VECTOR QUANTIZATION PROCESSOR FOR MOBILE VIDEO COMMUNICATION

Takuya Iwahashi, Takehide Shibayama, Masanori Hashimoto,
Kazutoshi Kobayashi, Hidetoshi Onodera

Graduate School of Informatics, Kyoto University, Japan

ABSTRACT

A vector quantization processor suitable for video communication has been developed. It performs 10 frames/sec of encoding and decoding QCIF with the FRMSHVQ algorithm. It consumes 49mW at 25MHz. The chip is fabricated in a 0.35μm CMOS technology.

I. INTRODUCTION

Today, rapid growth of communication on cellular phones makes it possible to talk with anybody, anytime, anywhere. On the other hand, videophones become popular by the progress of computer networks and image compression technology. Compared with voice data, picture data involves much more information. Both narrow bandwidth and high power consumption for image compression make it difficult to implement mobile videophones.

MPEG video compression technique is based on discrete cosine transform(DCT). DCT requires many multiply and accumulate(MAC) operations. Multiplier consumes large area and much power.

We have proposed a video compression algorithm called FRMSHVQ based on vector quantization(VQ)[1]. Since VQ needs no multiply operations, power and area must be minimized. The FRMSHVQ algorithm contains two time-consuming operations, VQ and motion estimation(ME). They are similar operations. A vector nearest to an input one is searched from a set of reference vectors. A lot of distances between the input and reference vectors must be computed. Since the distance computation accumulates the distances of all elements, an SIMD parallel processor adapted to VQ and ME can process the FRMSHVQ algorithm efficiently. We designed a vector processor, VP-DSP, suitable for such operation. Employing vector processor architecture and specific operations, VP-DSP performs compression and decompression under a low clock frequency of 25MHz. We have implemented a VP-DSP LSI with a 0.35μm CMOS process. It works properly and consumes 49mW at 25MHz/1.6V. In this paper, features of VP-DSP is described in detail.

II. FIXED-RATE MULTI-STAGE HIERARCHICAL VECTOR QUANTIZATION ALGORITHM

The Fixed-Rate Multi-Stage Hierarchical Vector Quantization (FRMSHVQ) algorithm[1] has proposed for mobile videophones. It consists of ME and VQ and reduces the consuming power at the decoder side since the compressed data is expanded by a simple table look-up method instead of the complicated IDCT of MPEG. This section describes the FRMSHVQ algorithm in detail.

E-mail:iwahashi@vlsi.kuee.kyoto-u.ac.jp

A. Picture Compression by Vector Quantization

On compressing an image by vector quantization, original image is divided into fixed size blocks. These blocks are compared with code vectors in a codebook (a table of typical patterns of images). All blocks of an image are converted to indexes. Decoded image can be restored from indexes by a simple table look-up method. Fig. 1 shows how an image is encoded and decoded using vector quantization.

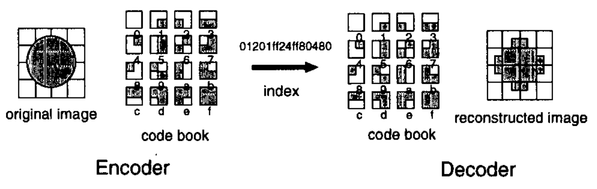

Fig. 1. Video compression by Vector Quantization.

B. Fixed-Rate Multi-Stage Hierarchical VQ Algorithm

Fig. 2 shows the flow of the Fixed-Rate Multistage Hierarchical Vector Quantization (FRMSHVQ) algorithm. Images are hierarchically partitioned into blocks in the four stages. Here a group of pixels is called a block. The size of the block is varied at each stage. At stage 1, a 8×8-pixel block is compressed to the average value of these 64 pixels. Average values are computed in every other block. The pixels in the blocks with no average values are interpolated from the adjacent blocks. at the subsequent frames, ME is applied instead of Stage 1.

The multi-stage hierarchical VQ (MSHVQ) algorithm [2], [3] is applied from Stage 2 to 4. A frame of image is vector-quantized hierarchically in multiple stages. Although lower stages deal with larger blocks, the dimension of vector is fixed to 16. In Stage 2, which handles 16×16 pixels, shaded four pixels are decimated to one as shown in Fig. 2. Pixels between shaded ones are interpolated. The difference between the original image and the decoded image is computed in every stages. The block where the dif-

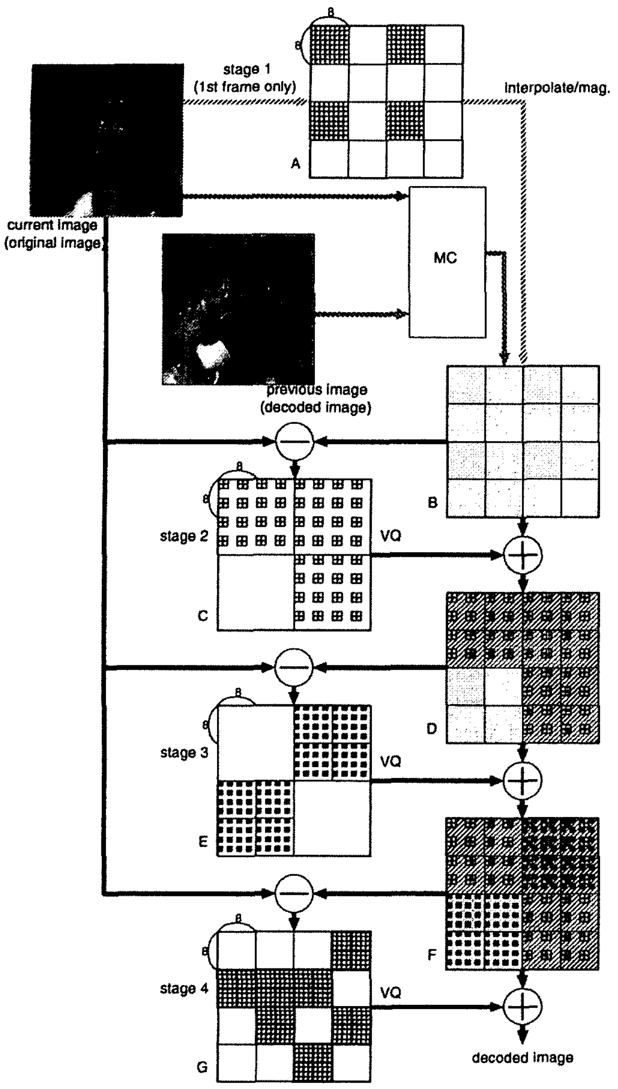

Fig. 2. FRMSHVQ algorithm.

ference is smaller than a threshold value is marked and it is not compressed in the subsequent stages. The FRMSHVQ algorithm compresses a single frame to a fixed size. When the compressed data reaches the limit, it stops compressing the current frame and moves to the next frame. For the video sequence including fast motions, the compression tends to stop lower stages, while it continues to higher stages for those including almost still motions.

The number of code vectors in a codebook must carefully be chosen according to the quality of image, compression efficiency and computation complexity. As the size of a codebook become larger, the quality is improved. But the compression becomes worse and the computation complexity becomes heavier. In the FRMSHVQ algorithm a codebook has 64 code vectors. A code vector is expanded to 16 by rearranging elements. Thus, the nearest vector is chosen from 64×16(=1024) code vectors. The size of a codebook is equivalent to keep the compression ratio, while the number of effective code vectors become large to obtain better quality. The proposed vector-pipeline DSP, VP-DSP simultaneously computes 16 individual distances between an input vector and 16 vectors generated from a code vector in an SIMD manner.

ME requires huge computation resources when it is realized by the exhaustive full-search block matching. In the PC-based system in [1], the orthogonal search method [4] is used to implement ME by the software on a PC. In VP-DSP, the sub-sampling full-search block matching[5] is adopted for ME to estimate motions efficiently in an SIMD manner.

Simulations of the FRMSHVQ algorithm show that the average PSNRs of standard video sequences such as Miss America, Susie and Salesman are over 30dB, which are about 3dB worse than an H.263 encoder[1].

III. Features of VP-DSP

We propose a vector quantization processor suitable for VQ-based video compression algorithm. It performs 10 frames/sec of encoding and decoding of QCIF. This section describes features of the processor called vector-pipeline DSP abbreviated as VP-DSP.

A. Specifications of VP-DSP

VP-DSP is a vector-pipeline processor which can execute vector operations. It has vector registers to operate vector operations. Specifications of VP-DSP is shown in Table I. block diagram of VP-DSP, which has 5 pipeline stages, is shown in Fig. 3.

TABLE I
Specifications of VP-DSP.

architecture	DLX-based RISC processor
vector register width	10 bit × 16 (160bit)
scalar register width	20bit
memory word width	6 bit
registers	11 scalar + 4 vector

VQ and ME perform same operations to a group of pixels according an SIMD manner. In scalar RISC processors, such SIMD operations are mapped to scalar instructions like branch and address increment inside a loop, which may stall pipeline stages. Such scalar instructions can be replaced with a vector instruction on vector processors. VP-DSP has great advantages against scalar processors. In the FRMSHVQ algorithm, dimensions of vectors are 16 on both VQ and ME. Thus, vector registers in VP-DSP have 16 elements.

B. Specific Operations

VP-DSP has specific operations such as absolute distance, accumulation and addition with saturation. These specific operations reduce clock cycles and consequently reduce power consumption. Table II shows vector instructions of VP-DSP.

TABLE II
VECTOR SPECIFIC INSTRUCTIONS.

Mnemonic	Assembly	Semantics
suba(sub /w abs)	suba rdv,rsv1,rsv2	rdv[i] ← |rsv1[i]-rsv2[i]|
	suba rdv,rsv1,rsv2,point	rdv[i]← |rsv1[point]-rsv2[i]|
adds(add /w saturation)	adds rdv,rsv1,rsv2	rdv ← rsv1 + rsv2 or 0 or 63
thin(thin down)	thin rdv,rsv1,rsv2,dir	rdv[dir×4] ← rsv1[0,4,8,12], rdv[others] ← rsv2
accu(accumulate up)	accu rdv,rsv1,rsv2	rdv[i] ← rsv1[i+1]+rsv2[i]
accd(accumulate down)	accd rdv,rsv1,rsv2	rdv[i] ← rsv1[i-1]+rsv2[i]

rsv:	source vector register	rss:	source scalar register
rdv:	destination vector register	rds:	destination scalar register

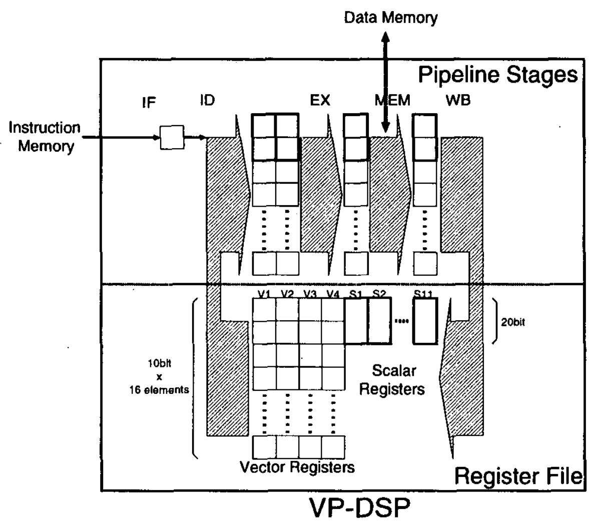

Fig. 3. Block diagram of VP-DSP.

VQ and ME operations require nearest neighbor search (NNS). When operating NNS, a sum of absolute distance (SAD) is calculated. An SAD consists of accumulation of several absolute distances. Clock cycles needed for these operations must be reduced. Absolute distance operation (e.g. $Y = |X_1 - X_2|$) usually needs two or three steps – subtraction and inversion.

In the FRMSHVQ algorithm, a code vector $\vec{y} = (y_0, y_1, .., y_{15})$ is expanded to rearranged 16 vectors, $\vec{Y_0} = (y_0, y_1, .., y_{15}), \vec{Y_1} = (y_1, y_2, .., y_0), .., \vec{Y_{15}} = (y_{15}, y_0, .., y_{14})$. These 16 SADs between an input vector $\vec{x} = (x_0, x_1, .., x_{15})$ and a code vector $\vec{y} = (y_0, y_1, .., y_{15})$ are computed as follows. An input vector $\vec{x}$ and a code vector $\vec{y}$ are loaded to two vector registers, V1 and V2. The combination of suba V3,V1,V2 and accd V4,V4,V3 instructions accumulates the absolute distances of 16 elements to the vector register V4. After the accumulation of the first element, V4 contains $(|x_0 - y_0|, |x_0 - y_1|, ..., |x_0 - y_{15}|)$. Iterating these instructions 16 times, V4 contains 16 independent absolute distances between an input vector and 16 vectors generated from a code vector, as described in Eq. (1). No load operation is required during computation of absolute distance. Fig. 4 illustrates suba and accd instructions for the second attempt among 16 iterations.

$$\begin{aligned} V4 &= (\sum_{i=0}^{15} |x_i - y_{(i+1)\%16}|, \sum_{i=0}^{15} |x_i - y_{(i+2)\%16}|, \\ &\quad .., \sum_{i=0}^{15} |x_i - y_{(i+0)\%16}|) \\ &= (|\vec{x} - \vec{Y_1}|, |\vec{x} - \vec{Y_2}|, .., |\vec{x} - \vec{Y_0}|) \qquad (1) \end{aligned}$$

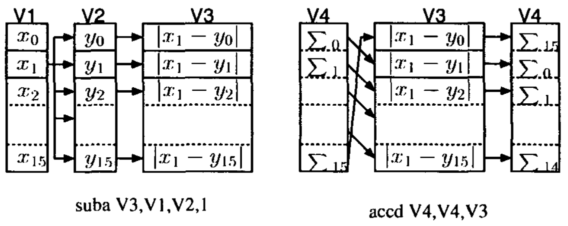

Fig. 4. suba and accd instructions.

IV. PERFORMANCE EVALUATION

VP-DSP is designed suitable for VQ and ME. For the purpose of comparison, we designed a scalar processor which has similar instruction sets (without vector related instructions). In this section, performance of VP-DSP is examined.

First, we estimate number of clocks for a single VQ operation, where an input vector is compared with 64 code vectors. Note that the FRMSHVQ algorithm expand 64 code vectors to 1024. To obtain the nearest vector among 1024 code vectors, 16384 (=1024×16) SADs are computed. The scalar processor computes and accumulates these absolute distances element by element and takes 138,243 clock cycles. On the other hand, VP-DSP computes and accumulates them in parallel and takes only 4,872 clock cycles.

An ME operation for a reference block takes 1294 clock cycles on VP-DSP. This is about 1/5 of that of the scalar processor. In ME, load instructions occupies 790 out of 1294. A load instruction on VP-DSP loads four elements simultaneously to a vector register, which is only four times faster than the scalar processor. This is why VP-DSP is only 5 times faster than the scalar processor. In VQ, VP-DSP computes 16 absolute distances without any load operation, while the scalar processor has to load code vectors from data memory for each element. This is why VP-DSP is 28 times faster, which is larger than 16, the number of parallel operations.

From the simulation of the FRMHVQ algorithm, we estimate that the number of VQ per frame is about 200. The number of ME for a QCIF frame is 396 (=176 × 144/8/8). Other operations needed for encoding and decoding take about 500k clock cycles. Thus, total clock cycles needed for a frame is about 2.5M ($\simeq 4872\times200+1294\times396+500$k). 10 frames/sec QCIF is processed at 25MHz. Fig. 5 shows the total clock cycles needed for the algorithm. VP-DSP is 15 times faster than the scalar processor.

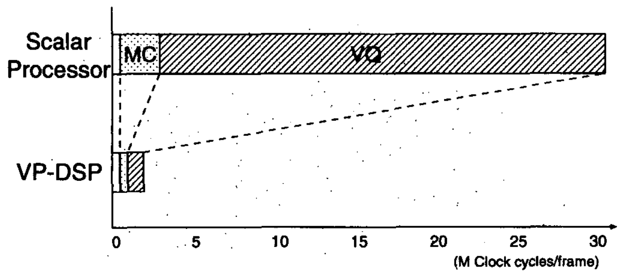

Fig. 5. Total clock cycles.

The area of VP-DSP synthesized for the 0.35 μm process is 2.26mm^2. This is only 4 times larger than that of the scalar processor, which is 0.57mm^2.

V. IMPLEMENTATIONS

In this section, we discuss a VP-DSP LSI. We first explain the low-power cell library used for VP-DSP design, and the specifications of VP-DSP. Then we show the measurement results of the fabricated VP-DSP LSI chip.

A. Specifications

VP-DSP is designed and fabricated in a 0.35μm technology with three metal layers. In order to reduce power dissipation, we developed a low-power cell library(Lib. A) specifically for VP-DSP design by a cell layout generation system VARDS[6]. The features of Lib. A are:

- Cell height is 9 interconnect pitches, whereas that of a generic library(Lib. B)[1] is 11 pitches.
- There are weak and intermediate driving-strength cells, such as x0.5, x0.75 and x1.5.

As explained in Sec. IV, required clock frequency is only 25MHz. The cells included in the generic library are too fast, i.e. too large for VP-DSP. We then generate smaller-height cells that are low-power but enough fast for 25MHz. The basic gate width of Lib. A is 3.4μm, which is 24% smaller than the width of the generic library. We also generate weak driving-strength cells(x0.5, x0.75) and intermediate cells(x1.5, x3) to reduce the power dissipation at the gates where slack is positive. In order to evaluate the effectiveness of Lib. A, we designed another DSP core using Lib. B. Fig. 6 shows the chip micrograph. Two DSP cores are implemented on the same die of 4.9mm^2 using the above two cell libraries. The 512-word 24-bit SRAM is generated by Alliance[7]. The detailed specifications of this chip are shown in Table III. The area of core #1 is 9% smaller than the area of core #2.

[1]provided by VLSI Design and Education Center for this technology.

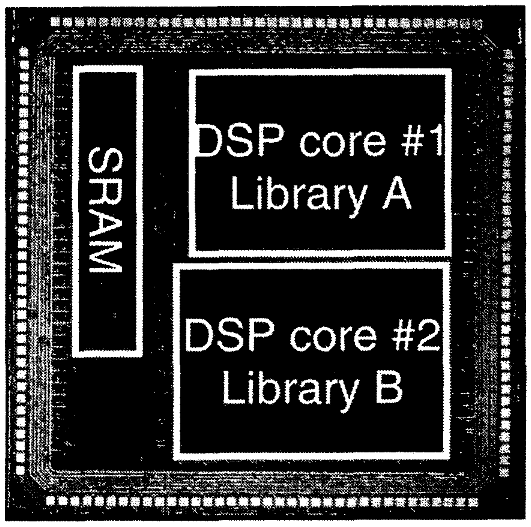

Fig. 6. Chip micrograph.

TABLE III
SPECIFICATIONS OF VP-DSP CHIP.

Die Area		24.0mm^2
Process		0.35μm 3M 1P CMOS
Lib. A	Area	4.26mm^2(2.46×1.73)
(#1)	#gates	31393
Lib. B	Area	4.68mm^2(2.6×1.8)
(#2)	#gates	28956
SRAM	Area	1.41mm^2
(512w×24b)	Access Time	1.68ns

B. Measurement Results

We have measured the VP-DSP LSI by an LSI tester. Fig. 7 shows a shmoo plot of the VP-DSP core #1 to sweep supply voltage and clock cycle. Note that the constraint at design time is 50MHz and 3.3V. Fig. 7 contains several measured power consumption values. To complete the FRMSHVQ algorithm in real time, the power consumption is 49mW at 25MHz/1.6V.

Table IV shows power dissipation of core #1 and core #2. The power consumption is reduced by about 10% using Lib. A, when supply voltage is 3.3V. The power

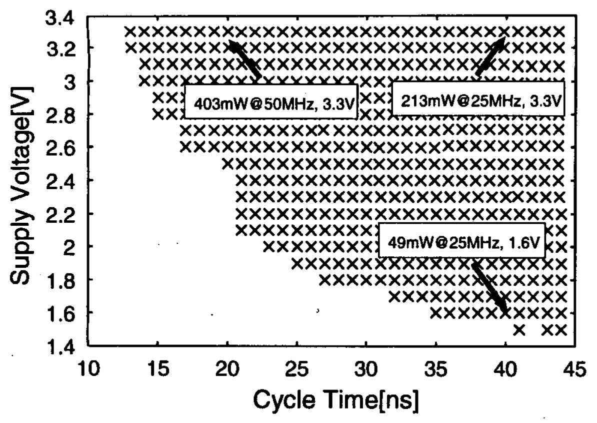

Fig. 7. Shmoo plot and power dissipation of VP-DSP core #1.

reduction is smaller than we expected, because D-FFs in Lib. A are not well designed for low-power. When the revised low-power D-FFs are used, we can expect that the power dissipation is reduced by about 12% from the simulation results.

TABLE IV
COMPARISON OF POWER DISSIPATION.

Frequency & Voltage	Lib. A(#1)	Lib. B(#2)
50MHz, 3.3V	403mW	439mW
25MHz, 3.3V	213mW	234mW
25MHz, 1.6V	49mW	50mW

VI. CONCLUSIONS

We propose a vector pipline processor (VP-DSP) suitable for mobile videophones. The chip was fabricated in a 0.35μm CMOS technology. By employing vector processor architecture, VP-DSP performs 10 frames/sec of encoding and decoding with QCIF at 25 MHz. Power consumption is 49mW at 1.6V supply voltage.

ACKNOWLEDGMENT

The VLSI chip in this study has been fabricated in the chip fabrication program of VLSI Design and Education Center(VDEC), the University of Tokyo with the collaboration by Rohm Corporation and Toppan Printing Corporation.

REFERENCES

[1] K. Kobayashi, K. Terada, H. Onodera and K. Tamaru, "A Real-Time Low-Rate Video Compression Algorithm Using Multi-Stage Hierachical Vec tor Quantization.", *IEICE Trans. on Fundementals*, vol. E82-A(2), pp. 215–222, 1999.
[2] Y. Ho and A. Gersho, "Variable-Rate Multi-Stage Vector Quantization For Image Coding", *ICASSP*, pp. 1156–1159, 1988.
[3] A. Gersho, "Hierarchical Vector Quantization for Speech Coding", *ICASSP*, pp. 10.9.1–10.9.4, 1984.
[4] A. Puri, H.M. Hang and D.L. Schilling, "An efficient block matching algorithm for motion-compensated coding", *ICASSP*, pp. 1063–1066, 1987.
[5] J.-S. Choi H.-K. Jung, C.-P. Hong and Y.-H. Ha, "A VLSI architecture for the alternative subsampling-based block matching algorithm", *IEEE Trans. Consum. Electron. (USA)*, vol. 41-2, pp. 239–47, 1995.
[6] T. Hashimoto and H. Onodera, "Layout Generation of Primitive Cells with Variable Driving Strength", *Proc. of SASIMI2000*, pp. 122–129, 2000.
[7] A. Greiner and F. Pecheux, "ALLIANCE: A Complete Set of CAD Tools for Teaching VLSI Design", *EUROCHIP. Proceedings of the Third Eurochip Workshop on VLSI*, pp. 230–7, 1992.
[8] J.L. Henessy and D.A. Patterson, "Computer Architecture, A Quantitative Approach", *Morgan Kaufmann*, 1996.