IEICE TRANS. FUNDAMENTALS, VOL.E83–A, NO.2 FEBRUARY 2000

PAPER *Special Section on Intelligent Signal and Image Processing*

A Bit-Operation Algorithm of the Median-Cut Quantization and Its Hardware Architecture

Shogo MURAMATSU[†], **Hitoshi KIYA**[††], *and* **Akihiko YAMADA**[††], *Members*

SUMMARY In this paper, an algorithm of the median-cut quantization (MCQ) is proposed. MCQ is the technique that reduces multi-valued samples to binary-valued ones by adaptively taking the median value as the threshold. In this work, the search process of the median value is derived from the quick-sort algorithm. The proposed algorithm searches the median value bit by bit, and samples are quantized during the search process. Firstly, the bit-serial procedure is shown, and then it is modified to the bit-parallel procedure. The extension to the multi-level quantization is also discussed. Since the proposed algorithm is based on bit operations, it is suitable for hardware implementation. Thus, its hardware architecture is also proposed. To verify the significance, for the application to the motion estimation, the performance is estimated from the synthesis result of the VHDL model.

key words: *adaptive quantization, median value, motion estimation, VLSI*

1. Introduction

The median-cut quantization (MCQ) is the technique that reduces multi-valued samples to binary-valued ones by adaptively altering the threshold as the median value. MCQ has been found several applications in signal and image processing so far, such as color reduction and preprocessing of low-power implementation [1]–[3].

The basic procedure of MCQ is as follows: firstly sort given samples, secondly extract the median value, and lastly quantize all samples by using the median value as the threshold. The most significant step is the sorting process. The sorting process has been a principal problem in the area of computer engineering so far. Thus, there are a lot of sophisticated techniques for that [4].

Extracting the median value, however, does not require the complete sorting. Indeed, in order to obtain the median value, the pruning technique of the quick-sort algorithm (QSA) is available. This algorithm can extract an arbitrary position value as well as the median value.

During the process of QSA, some values are used to recursively divide a given set to two subsets. Such a value is referred to as a pivot. As well as the normal QSA, the question on the pivot selection, however, still exists in the pruned QSA. In the article [5], a solution to select the pivot is shown, and the number of operations is improved.

In this paper, we consider developing an algorithm of MCQ which is suitable for hardware implementation. This is motivated from the fact that most of the applications are image and video processing, and they require large amount of computation within a specified time [6], [7]. In order to achieve this purpose, we adopt the pruned QSA. The problem of this approach is how to select the pivot. The method shown in the article [5], however, does not take the quantization into account.

From this background, we propose a new pivot selection method for MCQ with the pruned QSA. The proposed method concurrently quantize given samples while searching the median value bit by bit. Indeed, a value consisting of some most significant bits of the median value is used as the pivot during the search process. This bit-level operation leads to an efficient hardware implementation. Since the proposed algorithm can take an arbitrary position value as the threshold, the multiple execution with different thresholds yields a multi-level quantization.

The organization of this paper is as follows. As a preliminary, in Sect. 2, we briefly review MCQ and the pruned QSA. In Sect. 3, we discuss the basic idea of the proposed algorithm, and then show the bit-serial procedure. We also modify it to the bit-parallel procedure. The extension to the multi-level quantization is also discussed. The hardware architecture is proposed in Sect. 4. The architecture is simple in the data and control flow. To show the significance, in Sect. 5, the application to the low-bit block matching motion estimation is discussed [2], [3], [7]–[9], and estimate the architecture by some synthesis results from the VHDL model, followed by the conclusions in Sect. 6.

2. Review

As a preliminary, this section briefly reviews MCQ and the pruned QSA.

2.1 Median-Cut Quantization (MCQ)

Let $x(n)$ and $y(n)$ be an input and the output for

Manuscript received May 18, 1999.
Manuscript revised August 12, 1999.

[†]The author is with the Department of Electrical and Electronic Engineering, Faculty of Engineering, Niigata University, Niigata-shi, 950–2181 Japan.
[††]The authors are with the Department of Electrical Engineering, Graduate School of Engineering, Tokyo Metropolitan University, Hachioji-shi, 192–0397 Japan.

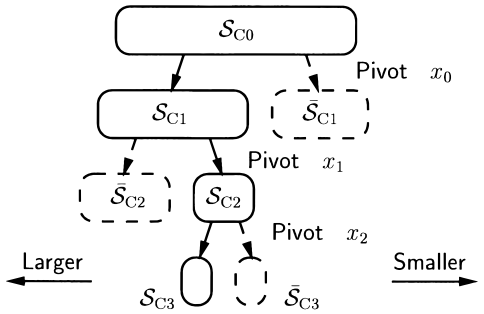


Fig. 1 Pruning technique of QSA.

MCQ, respectively, and let N be the number of samples. Then, the function of MCQ is represented as follows:

$$y(n) = \begin{cases} 1, & x(n) \geq T_M \\ 0, & \text{otherwise} \end{cases}, \ n = 0, 1, \cdots, N-1 \quad (1)$$

where T_M is the median value. Let $u(n)$ be the ordered sequence of $x(n)$ with the increasing order. Then, T_M is obtained as follows:

$$T_M = u(M), \ M = \lfloor N/2 \rfloor, \quad (2)$$

where the function $\lfloor x \rfloor$ means the integer part of its argument x. The threshold adaptively alters according to the distribution of $x(n)$.

The sorting process of $x(n)$ is not necessary to obtain T_M. It is sufficient to use the pruned QSA, which will be shown in the followings.

2.2 Pruning Technique of QSA

Figure 1 briefly illustrates the pruning technique of QSA [4], [5]. We firstly explain the case that the target is the median value T_M. Then, we generalize the discussion to any position value.

In Fig. 1, $\mathcal{S}_{C0}$ denotes the set of given samples $x(n)$, $\mathcal{S}_{Ci}$ is the subset of $\mathcal{S}_{C(i-1)}$ which includes the median value T_M, that is, the candidates, and $\bar{\mathcal{S}}_{Ci}$ is its complement in $\mathcal{S}_{C(i-1)}$, that is, $\mathcal{S}_{C(i-1)} = \mathcal{S}_{Ci} \oplus \bar{\mathcal{S}}_{Ci}$. x_{i-1} denotes the pivot of the i-th iteration.

For extracting T_M, the procedure is represented as follows, where M is the median position $M = \lfloor N/2 \rfloor$:

Step 0: Initialize i, c as $i = 0$, $c = 0$.
Step 1: Increment i as $i = i + 1$.
Step 2: Select an element $x_i \in \mathcal{S}_{C(i-1)}$ as the pivot.
Step 3: Divide $\mathcal{S}_{C(i-1)}$ into the following two subsets:

$$\mathcal{S}_{Li} = \{x(n) \in \mathcal{S}_{C(i-1)} : x(n) \geq x_{i-1}\},$$
$$\mathcal{S}_{Si} = \{x(n) \in \mathcal{S}_{C(i-1)} : x(n) < x_{i-1}\}.$$

Step 4: Let $\mathcal{S}_{Ci} = \mathcal{S}_{Li}$ when $c + |S_{Li}| > N - M - 1$, that is, $T_M \in \mathcal{S}_{Li}$. Otherwise, let $\mathcal{S}_{Ci} = \mathcal{S}_{Si}$ and update c as $c = c + |\mathcal{S}_{Li}|$.
Step 5: Break if $\mathcal{S}_{Ci} = \emptyset$. x_{i-1} is the target.
Step 6: Go to Step 1 until $|\mathcal{S}_{Ci}| = 1$ or all elements in $\mathcal{S}_{Ci}$ equal to each other.
Step 7: End. The/an element in $\mathcal{S}_{Ci}$ is the target.

Although the above procedure is described for extracting the median value T_M, the target does not limited to it. Indeed, any position in the range from 0 to $N-1$ can take the place of the median position M. The efficiency of this procedure highly depends on how to select the pivot $x_{i-1} \in S_{C(i-1)}$ in Step 2.

3. The Algorithm

In this section, we consider applying the pruned QSA to MCQ. The key point is the way of selecting the pivot. Indeed, the pivot is chosen as a value consisting of some most significant bits of the median value during the process. Firstly, we propose the bit-serial procedure, and then, modify it to the bit-parallel procedure to improve the throughput.

3.1 Bit-Serial Procedure

In this paper, we assume that $x(n)$ is an unsigned integer, and let B and N be the number of bits and the number of samples, respectively. Let us remind the procedure described in Sect. 2.2. The basic idea of our proposed algorithm is based on the following facts:

Fact 1: If the number of 1s in MSBs, or $(B-1)$-th bits, of all samples is larger than $N - M - 1$, the MSB of T_M must be '1,' otherwise '0.' Thus, by dividing $\mathcal{S}_{C0}$ into $\mathcal{S}_{L1}$ and $\mathcal{S}_{S1}$ according to each MSB, which implies the pivot $x_0 = T_M \& 2^{B-1}$, it is easily identified which the candidate is, where '&' denotes the logical product.
Fact 2: If $\mathcal{S}_{Si}$ is not the candidate, then the elements all are necessarily smaller than T_M, and should be quantized to '0.' Otherwise, the elements in $\mathcal{S}_{Li}$ should be quantized to '1.'
Fact 3: Assume that $\mathcal{S}_{C(i-1)}$ has been divided into $\mathcal{S}_{Ci}$ and $\bar{\mathcal{S}}_{Ci}$ with the pivot $x_{i-1} = T_M \& (\sum_{b=1}^{i} 2^{B-b})$. Then, dividing $\mathcal{S}_{Ci}$ into $\mathcal{S}_{L(i+1)}$ and $\mathcal{S}_{S(i+1)}$ according to only the $(B-i-1)$-th bit corresponds to the selection of the pivot as $x_i = T_M \& (\sum_{b=1}^{i+1} 2^{B-b})$.
Fact 4: The check if $\mathcal{S}_{L(i+1)}$ is the candidate or not is achieved by firstly counting 1s in the already-quantized samples and on the $(B-i-1)$-th bits of elements in $\mathcal{S}_{Ci}$, and then evaluating if it is larger than $N - M - 1$ or not. If not, $\mathcal{S}_{S(i+1)}$ is the candidate.

For the sake of simplification, we explained the above facts by using the masked median value $x_i = T_M \& (\sum_{b=1}^{i+1} 2^{B-b})$, which consists of the $(i+1)$ most significant bits of T_M, as the pivot. The pivot, however, may be out of $\mathcal{S}_{Ci}$. Precisely, the division according to a bit corresponds to selecting the smallest element $x(n)$

satisfying Eq. (3) as the pivot x_i.

$$x(n) \ \geq \ T_M \ \& \ \left(\sum_{b=1}^{i+1} 2^{B-b} \right). \tag{3}$$

Figure 2 shows the C-like pseudo code of the proposed algorithm. We refer to it as the bit-serial procedure. In this paper, $[x]_b$ denotes the b-th bit of x. $r(n)$ and $f(n)$ are one bit variable, where the former contains both of the quantized samples and the b-th bit of candidates, and the latter is the flag representing if it has already been quantized. Although no quantization operation appears in the procedure, flagging in the line 11 implies it. Table 1 shows an example of the process, where $B = 3$ and $N = 8$.

With the proposed algorithm, $NB + N$ cycles are required for completion. Let us consider applying the procedure to the block processing of size N, and define the throughput by the number of blocks processed per cycle. Then, the throughput is evaluated as $1/NB$, because the output process in the line 17 can be overlapped right before the line 5 in the process for the next block, and only NB cycles are required to complete one block.

```
0:  Bit_Serial_MCQ (y, T_M, x, N) {
1:      for ( b = B−1; b >= 0; b-- ) { /* Sweep bits */
2:          c = 0; /* Clear counter C */
3:          for ( n = 0; n < N; n++ ) { /* Sweep samples */
4:              if ( b == B−1 ) {
5:                  r(n) = [x(n)]_{B−1}; /* Set the (B−1)-th bit of x(n) to R */
6:                  f(n) = 0; /* Clear flag F */
7:              }
8:              else if ( (r(n) == [T_M]_{b+1}) && (f(n) == 0) ) /* Candidate */
9:                  r(n) = [x(n)]_b; /* Set the (B−1)-th bit of x(n) to R */
10:             else /* Not candidate */
11:                 f(n) = 1; /* Set flag F */
12:             c += r(n); /* Count up 1s in R */
13:         }
14:         [T_M]_b = c > N−M−1 ? 1 : 0; /* Update the median value T_M */
15:     }
16:     for ( n = 0; n < N; n++ )
17:         y(n) = r(n) | ~(f(n) | [T_M]_0); /* Output */
18: }
```

Fig. 2 Proposed bit-serial procedure of the median-cut quantization.

Table 1 Example of the signal flow in the proposed method, where $B = 3$, $N = 8$ and $M = 4$. $r^{(b)}(n)$ denotes $r(n)$ in the iteration for the b-th bit, 'f' is the flag which implies $f(n) = 1$ and $(\cdot)_2$ means the binary representation. In this example, the median value T_M results in $5 = (101)_2$.

n	$x(n)$		$r^{(2)}(n)$		$r^{(1)}(n)$		$r^{(0)}(n)$		$y(n)$
0	1	$(001)_2$	-	0	f	0	f	0	0
1	7	$(111)_2$	-	1	-	1	f	1	1
2	6	$(110)_2$	-	1	-	1	f	1	1
3	5	$(101)_2$	-	1	-	0	-	1	1
4	2	$(010)_2$	-	0	f	0	f	0	0
5	6	$(110)_2$	-	1	-	1	f	1	1
6	4	$(100)_2$	-	1	-	0	-	0	0
7	0	$(000)_2$	-	0	f	0	f	0	0
c			5		3		4		-
T_M			$(1--)_2$		$(10-)_2$		$(101)_2$		$(101)_2$

3.2 Bit-Parallel Procedure

Some applications require for the throughput to be larger than $1/NB$. In the following, we modify the bit-serial procedure so as to improve the throughput. We refer to the technique as the bit-parallel procedure because it operate multiple bits a cycle as a digit in parallel.

Note that, in Fig. 2, the check in the line 8 depends only on the result of the previous iteration. Thus, the checking processes in the same iteration can be executed concurrently. Based on this idea, we propose the bit-parallel procedure. Let P be the digit size and a divisor of N, and let $K = N/P$. We show the algorithm in Fig. 3, where it is assumed that the bit array of the input sequence $x(n)$ is prearranged to a P-bit digit $\hat{x}(m)$ such that

$$\begin{aligned}[\hat{x}(m)]_{P-p-1} &= [x(P \cdot (\!(m)\!)_K + p)]_{B-\lfloor \frac{m}{K} \rfloor - 1}, \\ p &= 0, 1, \cdots, P-1, \ m = 0, 1, \cdots, KB-1,\end{aligned} \tag{4}$$

where $(\!(x)\!)_N$ denotes the integer of x modulo N.

For example, when $P = 4$, the sequence shown in Table 1 is prearranged as follows:

$$\begin{aligned}\hat{x}(0) &= ([x(0)]_7 \ [x(1)]_7 \ [x(2)]_7 \ [x(3)]_7)_2 = (0111)_2 \\ \hat{x}(1) &= ([x(4)]_7 \ [x(5)]_7 \ [x(6)]_7 \ [x(7)]_7)_2 = (0110)_2 \\ \hat{x}(2) &= ([x(0)]_6 \ [x(1)]_6 \ [x(2)]_6 \ [x(3)]_6)_2 = (0110)_2 \\ &\vdots \\ \hat{x}(15) &= ([x(4)]_0 \ [x(5)]_0 \ [x(6)]_0 \ [x(7)]_0)_2 = (0000)_2\end{aligned}$$

The throughput of this prearrangement depends on the peripheral circuit and how to implement it. Hence, we here temporarily give an example. Provided one data per cycle via one bus of width B bits, it can be implemented by a buffer of size NB bits. The throughput meets P/NB blocks/cycle, where $P \leq B$.

In Fig. 3, the output $\hat{y}(k)$, the flag $\hat{f}(k)$ and the intermediate result $\hat{r}(k)$ are P-bit digits. As a result, the bit-parallel procedure increases the throughput from $1/NB$ blocks/cycle to $1/KB = P/NB$ blocks/cycle.

Note that the bit-serial procedure can be regarded as a special case of the bit-parallel procedure for $P = 1$.

3.3 Multi-Level Quantization

The pruning technique of QSA described in Sect. 2.2 can be used to obtain any position element of the input $x(n)$. This implies that a Q-level quantization is simply achieved by multiply executing the $Q-1$ processes with $Q-1$ different thresholds T_{M_q} given as in Eq. (5).

$$\begin{aligned}T_{M_q} &= u(M_q), \ M_q = \lfloor Nq/Q \rfloor, \\ & \qquad q = 1, 2, \cdots, Q-1,\end{aligned} \tag{5}$$

where $u(n)$ is the ordered sequence of $x(n)$. As a result,

```
0:  Bit_Parallel_MCQ (ŷ, T_M x̂, N, P) {
1:     K = N/P;
2:     for ( b = B−1; b >= 0; b-- ) { /* Sweep bits */
3:        c = 0; /* Clear counter C */
4:        for ( k = 0; k < K; k++ ) { /* Sweep digits */
5:           if ( b == B−1 ) {
6:              r̂(k) = x̂(Kb+k); /* Set x̂(m) to R */
7:              f̂(k) = ext(0); /* Clear flag F */
8:           }
9:           else { /* Set 1s to candidate bits in d̂ */
10:             d̂ = ~( ( r̂(k) ^ ext([T_M]_{b+1}) ) | ( f̂(k) ^ ext(0) ) );
11:             r̂(k) = ( x̂(Kb+k) & d̂ ) | ( r̂(k) & ~d̂ );
12:             f̂(k) = f̂(k) | ~d̂ ; /* Set flag F */
13:          }
14:          c += ones(r̂(k)); /* Count up 1s in R */
15:       }
16:       [T_M]_b = c > N−M−1 ? 1 : 0; /* Update the median value T_M */
17:    }
18:    for ( k = 0; k < K; k++ )
19:       ŷ(k) = r̂(k) | ~( f̂(k) | ext([T_M]_0) ); /* Output */
20: }
```

Fig. 3 Proposed bit-parallel procedure of the median-cut quantization, where "`ones`" is the function that returns the number of 1s on its argument, and "`ext`" is the function that extends its 1-bit input to P-bit digit.

```
0:  Bit_Parallel_MLQ ({ŷ_ℓ}, {T_{M_q}}, x̂, N, P, L) {
1:     K = N/P;
2:     Q = 2^L;
3:     for ( b = B−1; b >= 0; b-- ) { /* Sweep bits */
4:        do in parallel for ( q = 1; q < Q; q++ )
5:           c_q = 0; /* Clear counter C */
6:        for ( k = 0; k < K; k++ ) { /* Sweep digits */
7:           do in parallel for ( p = P−1; p >= 0; p-- )
8:              ([ŝ_{Q−1}]_p ··· [ŝ_1]_p)_2 = decode( ([r̂_{L−1}(k)]_p ··· [r̂_0(k)]_p)_2 );
9:           do in parallel for ( q = 1; q < Q; q++ ) {
10:             if ( b == B−1 ) {
11:                ŝ_q = x̂(Kb+k); /* Set x̂(m) to S */
12:                f̂_q(k) = ext(0); /* Clear flag F */
13:             }
14:             else { /* Set 1s to candidate bits in d̂_q */
15:                d̂_q = ~( ( ŝ_q ^ ext([T_{M_q}]_{b+1}) ) | ( f̂_q(k) ^ ext(0) ) );
16:                ŝ_q = ( x̂(Kb+k) & d̂_q ) | ( ŝ_q & ~d̂_q );
17:                f̂_q(k) = f̂_q(k) | ~d̂_q ; /* Set flag F */
18:             }
19:             c_q += ones(ŝ_q); /* Count up 1s */
20:          }
21:          do in parallel for ( p = P−1; p >= 0; p-- )
22:             ([r̂_{L−1}(k)]_p ··· [r̂_0(k)]_p)_2 = encode( ([ŝ_{Q−1}]_p ··· [ŝ_1]_p)_2 );
23:       }
24:       do in parallel for ( q = 1; q < Q; q++ )
25:          [T_{M_q}]_b = c_q > N−M_q−1 ? 1 : 0; /* Update T_{Mq} */
26:    }
27:    for ( k = 0; k < K; k++ )
28:       do in parallel for ( p = P−1; p >= 0; p-- )
29:          ([ŝ_{Q−1}]_p ··· [ŝ_1]_p)_2 = decode( ([r̂_{L−1}(k)]_p ··· [r̂_0(k)]_p)_2 );
30:       do in parallel for ( q = 1; q < Q; q++ )
31:          t̂_q = ŝ_q | ~( f̂_q(k) | ext([T_{M_q}]_0) );
32:       do in parallel for ( p = P−1; p >= 0; p-- ) /* Output */
33:          ([ŷ_{L−1}(k)]_p ··· [ŷ_0(k)]_p)_2 = encode( ([t̂_{Q−1}]_p ··· [t̂_1]_p)_2 );
34:    }
35: }
```

Fig. 4 Proposed procedure of the multi-level quantization, where $\{\hat{y}_\ell\}$ and $\{\hat{T}_{M_q}\}$ denote the sets of $\hat{y}_\ell(k)$ and T_{M_q}, respectively. "`encode`" encodes its $Q-1$-bit inputs to L-bit outputs, and "`decode`" is the inverse function, where $[x]_p$ means the p-th bit of x and $(\cdot)_2$ is the binary representation. All of $\hat{x}(m)$, $\hat{r}_\ell(k)$, $\hat{f}_q(k)$, $\hat{d}_q$, $\hat{s}_q$ and $\hat{t}_q$ are P-bit digits.

encoding the results of $Q-1$ processes yield the Q-level quantized output.

Let Q be a power of 2 and L be $\log_2(Q)$. The algorithm of the Q-level quantization can be represented as shown in Fig. 4, where only the bit-parallel type is considered because it covers the bit-serial procedure.

Note that each process is not required to independently store the intermediate results. The functions "`encode`" and "`decode`" realize the sharing of them. The function "$y =$ `decode`(x)" decodes the unsigned integer x of $(L-1)$-bit into the $(Q-1)$-bit digit y such that

$$[y]_b = \begin{cases} 1, & x > b \\ 0, & \text{otherwise} \end{cases}, \quad b = 0, 1, \cdots, Q-2. \qquad (6)$$

For example, when $L = 2$, `decode`(0), `decode`(1), `decode`(2) and `decode`(3) return $(000)_2$, $(001)_2$, $(011)_2$ and $(111)_2$, respectively. On the other hand, "$x =$ `encode`(y)" is the inverse.

Note that the procedure described in Sects. 3.1 and 3.2 can be regarded as a special case of the multi-level procedure for $L = 1$.

4. The Architecture

Our proposed algorithm is suitable for hardware implementation since the data and control flow are quite simple. In this section, we propose the hardware architecture. As was mentioned before, the multi-level bit-parallel algorithm as shown in Fig. 4 covers the two-level bit-serial and bit-parallel procedures. Thus, we consider the architecture only for the multi-level bit-parallel algorithm. We firstly show the data path architecture, and then the control unit.

4.1 Data Path

The architecture of the proposed data path is shown in Fig. 5. It is considered as a synchronous system. Figs. 5 (a) and (b) show the architectures of the top module and the q-th processing element (PE_q), respectively. Each PE_q contributes the parallel processing for q in the list shown in Fig. 4.

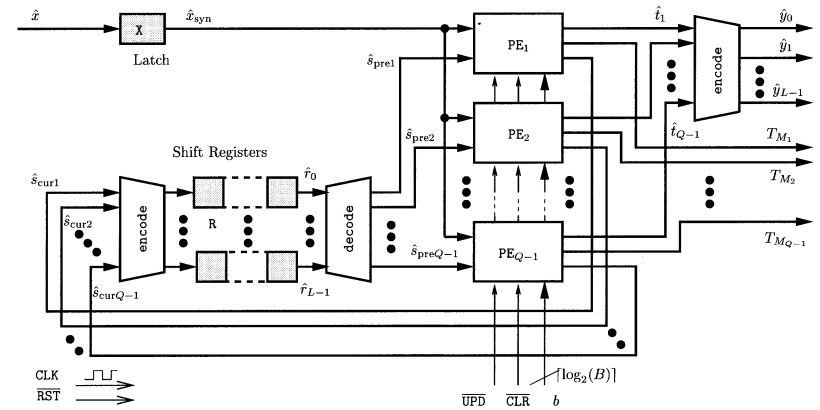
`CLK` and $\overline{\texttt{RST}}$ are the clock and reset signals, respectively. $\overline{\texttt{UPD}}$, $\overline{\texttt{CLR}}$ and b are the update signal for Counter `C` and Register `T`, and the clear signal for Shift Registers `F` and `R`, and the bit pointer of width $\lceil \log_2(B) \rceil$, respectively, which are generated by the control unit as shown later, where $\lceil x \rceil$ denotes the smallest integer larger than or equal to x.

All of $\hat{x}$, $\hat{x}_{\mathrm{syn}}$, $\hat{t}_q$, $\hat{y}_q$, $\hat{s}_{\mathrm{cur}q}$, $\hat{s}_{\mathrm{pre}q}$, $\hat{r}_\ell$ are P-bit digits, where $\hat{x}_{\mathrm{syn}}$ is the synchronized signal of $\hat{x}$ to `CLK`. $\hat{s}_{\mathrm{cur}q}$ and $\hat{s}_{\mathrm{pre}q}$ correspond to $\hat{s}_q$ in the list shown in Fig. 4, where the former is the value in the current iteration, and the latter is that of the previous iteration, respectively. "$N - M_q - 1$" is a constant.

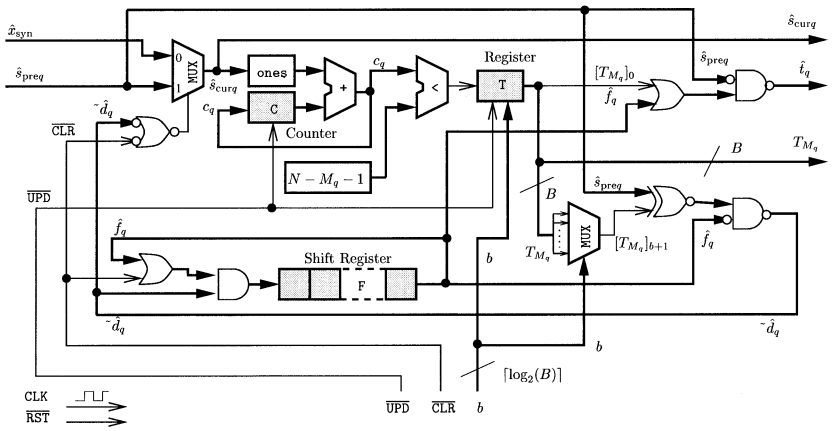
In the proposed architecture, we adopt shift registers for storing $\hat{f}_q$ and $\hat{r}_q$, because they remove the logic circuit required for pointing the k-th elements of $\hat{f}_q(k)$ and $\hat{r}_q(k)$.

The timing performance is as follows:

- Throughput [blocks/cycle]: P/NB
- Latency [cycles]: NB/P + (Prearrangement)



(a) Top module



(b) The q-th processing element (PE_q)

Fig. 5 Architecture of data path, where hatched box consists of noncombinational logic component. CLK and $\overline{\mathtt{RST}}$ are the clock and reset signals, respectively. $\overline{\mathtt{UPD}}$, $\overline{\mathtt{CLR}}$ and b are the control signals.

Table 2 Size of registers. N, B, P and L are the number of samples per block, the number of bits, the digit size and the number of quantization bit. P is assumed to be a divisor of N.

Component	Size of Register (bit)
Shift Reg. R	$L \times N/P \times P$
Shift Reg. F	$(2^L-1) \times N/P \times P$
Counter C	$(2^L-1) \times \lceil \log_2(N) \rceil$
Register T	$(2^L-1) \times B$
Latch X	P
Control Unit	$\lceil \log_2(N/P) \rceil + \lceil \log_2(B) \rceil$

In addition, Table 2 summarizes the size of registers, where P is assumed to be a divisor of N.

Note that, when the quantization bit L is one, the number of PEs is reduced to one. Thus, the circuit for "encode" and "decode" is removed. When the digit size P is one, the circuit "ones" of each PE is removed.

4.2 Control Unit

Figure 6 shows the architecture of the control unit. The control unit has KB states $\{\mathtt{S}_{b,k}\}$ for $b = 0, 1, \cdots, B-1$ and $k = 0, 1, \cdots, K-1$, where $K = N/P$. These states translate to another as follows:

- Any state $\mathtt{S}_{k,b}$ translates to $\mathtt{S}_{0,0}$ if $\overline{\mathtt{RST}} = 0$.
- $\mathtt{S}_{k,b}$ translates to $\mathtt{S}_{((k+1))_K,b}$ if $k \neq 0$ and $\overline{\mathtt{RST}} = 1$.
- $\mathtt{S}_{0,b}$ translates to $\mathtt{S}_{1,((B-1+b))_B}$ if $\overline{\mathtt{RST}} = 1$.

The output functions are defined by

- $\overline{\mathtt{UPD}} = 0$ when $\mathtt{S}_{0,b}$ for any b, otherwise $\overline{\mathtt{UPD}} = 1$.
- $\overline{\mathtt{CLR}} = 0$ when $\mathtt{S}_{k,B-1}$ for any k, otherwise $\overline{\mathtt{CLR}} = 1$.

The control flow is quite simple because each state translates to either $\mathtt{S}_{0,0}$ or another.

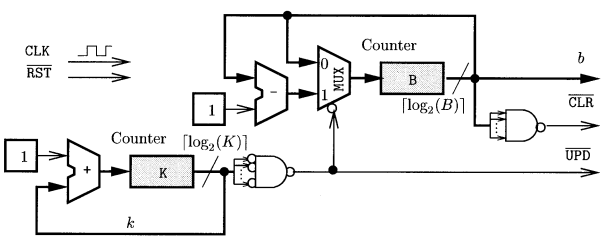


Fig. 6 Architecture of the control unit, where $K = N/P$.

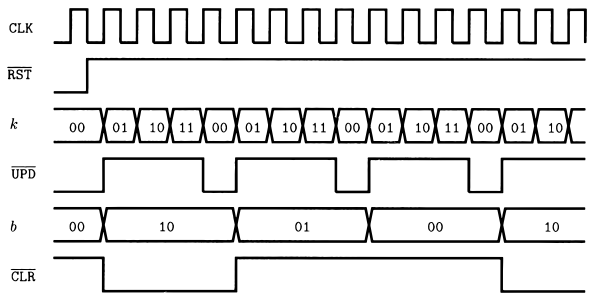


Fig. 7 Timing chart, where $B = 3$, $N = 8$ and $P = 2$.

An example of the timing chart is given in Fig. 7, where $B = 3$, $N = 8$ and $P = 2$. The threshold T_{M_q} and the output $\hat{y}_q$ are available when $\overline{\texttt{CLR}}$ is low.

5. Estimation

In this section, in order to show the significance of the proposed architecture, we show the timing, area and the power estimation. Firstly, we consider applying it to the low-bit motion estimation [2], [3], [7]–[9], and then the influence of the parameters to the area is discussed.

5.1 Application to Low-Bit Motion Estimation

The video compression standards MPEG and H.263 employ the motion compensated prediction coding technique for exploiting the temporal correlation [7]. The motion compensation requires to be preceded by the motion estimation, which is known to dominate most of the computation. Thus, a lot of realization issues has been discussed so far. The low-bit motion estimation technique is a solution to efficiently implement it [2], [3], [7]–[9]. The estimation follows a quantization, and, in the articles [2], [3], [9], it is shown that an adaptive quantization is suitable and MCQ is the good candidate for it.

Let $F_h \times F_v$ be the number of pixels per frame, and F_t be the number of frames per second (fps). Since the quantization is applied to each macro block, the number of blocks that the quantization has to complete a second is derived as $(F_h \times F_v \times F_t)/N$ [blocks/sec], where N is the number of pixels in a macro block.

The throughput of the proposed MCQ is P/NB blocks/cycle. Thus, the condition for the clock period $t_{\texttt{CLK}}$ is derived as follows:

$$t_{\texttt{CLK}} < \frac{P}{F_h \times F_v \times F_t \times B} \text{ [sec]}. \tag{7}$$

The broadcast video quality is specified by $F_h \times F_v = 720 \times 480$ pixels, $F_t = 30$ fps and $B = 8$ bits [7], where we assume that the quantization is applied only to the luminance component. Thus, $t_{\texttt{CLK}}$ is required to be smaller than $12P$ [nsec].

In order to estimate the performance, we modeled the proposed architecture by VHDL as an edge-triggered synchronous system [10]. Table 3 shows the estimation of the synthesis results, where the Synopsis design tools Ver.1998.02 [11] are used with the linear model of the standard cell library EXDLIB provided by VDEC for 0.5 μm triple-metal CMOS technology [12]. The environments are set as follows:

- No driving cell is set to CLK. Inverters are set to all inputs except for CLK as driving cells.
- An inverter is set to each output as load.
- The operating condition is set to "WCCOM."
- The wire load is set to "10×10."
- Toggles are occurred every rising edge of CLK for all inputs and each static probability is 0.5, except for $\overline{\texttt{RST}}$.

Table 3 Estimation from the synthesis results of the proposed MCQ, where $B = 8$ and CLK is constrained to the period $t_{\texttt{CLK}} = 10P$ [nsec]. "(met)" means no violation is occurred.

Digit size P		1	2	4	8
Clock period $t_{\texttt{CLK}}$ [nsec]		10	20	40	80
Max. data arrival time [nsec]	$L = 1$	10.16 -	19.51 (met)	33.10 (met)	36.52 (met)
	$L = 2$	13.04 -	19.52 (met)	36.03 (met)	39.08 (met)
Total cell area $\times 10^5$ [μm^2]	$L = 1$	7.16	7.09	7.05	7.34
	$L = 2$	17.88	17.82	17.79	18.68
Net switching power [mW]	$L = 1$	243.7	130.3	69.6	35.0
	$L = 2$	611.3	329.4	182.1	87.3

The above environments are chosen as a temporary example. Because more practical environments highly depend on the peripheral circuits and operating condition, we here simply assume that all input and output ports are connected to inverters and a clock tree would be generated for CLK. WCCOM is the worst operating condition defined in EXDLIB. The toggles are set as the worst case in terms of the input switching power.

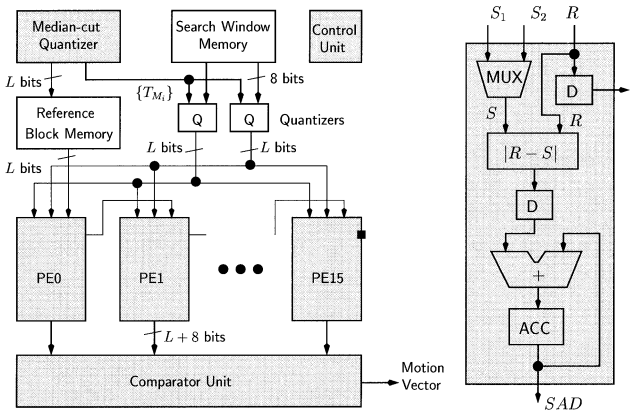
The constraints are as follows:

- CLK is constrained to $t_{\texttt{CLK}} = 10P$ [nsec].
- Area is not constrained.

From Table 3, we see that the bit-parallel procedure meets the specified processing time whereas the bit-serial procedure ($P = 1$) cannot. This is because the larger digit size P is, the larger throughput is. Increasing the period $t_{\texttt{CLK}}$ results in the considerable saving of the power consumption. The main reason is that the switching power consumption is almost proportional to the clock frequency among comparable scale CMOS circuits [13].

On the other hand, area is estimated to be almost flat. From Table 2, it is seen that the digit size is insensitive to the size of registers. The total cell area in the case $P = 1$ is larger than those for $P = 2$ and $P = 4$. This is because the synthesis tool tries to satisfy the timing constraint by using a lot of large cells.

Each critical path, which occurs maximum data



(a) MCQ and linear array (b) PEn

Fig. 8 Linear array architecture of motion estimator with MCQ, where L denotes the number of quantized bits.

arrival time, results in one from either Register `T` or the control unit to Register `T` through the nets for $[T_{M_q}]$, $\tilde{\hat{d}}_q$, $s_{\mathrm{curr}q}$, c_q and the succeeding components "ones," adder and comparator.

5.2 1-Bit and Full-Bit Motion Estimator

Let us investigate a 1-bit motion estimator (ME) with MCQ and compare it with a full 8-bit architecture. As an example, we consider a linear array architecture with 16 processing elements (PEs) of sum of absolute difference (SAD) for full-search block matching motion estimation, where the search window is of size 31×31 [7].

Figure 8 (a) shows an architecture with MCQ, where L denotes the number of quantized bits and the component Median-cut Quantizer is assumed to be a bit-serial MCQ. The component Q quantizes the input according to the thresholds $\{T_{M_i}\}$. Figure 8 (b) is the internal architecture of each PE, where the components D, MUX, $|R - S|$, + and ACC are a delay element, multiplexer, combinational logic for absolute difference, adder and an accumulator, respectively.

For the 1-bit case, L equals 1, and the size of the buffer memory Reference Block Memory, which contains samples in a macro block, is $16 \times 16 = 256$ bits. On the other hand, the full-bit architecture takes 8 as L. The components Median-cut Quantizer and Q are removed. However, Reference Block Memory become of size $16 \times 16 = 256$ bytes, that is 2048 bits.

In the followings, the common intermediate format (CIF) is assumed. CIF is specified by $F_h \times F_v = 352 \times 288$ pixels, $F_t = 30$ fps and $B = 8$ bits [7]. Thus, from Eq. (7), the clock period $t_{\mathtt{CLK}}$ should be less than 41.10 [nsec], where $P = 1$. The linear array ME, however, requires 4096 cycles to complete the operations for a macro block, and the clock period $t'_{\mathtt{CLK}}$ should satisfy the condition $t'_{\mathtt{CLK}} < N/(F_h \times F_v \times F_t \times 4096)$. It results in $t'_{\mathtt{CLK}} < 20.55$ [nsec].

From the above discussion, we estimate the area and power of MCQ and linear array ME by the synthesis results from their VHDL models with the constraints $t_{\mathtt{CLK}} = 40$ [nsec] and $t'_{\mathtt{CLK}} = 20$ [nsec], respectively. Environments are chosen to be the same with that made in the previous subsection. We estimate the performance of MCQ and linear array ME independently because of the difference in their clock periods. There are several strategies to implement the whole system with one clock signal such as clock gating for MCQ, doubling the throughput of each PE in ME and so forth. In this paper, we omit to detail the implementation, but give simple estimation.

Table 4 Comparison between 1-bit ME with MCQ and full 8-bit ME, where L is the number of quantized bits. The clocks of periods 40 [nsec] and 20 [nsec] are supplied to MCQ and ME, respectively.

(a) Total cell area (μm^2)

L	1 bit	8 bits
MCQ	6.90	-
ME	7.87	21.56
Total	14.77	21.56

(b) Net switching power (mW)

L	1 bit	8 bits
MCQ	55.16	-
ME	137.50	693.35
Total	192.66	693.35

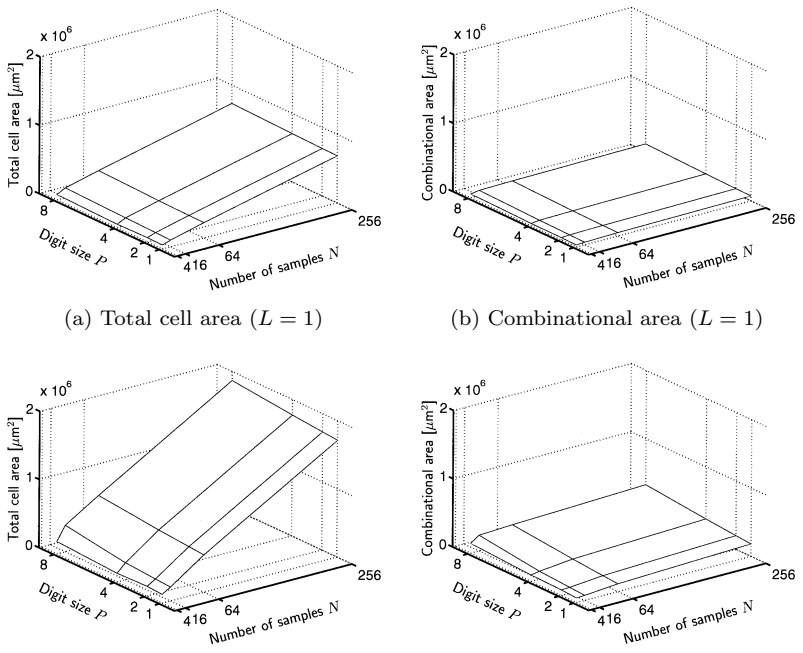
Table 4 (a) shows the total cell area for both MCQ and ME unit, that is, the shaded region in the figure. From the table, the area of 1-bit ME is about 70% of that of full-bit one. Furthermore, considerable reduction on the area for the buffer memory, Reference Block Memory, is achieved when L is 1. On the assumption that the buffer area proportionally increases with the capacity, we can roughly estimate that the area is reduced to 1/8. The area for Q is relatively quite smaller than the other components.

The estimated power consumption is given in Table 4 (b). It is seen that about 70% saving of the power consumption is achieved by introducing a 1-bit ME with MCQ, where the other components such as the buffer memory are not taken account of.

5.3 Discussions on Area

The proposed architecture is quite simple in the data and control flow. Hence, the total cell area is mostly occupied by the non-combinational area. Figure 9 shows the estimation of the total cell area and the combinational area, where the `CLK` is constrained to $t_{\mathtt{CLK}} = 80$ [nsec] for all case. The environments are the same with that used in the estimation given in Sect. 5.1.

From Fig. 9, we see that the total cell area increases approximately in proportion to the number of samples N whereas the combinational area relatively keeps small area. P does not affect the area as much as N, since it affects just the size of Latch `X` and some combinational logic circuits such as "`ones`" and the adder for Counter `C`. The result is explained by Table 2.



(a) Total cell area ($L = 1$) (b) Combinational area ($L = 1$)

(c) Total cell area ($L = 2$) (d) Combinational area ($L = 2$)

Fig. 9 Cell area, which is estimated by Synopsis design compiler Ver.1998.02 from the VHDL model with the standard cell library EXDLIB provided by VDEC for 0.5 μm triple-metal CMOS technology, where $B = 8$. `CLK` is constrained to 80 [nsec].

6. Conclusions

In this paper, we proposed an algorithm of the median-cut quantization, and the hardware architecture. The search process of the median value is derived from the pruning technique of the quick-sort algorithm. In the proposed method, a bit-level pivot selection is proposed. The selection was shown to correspond to selecting the masked median value, and enables us to quantize given samples concurrently. Since the search process is done bit by bit and gives quite simple data and control flow, it is suitable for hardware implementation.

Firstly, we proposed the two-level bit-serial procedure and then modified it to the bit-parallel one so as to improve the throughput. The extension to the multi-level quantization was also discussed. Then, we proposed the hardware architecture for the multi-level bit-parallel procedure.

In order to show the significance, we gave the timing, area and the power estimation from the synthesis result of the VHDL model. We firstly considered applying it to the low-bit motion estimation, and discussed the influence of the parameters to the total cell area. It was shown that the larger digit size P releases the time constraint and decrease the power consumption without significant increasing of the total cell area.

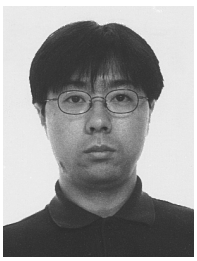
Acknowledgement

The design estimation in this study has been derived by VLSI design tools licensed via the VLSI Design and Education Center (VDEC), the University of Tokyo. We used the CMOS standard cell libraries developed in the VDEC program [12]. We would like to appreciate their cooperation.

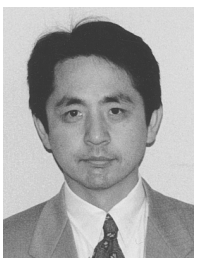
References

[1] P. Heckbert, "Color image quantization for frame buffer display," Proc. of SIGGRAPH '82, pp.297–307, 1982.
[2] H. Kiya, J. Furukawa, and Y. Noguchi, "Block matching motion estimation using less gray level images for MPEG video," ITC-CSCC '98, vol.1, pp.167–170, 1998.
[3] H. Kiya, J. Furukawa, and Y. Noguchi, "Block matching motion estimation based on median cut quantization for MPEG video," IEICE Trans. Fundamentals, vol.E82-A, no.6, pp.899–903, June 1999.
[4] D.E. Knuth, The art of Computer Programming vol.III, Addison Wesley, 1973.
[5] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest, and R.E. Tarjan, "Time bounds for selection," J. Computer and System Sciences, vol.7, pp.448–461, 1972.
[6] E.R. Dougherty and P.A. Laplante, Introduction to Real-Time Imaging, IEEE Press, 1995.
[7] V. Bhaskaran and K. Konstantinides, Image and Video Compression Standards, Algorithms and Architectures, 2nd Edition, Kluwer Academic Publishers, 1997.
[8] B. Natarajan, V. Bhaskaran, and K. Konstantinides, "Low-complexity block-based motion extimation via one-bit transforms," IEEE Trans. Circuits & Syst., vol.7, no.4, pp.702–706, 1997.

[9] S. Lee, J.-M. Kim, and S.-I. Chae, "New motion estimation algorithm using adaptively quantized low bit-resolution image and its VLSI architecture for MPEG2 video encoding," IEEE Trans. Circuits & Syst. for Video Technology, vol.8, no.6, pp.734–743, 1998.
[10] D.L. Perry, VHDL, McGraw-Hill, Inc., 1994.
[11] Synopsys, Inc, 700 East Middlefield Road, Mountain View, CA 94042 USA, Online Documentation Version 1998.02, 1998.
[12] K. Shiomi, K. Okino, T. Kawasaki, T. Ishihara, and H. Yasuura, "Development of a standard cell livrary for VDEC," IEICE Technical Report, CAS97-31, VLD97-31, DSP97-46, 1997.
[13] P. Pirsch, Architectures for Digital Signal Processing, John Wiley & Sons Ltd., 1998.



Shogo Muramatsu was born in Tokyo, Japan, on May 2, 1970. He received the B.E., M.E. and D.E. degrees in electrical engineering from Tokyo Metropolitan University in 1993, 1995 and 1998, respectively. In 1997, he joined Tokyo Metropolitan University. In 1999, he joined Niigata University, where he is currently a research associate of Electrical and Electric Engineering, Faculty of Engineering. His research interests are in digital signal processing, multirate systems, image processing and VLSI architecture. Dr. Muramatsu is a Member of the Institute of Electrical and Electronics Engineers, Inc. (IEEE) of USA.



Hitoshi Kiya was born in Yamagata, Japan, on November 16, 1957. He received the B.E. and M.E. degrees in electrical engineering from Nagaoka University of Technology, Niigata, Japan, and the D.E. degree in electrical engineering from Tokyo Metropolitan University, Tokyo, Japan, in 1980, 1982 and 1987, respectively. In 1982, he joined Tokyo Metropolitan University, where he is currently a Professor of Electrical Engineering, Graduate School of Engineering. He was a visiting researcher of the University of Sydney in Australia from Oct. 1995 to March 1996. His research interests are in digital signal processing, multirate systems, adaptive filtering, image processing, and efficient algorithms for VLSI implementation. Dr. Kiya is a Member of the Institute Electrical and Electronics Engineers, Inc. (IEEE) of USA, the Image Electronics Engineers of Japan and the Institute of Image Information and Television Engineers of Japan. He is an Associate Editor of IEEE Transactions on Signal Processing.

Akihiko Yamada was born in Osaka, Japan, on September 5, 1936. He received the B.E. and the D.E. degrees from Osaka University, Osaka, Japan in 1955 and 1980, respectively. In 1955, he joined NEC Corporation. In 1993, he joined Tokyo Metropolitan University, where he is currently a Professor of Electrical Engineering, Graduate School of Engineering. His research interests are in hardware/software codesign and VLSI design methodologies. Dr. Yamada is a senior member of the Institute Electrical and Electronics Engineers, Inc. (IEEE) of USA, and a member of the Association for Computing Machinery (ACM) of USA and Information Processing Society of Japan.