Regular Paper

Multimedia Accelerator ACE: A Practical Approach for Multimedia Applications

Chikako Nakanishi,† Isao Minematsu,† Hisakazu Sato,† Kiyoshi Nakakimura,† Takahiko Arakawa† and Shuhei Iwade†

In a highly information-oriented society, demands for information appliances are rapidly increasing, and high performance LSIs to meet this demand are being widely developed. In particular, an LSI that has high performance both in control tasks and signal-processing tasks is in high demand. There have been several approaches to address this issue, but they each have their respective strengths and limitations. In this paper a novel approach to solve conventional problems is proposed utilizing a DSP accelerator "ACE" that can be implemented with a general-purpose MCU. It features high programmability so that it can be adapted to various applications. Also it can be realized at low cost by limiting its usage as an accelerator. Moreover, in order to boost the total performance of the MCU-ACE system, independent parallel execution is exploited. In order to reduce the overhead of the transactions between the cores, a double-buffered shared memory and a well thought out simple interface are employed. A software development methodology to improve productivity in multi-processor systems by unifying the development platforms is also proposed. For a fair comparison with various architectures, an instruction set simulator and an instruction scheduler were developed. Using benchmark programs in the audio-voice area with these tools, the new approach performs 53–61% better than the conventional one. Thus the superiority of the proposed approach over conventional approaches is proven.

1. Introduction

In the highly information-oriented society that we are facing, information appliances are increasing their number of functions rapidly and becoming more and more complex[1)~3)]. For example, digital handsets equipped with a Java-enabled Web browser are becoming popular. Similarly, set-top-boxes incorporating such functions as Voice-Over-IP (VoIP) for the Internet telephones and digital audio decoding of MPEG1 audio layer-3 (MP3) while maintaining conventional user interface operations are also growing in number.

As a result of such a rapid improvement in appliances with multimedia applications, the LSIs used in these appliances need to meet various requirements including higher performance and a shorter development period. In particular, since such applications need to operate complicated control tasks and data-intensive signal processing tasks at the same time, an LSI which sustains high performance for both these kinds of processing is in high demand.

One solution to this requirement might be to adopt a general purpose micro-controller unit (MCU) that would deliver a high enough performance to meet all these diverse operations. However, since low cost and low power are among the most critical requirements for these applications, this MCU-centric solution is inappropriate in most cases. Another solution has usually been adopted in which an ASIC is used with an MCU or with a digital signal processor (DSP). However, since a quick turn around time (TAT) is becoming more critical due to both an even shorter product life cycle and ongoing specification changes in multimedia standards, this approach, which requires a considerably long TAT, is becoming less attractive. As a result, a new solution with high programmability to meet this quick TAT requirement while conforming to conventional demands is now required for multimedia applications.

Recently, many approaches to address this problem have been proposed. They include an MCU with enhanced DSP-like functions[4)~6)], an MCU combined with a DSP in a single LSI[7)], and mediaprocessors which feature novel architectures such as very long instruction word (VLIW)[8),9)]. These approaches, however, have their respective strengths and limitations.

Aside from these possible solutions, we are interested in still another approach that features a combination of a general-purpose MCU and

† System LSI Development Center, Mitsubishi Electric Corporation

an application specific accelerator in order to achieve a well-balanced performance between control and signal processing tasks. We consider that this approach has inherent advantages over other approaches as follows. Firstly, it is expected to achieve higher performance with a smaller die area by limiting the usage of the accelerator. Secondly, it can realize a lower power dissipation since a dynamic power management feature can easily be adopted for the accelerator, which is separated from the accompanying MCU. Finally, since the accelerator can be implemented with a developed MCU, utilizing resources for the MCU can reduce the development periods for its software tools and LSI hardware. Overall, this approach should provide the best tradeoff between high performance, low cost, low power and a short TAT.

Our approach highlights a DSP accelerator that commands high programmability and high portability so as to be compatible with a general-purpose MCU in order to be used in many applications. This DSP accelerator, which we have dubbed ACE, can be realized with a small die area by limiting its usage only for the accelerator accompanying an MCU and, as a result, by removing the hardware only needed for standalone usage. It also features efficient interfaces with MCUs in order to reduce the overhead of transactions between MCU and ACE. Also, by utilizing this inter-core interface, we propose a solution to software development problems that are generic in multi-core hardware configurations.

In this work, we compare the characteristics of conventional approaches, and describe the motivation for our approach in Section 2. Next, we describe the features of our hardware specification in Section 3, and some novel ideas to simplify software development in Section 4. In Section 5 we evaluate the performance of our approach by comparing with some conventional approaches, and we engage in a detailed discussion regarding the comparison. Finally, we offer our conclusions in Section 6.

2. Conventional Approaches

MCUs deliver high performance for control processing while DSPs deliver high performance for signal processing. In order to achieve high performance for both processes at the same time, several approaches are possible, as follows.

(1) **MCU enhancing DSP functions**
An MCU can introduce DSP-like instructions or mechanisms to compensate for the lack of signal processing performance. Such instructions include multiply-and-accumulate (MAC), saturation, count-leading-zero's and so on. ARM9E [4] and MIPS "Jade" [6] are among the examples. Conversely, a DSP can introduce MCU-like mechanisms, such as flexible input-output (I/O) functions, for the corresponding reason.

(2) **Combining MCU with DSP via a bus**
This approach combines an MCU with a DSP by connecting then with a dedicated bus interface, examples of this approach include M*Core[7]. A derivative approach is to connect an MCU and signal-processing unit in a loosely coupled way[10),11)].

(3) **Mediaprocessors**
A newly developed mediaprocessor can be another solution, and VLIW architecture is likely to be adopted in these processors. One example is SH-DSP [8] which combines an MCU core with a DSP in a tightly coupled way, the resulting core behaving as a single VLIW processor. Another example is D30V [9] which realizes both MCU and DSP functions within one core using VLIW.

The first approach has an advantage over others in that the development cost of such an LSI is expected to be the lowest. However, this approach has the possible drawback of low effective performance since the optimization for an existing core can be less efficient than a core designed from scratch. Furthermore, introducing sophisticated mechanisms to overcome this drawback might reduce the advantage of a low development cost.

The second approach has an inherent advantage in that the hardware and software resources of the original cores can basically be appropriated. For example, the hardware development period of an LSI in which the cores are already implemented is expected to be small. Also, some peripheral modules in such an LSI can be shared by the cores, which results in a small die size. In addition, since both cores can execute instructions independently, the effective performance can be the best of these approaches. However, this approach has serious drawbacks in terms of software development. Since the cores execute their respective programs with different instruction sets, soft-

ware development including debugging and optimizing is by its nature fiendishly difficult. Although a unified software development environment that handles both cores seamlessly would solve these difficulties, this tool is still too immature to be used in practice. Another serious drawback of this approach is that the transactions between the cores including inter-core data transfer and synchronizing could generate an adverse effect on performance to a considerable degree.

The third approach, on the other hand, has an advantage over the second one in terms of software development productivity. Firstly programmers need learn the instruction set and the software tools of only a single core. Secondly, the concurrent development of control and signal processing tasks can be realized, which result in simpler procedures in debugging and verifying. Since the importance of software development is surpassing that of hardware development in order to achieve a short TAT, this advantage regarding software programming cannot be overestimated in many applications. However, this approach has the serious drawback of the high cost of hardware development because it needs to develop larger hardware than the others do. Moreover, a VLIW approach, which consumes considerable hardware resources in large register files or cross-bar switches, may suffer from ineffective hardware usage since its inherent parallelism cannot be fully exploited in many applications. Therefore, this approach might be the least effective one in terms of hardware cost-performance as well as power dissipation.

Of these three approaches, we consider that the second one, which combines an MCU with a DSP, is the most practical way to realize well-balanced high performance of control and signal processing tasks at the same time. Specifically, to enjoy the inherent advantage of its small hardware size, a combination of a general-purpose MCU and an application specific hardware accelerator would be quite an attractive derivative of the approach.

However, as stated above, the second approach has serious problems in interfacing the cores both in the light of software development productivity and of the transaction overhead. In addition, if an application specific accelerator were introduced in this approach, another advantage of a quick TAT would be spoiled due to the core development period.

In order to solve the problems of this approach as well as the possible drawbacks of introducing an accelerator, we propose a solution featuring a special DSP accelerator. This accelerator, known as ACE, realizes high programmability so that it can be adapted to various applications. Also, by removing the hardware needed only for standalone usage, the accelerator core can be realized with a small die area. Moreover, a simple interface mechanism is introduced to overcome the problem of inter-core transactions that would degrade overall performance. This inter-core interface mechanism also realizes a simple software development environment, which will solve the inherent problems of productivity in multi-core processor systems.

In the following sections, we describe the basic characteristics and software development environment using the ACE accelerator.

3. Architecture Overview

The design goal of the ACE accelerator is to provide a portable DSP core with high programmability so that it can be used with any MCU while maintaining the advantages of an MCU-DSP combination. In order to achieve this goal, we decided on two basic architectural features before breaking down its specifications.

(1) The ACE core can be controlled from an MCU via dedicated memory-mapped registers so that the accompanying MCU basically does not have to be modified.
(2) The ACE core is not equipped with flexible operation for I/O or for memory access in order to reduce hardware size while maintaining high programmability

Moreover, we adopted the following ideas to solve the problems of this approach discussed in the previous section.

(1) In order to reduce the overhead of data transfer, a shared memory placed between the cores is employed so that explicit data transfer is unnecessary.
(2) In order to facilitate transactions between the cores, the shared memory is implemented as double-buffered, each core can only access one side of this exclusively.
(3) In order to reduce the control overhead from the MCU, a few dedicated control registers are employed to synchronize the programs running on both cores.
(4) In order for MCU programmers to eas-

ily control the ACE, the control functions used in MCU programs written in C are provided with simple application programming interface (API). The functions are implemented in a non-blocking way with the result that the MCU program does not have to waste cycles.

(5) In order to make the software development environment of the cores as seamless as possible, ACE's debugging tools are incorporated into those of the MCU. As a result, the concurrent development of the programs on the MCU and the ACE can be realized, which we expect will result in higher productivity in programming and debugging.

The double-buffered memory mechanism with a few control registers that enables exclusive access from the MCU and the ACE leads to a natural solution of the race condition problem, which is inherent in multi-processor systems. In addition, such a mechanism enables simple synchronization. These two features greatly alleviate the difficulties in developing software for multiprocessor system while maintaining high performance due to the parallel processing of the cores. In the following subsections, we describe the overall hardware structures and the details of the ACE.

3.1 Overall Hardware Structure

Figure 1 shows a typical hardware structure of an ACE combined with an MCU. The ACE consists of a core, which includes a processing unit, a data memory (XMEM), a coefficient memory (YMEM), an instruction memory (IMEM), and a set of control registers. The XMEM is shared by both cores and consists of a pair of memory areas. The MCU and the ACE can access only one of the memory spaces at a time, exclusively. In addition, one of the memory spaces is mapped to the same address alternately when accessible. In other words, XMEM is implemented as double-buffered.

The ACE can access XMEM, YMEM and IMEM with no-waiting. The XMEM and YMEM are accessed using address registers in a similar way as conventional DSPs. It should be remarked here that only one side of the double-buffered XMEM is accessible at a time.

On the other hand, the MCU can access all the resource of ACE including XMEM, YMEM, IMEM and all the control registers when ACE is inactive. When ACE is active and executing a program, the MCU can access only that one side of XMEM selected for its exclusive access and a control register indicating the status of ACE. The resources of ACE can be accessed in the same way as an ordinary memory-mapped peripheral device by using general memory access instructions such as load and store. The details of the control registers are described in the following subsections.

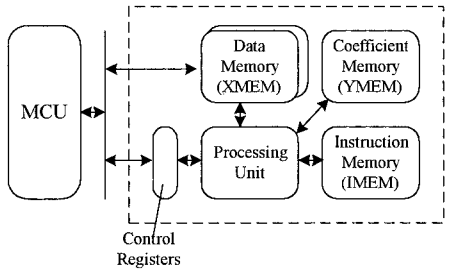

Fig. 1 Typical hardware structure of ACE combined with MCU.

3.2 Shared Memory

As described above, the shared memory XMEM consists of a pair of memory spaces, each of which can be accessed only by the MCU or the ACE respectively. In this section, we describe how this double-buffer memory is realized as well as its behavior in detail.

Figure 2 shows an example of the memory map of ACE from MCU. Note that one side of the double-buffered XMEM, which is selected with one of the control registers "BUFFER_NUM", is mapped onto a different address in addition to its real address. As a result, when ACE is active, MCU can exclusively access the selected side of XMEM with the same address, while ACE is exclusively accessing the other side. This address is called the "shadow address". When ACE is inactive, on the other hand, MCU can access both sides of XMEM at their real address as well as one side at its shadow address. The BUFFER_NUM register can be modified from MCU only when ACE is inactive.

The behavior of the write access to the shared memory XMEM when ACE is active is described in more detail with **Fig. 3**. The address data from MCU (MCU address) is transferred to one of the two memory spaces of XMEM according to buffer_num which is the value of the BUFFER_NUM register. Conversely, the address data from ACE (ACE address) is transferred to the other side of XMEM. Similarly, data that is read out from one of the buffers is transferred to either MCU or ACE according

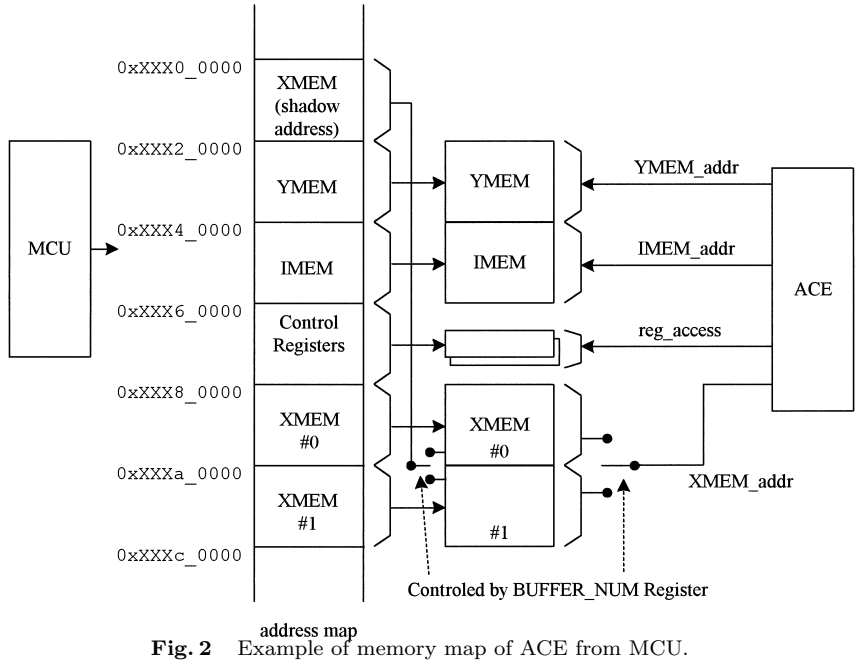

Fig. 2 Example of memory map of ACE from MCU.

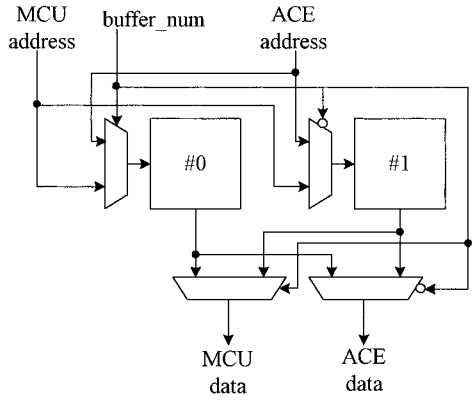

Fig. 3 Shared memory.

to BUFFER_NUM. As a result, the MCU and ACE can access one of the buffers exclusively. For example, if BUFFER_NUM is 0, the MCU can access XMEM #0 at the shadow address.

As described earlier, BUFFER_NUM can be modified only by MCU when ACE is inactive. This interlock mechanism avoids the inherent race condition problem. Otherwise, MCU or ACE may toggle the BUFFER_NUM by mistake when the other core is accessing, or MCU and ACE may try to access the same buffer. We do not believe that this access methodology will pose any disadvantages in software development. It should be noted that when MCU tries to modify the BUFFER_NUM when ACE is active, it has no effect; the program must confirm whether ACE is active or not before it tries to toggle the buffer.

3.3 Control Registers

In this section, the control registers of ACE are listed with brief functional explanations.

- `RESET_CNT`
  Used for resetting ACE from MCU. ACE cannot access this register.
- `ACE_CNT`
  Used for examining and changing the status of ACE from MCU. MCU can modify this register only when ACE is inactive, though it can read the register anytime. By modifying the register, MCU can change its status to either ACTIVE or HALT. In HALT status ACE ceases executing its program and all the resources are accessible from MCU. Though ACE cannot access the register directly, it can change the value when it falls into the HALT state by issuing a "halt" instruction.
- `PC_CNT`
  Mirror register of PC in ACE. ACE starts executing a program from the address specified in this register. MCU can modify this register only when ACE is inactive.
- `BUFFER_NUM`
  Used to specify the buffer that MCU can access exclusively as described in the previous subsection.
- `BD_CNT`

Table 1 ACE API.

Name	Operation
reset_ace	Initializing and resetting (setting RESET_CNT)
start_ace	Start executing program (setting PC_CNT and ACE_CNT)
check_ace_st	Checking the status (reading ACE_CNT)
halt_ace	Halting (setting ACE_CNT)
set_ace_buffer	Specifing buffer of shared memory (setting BUFFER_NUM)
req_ace_db	Requesting debug interrupt (setting DB_CNT)

Request a debug interrupt from MCU. When the request is accepted, ACE stops executing the program and falls into the DEBUG state.

The basic procedure of starting ACE from MCU is explained as follows.

(1) MCU confirms whether ACE is in the HALT state by examining ACE_CNT.
(2) MCU sets the start address of the program to PC_CNT.
(3) MCU changes ACE to its ACTIVE state by changing ACE_CNT.

Note that ACE does not notify MCU when it falls into the HALT state. In order to synchronize the programs running on MCU and ACE, MCU needs to check the status of ACE.

3.4 Control Functions (ACE API)

In order for software programmers to use the control procedures from MCU described in the previous section, the control functions are defined with simple APIs. The number of such control functions can be reduced for several reasons. First, since ACE is equipped with a hardware mechanism to interlock access to the double-buffered XMEM, software does not need to pay regard to race condition problems and a function for such a purpose is not required. Second, since XMEM, YMEM and IMEM can be accessed from MCU like conventional memory, APIs for transferring data are also unnecessary. Consequently, the only functions required are for resetting, starting, checking, halting, specifying the buffer of the shared memory, and requesting debug interrupt.

Those functions are summarized in **Table 1** with simple explanations regarding how they are implemented in terms of the usage of the control registers. The functions can be used from C programs executed on MCU.

3.5 ACE Core

The ACE core has a similar functional structure as conventional 16-bit fixed-point DSPs as shown in **Fig. 4**. The address generation unit comprises of four address registers (AR0-AR3), four address modification registers (AMD0-AMD3) and two adders, one of which supports modulo addressing. It generates two addresses for XMEM and YMEM concurrently. The data arithmetic unit, on the other hand, has four 16-bit general purpose registers (R0-R3), four 16-bit temporary registers (TR0-TR3) and two 40-bit accumulators (A0-A1). It comprises of a shifter (SFT) an ALU and a multiplier. The multiplier product can be fed into the ALU. A MAC operation is realized by pipelining the multiplier and the ALU. **Table 2** summarizes the basic characteristics of the ACE core.

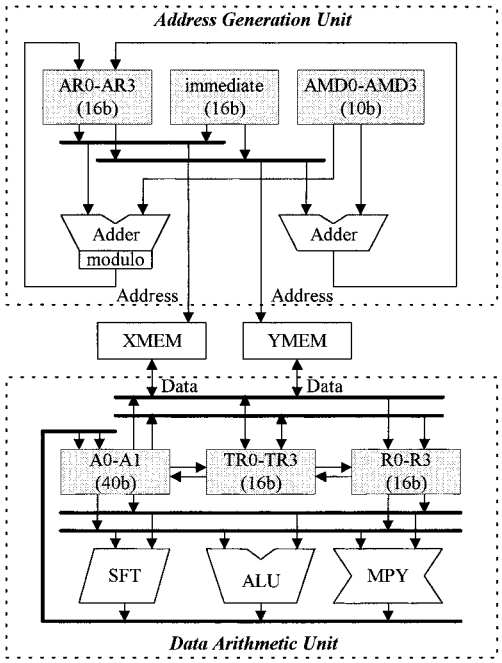

Fig. 4 Block diagram of ACE core.

Table 2 ACE core.

Instruction word	16-bit
Data word	16-bit (integer/fixed-point)
Multiply-and-accumulate	16 x 16 + 40-bit (1 cycle throughput)
Clock frequency	100 MHz
Performance	100 MMAC/s
Design methodology	Soft Macro
Process technology	0.18 um (CMOS)
Logic size	24 K gates (approx.)

4. Software Development

Since our approach with ACE is similar to

the conventional one of an MCU combined with a DSP, it has the same drawbacks in terms of software development. Furthermore, since ACE is expected to be used with a finer granularity to its software structure in order to exploit its advantage of less overhead in data transfers and synchronization, software development can be more elaborate, for example in balancing workload between the cores. As a result, it is very important to assuage a possible increase of problems in developing software for ACE.

In this section, we propose a methodology to improve software productivity, especially in debugging. First, we explain the software development flow by comparing that of an MCU alone with that of an MCU combined with a DSP. We then propose a software toolchain methodology for the concurrent development of a program running on MCU and ACE.

4.1 Software Development Flow

Generally speaking, a program coded in a high-level language like C results in a less effective implementation than one coded in assembly language. Therefore, the intensive portions in a program are coded in assembly to meet this performance requirement. This tendency is strengthened in the case of signal processing. **Figure 5** shows a typical flow in developing a program that is executed on either an MCU or a DSP alone where the intensive portion in a program coded in assembly is called from C as a function.

On the other hand, **Fig. 6** illustrates a typical flow in programming for an MCU combined with DSP. The differences from Fig. 5 are written in italics. It is apparent this flow is much more complicated with a longer feedback loop than the previous one, especially when the software development environment is not unified.

Since our approach with ACE is similar to this multi-core approach, the software development environment would resemble the one shown in Fig. 6 if we were to implement it simply. Therefore, we decided to set one of the design goals of ACE to be that its software development would be as seamless as possible like the one for single core approaches.

We will now explain the flow in developing our software, shown in **Fig. 7**, by comparing with Fig. 5. The differences from Fig. 5 are written in italics. Similar procedures are taken until the benchmarking step. The only major difference is that a discussion toward parallel execution utilizing the two cores is added, i.e.,

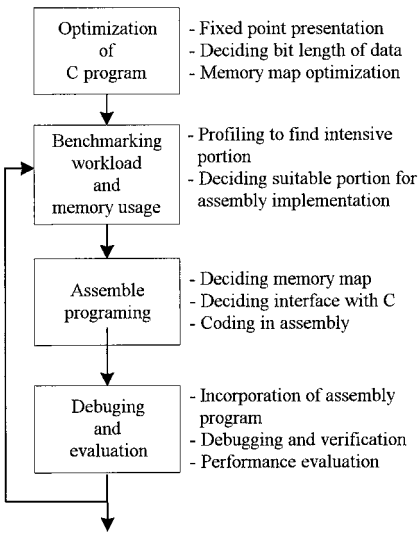

Fig. 5 Software development flow of conventional MCU.

whether the intensive portion is suitable for parallel execution with ACE is discussed by estimating the performance gain. This is a very important step in boosting performance with this method. Another difference in the flow is that in Fig. 7 interfacing using an API must be considered, whereas Fig. 5 includes a discussion about interfacing with C. Since ACE is equipped with the API functions explained in the previous section, this disparity does not complicate the flow with ACE. In addition, since the software tools are integrated into those of the MCU, as described in the next subsection, software development, especially in debugging and evaluation, can be realized in a natural way. Overall, the whole flow of software development and its methodologies are less complicated than those of the MCU combined with DSP approach. It is also apparent that they are as simple as those shown in Fig. 5. This results in a shorter feedback loop in optimizing and balancing programs executed on both cores.

4.2 Software Development Environment

Here, a methodology for software development is proposed which utilizes a single debugging and evaluating environment in order to simplify these procedures. With this methodology, programmers are able to concurrently, and more easily, debug programs executed on both cores with a single environment. This methodology will provide much higher software produc-

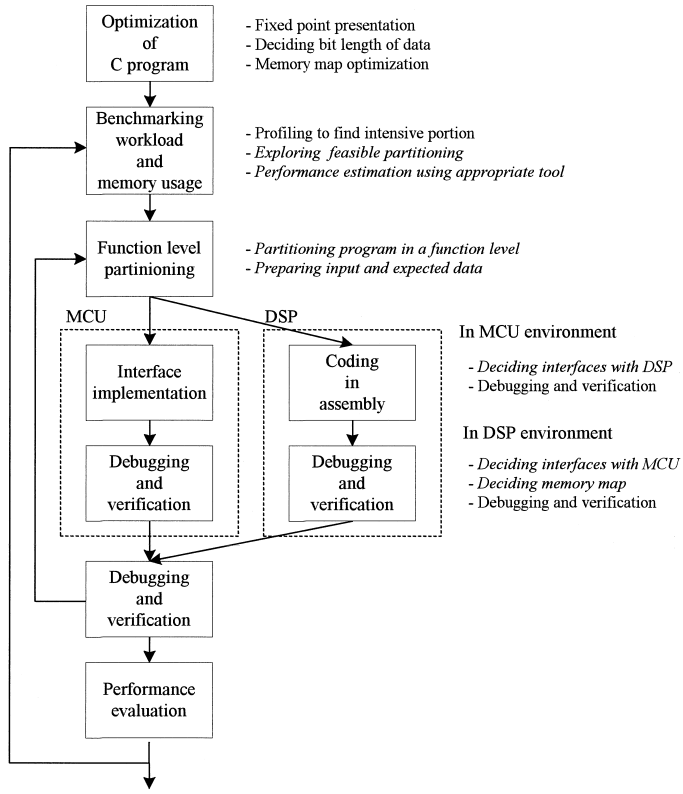

Fig. 6 Software development flow of MCU combined with general-purpose DSP.

tivity due to the following reasons.

(1) You do not have to arrange a separate testbench for ACE (DSP), since the input data is provided and the results are checked by a program executed on MCU (or its simulator) on the fly.

(2) The same program can be used when MCU emulates ACE and ACE hardware is actually executing. A library implementing API conceals the implementation of ACE.

(3) Balancing the workload is much simplified with a single tool.

Next, we describe the ACE emulator used in this methodology. The ACE emulator is a software module that emulates the total behavior of the ACE including interfaces to MCU. It is implemented in a library with a compatible interface with APIs. It also has debugging functions such as setting a breakpoint.

The ACE emulator starts executing a program on executing an API function from the address which has been specified with another API. ACE returns its control to the application program running on MCU when it stops executing a program by executing a "halt" instruction. Thus basically MCU does not stop

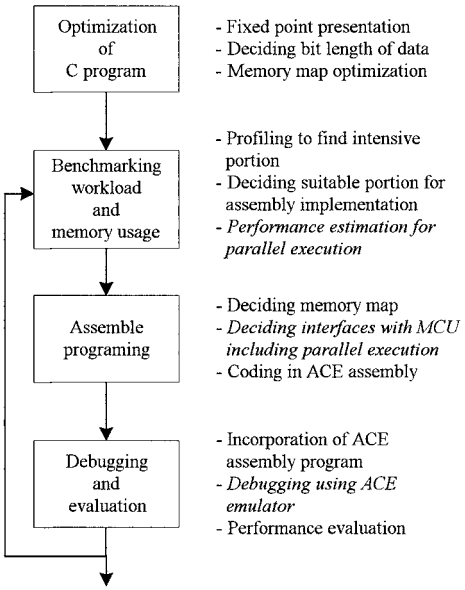

Fig. 7 Software development flow of MCU combined with ACE.

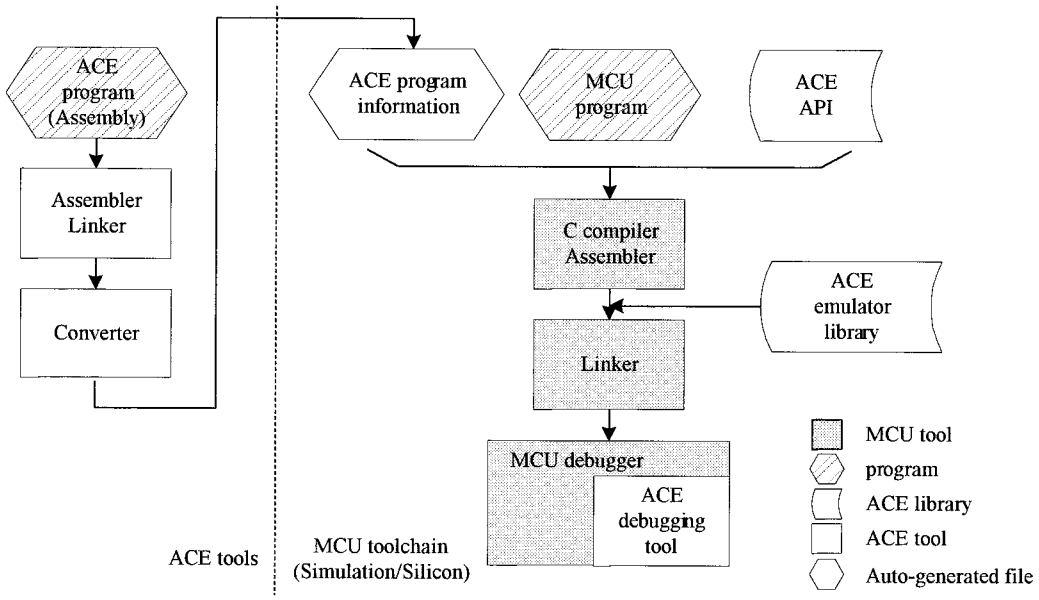

Fig. 8 Software development toolchain of MCU with ACE.

ACE while it is executing a program. This is also the case when you use ACE hardware instead of the emulator library. Note that this restricted control flow does not cause possible limitations in debugging.

By executing a program linked with the API emulator library, the MCU program can debug a program easily. Since the ACE emulator utilizes the console I/O for operations, it is possible to examine the values of all the registers, the contents of memory, set a breakpoint and execute in single step mode by issuing a command. In addition, if this program were executed on a debugger for the MCU, the concurrent debugging of the programs running on both cores could easily be realized. We believe that we are able to provide an adequate debugging environment with this methodology.

Figure 8 is a schematic diagram of the software development toolchain. The ACE program coded in assembly is converted into a few C header files in its memory images in XMEM, YMEM and IMEM as well as some mapping information. A program running on MCU with this ACE program includes these header files when compiling and is linked with a proper API library. In this way, the ACE program is incorporated in the MCU program. As for the initialization of ACE, the MCU program copies ACE program images to IMEM and coefficient images to YMEM, by using memory access instructions. As with the ACE emulator, a memory image of an ACE program is included in its memories IMEM, XMEM, and YMEM.

5. Evaluation

In this section, we try to evaluate the performance of ACE by comparing it with conventional approaches that have been described in Section 2. In Section 5.1, we describe the comparison platform as well as the architectural models characterizing each approach. Then, we give explanations of the benchmark programs used in Section 5.2. Finally, we present the comparison method, and discuss the results in Section 5.3.

5.1 Evaluation Models and Platform

For a fair comparison of the processor architectures, we developed a platform in which the number of parallel executions and the overhead between the cores have the sole effect on the performance. In order to conceal the effect of the detailed instruction sets, we employed ACE to represent a general-purpose DSP.

In order to compare the performance gain due to several DSP enhancement approaches, a reference MCU should be so chosen that it has little DSP functions in itself. We believe that the early generations of the MIPS architecture are well-suited for this purpose, and we employed a reference MCU based on MIPS R3000 with one multiplier added; we call this model the "basic MCU" below. Also, we assume a perfect cache. Next, we explain the architectural models used

Table 3 Penalties and overhead.

	Basic MCU	Model A	Model C	Model D	Model B	Model E
Branch	2	2	2	2	-	-
Load	2	2	2	2	-	-
Multiply	2	2	2	2	-	-
Other Instructions	1	1	1	1	-	-
Activate	-	-	-	-	20	5
Synchronize	-	-	-	-	12	6
Reset	-	-	-	-	10	5
Data transfer	-	-	-	-	N*	0
Specifying a buffer	-	-	-	-	-	6

* The data transfer overhead of model B is proportional to the transferred data word size N.

Table 4 Benchmark programs and their code sizes (Kbytes).

Program Name	General description	Original	Optimized		ACE
			MCU		
MP1	MPEG1 audio layer 1 decoding	8.6	5.1	(−3.5)	2.2
MP3	MPEG1 audio layer 3 decoding	19.7	13.6	(−6.1)	2.8
G.723.1	Voice codec	99.5	66.5	(−33.0)	8.8
autcor	Auto-correlation	0.5	0.4	(−0.1)	0.06
fbital	Bit manipulation	0.8	0.5	(−0.3)	0.08
FFT	128-point fast-fourier-transform	1.2	0.5	(−0.7)	0.15
viterb	Viterbi-decoding	2.3	0.3	(−2.0)	0.19
conven	Convolution encoding	0.7	0.4	(−0.3)	0.10

for the comparison.

Model A: MCU with DSP enhancement (single-issue)

This model is a single-core approach where some DSP functions are added to the basic MCU as follows. It includes MAC, multiply with shift, and a loop instruction. Also, it has a saturation function, and auto increment/decrement addressing with load/store instructions. Two 32-bit registers can be used as a single accumulator.

Model B: Multi-core

This is a model where a basic MCU and a general-purpose DSP are connected in a loosely coupled way. The cores do not have a shared memory. The cores can execute programs independently.

Model C: VLIW

This model is a four-way VLIW architecture that has the same instruction set as the basic MCU. The execution units include one multiplier, one arithmetic unit, two memory units, one shifter and one branch unit. Since this model is so defined that the combination of the execution units realizes a MAC operation with one-cycle throughput, it properly represents a conventional VLIW approach.

Model D: MCU with DSP enhancement (four-issue)

This model is a further enhancement over model A. It has five heterogeneous execution units, four of which can execute their respective instructions at a maximum. The execution units consist of one multiplier, one arithmetic unit, one memory unit, one shifter, and one branch unit. As with model C, this model is so designed that it is able to realize a MAC operation with one-cycle throughput.

Model E: ACE

This is a model where a basic MCU and a general-purpose DSP (ACE) are combined as described in Section 3.

Table 3 summarizes the penalties of the instruction groups as well as the overhead between the cores for each of the models.

5.2 Benchmark Programs

We employed several benchmark programs with specific applications in mind, MPEG-1 audio layer-1 and layer-3 (MP1 and MP3) are popular standards in the digital audio area, and G.723.1 represents a voice-band application. In addition, five small programs widely used in many telecommunication applications are employed. All the benchmark programs were originally written in C. **Table 4** summarizes the benchmark programs with their original code sizes when compiled with a MIPS C-compiler for R3000 (cc Ver. 3.0) and code for MCU and ACE after their respective intensive portions have been transferred to ACE.

As for MP1 and MP3, the workloads of MCU and ACE are balanced, and some optimizations

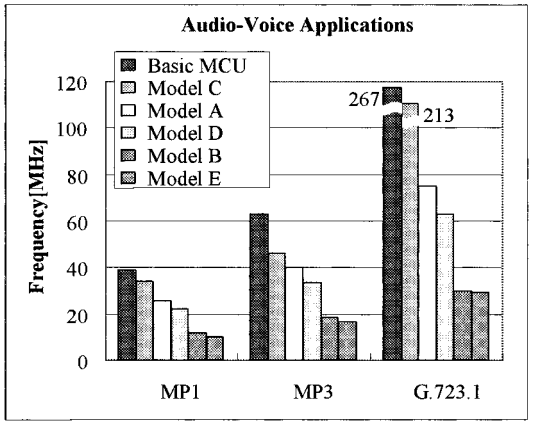

Fig. 9 Performance comparisons using audio-voice applications.

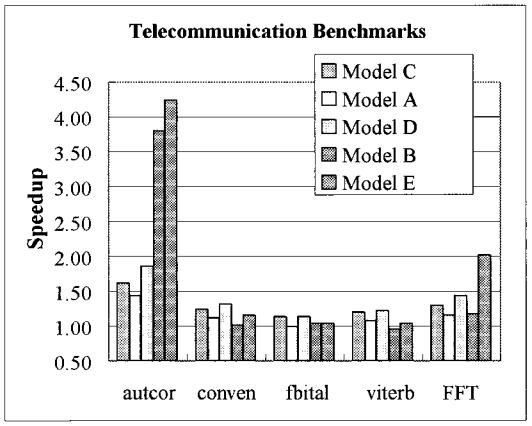

Fig. 10 Performance comparisons using telecommunication benchmarks.

have been done to the MCU program in order to exploit the parallel executions of the cores. As for G.723.1, the intensive portions are executed by ACE but the parallel executions of the cores are not utilized. On the other hand, five telecommunication benchmarks are totally ported to ACE except for their respective top routines.

5.3 Results and Discussion

We have developed a cycle-accurate instruction set simulator for all the models described in the previous section. Also, we have developed an instruction scheduler that post-processes an assembler program generated by the MIPS C-compiler to run on the simulator. This instruction scheduler analyzes the dependencies within a basic block or within a function, and generates a program that utilizes the function and inherent parallelism of the target model. With this scheduler, the programs are generated which are executed on model A, model C, and model D. On the other hand, programs executed on ACE (DSP) are hand-coded in the assembly, and they are used for models B and E.

The performances of the large benchmarks, MP1, MP3, and G.723.1, are compared at the necessary frequencies in **Fig. 9** where a shorter bar represents a better performance. On the other hand, all the small benchmarks from the telecommunications area are compared for their relative performance to the basic MCU in **Fig. 10** where a longer bar represents a better performance.

It is apparent from Fig. 9 that model E (ACE) provides the best performance in all the three benchmarks, followed by model B, model D, model A and model C. Specifically, since MP1 and MP3 include an intensive filter using a MAC, introducing the DSP function proved to be very effective. Moreover, since model D provides better performance than model A, the effect of parallel execution is evident. On the other hand, model E and model B provide better performance than model D by 45–55%. This result indicates that utilizing function-level parallel execution can be more effective than utilizing instruction-level parallel execution. A comparison between model B and model E gives a noteworthy result in that model B requires 12–20% more cycles. This result provides the insight that the overhead of inter-core transactions has a considerable impact on the total performance.

Since G.723.1 requires the saturation function very often, the basic MCU and model C provide much worse performances than the other models. We can thus conclude that these approaches are not suitable for this application. Furthermore, model B and model E which feature independent executions of the cores provide better performance than model A by 60–61%, and model D by 53%. This result indicates that the DSP functions added to models A and D are insufficient to gain good performance. Moreover, model E provides better results than model B by 0.6 MHz, which is equivalent to 29% of the optimized MCU workload. Since the MCU is assumed to execute complicated protocol processing, this difference may generate a considerable difference in the overall performance. This is another example where the overhead of inter-core transactions has a considerable impact on the MCU workload.

However the telecommunication benchmarks

provide less evidence of the apparent superiority of the B and E architectures. Model E presents the best performance in autcor and FFT, which contain intensive DSP operations. On the other hand, model C and model D give better results than model E in the other benchmarks, which contain less DSP operations. Overall, the relative performances are greatly dependent on the characteristics of a program, and general architectural superiority cannot be determined. In other words, in order to select the proper processor architecture, it is important to know in advance the characteristics of the program.

Another result shown in Fig. 10 is that model E presents a much better performance than model B for the FFT benchmark whereas it presents comparable performance for the other benchmarks. This disparity can be accounted for by the fact that the FFT involves the most data transfer of all the benchmarks and, as a result, the inter-core data transfer consumes a majority of the execution cycles for model B. This is still another example that indicates the transaction overhead when using an MCU has a strong impact on performance. In addition, the results of comparing the two models implies that data transfer overhead would be even more conspicuous in the case of benchmarks where DSP enhancements are effective in reducing the computation cycle.

Finally, we can conclude from the results in this section that the introduction of an application specific hardware accelerator boosts performance by utilizing parallel processing. Parallel processing is more effectively exploited with the help of the double-buffered shared memory since both cores can execute programs more independently. Moreover, it is evident that an inter-core transaction causes a serious impact on performance, and that smaller overheads in data transfer and in synchronization assuage this effect considerably. Overall, we have demonstrated the effectiveness of our approach in achieving higher performance.

6. Conclusion

In a highly information-oriented society, demands for information appliances are rapidly increasing, and high performance LSIs to meet these demands are being widely developed. In particular, an LSI that has high performance both in control tasks and signal-processing tasks is in high demand. In such applications where low cost and low power are among the most critical requirements, a general purpose MCU that delivers a high enough performance is inappropriate in most cases. A conventional approach using ASIC is becoming less appropriate because its considerable TAT is crucial in some applications. Several new approaches have been proposed to address these problems, but they each have their respective strengths and limitations.

In this paper, we have proposed a novel approach to solve these problems utilizing a DSP accelerator "ACE" that can be implemented with any general-purpose MCU. It features high programmability so that it can be adapted to various applications. Also, it can be realized at low cost by limiting its usage as an accelerator. Moreover, in order to boost the total performance of the MCU-ACE system, independent parallel execution is exploited. In order to reduce the overhead of the transactions between the cores, a double-buffered shared memory and a well thought out simple interface are employed. We have also proposed a software development methodology to improve productivity in multi-processor systems by unifying the development platforms.

We have developed an instruction set simulator and an instruction scheduler in order to evaluate the performance of ACE by comparing it with conventional approaches. The results using audio applications show that, when independent parallel execution is exploited, the MCU-ACE combination performs 45–55% better than conventional approaches, where the proposed interface between the cores accounts for 12–20% in boosting performance. On the other hand, the results using a voice-band application show that, when sequential execution is exploited, our approach performs 53–61% better than the conventional ones, where the proposed interface accounts for a 29% reduction of the required MCU operation frequency.

We can conclude from these results that the introduction of a hardware accelerator that specializes in DSP functions greatly boosts performance when utilizing independent parallel execution. Another conclusion is that the proposed interface contributes to considerable performance gain by reducing the overhead of transactions between the cores.

The ACE can be implemented with a logic size as small as about 24 K gates, which is assumed to be less than half of an ordinary 32-bit

MCU. Consequently, an excellent cost performance can be realized by incorporating ACE with an MCU. Moreover, since ACE is so designed that it can be implemented with any MCU without changes, this characteristic can contribute to a short development period of a silicon-on-chip. Another characteristic of being an independent core is that it is easy to incorporate a dynamic power management feature.

In this paper, we have shown a superiority of our approach in term of performance. In the future, we are going to look at the overall power dissipation as well as hardware sizes in detail. At the same time, we are going to improve the ACE core to be more compact in terms of logic size and power dissipation. Moreover, in order to be used in a wider range of applications, we are planning further improvements in the architecture and the implementation; possible improvements include a combination of an MCU with multiple ACEs and a more flexible and reusable implementation.

References

1) Diefendorff, K. and Dubey, P.K.: How Multimedia Workloads Will Change Processor Design, *IEEE Computer*, Vol.30, No.9, pp.43–45 (1997).
2) Kozyrakis, C.E. and Patterson, D.A.: A New Direction for Computer Architecture Research, *IEEE Computer*, Vol.32, No.11, pp.24–32 (1997).
3) Eyre, J. and Eier, J.: DSP Processors Hit the Mainstream, *IEEE Computer*, Vol.31, No.8, pp.51–59 (1998).
4) Yarlagadda, K.: Arm Refocuses DSP Effort, *Microprocessor Report*, Vol.13, No.8, pp.11–13 (1999).
5) Yarlagadda, K.: Lexra Adds DSP Extensions, *Microprocessor Report*, Vol.13, No.11, pp.19–21 (1999).
6) Halfhill, T.R.: Jade Enriches MIPS Embedded Family, *Microprocessor Report*, Vol.13, No.7, pp.18–21 (1999).
7) Scott, J., Lee, L.H., Arends, J. and Moyer, B.: Designing the Low-Power M*Core Architecture, *Proc. IEEE Power Driven Microarchitecture Workshop at ISCA98*, pp.145–150, IEEE (1998).
8) Turley, J.: Hitachi Adds FP, DSP Units to SuperH Chips, *Microprocessor Report*, Vol.9, No.16, pp.10–11 (1995).
9) Yoshida, T., et al.: A 2V 250 MHz Multimedia Processor, *ISSCC Digest of Technical Papers*, pp.266–267 (1997).
10) Turley, J.: ARM Tunes Piccolo for DSP Performance, *Microprocessor Report*, Vol.10, No.15, pp.17–21 (1996).
11) Dolle, M. and Schlett, M.: A Cost-Effective RISC/DSP Microprocessor for Embedded Systems, *IEEE Micro*, Vol.15, No.5, pp.32–40 (1995).

(Received January 25, 2002)
(Accepted May 9, 2002)

Chikako Nakanishi received a B.S. degree from the Dept. of Information and Computer Sciences, Faculty of Engineering Science, Osaka University, in 1988, and a Ph.D. degree in Engineering Science from Osaka University, in 2000. She joined Mitsubishi Electric Corporation in April 1988. Since then she has been engaged in the research and development of microprocessors and digital signal processors. She is now a staff member of the system LSI Development Center, Itami, Japan.

Isao Minematsu received B.S. and M.S. degrees in Geophysics from Kyoto University, Japan, in 1989 and 1991 respectively. In 1991, he joined Mitsubishi Electric Corporation, where he has been engaged in developing embedded software and microprocessors. He is now a staff member of the System LSI Development Center in Itami, Japan. Since 2001, he has doubled as a researcher at Semiconductor Technology Academic Research Center (STARC). He is a member of the IEEE-CS. His research interests include digital signal processor architectures and low-power SoC designs.

Hisakazu Sato received B.S. and M.S. degrees in Electrical and Electronic Engineering from Toyohashi University of Technology, Toyohashi, Japan, in 1986 and 1988, respectively. He joined Mitsubishi Electric Corporation in April 1988. Since then he has been engaged in research and development of microprocessors and digital signal processors. He is currently a senior engineer of Design Group B in the System LSI Design R&D Department, System LSI Development Center.

Kiyoshi Nakakimura was born in Hyogo, Japan, in 1964. He received B.S. and M.S. degrees in Instrumentation Engineering from Keio University, Kanagawa, Japan, in 1988 and 1990, respectively. He joined Mitsubishi Electric Corporation in 1990, where he has been engaged in the development of digital signal processors. He is now a staff member of the system LSI Development Center, Itami, Japan.

Takahiko Arakawa was born in Kagawa, Japan, in 1958. He received a B.E. degree in Electrical Engineering from Hiroshima University, Hiroshima, Japan, in 1981 and a Dr.E. degree in Electronic Engineering from Tokushima Bunri University, Kagawa, Japan, in 2000. In 1981 he joined the LSI Research and Development Laboratory, Mitsubishi Electric Corporation, Itami, Japan, where he worked on the research and development of high-performance semi-custom CMOS LSIs until 1997. Since 1998, he has been engaged in the research and development of system LSIs and low power digital circuits in the System LSI Development Center. His current works are the research and development of digital signal processors and SoC design methodology. He is a member of the Institute of Electronics, Information and Communication Engineers (IEICE) of Japan.

Shuhei Iwade was born in Osaka, Japan, in 1952. He received B.S., M.S., and Ph.D. degrees in Physics from Osaka University in 1976, 1978, and 1982, respectively. In 1978, he joined the LSI Research and Development Laboratory, Mitsubishi Electric Corporation, Hyogo, Japan. From 1978, he was engaged in the design of high speed Gate Arrays, infrared image sensors and MOS analog circuits. Since 1997, he has been engaged in the development of high speed & low power processors and circuits as a Department Manager of the System LSI Design R&D Department in the System LSI Development Center, Hyogo, Japan. He is a member of the Institute of Electronics, Information and Communication Engineers (IEICE) of Japan.