MDSP-II: A 16-Bit DSP with Mobile Communication Accelerator

Byoung-Woon Kim, *Student Member, IEEE*, Jin-Hyuk Yang, Chan-Soo Hwang, *Student Member, IEEE*, Young-Su Kwon, *Student Member, IEEE*, Keun-Moo Lee, *Student Member, IEEE*, In-Hyoung Kim, *Student Member, IEEE*, Yong-Hoon Lee, *Member, IEEE*, and Chong-Min Kyung, *Member, IEEE*

Abstract—**This paper describes a 16-bit programmable fixed-point digital signal processor, called MDSP-II, for mobile communication applications. The instruction set of MDSP-II was determined after a careful analysis of the global system for mobile communications (GSM) baseband functions. An application-specific hardware block called the mobile communication accelerator (MCA) was incorporated on-chip to accelerate the execution of the key operations frequently appearing in Viterbi equalization.**

With the assistance of MCA, the GSM baseband functions, which need 53 million instructions per second (MIPS) on the general-purpose digital signal processors, can be performed only with 19 MIPS. The MDSP-II was implemented with a 0.6-μm triple-layer metal CMOS process on a 9.7 $\times$ 9.8 mm^2 silicon area and was operated up to 50 MHz clock frequency.

Index Terms—**Accelerator, digital signal processors, global system for mobile communications (GSM), mobile communication, Viterbi algorithm.**

I. Introduction

PROGRAMMABLE digital signal processors (DSP's) have been widely used to reduce the development cost and time-to-market of many applications [1]. In the case of mobile communication systems such as the global system for mobile communications (GSM) and IS-136, however, DSP's have only been applied to specific areas such as speech codec because of their limited performance. To overcome this limitation, a number of application-specific DSP's have been developed. For example, Lucent's DSP1618 performs the Viterbi decoding using a coprocessor [2], which supports various decoding modes with control registers at the cost of chip area. On the other hand, TMS320C54x supports specific instructions for the Viterbi decoding [3], which makes it very popular for mobile communication. However, since it has only one multiplier, it is difficult to handle the multiplication and accumulation (MAC) of complex numbers, which is quite often used in such applications as channel equalization. Recently, several DSP's that can support two MAC operations per cycle were developed. For example, very-long-instruction-word-based TMS320C6x, which lacks a dedicated MAC unit but has two multipliers and six arithmetic units, performs MAC operations by using separate multiply and add instructions [4]. Also, the dual MAC unit in Lucent's DSP16000 performs two multiplications and two accumulations in one cycle and supports instructions for the Viterbi decoding [5].

This paper describes MDSP-II, a 16-bit DSP with a special functional block called the mobile communication accelerator (MCA) developed using the MetaCore framework [6], which is a design framework for generating application-specific instruction set processors for DSP applications, along with the generator set for software tools such as compiler, assembler, and instruction-set simulator. The design of MDSP-II was initially started with a basic hardware architecture and a basic instruction set provided by the MetaCore DSP design framework. The hardware architecture and the instruction set of MDSP-II have been customized for mobile communication applications, including the application-specific functional block called MCA, the details of which can be found in [6].

MDSP-II was designed with the objective of performing several crucial mobile communication functions faster and with less processing power requirement due to the adoption of MCA, a special functional block, as well as an instruction set optimized toward the mobile communication applications. Therefore, the remaining processing power can be used for other features such as echo cancellation and speech recognition for voice dialing. Even the half-rate complex vocoder function to enhance the channel utilization can be implemented with the remaining processing power. At least, the remaining processing power always helps to reduce the power consumption to make MDSP-II more competitive for portable applications, simply by performing the same functionality at the reduced clock rate.

In this paper, we first analyze the key algorithms in the mobile communication applications requiring many clock cycle counts, then extract frequent and time-consuming operations to effectively reduce the total clock cycles required. The architectures of MDSP-II and MCA are described, along with the key operations supported. Last, experimental results on the proposed accelerator are shown with the overall performance improvement of MDSP-II in GSM, one of the major mobile communication applications.

II. Key Operations of Mobile Communication

Fig. 1 shows the block diagram of mobile communication systems such as GSM and IS-136 [5], [6]. GSM is the European standard for digital cellular systems. We selected GSM as a major application for analysis and benchmarking because it is one of the popular mobile communication standards in

Manuscript received August 9, 1998; revised October 21, 1998.

The authors are with the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, Taejon 305-701 Korea.

Publisher Item Identifier S 0018-9200(99)01652-2.

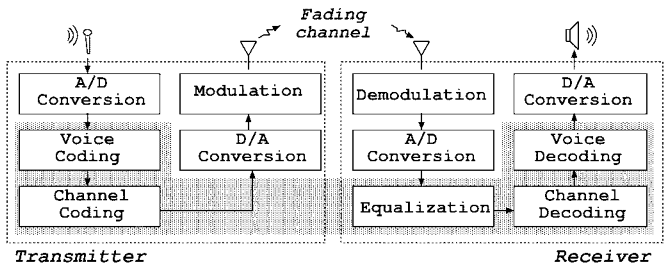


Fig. 1. Block diagram of GSM.

the world, and many of its algorithms are similar to those of other standards.

The baseband processing blocks of GSM are shown as shaded in Fig. 1, where the functional blocks with relatively less significant complexities such as interleaving/deinterleaving and enciphering/deciphering are not included. To use as a reference in estimating the number of clock cycles necessary for each block, we chose MDSP-II without MC, because the performance of MDSP-II without MCA is similar to that of TMS320C5x, a widely used fixed-point DSP. For both voice coding and voice decoding, a regular pulse excitation/long-term predictor (RPE/LTP)-based voice coder was employed. In the channel decoding, the Viterbi algorithm was used to decode a convolutional coding. For the equalization to fight the intersymbol interference due to multipath fading, the Viterbi equalization algorithm based on maximum likelihood sequence estimation was used.

Fig. 2 shows the required processing power of each component of the GSM baseband functions. Most general-purpose DSP's cannot support all the baseband functions, as the total required processing power for GSM itself is 53 million instructions per second (MIPS). Furthermore, future systems will have a bigger demand on the power requirement, aggravating the already tight power budget. Fig. 2 shows that nearly 80% of the total processing power (42 out of the total 53 MIPS) is consumed just for equalization. Therefore, it is important to accelerate the execution of equalization for performance improvement. The detailed behavior of the Viterbi equalization, the required processing power analysis, and three key operations are explained in the following section.

A. Viterbi Equalization for GSM

Fig. 3 shows the block diagram of the Viterbi equalization, where a channel is modeled with N-tap finite impulse response (FIR) filter. The role of channel estimation is to estimate adaptively a set of filter coefficients based on the received signal. Since the input is binary, there are 2^N possible channel input vectors and the corresponding channel outputs, which will be referred to as the reference values. The Viterbi algorithm is based on the trellis diagram having 2^{N-1} states at each time, where a state at time k is connected to two states at time $(k-1)$. For each state transition, there is one possible input vector and the corresponding channel output (reference value). At time k, the difference between the kth received signal value and the reference value is called the *branch metric*, and its accumulation along a path is called the *path metric*. After obtaining the branch metrics, the Viterbi algorithm computes path metrics of the two paths entering each state and determines a path with the smaller path metric, called the *survivor*. Last, the transmitted sequence is estimated by tracing back the survivors. A comprehensive tutorial on the Viterbi equalization is given in [7].

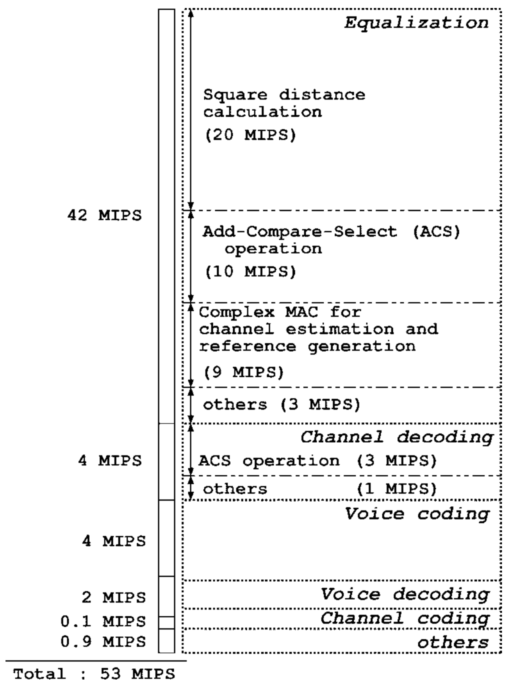


Fig. 2. Required processing power, in MIPS, of each of GSM's baseband functions consisting of equalization, channel decoding, voice coding, etc. Three major operations in the equalization and one major operation in the channel decoding are also shown.

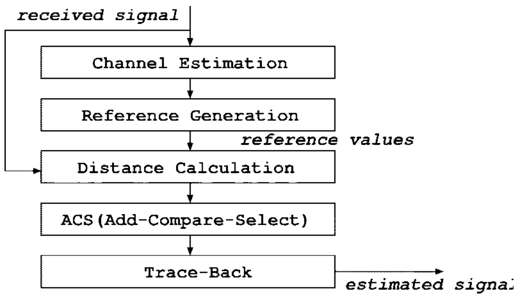


Fig. 3. Block diagram of the Viterbi equalization.

B. Computational Considerations

Again referring to Fig. 3, the received signal is complex, consisting of in-phase and quadrature components. As a consequence, *complex multiplication and accumulation* (complex MAC) is frequently used for channel estimation and reference generation. Each complex MAC needs six clock cycles in the general-purpose DSP's, and about 9 MIPS are necessary for the complex MAC operations in GSM. The square distance calculation for obtaining the branch metric requires heavy computation. One such operation needs ten clock cycles in the general-purpose DSP's, and a total of 20 MIPS are

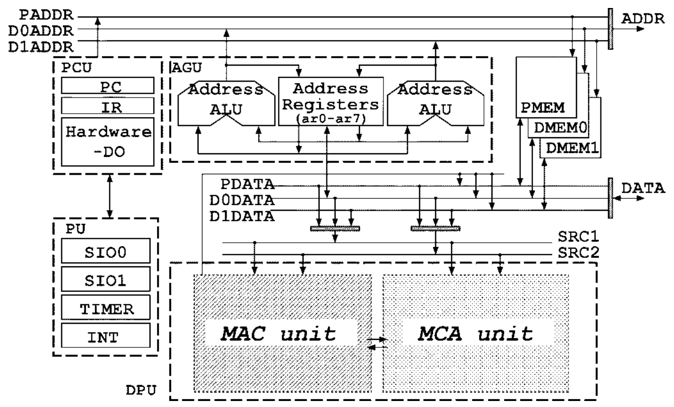


Fig. 4. Architecture of MDSP-II consisting of DPU, AGU, PCU, and two-bank data memory and program memory with the associated address and data bus connections.

occupied for the distance calculation in GSM. Add-compare-select (ACS) is a key operation of the Viterbi algorithm, which is used for both equalization and channel decoding. In GSM, the ACS operations for equalization and decoding need about 10 MIPS and 3 MIPS, respectively.

Based on the above analysis of the required processing power in GSM applications, three key operations (complex MAC, square distance calculation, and ACS) are identified for acceleration using application-specific hardware in the design of MDSP-II. Details of the architectures of MDSP-II and MCA are explained in the following section.

III. MDSP-II

A. *Architecture of MDSP-II*

Fig. 4 shows the architecture of MDSP-II based on 16-bit instruction and 16-bit data. Data memory is divided into two banks (DMEM0 and DMEM1) for efficient operand fetch, and instructions are stored in the program memory (PMEM). The heart of the architecture consists of three execution units operating in parallel, i.e., a fixed-point data processing unit (DPU), a memory address generation unit (AGU), and a program control unit (PCU). A peripheral unit (PU) is also included for communication with external devices.

A brief explanation of each unit follows. DPU, which consists of various basic hardware units, has been designed to optimize the time-critical inner-loop functions of most DSP algorithms. With the help of the MAC unit, the MCA unit performs such operations for the Viterbi equalization as complex MAC, distance calculation, ACS operation, and trace-back operation. The details of these operations will be explained in the following section. AGU performs effective address calculations necessary for fetching operands in memory. It can generate two addresses and modify them in one clock cycle, which occurs simultaneously with the operation of DPU. AGU contains eight address registers (ar0, ar1, $\cdots$, ar7), each of which can be controlled independently to support three addressing modes generally used in DSP algorithms, i.e., linear, modular, and bit-reverse modes. PCU supports zero-overhead loop control as well as branching, subroutine control, and exception handling using a program counter (PC) and an instruction register (IR). PU has two serial ports (SIO0 and SIO1), a timer (TIMER), and an interrupt controller (INT).

Fig. 5 shows three instruction formats of MDSP-II with the corresponding examples. INST denotes the mnemonic of instructions, and ACC denotes the accumulator (a0 or a1), while S1 and S2 denote either memory operand or accumulator. When used as a memory operand, S1 or S2 must specify both the address pointer (e.g., ar0) and its update rule (e.g., ++ for the autoincrement). The optional part ([+/−]) is used to discriminate between MAC+(multiply and add) and MAC−(multiply and subtract) instructions. If ACC is not specified as the destination, the relevant special register is used as the destination.

The first example shows the usage of ADD instruction, which is common to arithmetic and logic instructions. MAC+ instruction performs two operand fetches (from memory, which is pointed by address registers ar0 and ar1) followed by two address register updates ($\mathtt{ar0} = \mathtt{ar0} + 1, \mathtt{ar1} = \mathtt{ar1} + 1$) and two arithmetic operations (multiply and add). Hamming distance calculation (HDIS) shows an example where the destination is implicitly specified.

B. *MAC and MCA Units*

Fig. 6 shows the block diagram of the MAC and MCA units. Two operand buses (SRC1, SRC2) provide data coming from either data memory or the accumulator. The result of one unit (MAC_OUT or MCA_OUT) can be used as the input data for the other unit. The MAC unit, which handles all the data processing of the basic instruction set, consists of a 16 $\times$ 16 multiplier, a 36-bit arithmetic logic unit (ALU), and two 36-bit accumulators (ACC0, ACC1). The MCA unit executes the instructions for complex MAC, square distance calculation, and ACS of the Viterbi algorithm, with the help of the MAC unit. The MCA unit consists of a 16 $\times$ 16 multiplier, a 36-bit adder, a Hamming-distance unit, a queue, a product register (CPR), two operand latches (CS1, CS2), and three 36-bit accumulators (CACC0, CACC1, CACC2).

IV. Accelerated Key Operations

In this section, a detailed description on the behavior of each key operation is given.

A. *Complex MAC*

Since the MAC operation occurs very often in most DSP applications, every DSP has the MAC unit. In the equalization of GSM, the received signal is a complex number that is composed of the in-phase and quadrature terms. However, complex MAC, which is necessary in the Viterbi equalization, cannot be efficiently handled in the traditional MAC unit due to the difficulty in maintaining both the real part and the imaginary part. In MDSP-II, complex MAC is performed by assigning the MAC unit for calculating the real-part value and the MCA unit for the imaginary-part value.

```
[Format]
  INST  ACC, S1             ;; ACC = ACC op S1
  INST  ACC, S1, S2         ;; ACC = ACC [+/-] S1 op S2
  INST  S1, S2              ;; special register = S1 op S2


[Example]
  ADD   a1, *ar0            ;; a1 = a1 + (*ar0)
  MAC+  a0, *ar0++, *ar1++  ;; a0 = a0 + (*ar0++ x *ar1++)
  HDIS  *ar0++, *ar1++      ;; Hamming distance between *ar0++ and *ar1+
                            ;;   is stored in a special register
```

Fig. 5. Three instruction formats of MDSP-II and the corresponding examples.

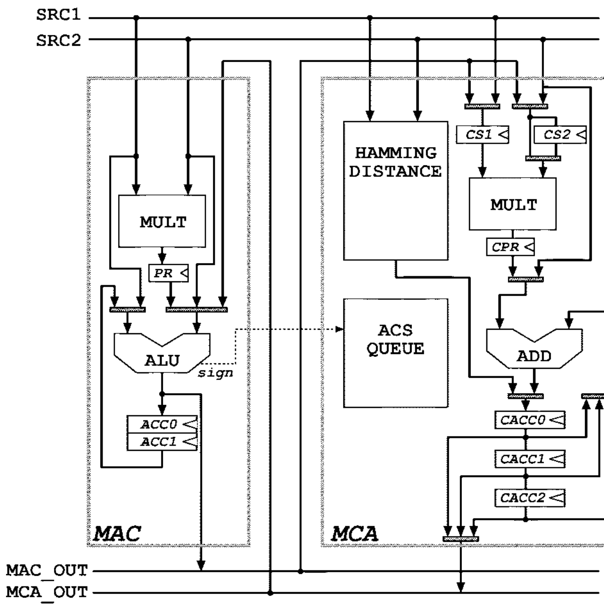


Fig. 6. Block diagram of MAC unit and MCA unit.

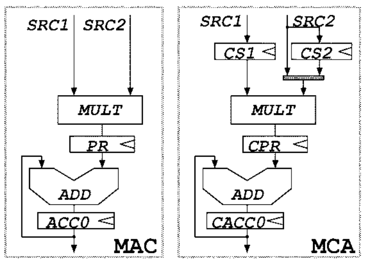
The complex MAC operation for $X_i = a_i + jb_i$ and $Y_i = c_i + jd_i$ is defined as follows:

$$\sum_i X_i \times Y_i = \sum_i (a_i + jb_i) \times (c_i + jd_i)$$
$$= \sum_i (a_i c_i - b_i d_i) + j(a_i d_i + b_i c_i). \quad (1)$$

One complex MAC corresponds to the evaluation of the real and imaginary part of the ith entry and adding them to (or subtracting them from) the corresponding accumulator. Therefore, one complex MAC is composed of eight data loads, four multiplications, and four accumulations. Compared with the general-purpose DSP's with only one MAC unit, where at least four clock cycles are necessary for executing one complex MAC operation, MDSP-II executes one complex MAC operation in two clock cycles by exploiting the MCA unit as well as the MAC unit.

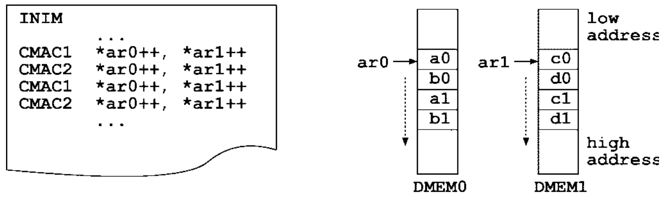
Fig. 7 shows the whole datapath for the complex MAC operation consisting of the MAC unit and the MCA unit. The real part is calculated in the MAC unit, and its result is accumulated in the first accumulator (ACC0), whereas the imaginary part is calculated in the MCA unit with its result accumulated in CACC0.

Even though there are eight data loads in (1), four duplicate data loads can be eliminated if two latches (CS1, CS2) are properly controlled to reuse the previously latched values.



(a) (b)

Fig. 7. Datapath for the complex MAC operation, where (a) MAC performs the real-part calculation ($a_i \times c_i$ followed by $b_i \times d_i$, which is subtracted from the former) while (b) the imaginary-part calculation is done in MCA ($a_i \times d_i$ followed by $b_i \times c_i$ and accumulation).



```
INIM
        ...
CMAC1   *ar0++,  *ar1++
CMAC2   *ar0++,  *ar1++
CMAC1   *ar0++,  *ar1++
CMAC2   *ar0++,  *ar1++
        ...
```

(a) (b)

Fig. 8. (a) Program and (b) data stored in data memory (DMEM0, DMEM1) for the complex MAC operation.

The behavior of complex MAC is further explained with the program and data structure shown in Fig. 8. The program for complex MAC is just a repeating sequence of CMAC1 $*$ ar0++, $*$ar1++ and CMAC2 $*$ ar0++, $*$ar1++. INIM instruction needs to be executed *a priori* to initialize all the MCA-related registers.

The behavior of a complex MAC operation can be easily understood with Table I, which is produced based on the behavior of CMAC1 and CMAC2 instructions as shown in Fig. 9.

Table I indicates that ACC0 contains the real part of the complex MAC $(a_0c_0 - b_0d_0) + (a_1c_1 - b_1d_1) + \cdots$ while CACC0 contains the imaginary part $(a_0d_0 + b_0c_0) + (a_1d_1 + b_1c_1) + \cdots$.

B. Distance Calculation for the Viterbi Algorithm

There are two types of distances that are evaluated in the Viterbi algorithm of GSM. One is the square distance for equalization and the other is the Hamming distance for channel decoding. The hardware configuration for the distance calculation and the behavior of the relevant instructions are explained below.

TABLE I
CLOCK-LEVEL BEHAVIOR OF INSTRUCTIONS FOR THE COMPLEX MAC OPERATION

Time	Instruction	MAC		MCA			
		PR	ACC0	CS1	CS2	CPR	CACC0
T_0	INIM	0	0	0	0	0	0
$T_0 + T_c$	CMAC1	$a_0 \times c_0$	0	a_0	c_0	0	0
$T_0 + 2T_c$	CMAC2	$b_0 \times d_0$	$a_0 \times c_0$	b_0	c_0	$a_0 \times d_0$	0
$T_0 + 3T_c$	CMAC1	$a_1 \times c_1$	$a_0 \times c_0 - b_0 \times d_0$	a_1	c_1	$b_0 \times c_0$	$a_0 \times d_0$
$T_0 + 4T_c$	CMAC2	$b_1 \times d_1$	$a_0 \times c_0 - b_0 \times d_0 + a_1 \times c_1$	b_1	c_1	$a_1 \times d_1$	$a_0 \times d_0 + b_0 \times c_0$

```
CMAC1  S1, S2

MAC: PR    <- S1 x S2
  || ACC0  <- ACC0 - PR

MCA: CS1   <- S1
  || CS2   <- S2
  || CPR   <- CS1 x CS2
  || CACC0 <- CACC0 + CPR
```

```
CMAC2  S1, S2

MAC: PR    <- S1 x S2
  || ACC0  <- ACC0 + PR

MCA: CS1   <- S1
  || CPR   <- CS1 x S2
  || CACC0 <- CACC0 + CPR
```

Fig. 9. Register-transfer-level (RTL) behavior of CMAC1 and CMAC2 instructions for the complex MAC operation($\leftarrow$ denotes "data transfer," while $||$ denotes "parallel execution").

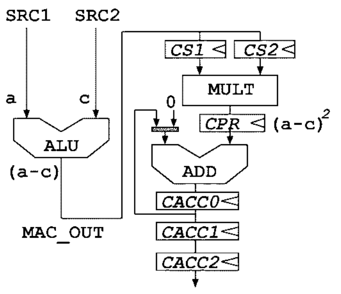


Fig. 10. Datapath for the square distance calculation, where the MAC unit calculates the distance between two numbers, while the square of the distance is accumulated on CACC0 by the MCA unit.

1) Square Distance for Equalization: Given two complex numbers $X = a + jb$ and $Y = c + jd$, the square distance of the numbers is defined as follows:

$$\texttt{Square Distance}\ (X, Y) \equiv (a - c)^2 + (b - d)^2.$$

A general-purpose DSP needs about ten instructions for this calculation. However, MDSP-II can calculate one distance every two cycles using the datapath shown in Fig. 10. To calculate one square distance, two one-cycle instructions, i.e., SDIS1 and SDIS2 need to be executed in sequence, where the behavior of each instruction is described in Fig. 11.

The behavior of the square distance calculation is as follows. After the difference $(\texttt{a} - \texttt{c})$ is obtained by the ALU in the MAC unit and squared by the multiplier in the MCA unit, the squared value $(\texttt{a} - \texttt{c})^2$ is accumulated in the CACC0. Since these operations operate in fully pipelined fashion, another squared value $(\texttt{b} - \texttt{d})^2$ is accumulated on $(\texttt{a} - \texttt{c})^2$ after one cycle. As a result, CACC0 contains the square distance $(\texttt{a} - \texttt{c})^2 + (\texttt{b} - \texttt{d})^2$ after two clock cycles.

```
SDIS1  S1, S2

MAC: MAC_OUT <- S1 - S2

MCA: CS1   <- MAC_OUT
  || CS2   <- MAC_OUT
  || CPR   <- CS1 x CS2
  || CACC0 <- 0 + CPR
```

```
SDIS2  S1, S2

MAC: MAC_OUT <- S1 - S2

MCA: CS1   <- MAC_OUT
  || CS2   <- MAC_OUT
  || CPR   <- CS1 x CS2
  || CACC0 <- CACC0 + CPR
```

Fig. 11. RTL behavior of SDIS1 and SDIS2 instructions for the square distance calculation($\leftarrow$ denotes "data transfer," while $||$ denotes "parallel execution").

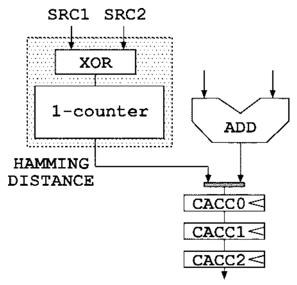


Fig. 12. Datapath for the Hamming distance calculation, where the counter of "1's" is implemented as a six-depth Wallace tree.

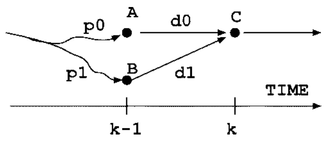


Fig. 13. Path metric selection for an optimal path, where states (A, B, and C), path metrics (p0 and p1), and distances (d0 and d1) are shown.

2) Hamming Distance for Channel Decoding: The Hamming distance of two integer numbers is used as a metric in the convolutional decoder. The Hamming distance can be simply obtained by counting the number of "1's" after the exclusive-OR operation of two 16-bit numbers. To count the number of "1's" is loop intensive if implemented in software. Instead, the counting is implemented in one clock cycle using a Wallace tree [10] of depth six. The Wallace tree is composed of 15 full-adder cells having the worst delay of 1.1 ns.

Fig. 12 shows the datapath for the Hamming distance calculation.

C. ACS of the Viterbi Algorithm

A part of the trellis diagram is shown in Fig. 13, where A, B, and C are states at each time and p0 is a path metric up to A, while p1 is a path metric up to B. d0 is the distance between A and C, while d1 is the distance between B and C. The path metric up to C is, therefore, the smaller of p0+d0 and p1+d1 [9].

Given path metrics and distances, the path metric of C is determined by the ACS operation, which is composed two additions, one comparison, and one selection.

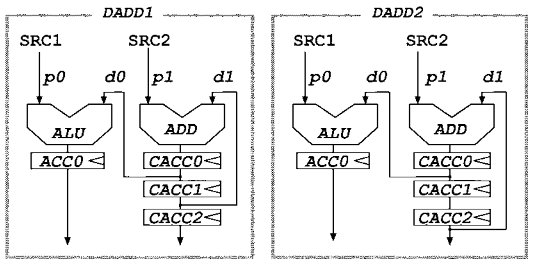
MDSP-II performs two additions simultaneously in one cycle using the MAC unit and the MCA unit, which is called

```
DADD1  S1, S2
 MAC: ACC0  <- S1 + CACC0
 MCA: CACC0 <- S2 + CACC1

DADD2  S1, S2
 MAC: ACC0  <- S1 + CACC0
 MCA: CACC0 <- S2 + CACC2

CAS    dest
 MAC: dest <- MINIMUM( ACC0, CACC0 )
 MCA: ACS QUEUE <- Sign bit of ALU
```

Fig. 14. RTL behavior of DADD1, DADD2, and CAS instructions for the ACS operation($\leftarrow$ denotes "data transfer," while $\|$ denotes "parallel execution").



(a) (b)

Fig. 15. Datapaths for the dual addition mode, where (a) DADD1 instruction is used for Hamming distance and (b) DADD2 instruction is used for square distance.

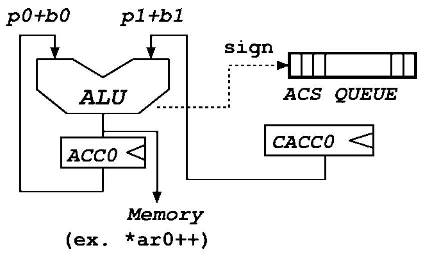


Fig. 16. Datapath for CAS.

TABLE II
FEATURES OF MDSP-II

Features	MDSP-II
Pipeline	5-stage pipeline
Clock frequency	50 MHz (@ 5.0 V)
Performance	100 M MAC's/second
Die size	9.7 mm × 9.8 mm
Core size (excluding the on-chip memory)	5.0 mm × 4.9 mm
On-chip memory	12 KB (program), 8 KB (data)

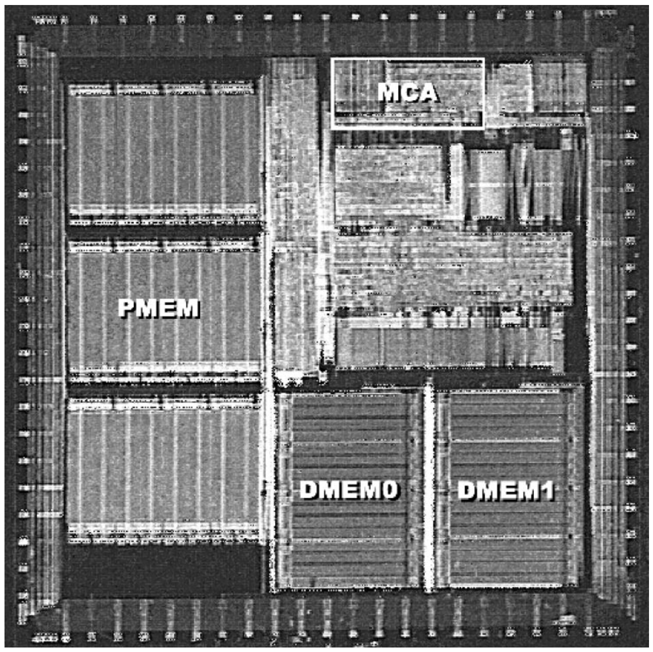


Fig. 17. Photomicrograph of the fabricated chip.

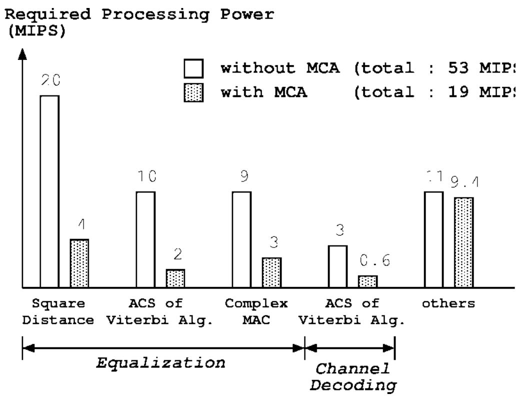


Fig. 18. Performance improvement in GSM.

TABLE III
REQUIRED PROCESSING POWER FOR VITERBI DECODING, WHERE r AND (*) DENOTE CODE RATE AND THE GSM-SPECIFIC GENERATOR POLYNOMIAL, RESPECTIVELY

	TMS320C54x	MDSP-II
r = 1/2 (*)	0.58 MIPS	0.8 MIPS
r = 1/2	1.51 MIPS	0.8 MIPS
r = 1/3	1.74 MIPS	0.8 MIPS
r = 1/4	2.01 MIPS	0.8 MIPS

the *dual addition mode*. There are two instructions supporting the dual addition mode; DADD1 for the Hamming distance and DADD2 for the square distance because the latency of the Hamming distance calculation is one and the latency of the square distance calculation is two.

The behavior of the ACS operation is as follows, where the behavior of related instructions (DADD1, DADD2, and ACS) is shown in Fig. 14. When the square distance has been calculated using SDIS1 and SDIS2 instructions, two distance values, i.e., d0 and d1, are stored in CACC0 and CACC2, respectively. Then, DADD2 instruction is executed such that ACC0 contains p0 + d0 and CACC0 contains p1 + d1 eventually, as shown in Fig. 15. For the Hamming distance calculation, CACC0 and CACC1 contain d0 and d1, respectively, which are used for the dual addition by the following DADD1 instruction.

After the dual addition, compare and select (CAS) instruction performs the comparison in the ALU of the MAC unit, and the sign of the difference between two values p0 + d0 and p1 + d1 is pushed into a queue. The smaller of the two values is stored in memory because it is the path metric of C state. Last, an optimal path is obtained by tracing back the values of the queue. Fig. 16 shows the datapath of the CAS instruction.

V. Experimental Results

MDSP-II was fabricated using a 0.6-μm triple-layer-metal CMOS process as a 9.7 $\times$ 9.8 mm^2 die and operates up to 50 MHz clock based on a five-stage pipeline. Details are shown in Table II, and the photomicrograph of MDSP-II is shown in Fig. 17.

Fig. 18 shows the performance improvement obtained with the MCA unit. Compared to the case of MDSP-II without MCA, where 53 MIPS are needed for the GSM baseband functions, only 19 MIPS are required in MDSP-II with MCA, mainly due to the dramatic performance improvement in the equalization and the channel decoding. The size of the MCA unit only is 0.9 $\times$ 1.4 mm^2, which corresponds to 5.1% of the whole MDSP-II core (5.0 $\times$ 4.9 mm^2) excluding the on-chip program (12 KB) and data (8 KB) SRAM's. The performance improvement is 170%, which is significant compared to the area overhead of 5.1%.

We compared MDSP-II with TMS320C54x, which is one of the most popular DSP chips for mobile communication applications. The result of performance comparison for the Viterbi decoding is shown in Table III, where r denotes the *code rate* and $(*)$ denotes that GSM-specific generator polynomials were used. The constraint length is five for all cases.

MDSP-II performs the Viterbi decoding in 0.8 MIPS for all cases. MDSP-II is superior to TMS320C54x except for the case $r = 1/2(*)$ where the trellis diagram is symmetric so that the special instructions of TMS320C54x, which are applicable only to the symmetric trellis diagram, can be used to simplify the decoding [3]. However, the symmetry of a trellis diagram does not exist in the Viterbi equalization that is the most time-consuming portion in the GSM applications.

VI. Conclusions

In this paper, we presented a 16-bit DSP called MDSP-II designed for GSM based on the mobile communication accelerator, which supports the complex MAC operation and the Viterbi algorithm. In MDSP-II, the inclusion of MCA has produced a performance improvement by 170% for GSM at the additional cost of 5.1% in chip area. Because the input/output interface of MCA is very simple and is controllable at the instruction level, it can be used as a functional block in designing other appropriate application-specific DSP's.

References

[1] E. A. Lee, "Programmable DSP's: A brief overview," *IEEE Micro Mag.*, vol. 10, no. 5, pp. 14–16, Oct. 1990.

[2] "DSP1618 digital signal processor," AT&T Data Sheet, Feb. 1994.

[3] H. Hendrix, "Viterbi decoding techniques in the TMS320C54x family," TI Application Rep., June. 1996.

[4] "TMS320C62x/C67x CPU and instruction set Texas Instruments, reference guide," SPRU198C, Mar. 1998.

[5] M. Levy. (Apr. 1998). EDN's 1998 DSP-architecture directory. [Online]. Available WWW: http://www.ednmag.com/reg/1998/042398/09dsp.cfm.

[6] J.-H. Yang, B. W. Kim, S. J. Nam, J. H. Cho, S. W. Seo, C. H. Ryu, Y. S. Kwon, D. H. Lee, J. Y. Lee, J. S. Kim, H. D. Yoon, J. Y. Kim, K. M. Lee, C. S. Hwang, I. H. Kim, J. S. Kim, K. I. Park, K. H. Park, Y. H. Lee, S. H. Hwang, I. C. Park, and C. M. Kyung, "MetaCore: An application specific DSP development system," in *Proc. Design Automation Conf.*, 1998, pp. 800–803.

[7] L. Hanzo and R. Steele, 'The Pan-European mobile radio system," *Eur. Trans. Telecommun.*, pp. 245–276, 1994.

[8] L. Hanzo and J. Stefanov, "The Pan-European digital cellular mobile radio system-known as GSM," in *Mobile Radio Communications*. New York: IEEE Press/Pentech, 1992, ch. 8, pp. 677–773.

[9] G. Forney, "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268–278, Mar. 1973.

[10] C. S. Wallace, "A suggestion for parallel multiplier," *IEEE Trans. Electronic Computers*, vol. EC-13, pp. 14–17, Feb. 1964.

Byoung-Woon Kim (S'98) was born in Masan, Korea, in 1971. He received the B.S. degree in electronics engineering from Kyungpook National University, Taegu, Korea, in 1994 and the M.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology, Teajon, in 1996, where he currently is pursuing the Ph.D. degree in electrical engineering.

His research interests include hardware–software codesign for application-specific DSP's and low-power DSP processor design.

Jin-Hyuk Yang was born in Taegu, Korea, in 1968. He received the B.S. degree in electronics engineering from Kyungpook National University, Taegu, Korea, in 1992 and the M.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology, Teajon, in 1994, where he currently is working toward the Ph.D. degree in electrical engineering.

His research interests include hardware–software codesign for application-specific DSP's and VLSI architecture for DSP and general-purpose microprocessors.

Chan-Soo Hwang (S'96) was born in Suwon, Korea, on July 4, 1975. He received the B.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology, Teajon, in 1997, where he currently is pursuing the M.S. degree.

His research interests include equalization in wired, wireless, and storage channels, multirate signal processing, and their implementations.

Young-Su Kwon (S'98) was born in Bong-Hwa, Korea, on January 19, 1976. He received the B.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology, Taejon, in 1997, where he currently is pursuing the M.S degree in electrical engineering.

His current research interests include DSP core development and three-dimensional graphics hardware design.

Keun-Moo Lee (S'98) was born in Seoul, Korea, on September 3, 1971. He received the B.S. and M.S degrees in electrical engineering from the Korea Advanced Institute of Science and Technology, Taejon, in 1993 and 1995, respectively, where he currently is pursuing the Ph.D. degree.

His research interests are in wireless communication and CPM.

In-Hyoung Kim (S'97) was born in Seoul, Korea, on December 6, 1973. He received the B.S. and M.S. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology, Taejon, in 1996 and 1998, respectively, where he currently is pursuing the Ph.D. degree.

His research interests include synchronization and equalization in CPM, OFDM systems, and their implementations.

Yong-Hoon Lee (S'81–M'84) was born in Seoul, Korea, on July 12, 1955. He received the B.S. and M.S degrees in electrical engineering from Seoul National University, Seoul, Korea, in 1978 and 1980, respectively, and the Ph.D. degree in systems engineering from the University of Pennsylvania, Philadelphia, in 1984.

From 1984 to 1988, he was an Assistant Professor with the Department of Electrical and Computer Engineering, State University of New York at Buffalo. Since 1989, he has been with the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, Taejon, where he is currently a Professor. His research activities are in the area of one- and two-dimensional digital signal processing, VLSI signal processing, and digital communication systems.

Chong-Min Kyung (S'76–M'81) received the B.S. degree in electronic engineering from Seoul National University, Seoul, Korea, in 1975 and the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Taejon, in 1977 and 1981, respectively.

He was with AT&T Bell Laboratories, Murray Hill, NJ, from April 1981 to January 1983 working in the area of semiconductor device and process simulation. In February 1983, he joined the Department of Electrical Engineering at KAIST, where he is now a Professor. His current research interests include microprocessor design, VLSI CAD, computer graphics, microprocessor architecture, and DSP chip design.