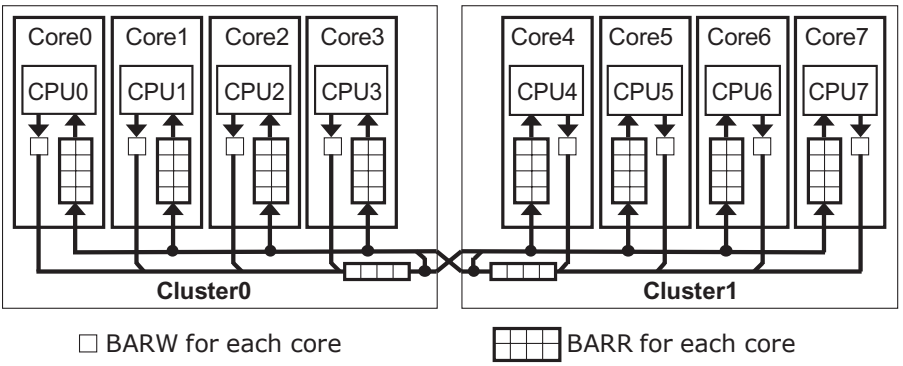

FIGURE 1.35. Barrier registers for synchronization.

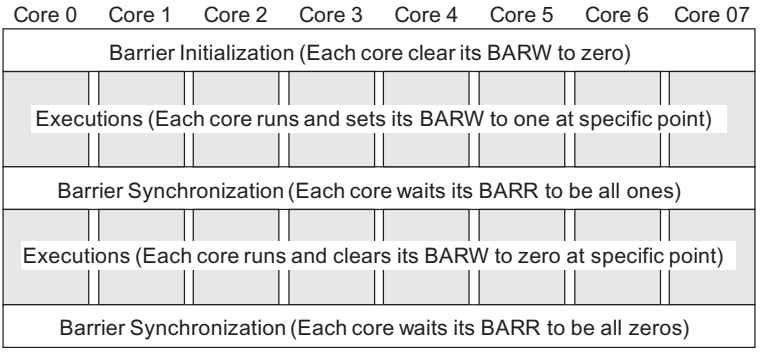

FIGURE 1.36. Synchronization example using barrier registers.

use these registers for fast synchronization between the cores. In the synchronization, one core waits for other cores to reach a specific point in a program. Figure 1.35 illustrates the barrier registers for the synchronization. In a conventional software solution, the cores have to test and set a specific memory location, but this requires long cycles. We provide three sets of barrier registers to accelerate the synchronization. Each CPU core has a 1-bit barrier write (BARW) register that it notifies when it reaches a specific point. The BARW values of all the cores are gathered by hardware to form an 8-bit barrier read (BARR) register of each core so that each core can obtain all the BARW values from its BARR register with a single instruction. As a result, the synchronization is fast and does not disturb other transactions on the Super-Hyway bus.

Figure 1.36 shows an example of the barrier register usage. In the beginning, all the BARW values are initialized to zero. Then each core inverts its BARW

TABLE 1.11. Eight-Core Synchronization Cycles

	Conventional Method (via External Memory)	RP-2 Method (via BARW/BARR registers)
Average clock cycles	52,396	8,510
Average difference	20,120	10

value when it reaches a specific point, and it checks and waits until all its BARR values are ones reflecting the BARW values. The synchronization is complete when all the BARW values are inverted to ones. The next synchronization can start immediately with the BARWs being ones, and is complete when all the BARW values are inverted to zeros.

Table 1.11 compares the results of eight-core synchronizations with and without the barrier registers. The average number of clock cycles required for a certain task to be completed with and without barrier registers is 8,510 and 52,396 cycles, respectively. The average differences in the synchronizing cycles between the first and last cores are 10 and 20,120 cycles, with and without the barrier registers, respectively. These results show that the barrier registers effectively improve the synchronization.

1.6.6.4 Chip Integration and Evaluations The RP-2 was fabricated using the same 90-nm CMOS process as that for the RP-1. Figure 1.37 is the chip micrograph of the RP-2. It achieved a total of 8,640 MIPS at 600 MHz by the eight SH-X3 cores measured with the Dhrystone 2.1 benchmark, and consumed 2.8 W at 1.0 V, including leakage power.

The fabricated RP-2 chip was evaluated using the SPLASH-2 benchmarks on an SMP Linux OS. Figure 1.38 plots the RP-2 execution time on one cluster based on the number of POSIX threads. The processing time was reduced to 51–63% with two threads and to 41–27% with four or eight threads running on one cluster. Since there were fewer cores than threads, the eight-thread case showed similar performance to the four-thread one. Furthermore, in some cases, the increase in the number of threads resulted in an increase in the processing time due to the synchronization overhead.

1.7 SH-X4: ISA AND ADDRESS SPACE EXTENSION

Continuously, embedded systems expand their application fields, and enhance their performance and functions in each field. As a key component of the system, embedded processors must enhance their performance and functions with maintaining or enhancing their efficiencies. As the latest SH processor core, the SH-X4 extended its ISA and address space efficiently for this purpose.

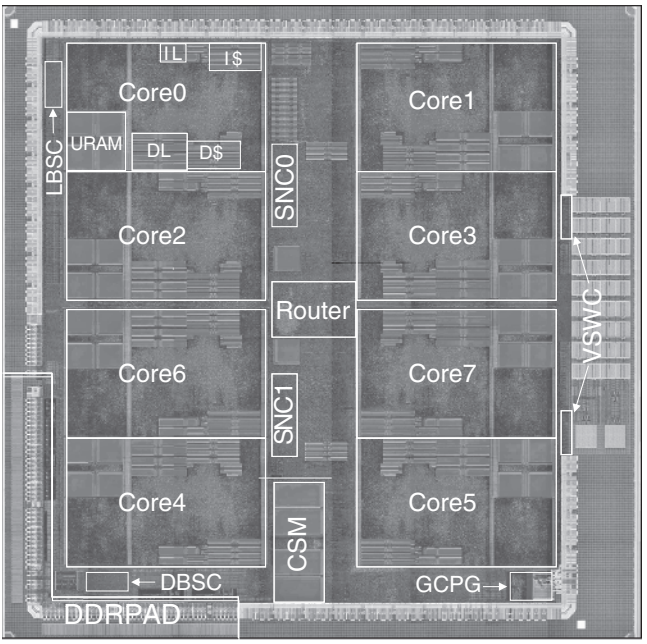

FIGURE 1.37. Chip micrograph of RP-2.

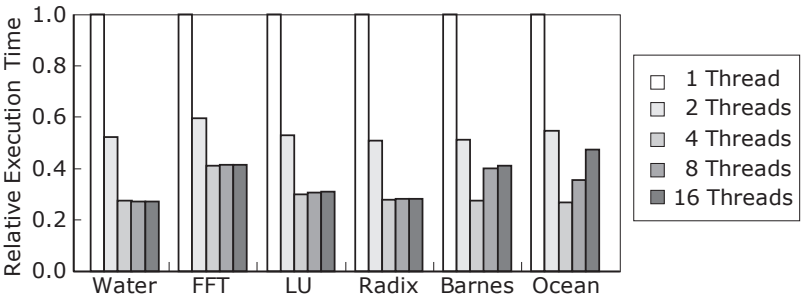

FIGURE 1.38. RP-2 execution time according to number of POSIX threads.

The SH-X4 was integrated on the RP-X heterogeneous multicore chip as two 4-core clusters with four Flexible Engine/Generic ALU Arrays (FE-GAs) [47, 48], two MX-2 matrix processors [49], a Video Processing Unit 5 (VPU5) [50, 51], and various peripheral modules.

1.7.1 SH-X4 Core Specifications

Table 1.12 shows the specifications of an SH-X4 core designed based on the SH-X3 core. The most of the specifications are the same as those of the SH-X3

TABLE 1.12. SH-X4 Processor Core Specifications

ISA	SuperH 16-Bit ISA with Prefix Extension
Operating frequency	648 MHz (45-nm low power CMOS process)
Performance	
Dhrystone 2.1	1,717 MIPS (2.65 MIPS/MHz)
FPU (peak)	4.5/0.6 GFLOPS (single/double)
Power/power efficiency	106 mW/16 GIPS/W
Address space	
Logical	32 bits, 4 GB
Physical	40 bits, 1 TB

core, and the same ones are not shown. The SH-X4 extended the ISA with some prefixes, and the cycle performance is enhanced from 2.23 to 2.65 MIPS/MHz. As a result, the SH-X4 achieved 1,717 MIPS at 648 MHz. The 648 MHz is not so high compared with the 600 MHz of the SH-X3, but the SH-X4 achieved the 648 MHz in a low power process. Then, the typical power consumption is 106 mW, and the power efficiency reached as high as 16 GIPS/W.

1.7.2 Efficient ISA Extension

The 16-bit fixed-length ISA of the SH cores is an excellent feature enabling a higher code density than that of 32-bit fixed-length ISAs of conventional RISCs. However, we made some trade-off to establish the 16-bit ISA. Operand fields are carefully shortened to fit the instructions into the 16 bits according to the code analysis of typical embedded programs in the early 1990s. The 16-bit ISA was the best choice at that time and the following two decades. However, required performance grew higher and higher, program size and treating data grew larger and larger. Therefore, we decided to extend the ISA by some prefix codes.

The week points of the 16-bit ISA are (1) short-immediate operand, (2) lack of three-operand operation instructions, and (3) implicit fixed-register operand. The short-immediate ISA uses a two-instruction sequence of a long-immediate load and a use of the loaded data, instead of a long immediate instruction. A three-operand operation becomes a two-instruction sequence of a move instruction and a two-operand instruction. The implicit fixed-register operand makes register allocation difficult, and causes inefficient register allocations.

The popular ISA extension from the 16-bit ISA is a variable-length ISA. For example, an IA-32 is a famous variable-length ISA, and ARM Thumb-2 is a variable-length ISA of 16 and 32 bits. However, a variable-length instruction consists of plural unit-length codes, and each unit-length code has plural meaning depending on the preceding codes. Therefore, the variable-length ISA causes complicated, large, and slow parallel-issue logic with serial code analysis.

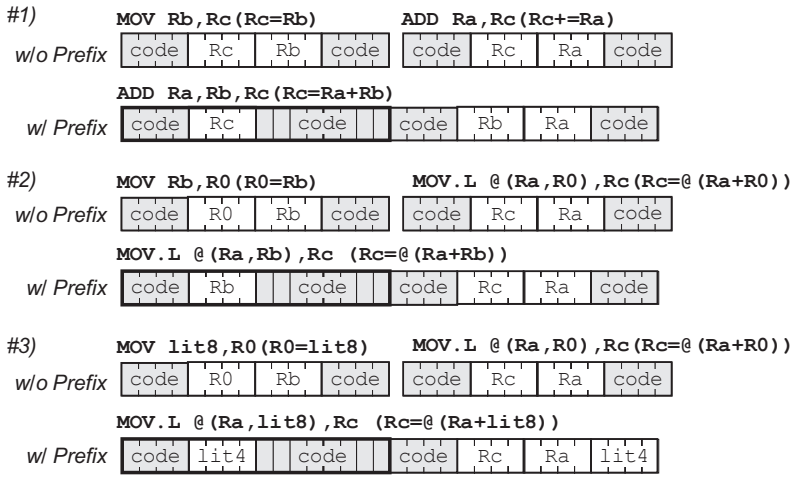

FIGURE 1.39. Examples of ISA extension.

Another way is using prefix codes. The IA-32 uses some prefixes, as well as the variable-length instructions, and using prefix codes is one of the conventional ways. However, if we use the prefix codes but not use the variable-length instructions, we can implement a parallel instruction decoding easily. The SH-X4 introduced some 16-bit prefix codes to extend the 16-bit fixed-length ISA.

Figure 1.39 shows some examples of the ISA extension. The first example (#1) is an operation "Rc = Ra + Rb (Ra, Rb, Rc: registers)", which requires a two-instruction sequence of "MOV Ra, Rc (Rc = Ra)" and "ADD Rb, Rc (Rc + = Rb)" before extension, but only one instruction "ADD Ra, Rb, Rc" after the extension. The new instruction is made of the "ADD Ra, Rb" by a prefix to change a destination register operand Rb to a new register operand Rc. The code sizes are the same, but the number of issue slots reduces from two to one. Then the next instruction can be issued simultaneously if there is no other pipeline stall factor.

The second example (#2) is an operation "Rc = @(Ra + Rb)," which requires a two-instruction sequence of "MOV Rb, R0 (R0 = Rb)" and "MOV.L @(Ra, R0), Rc (Rc = @(Ra + R0))" before extension, but only an instruction "MOV.L @(Ra, Rb), Rc" after the extension. The new instruction is made of the "MOV @(Ra, R0), Rc" by a prefix to change the R0 to a new register operand. Then we do not need to use the R0, which is the third implicit fixed operand with no operand field to specify. It makes the R0 busy and register allocation inefficient to use the R0-fixed operand, but the above extension solve the problem.

The third example (#3) is an operation "Rc = @(Ra + lit8) (lit8: 8-bit literal)," which requires a two-instruction sequence of "MOV lit8, R0

(R0 = lit8)" and "MOV.L @(Ra, R0), Rc (Rc = @(Ra + R0))" before extension, but only an instruction "MOV.L @(Ra, lit8), Rc" after the extension. The new instruction is made of the "MOV.L @(Ra, lit4), Rc (lit4: 4-bit literal)" by a prefix to extend the lit4 to lit8. The prefix can specify the loaded data size in memory and the extension type of signed or unsigned if the size is 8 or 16 bits, as well as the extra 4-bit literal.

Figure 1.40 illustrates the instruction decoder of the SH-X4 enabling a dual issue, including extended instructions by prefix codes. The gray parts are the extra logic for the extended ISA. Instruction registers at the I3 stage hold first four 16-bit codes, which was two codes for the conventional 16-bit fixed-length ISA. The simultaneous dual-issue of the instructions with prefixes consumes the four codes per cycle at peak throughput. Then, a predecoder checks each code in parallel if it is a prefix or not, and outputs control signals of multiplexers MUX to select the inputs of prefix and normal decoders properly.

The Table 1.13 summarizes all cases of the input patterns and corresponding selections. A code after the prefix code is always a normal code, and hardware

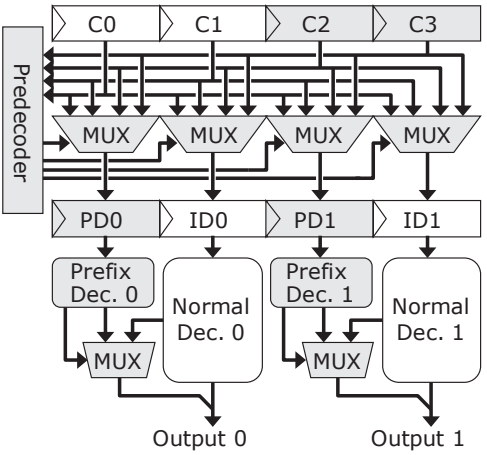

FIGURE 1.40. Instruction decoder of SH-X4.

TABLE 1.13. Input Patterns and Selections

Input				Output			
C0	C1	C2	C3	PD0	ID0	PD1	ID1
N	N	–	–	–	C0	–	C1
N	P	–	–	–	C0	C1	C2
P	–	N	–	C0	C1	–	C2
P	–	P	–	C0	C1	C2	C3

P, prefix; n, normal; –, arbitrary code.

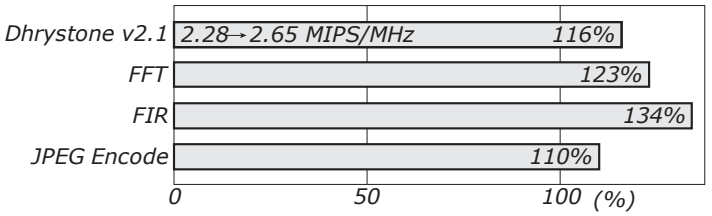

FIGURE 1.41. Performance improvement ratio by prefix codes.

need not check it. Each prefix decoder decodes a provided prefix code, and overrides the output of the normal decoder appropriately. As a result, the instruction decoder performs the dual issue of instructions with prefixes.

Figure 1.41 shows evaluation results of the extended ISA with four benchmark programs. The performance of Dhrystone 2.1 was accelerated from 2.24 to 2.65 MIPS/MHz by 16%. The performance of FFT, finite impulse response (FIR), and JPEG encoding were improved by 23, 34, and 10%, respectively. On the other hand, area overhead of the prefix code implementation was less than 2% of the SH-X4. This means the ISA extension by the prefix codes enhanced both performance and efficiency.

1.7.3 Address Space Extension

The 32-bit address can define an address space of 4 GB. The space consists of main memory, on-chip memories, various IO spaces, and so on. Then the maximum linearly addressed space is 2 GB for the main memory. However, the total memory size is continuously increasing, and will soon exceed 2 GB even in an embedded system. Therefore, we extended the number of physical address bits to 40 bits, which can define 1-TB address space. The logical address space remains 32-bit, and the programming model is unchanged. Then the binary compatibility is maintained. The logical address space extension would require the costly 32- to 64-bit extensions of register files, integer executions, branch operations, and so on.

Figure 1.42 illustrates an example of the extension. The 32-bit logical address space is compatible to the predecessors of the SH-X4. The MMU translates the logical address to a 32/40-bit physical address by TLB or privileged mapping buffer (PMB) in 32/40-bit physical address mode, respectively. The TLB translation is a well-known dynamic method, but the original PMB translation is a static method to avoid exceptions possible for the TLB translation. Therefore, the PMB page sizes are larger than that of the TLB to cover the PMB area efficiently.

The logical space is divided into five regions, and the attribute of each region can be specified as user-mode accessible or inaccessible, translated by TLB or PMB, and so on. In the example, the P0/U0 region is user-mode

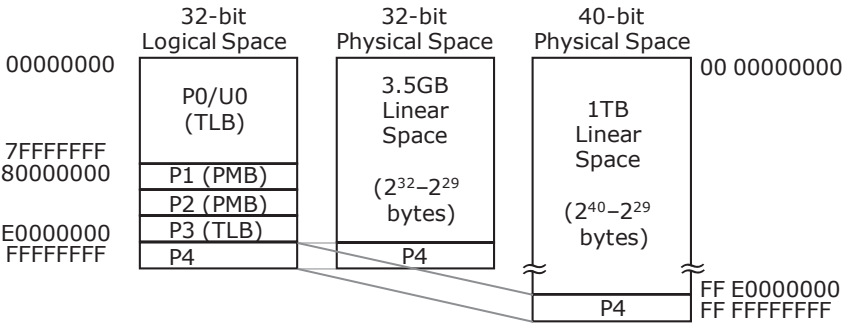

FIGURE 1.42. An example of logical and physical address spaces of SH-X4.

accessible and translated by TLB, the P1 and P2 region are user-mode inaccessible and translated by PMB, and the P3 region is user-mode inaccessible and translated by TLB. The P4 region includes a control register area that is mapped on the bottom of physical space so that the linear physical space is not divided by the control register area.

1.7.4 Data Transfer Unit

High-speed and efficient data transfer is one of the key features for multicore performance. The SH-X4 core integrates a DTU for this purpose. A DMAC is conventional hardware for the data transfer. However, the DTU has some advantage to the DMAC, because the DTU is a part of an SH-X4 core. For example, when a DMAC transfer the data between a memory in an SH-X4 core and a main memory, the DMAC must initiate two SuperHyway bus transactions between the SH-X4 core and the DMAC and between the DMAC and the main memory. On the other hand, the DTU can perform the transfer with one SuperHyway bus transaction between the SH-X4 core and the main memory. In addition, the DTU can use the initiator port of the SH-X4 core, whereas the DMAC must have its own initiator port, and even if all the SH-X4 cores have a DTU, no extra initiator port is necessary. Another merit is that the DTU can share the unified TLB (UTLB) of the SH-X4 core, and the DTU can handle a logical address.

Figure 1.43 shows an example of a data transfer between an SH-X4 core and an FE-GA. The DTU has a transfer TLB (TTLB) as a micro TLB that caches UTLB entries of the CPU for independent executions. The DTU can get a UTLB entry when the translation misses the TTLB. The DTU action is defined by a command chain in a local memory. The DTU can execute the command chain of plural commands without CPU control. In the example, the DTU transfers data in a local memory of the SH-X4 to a memory in the FE-GA. The source data specified by the source address from the command

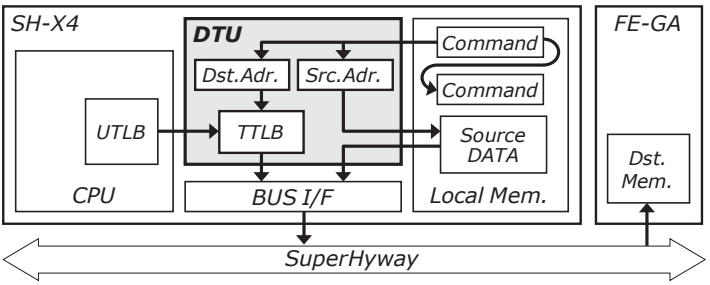

FIGURE 1.43. An example of DTU transfer between SH-X4 and FE-GA.

is read from the local memory, and the destination address specified by the command is translated by the TTLB. Then the address and data are output to the SuperHyway via the bus interface, and the data are transferred to the destination memory of the FE-GA.

1.7.5 RP-X Prototype Chip

A heterogeneous multicore is one of the most promising approaches to attain high performance with low frequency and power, for consumer electronics or scientific applications. The RP-X is the latest prototype multicore chip with eight SH-X4 cores, four FE-GAs, two MX-2s, a VPU5, and various peripheral modules. It was fabricated using 45-nm CMOS process. The RP-X achieved 13.7 GIPS at 648 MHz by the eight SH-X4 cores measured using the Dhrystone 2.1 benchmark, and a total of 114.7 GOPS with 3.07 W. It attained a power efficiency of 37.3 GOPS/W.

1.7.5.1 RP-X Specifications The RP-X specifications are summarized in Table 1.14. It was fabricated using a 45-nm CMOS process, integrating eight SH-X4 cores, four FE-GAs, two MX-2s, one VPU5, one SPU, and various peripheral modules as a heterogeneous multicore SoC, which is one of the most promising approaches to attain high performance with low frequency and power, for consumer electronics or scientific applications.

The eight SH-X4 cores achieved 13.7 GIPS at 648 MHz measured using the Dhrystone 2.1 benchmark. Four FE-GAs, dynamically reconfigurable processors, were integrated and attained a total performance of 41.5GOPS and a power consumption of 0.76 W. Two 1024-way MX-2s were integrated and attained a total performance of 36.9GOPS and a power consumption of 1.10 W. Overall, the efficiency of the RP-X was 37.3 GOPS/W at 1.15 V, excluding special-purpose cores of a VPU5 and an SPU. This was the highest among comparable processors. The operation granularity of the SH-X4, FE-GA and MX-2 processors are 32, 16, and 4 bits, respectively, and thus, we can assign the appropriate processor cores for each task in an effective manner.

TABLE 1.14. RP-X Specifications

Process technology		45-nm, 8-layer Cu, triple-Vth, CMOS
Chip size		153.76 mm^2 (12.4 mm × 12.4 mm)
Supply voltage		1.0–1.2 V (internal), 1.2/1.5/1.8/2.5/3.3 V (I/O)
Clock frequency		648 MHz (SH-X4), 324 MHz (FE-GA, MX-2)
Total power consumption		3.07 W (648 MHz, 1.15 V)
Processor cores and performances		
8× SH-X4	CPU	13.7 GIPS (Dhrystone 2.1, 8-core total)
	FPU	36.3 GFLOPS (8-core total)
4× FE-GA		41.5 GOPS (4-core total)
2× MX-2		36.9 GOPS (2-core total)
Programmable special purpose cores		VPU5 (video processing unit) for MPEG2, H.264, VC-1
		SPU (sound processing unit) for AAC, MP3
Total performances and power		114.7 GOPS, 3.07 W, 37.3 GOPS/W (648 MHz, 1.15 V)
External interfaces		2× DDR3-SDRAM (32-bit, 800 MHz), SRAM, PCIexpress (rev 2.0, 2.5 GHz, 4 lanes), serial ATA

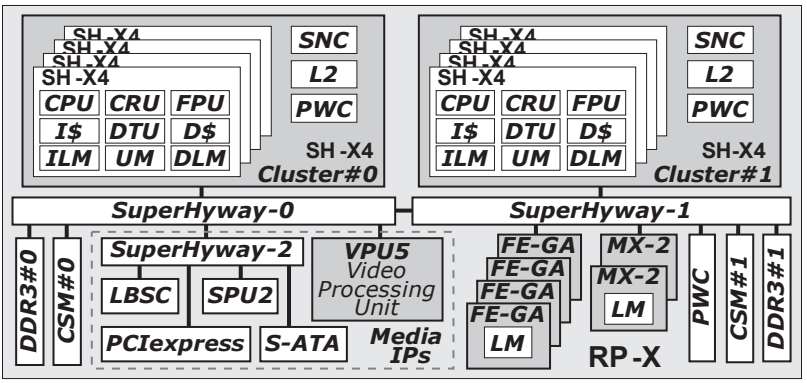

FIGURE 1.44. Block diagram of RP-X.

Figure 1.44 illustrates the structure of the RP-X. The processor cores of the SH-X4, FE-GA, and MX-2, the programmable special purpose cores of the VPU5 and SPU, and the various modules are connected by three SuperHyway buses to handle high-volume and high-speed data transfers. SuperHyway-0 connects the modules for an OS, general tasks, and video processing, SuperHyway-1 connects the modules for media acceleration, and SuperHyway-2 connects media IPs except for the VPU5. Some peripheral buses and modules are not shown in the figure.

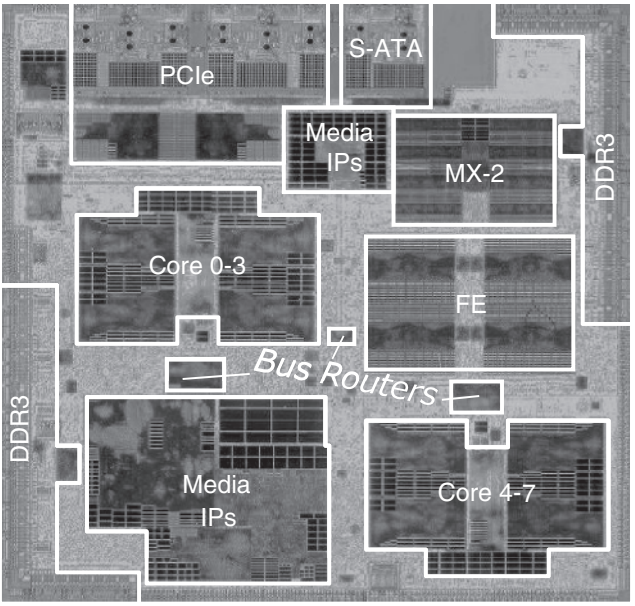

FIGURE 1.45. Chip micrograph of RP-X.

A DTU was implemented in each SH-X4 core to transfer data to and from the special-purpose cores or various memories without using CPU instructions. In this kind of system, multiple OSes are used to control various functions, and thus high-volume and high-speed memories are required.

1.7.5.2 Chip Integration and Evaluations The RP-X was fabricated using a 45-nm low power CMOS process. A chip micrograph of the RP-X is in Figure 1.45. It achieved a total of 13,738 MIPS at 648 MHz by the eight SH-X4 cores measured using the Dhrystone 2.1 benchmark, and consumed 3.07 W at 1.15 V including leakage power.

The RP-X is a prototype chip for consumer electronics or scientific applications. As an example, we produced a digital TV prototype system with IP networks (IP-TV), including image recognition and database search. Its system configuration and memory usage are shown in Figure 1.46. The system is capable of decoding 1080i audio/video data using a VPU and an SPU on the OS#1. For image recognition, the MX-2s are used for image detection and feature quantity calculation, and the FE-GAs are used for optical flow calculation of a VGA (640×480) video at 15 fps on the OS#2. These operations required 30.6 and 0.62 GOPS of the MX-2 and FE-GA, respectively. The SH-X4 cores are used for database search using the results of the above operations on the OS#3, as well as supporting of all the processing, including OS#1, OS#2, OS#3,

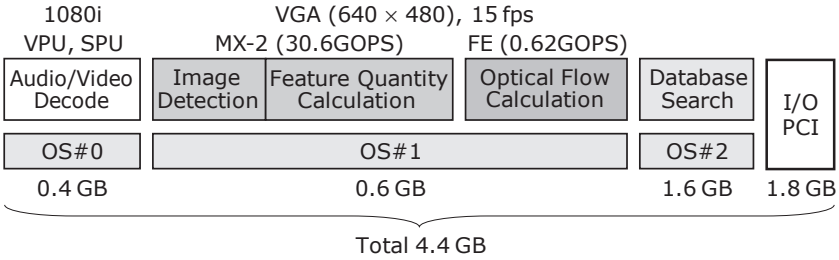

FIGURE 1.46. System configuration and memory usage of a digital TV.

TABLE 1.15. Performance and Power Consumption of RP-X

	Operating Frequency	Performance	Power	Power Efficiency
SH-X4	648 MHz	36.3 GFLOPS	0.74 W	49.1 GFLOPS/W
MX-2	324 MHz	36.9 GOPS	0.81 W	45.6 GOPS/W
FE-GA	324 MHz	41.5 GOPS	1.12 W	37.1 GOPS/W
Others	324/162/81 MHz	–	0.40 W	–
Total	–	114.7 GOPS	3.07 W	37.3 GOPS/W

and data transfers between the cores. The main memories of 0.4, 0.6, 1.6, and 1.8 GB are assigned to OS#1, OS#2, OS#3, and PCI, respectively, for a total of 4.4 GB.

Table 1.15 lists the total performance and power consumption at 1.15 V when eight CPU cores, four FE-GAs, and two MX-2s are used at the same time. The power efficiency of the CPU cores, FE-GAs, and MX-2s reached 42.9 GFLOPS/W, 41.5 GOPS/W, and 36.9 GOPS/W, respectively. The power consumption of the other components was reduced to 0.40 W by clock gating of 31 out of 44 modules. In total, if we count 1 GFLOPS as 1 GOPS, the RP-X achieved 37.3 GOPS/W at 1.15 V, excluding I/O area power consumption.

REFERENCES

[1] P.P. Gelsinger, "Microprocessors for the new millennium challenges, opportunities, and new frontiers," in ISSCC Digest of Technical Papers, Session 1.3, Feb. 2001.

[2] F. Arakawa, "Multicore SoC for embedded systems," in International SoC Design Conference (ISOCC) 2008, Nov. 2008, pp. I-180–I-183.

[3] R.P. Weicker, "Dhrystone: a synthetic programming benchmark," Communications of ACM, 27(10), Oct. 1984, pp. 1013–1030.

[4] R.P. Weicker, "Dhrystone benchmark: rationale for version 2 and measurement rules," ACM SIGPLAN Notices, 23(8), Aug. 1988, pp. 49–62.

[5] T. Hattori et al., "A power management scheme controlling 20 power domains for a sin-gle-chip mobile processor," in ISSCC Dig. Tech. Papers, Session 29.5, Feb., 2006.

[6] M. Ito et al., "A 390 MHz single-chip application and dual-mode baseband processor in 90 nm triple-Vt CMOS," in ISSCC Dig. Tech. Papers, Session 15.3, Feb. 2007.

[7] M. Naruse et al., "A 65 nm single-chip application and dual-mode baseband processor with partial clock activation and IP-MMU," in ISSCC Dig. Tech. Papers, Session 13.3, Feb. 2008.

[8] M. Ito et al., "A 65 nm single-chip application and dual-mode baseband processor with par-tial clock activation and IP-MMU," IEEE Journal of Solid-State Circuits, 44(1), Jan. 2009, pp. 83–89.

[9] K. Hagiwara et al., "High performance and low power SH2A-DUAL core for embedded microcontrollers," in COOL Chips XI Proceedings, Session XI, no. 2, April 2008. 1.

[10] H. Kido et al., "SoC for car navigation systems with a 53.3 GOPS image recognition engine," in HOT CHIPS 21, Session 6, no 3, Aug. 2009.

[11] R.G. Daniels, "A participant's perspective," IEEE Micro, 16(2), Apr. 1996, pp. 8–15.

[12] L. Gwennap, "CPU technology has deep roots," Microprocessor Report, 10(10), Aug. 1996, pp. 9–13.

[13] H. Nakamura et al., "A circuit methodology for CMOS microcomputer LSIs," in ISSCC Dig. Tech. Papers, Feb. 1983, pp. 134–135.

[14] S. Kawasaki, "SH-II A low power RISC microprocessor for consumer applications," in HOT Chips VI, Aug. 1994, pp. 79–103.

[15] A. Hasegawa et al., "SH-3: high code density, low power," IEEE Micro, 15(6), Dec. 1995, pp. 11–19.

[16] F. Arakawa et al., "SH4 RISC multimedia microprocessor," in HOT Chips IX Symposium Record, pp. 165–176, Aug. 1997.

[17] O. Nishii et al., "A 200 MHz 1.2 W 1.4GFLOPS microprocessor with graphic operation unit," in ISSCC Dig. Tech. Papers, Feb. 1998, pp. 288–289, 447.

[18] F. Arakawa et al., "SH4 RISC multimedia microprocessor," IEEE Micro, 18(2), March/April 1998, pp. 26–34.

[19] P. Biswas et al., "SH-5: the 64 bit SuperH architecture," IEEE Micro, 20(4), July/Aug. 2000, pp. 28–39.

[20] K. Uchiyama et al., "Embedded processor core with 64-bit architechture and its system-on-chip integration for digital consumer products," IEICE Transactions on Electronics, E84-C(2), Feb. 2001, pp. 139–149.

[21] F. Arakawa, "SH-5: a first 64-bit SuperH core with multimedia extension," in HOT Chips 13 Conference Record, Aug. 2001.

[22] F. Arakawa et al., "An embedded processor core for consumer appliances with 2.8GFLOPS and 36M polygons/s FPU," in ISSCC Digest of Technical Papers, vol. 1, Feb. 2004, pp. 334–335, 531.

[23] M. Ozawa et al., "Pipeline structure of SH-X core for achieving high performance and low power," in COOL Chips VII Proceedings, vol. I, pp. 239–254, April 2004.

[24] F. Arakawa et al., "An embedded processor core for consumer appliances with 2.8GFLOPS and 36M polygons/s FPU," IEICE Transactions on Fundamentals, E87-A(12), Dec. 2004, pp. 3068–3074.

[25] F. Arakawa et al., "An exact leading non-zero detector for a floating-point unit," IEICE Transactions on Electronics, E88-C(4), April 2005, pp. 570–575.

[26] F. Arakawa et al., "SH-X: an embedded processor core for consumer appliances," ACM SIGARCH Computer Architecture News, 33(3), June 2005, pp. 33–40.

[27] T. Kamei et al., "A resume-standby application processor for 3G cellular phones," in ISSCC Dig. Tech. Papers, Feb. 2004, pp. 336–337, 531.

[28] M. Ishikawa et al., "A resume-standby application processor for 3G cellular phones with low power clock distribution and on-chip memory activation control," in COOL Chips VII Proceedings, Vol. I, April 2004, pp. 329–351.

[29] M. Ishikawa et al., "A 4500 MIPS/W, 86 μA resume-standby, 11 μA ultra-standby application processor for 3G cellular phones," IEICE Transactions on Electronics, E88-C(4), April 2005, pp. 528–535.

[30] T. Yamada et al., "Low_power design of 90-nm SuperHTM processor core," in Proceedings of 2005 IEEE International Conference on Computer Design (ICCD), Oct. 2005, pp. 258–263.

[31] F. Arakawa et al., "SH-X2: an embedded processor core with 5.6 GFLOPS and 73M polygons/s FPU," in 7th Workshop on Media and Streaming Processors (MSP-7), Nov. 2005, pp. 22–28.

[32] T. Yamada et al., "Reducing consuming clock power optimization of a 90 nm embedded processor core," IEICE Transactions on Electronics, E89–C(3), March 2006, pp. 287–294.

[33] T. Kamei, "SH-X3: enhanced SuperH core for low-Power multi-processor systems," in Fall Microprocessor Forum 2006, Oct. 2006.

[34] F. Arakawa, "An embedded processor: is it ready for high-performance computing?" in IWIA 2007 Jan. 2007, pp. 101–109.

[35] Y. Yoshida et al., "A 4320 MIPS four-prcessor core SMP/AMP with individually managed clock frequency for low power consumption," in ISSCC Dig. Tech. Papers, Session 5.3, Feb. 2007.

[36] S. Shibahara et al., "SH-X3: flexible SuperH multi-core for high-performance and low-power embedded systems," in HOT CHIPS 19, Session 4, no 1, Aug. 2007.

[37] O. Nishii et al., "Design of a 90 nm 4-CPU 4320 MIPS SoC with individually managed frequency and 2.4 GB/s multi-master on-chip interconnect," in Proc. 2007 A-SSCC, Nov. 2007, pp. 18–21.

[38] M. Takada et al., "Performance and power evaluation of SH-X3 multi-core system," in Proc. 2007 A-SSCC, Nov. 2007, pp. 43–46.

[39] M. Ito et al., "An 8640 MIPS SoC with independent power-off control of 8 CPUs and 8 RAMs by an automatic parallelizing compiler," in ISSCC Dig. Tech. Papers, Session 4.5, Feb. 2008.

[40] Y. Yoshida et al., "An 8 CPU SoC with independent power-off control of CPUs and multicore software debug function," in COOL Chips XI Proceedings, Session IX, no. 1, April 2008.

[41] H.T. Hoang et al., "Design and performance evaluation of an 8-processor 8640 MIPS SoC with overhead reduction of interrupt handling in a multi-core system," in Proc. 2008 A-SSCC, Nov. 2008, pp. 193–196.

[42] Y. Yuyama et al., "A 45 nm 37.3GOPS/W heterogeneous multi-core SoC," in ISSCC Dig., Feb. 2010, pp. 100–101.

[43] T. Nito et al., "A 45 nm heterogeneous multi-core SoC supporting an over 32-bits physical address space for digital appliance," in COOL Chips XIII Proceedings, Session XI, no. 1, April 2010.

[44] F. Arakawa, "Low power multicore for embedded systems," in COMS Emerging Technology 2011, Session 5B, no. 1, June 2011.

[45] G. Hinton et al., "A 0.18-μm CMOS IA-32 processor with a 4-GHz integer execution unit," IEEE Journal of Solid-State Circuits, 36(11), Nov. 2001, pp. 1617–1627.

[46] S.C. Woo et al., "The SPLASH-2 programs: characterization and methodological considerations," in Proc. ISCA, 1995.

[47] M. Ito et al., ""Heterogeneous multiprocessor on a chip which enables 54x AAC-LC stereo encoding," in IEEE 2007 Symp. VLSI, June 2007, pp. 18–19.

[48] H. Shikano et al., "Heterogeneous multi-core architecture that enables 54x AAC-LC stereo encoding," IEEE Journal of Solid-State Circuits, 43(4), April 2008, pp. 902–910.

[49] T. Kurafuji et al., "A scalable massively parallel processor for real-time image processing," in IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers, Feb. 2010, pp. 334–335.

[50] K. Iwata et al., "256 mW 40 Mbps Full-HD H.264 high-Profile codec featuring a dual-macroblock pipeline architecture in 65 nm CMOS," IEEE Journal of Solid-State Circuits, 44(4), Apr. 2009, pp. 1184–1191.

[51] K. Iwata et al., "A 342 mW mobile application processor with full-HD multi-standard video codec and tile-based address-translation circuits," IEEE Journal of Solid-State Circuits, 45(1), Jan. 2010, pp. 59–68.