Published in final edited form as:
Integration (Amst). 2008 February 1; 41(2): 193–209. doi:10.1016/j.vlsi.2007.05.002.

On VLSI Design of Rank-Order Filtering using DCRAM Architecture

Meng-Chun Lin and **Lan-Rong Dung**
Dept. of Electrical and Control Engineering, National Chiao Tung University, Hsinchu City, Taiwan 300, ROC

Abstract

This paper addresses on VLSI design of rank-order filtering (ROF) with a maskable memory for real-time speech and image processing applications. Based on a generic bit-sliced ROF algorithm, the proposed design uses a special-defined memory, called the dual-cell random-access memory (DCRAM), to realize major operations of ROF: threshold decomposition and polarization. Using the memory-oriented architecture, the proposed ROF processor can benefit from high flexibility, low cost and high speed. The DCRAM can perform the bit-sliced read, partial write, and pipelined processing. The bit-sliced read and partial write are driven by maskable registers. With recursive execution of the bit-slicing read and partial write, the DCRAM can effectively realize ROF in terms of cost and speed. The proposed design has been implemented using TSMC 0.18 μm 1P6M technology. As shown in the result of physical implementation, the core size is $356.1 \times 427.7 \mu m^2$ and the VLSI implementation of ROF can operate at 256 MHz for 1.8V supply.

Keywords

CMOS memory integrated circuits; Coprocessors; Image processing; Median filters; Nonlinear filters

1 Introduction

Rank-order filtering (ROF), or order-statistical filtering, has been widely applied for various speech and image processing applications [1]–[6]. Given a sequence of input samples $\{x_{i-k}, x_{i-k+1}, \ldots, x_i, \ldots, x_{i+l}\}$, the basic operation of rank order filtering is to choose the r-th largest sample as the output y_i, where r is the rank-order of the filter. This type of ROF is normally classified as *the non-recursive ROF*. Another type of ROF is called *the recursive ROF*. The difference between the recursive ROF and the non-recursive ROF is that the input sequence of the recursive ROF is $\{y_{i-k}, y_{i-k+1}, \ldots; y_{i-1}, x_i, \ldots, x_{i+l}\}$ Unlike linear filtering, ROF can remove sharp discontinuities of small duration without blurring the original signal; therefore, ROF becomes a key component for signal smoothing and impulsive noise elimination. To provide this key component for various signal processing applications, we intend to design a configurable rank-order filter that features low cost and high speed.

Email addresses: asurada.ece90g@nctu.edu.tw, lennon@cn.nctu.edu.tw (Meng-Chun Lin, Lan-Rong Dung).

Many approaches for hardware implementation of rank-order filtering have been presented in the past decades [8,10–24]. Many of them are based on sorting algorithm [11,22,23,25–28]. They considered the operation of rank-order filtering as two steps: sorting and choosing. Papers [10,19] have proposed systolic architectures for rank-order filtering based on sorting algorithms, such as bubble sort and bitonic sort. These architectures are fully pipelined for high throughput rate at the expense of latency, but require a large number of compare-swap units and registers. To reduce the hardware complexity, papers [8,12,14,15,29–32] present linear-array structures to maintain samples in sorted order. For a sliding window of size *N*, the linear-array architectures consist of *N* processing elements and require three steps for each iteration: finding the proper location for new coming sample, discarding the eldest one, and moving samples between the newest and eldest one position. The three-step procedure is called delete-and-insert (DI). Although the hardware complexity is reduced to *O(N)*, they require a large latency for DI steps. Paper [31] further presents a micro-programmable processor for the implementations of the median-type filters. Paper [20] presents a parallel architecture using two-phase design to improve the operating speed. In this paper, they first modified the traditional content-addressable memory (CAM) to a shiftable CAM (SCAM) processor with shiftable memory cells and comparators. Their architecture can take advantages of CAM for parallelizing the DI procedure. Then, they use two-phase design to combine delete and insert operations. Thereafter, the SCAM processor can quickly finish DI operations in parallel. Although the SCAM processor has significantly increased the speed of the linear-array architecture, it can only process a new sample at a time and cannot efficiently process 2-D data. For a window of size *n-by-n*, the SCAM processor needs *n* DI procedures for each filtering computation. To have an efficient 2-D rank-order filter, papers [12,27] present solutions for 2-D rank-order filtering at the expense of area.

In addition to the sorting algorithm, the paper [35] applies the threshold decomposition technique for rank-order filtering. To simplify the VLSI complexity, the proposed approach uses three steps: decomposition, binary filtering, and recombination. The proposed approach significantly reduce the area complexity from exponential to linear. Papers [10,13,16–18,21, 33,34] employ the bit-sliced majority algorithm for median filtering, the most popular type of rank-order filtering. The bit-sliced algorithm [36,37] bitwisely selects the ranked candidates and generates the ranked result one bit at a time. Basically, the bit-sliced algorithm for median filtering recursively executes two steps: majority calculation and polarization. The majority calculation, in general, dominates the execution time of median filtering. Papers [10], [17] and [37] present logic networks for implementation of majority calculation. However, the circuits are time-consuming and complex so that they cannot take full advantages of bit-sliced algorithm. Some papers claim that this type of algorithm is impractical for logic circuit implementation because of its exponential complexity [31]. Paper [21] uses an inverter as a voter for majority calculation. It significantly improves both cost and processing speed, but the noise margin will become narrow as the number of inputs increases. The narrow noise margin makes the implementation impractical and limits the configurability of rank-order filtering.

Instead of using logic circuits, this paper presents a novel memory architecture for rank-order filtering based on a generic rank-order filtering algorithm. The generic rank-order filtering algorithm uses the threshold decomposition to bitwisely determine the rank-order result from the most significant bit (MSB) to the least significant bit (LSB) and applies the polarization simultaneously to polarize impossible candidates. Using this algorithm, we can pick the result, bit-by-bit, for a specific rank-order without sorting the numbers. Note that the sorting is the most complex part in conventional ROF implementations. Basically, there are three major tasks in the generic algorithm; they are parallel read, threshold decomposition, and parallel polarization. This paper presents a maskable memory structure, motivated from CAM architecture, to realize these tasks efficiently. The maskable memory structure, called dual-cell

random-access memory (DCRAM), is an extended SRAM structure with maskable registers and dual cells. The maskable registers allow the architecture to selectively read or write bit-slices, and hence speed up ”parallel read” and ”parallel polarization” tasks. The control of maskable registers is driven by a long-instruction-word (LIW) instruction set. The LIW makes the proposed architecture programmable for various rank-order filtering algorithms, such as recursive and non-recursive ROFs. The proposed architecture has been implemented using TSMC 0.18um 1P6M technology and successfully applied for 1-D/2-D ROF applications. For 9-point 1-D and 3-by-3 2-D ROF applications, the core size is $356.1 \times 427.7 um^2$. As shown in the post-layout simulation, the DCRAM-based processor can operate at 290 MHz for 3.3V supply and 256 MHz for 1.8V supply. For image processing, the performance of the proposed processor can process video clips of SVGA format in real-time.

The rest of the paper is organized as follows. In Section 2, we will first introduce the generic bit-sliced ROF algorithm with an example. Section 3 details the proposed architecture and its implementation. Section 4 particularly emphasizes the essence of the proposed design, DCRAM. Section 5 describes the instruction set with a programming example. Section 6 gives examples for application of the proposed processor. Section 7 presents a fully-pipelined version of the proposed ROF architecture at the expense of area. Section 8 demonstrates results of physical implementation and post-layout simulation. Section 9 qualitatively compares the proposed design with existing ROF architectures. Finally, Section 10 concludes our contribution and merits of this work.

2 The generic bit-sliced rank-order filtering algorithm

Let $W_i=\{x_{i-k}, x_{i-k+1}, \ldots; x_i, \ldots, x_{i+l}\}$ be a window of input samples. The binary code of each input x_j is denoted as $u_j^{B-1} \cdots u_j^1 u_j^0$. The output y_i of the r-th order filter is the r-th largest sample in the input window W_i, denoted as $v_i^{B-1} \cdots v_i^1 v_i^0$. The algorithm sequentially determines the r-th order value bit-by-bit starting from the most significant bit (MSB) to the least significant bit (LSB). To start with, we first count 1’s from the MSB bit-slice of input samples and use Z_{B-1} to denote the result. The b-th bit-slice of input samples is defined as $u_{i-k}^b u_{i-k+1}^b \ldots u_i^b \ldots u_{i+l}^b$. If Z_{B-1} is greater than or equal to r, then v_i^{B-1} is 1; otherwise, v_i^{B-1} is 0. Any input sample whose MSB has the same value as v_i^{B-1} is considered as one of candidates of the r-th order sample. On the other hand, if the MSB of an input sample is not equal to v_i^{B-1}, the input sample will be considered as a non-candidate. Non-candidates will be then polarized to either the largest or smallest value. If the MSB of an input sample x_j is 1 and v_i^{B-1} is 0, the rest bits (or lower bits) of x_j are set to 1’s. Contrarily, if the MSB of an input sample x_j is 0 and v_i^{B-1} is 1, the rest bits (or lower bits) of x_j are set to 0’s. After the polarization, the algorithm counts 1’s from the consecutive bit-slice and then repeats the polarization procedure. Consequently, the r-th order value can be obtained by recursively iterating the steps bit-by-bit. The following pseudo code illustrates the generic bit-sliced rank-order filtering algorithm:

Given the input samples, the window size $N=l+k+1$, the bitwidth B and the rank r, do:

Step 1: Set b=B−1.

Step 2: (Bit counting)

Calculate Z_b from $\{ u_{i-k}^b, u_{i-k+1}^b, \ldots, u_i^b, \ldots, u_{i+l}^b \}$.

Step 3: (Threshold decomposition)

If $Z_b \geq r$, $v_i^b = 1$; otherwise $v_i^b = 0$.

Step 4: (Polarization)

If $u_j^b \neq v_i^b$, $u_j^m = u_j^b$ for $0 \leq m \leq b-1$ and $i-k \leq j \leq i+l$

Step 5: $b=b-1$.

Step 6: If $b \geq 0$ go to Step 2.

Step 7: Output y_i.

Fig. 1 illustrates a bit-sliced ROF example for $N=7$, $B=4$, and $r=1$. Given that the input samples are $7(0111_2)$, $5(0101_2)$, $11(1011_2)$, $14(1110_2)$, $2(0010_2)$, $8(1000_2)$, and $3(0011_2)$, the generic algorithm will produce $14(1110_2)$ as the output result. At the beginning, the "Bit counting" step will calculate the number of 1's at MSBs, which is 3. Since the number of 1's is greater than r, the "Threshold decomposition" step sets the MSB of y_i to '1'. Then, the "Polarization" step will consider the inputs with $u_j^3 = 1$ as candidates of the ROF output and polarize the lower bits of the others to all 0's. After repeating the above steps with decreasing b, the output y_i will be $14(1110_2)$.

3 The dual-cell RAM architecture for rank-order filtering

As mentioned above, the generic rank-order filtering algorithm generates the rank-order value bit-by-bit without using complex sorting computations. The main advantage of this algorithm is that the calculation of rank-order filtering has low computational complexity and can be mapped to a highly parallel architecture. In the algorithm, there are three main tasks: bit counting, threshold decomposition, and polarization. To have these tasks efficiently implemented, this paper presents an ROF processor based on a novel maskable memory architecture, as shown in Fig. 2. The memory structure is highly scalable with the window size increasing, by simply adding memory cells. Furthermore, with the instruction decoder and maskable memory, the proposed architecture is programmable and flexible for different kinds of ROFs.

The dual-cell random-access memory (DCRAM) plays a key role in the proposed ROF architecture. In the DCRAM, there are two fields for reusing the input data and pipelining the filtering process. For the one-dimensional (1-D) ROF, the proposed architecture receives one sample at a time. For the *n-by-n* two-dimensional (2-D) ROF, the architecture reads n samples into the input window within a filtering iteration. To speed up the process of rank-order filtering and pipeline the data loading and filtering calculation, the data field loads the input data while the computing field is performing bit-sliced operations. Hence, the execution of the architecture has two pipeline stages: data fetching and rank-order calculation. In each iteration, the data fetching first loads the input sample(s) into the data field and then makes copies from the data field to the computing field. After having the input window in the computing field, the rank-order calculation bitwisely accesses the computing field and executes the ROF tasks.

The computing field is in the maskable part of DCRAM. The maskable part of DCRAM performs parallel reads for bit counting and parallel writes for polarization. The read-mask register (RMR) is configured to mask unwanted bits of the computing field during read operation. The value of RMR is one-hot-encoded so that the bit-sliced values can be read from the memory in parallel. The bit-sliced values will then go to the Level-Quantizer for threshold decomposition. When the ROF performs polarization, the write-mask register (WMR) is configured to mask untouched bits and allow the polarization selector (PS) to polar lower bits

of noncandidate samples. Since the structure of memory circuits is regular and the maskable scheme provides fast logic operations, the maskable memory structure features low cost and high speed. It obviously outperforms logic networks on implementation of bit counting and polarization.

To start with the algorithm, the RMR is one-hot-masked according to the value *b* in the generic algorithm and then the DCRAM outputs a bit-sliced value $\{u^b_{i-k}, u^b_{i-k+1}, \ldots, u^b_i, \ldots, \ldots u^b_{i+l}\}$ on "**c_d**". The bit-sliced value will go to both the Level-Quantizer and PS. The Level-Quantizer performs the Step 2 and Step 3 by summing up bits of the bit-sliced value to Z_b and comparing Z_b with the rank value *r*. The rank value *r* is stored in the rank register (RR). The bitwidth *w* of Z_b is $[log_2^N]$. Fig. 3 illustrates the block diagram of the Level-Quantizer, where FA denotes the full adder and HA denotes the half adder. The signals "S" and "C" of each FA or HA represent sum and carry, respectively. The circuit in the dash-lined box is a comparator. The comparator is implemented by a carry generator because the comparison result of Z_b and *r* can be obtained from the carry output of Z_b plus the two's complement of *r*. The carry output is the quantized value of the Level-Quantizer.

Normally, the comparison can be made by subtracting *r* from Z_b. Since Z_b and *r* are unsigned numbers, to perform the subtraction, both numbers have to reformat to two's complement numbers by adding a sign bit. In this paper, the reformated numbers of Z_b and *r* are expressed as $Z_{b,S}$ and r_S, respectively. Since both numbers are positive, their sign bits are equal to '0'. If $Z_{b,S}$ is less than r_S, the result of subtraction, Δ, will be negative; that is, the sign bit (or MSB) of Δ is '1'. Eq. 1 shows the inequation of the comparison, where $\overline{r_s}$ denotes the one's complement of r_S and **1** denotes $(00 \ldots 01)_2$. Because the MSB of $Z_{b,S}$ is '0' and the MSB of $\overline{r_s}$ is '1', to satisfy Eq. 1, the carry of $Z^{w-1}_{b,s}+\overline{r^{w-1}_s}$ must be equal to '0' so that the sign bit of Δ becomes '1'. To simplify the comparison circuit, instead of implementing an adder, we use the carry generator to produce the carry of $Z^{w-1}_{b,s}+\overline{r^{w-1}_s}$. Each cell of the carry generator is a majority (Maj) circuit that performs the boolean function shown in Eq. 2. Furthermore, we use an OR gate at the LSB stage because of Eq. 3. Thus, the dash-lined box is an optimized solution for comparison of Z_b and *r* without implementing the *bit-summation* and *signed-extension* parts.

$$\Delta = Z_{b,S} + \overline{r_s} + \mathbf{1} < 0. \tag{1}$$

$$Maj(A, B, C) = AB + BC + AC. \tag{2}$$

$$Z^0_b \cdot \overline{r^0} + Z^0_b \cdot 1 + 1 \cdot \overline{r^0} = Z^0_b + \overline{r^0}. \tag{3}$$

After the Level-Quantizer finishes the threshold decomposition, the quantized value goes to the LSB of the shift register, "sr[0]". Then, the polarization selector (PS) uses exclusive ORs (XORs) to determine which words should be polarized, as shown in Fig. 4. Obviously, the XORs can examine the condition of $u^b_j \neq v^b_i$ and select the word-under-polarization's (WUPs) accordingly. When "**c_wl**" is '1', the lower bits of selected words will be polarized; the lower bits are selected by WMR. According to the Step 4, the polarized value is u^b_j which is the inversion of v^b_i. Since v^b_i is the value of sr[0], we inverse the value of "sr[0]" to "c_in", as shown in Fig. 2.

As seen in the generic algorithm, the basic ROF repeatedly executes *Bit-counting*, *Threshold decomposition*, and *Polarization* until the LSB of the ROF result being generated. Upon executing *B* times of three main tasks, the ROF will have the result in the Shift Register. A cycle after, the result will then go to the output register (OUTR). Doing so, the proposed architecture is able to pipeline the iterations for high-performance applications.

4 Implementation of Dual-Cell Random-Access Memory

Fig. 5 illustrates a basic element of DCRAM. Each element has two cells for data field and computing field, respectively. The data cell is basically an SRAM cell with a pair of bitlines. The SRAM cell is composed of INV1 and INV2 and stores a bit of input sample addressed by the wordline "d_wl[i]". The computing cell performs three tasks: *copy*, *write*, and *read*. When the copy-line "cp" is high, through INV5 and INV6, the pair of INV3 and INV4 will have the copy of the 1-bit datum in the data cell. The *copy* operation is unidirectional, and the pair of INV5 and INV6 can guarantee this directivity. When the one-bit value stored in the computing cell needs to be polarized, the "wm[j]" and "c_wl[i]" will be asserted, and the computing cell will perform the *write* operation according to the pair of bitlines "c_bl[j]" and "$\overline{\text{c\_bl[j]}}$". When the ROF reads the bit-sliced value, the computing cell uses an NMOS, gated by "rm[j]", to output the complement value of the stored bit to the dataline "$\overline{\text{c\_d[i]}}$". The datalines of computing cells of each word will be then merged as a single net. Since the RMR is one-hot configured, each word has only a single bit being activated during the *read* operation.

As shown in Fig. 6, the dataline "$\overline{\text{c\_d[i]}}$" finally goes to an inverter to pull up the weak '1', which is generated by the "rm[j]"-gated NMOS, and hence the signal "c_d[i]" has the value of the *i*-th bit of each bit-slice. Because the ROF algorithm polarizes the non-candidate words with either all zeros or all ones, the bitline pairs of computing cells are merged as a single pair of "c_in" and "$\overline{\text{c\_in}}$".

Fig. 7 illustrates the implementation of DCRAM with the floorplan. Each D_i–C_i pair is a maskable memory cell where D_i denotes D_cell(*i*) and C_i denotes C_cell(*i*). Each word is split into higher and lower parts for reducing the memory access time and power dissipation [38]. The control block is an interface between control signals and address decoder. It controls wordlines and bitlines of DCRAM. When the write signal "wr" is not asserted, the control block will disassert all wordlines by the address decoder.

5 Instruction set of proposed ROF processor

The proposed ROF processor is a core for the impulsive noise removal and enabled by an instruction sequencer. Fig. 8 illustrates the conceptual diagram of the ROF processor. The instruction sequencer is used for the generation of instruction codes and the control of input/output streams. The instruction sequencer can be a microprocessor or dedicated hardware.

Fig. 9 lists the format of the instruction set. An instruction word contains two subwords: the data field instruction and the computing field instruction. Each instruction cycle can concurrently issue two field instructions for parallelizing the data preparation and ROF execution; hence, the proposed processor can pipeline ROF iterations. When one of the field instructions performs "no operation", `DF_NULL` or `CF_NULL` will be issued. All registers in the architecture are updated a cycle after instruction issued.

The instruction `SET` resets all registers and set the rank register RR for a given rank-order *r*. The instruction `LOAD` loads data from "**d_in**" by asserting "wr" and setting "addr". The instruction `COPY/DONE` can perform the "COPY" operation or "DONE" operation. When the bit value of *c* is '1', the DCRAM will copy a window of input samples from the data field to

the computing field. When the bit value of *d* is '1', the DCRAM wraps up an iteration by asserting "en" and puts the result into OUTR.

The instruction `P_READ` is issued when the ROF algorithm executes bit-sliced operations. The field <*mask*> of `P_READ` is one-hot coded. It allows the DCRAM to send a bit-slice to the Level-Quantizer and PS for the *Threshold decomposition* task. The instruction `P_WRITE` is issued when the ROF algorithm performs the *Polarization* task. The field <*mask*> of `P_WRITE` is used to set a consecutive sequence of 1's. The sequence can mask out the higher bits for polarization.

Given a 1-D rank order filter application with N =9 and r =3, for instance, the pseudo-code for programming the ROF processor is as follows:

```
SET 3;

i=0; -- IS

start loop; -- IS

LOAD i, P_READ 00000001;

COPY, P_READ 10000000;

DONE, P_WRITE 01111111;

P_READ 01000000;

P_WRITE 00111111;

P_READ 00100000;

P_WRITE 00011111;

P_READ 00010000;

P_WRITE 00001111;

P_READ 00001000;

P_WRITE 00000111;

P_READ 00000100;

P_WRITE 00000011;

P_READ 00000010;

P_WRITE 00000001;

i++; -- IS

i=i mod 9; -- IS

end loop; -- IS
```

To generate instructions to the ROF processor, the complete 1-D non-recursive ROF circuit includes an instruction sequencer, as shown in Fig. 10. Based on the pseudo-code of the 1-D ROF, the instruction sequencer will generate the instruction codes to the ROF processor. In the pseudo-code, the lines with the comment “ `--IS`” are not parts of the instruction sequence, but realized in the instruction sequencer. Given the above pseudo-code, the instruction sequencer will generate instructions as shown below. The instructions other than “ `SET 3;`” will be repeatedly sent to the instruction decoder.

```
SET 3;

LOAD 0000, P_READ 00000001;

COPY, P_READ 10000000;

DONE, P_WRITE 01111111;

P_READ 01000000;

P_WRITE 00111111;

P_READ 00100000;

P_WRITE 00011111;

P_READ 00010000;

P_WRITE 00001111;

P_READ 00001000;

P_WRITE 00000111;

P_READ 00000100;

P_WRITE 00000011;

P_READ 00000010;

P_WRITE 00000001;

LOAD 0001, P_READ 00000001;

COPY, P_READ 10000000;

DONE, P_WRITE 01111111;

P_READ 01000000;

P_WRITE 00111111;

P_READ 00100000;

P_WRITE 00011111;

P_READ 00010000;
```

```
P_WRITE 00001111;

P_READ 00001000;

P_WRITE 00000111;

P_READ 00000100;

P_WRITE 00000011;

P_READ 00000010;

P_WRITE 00000001;

… … …

LOAD 1000, P_READ 00000001;

COPY, P_READ 10000000;

DONE, P_WRITE 01111111;

P_READ 01000000;

P_WRITE 00111111;

P_READ 00100000;

P_WRITE 00011111;

P_READ 00010000;

P_WRITE 00001111;

P_READ 00001000;

P_WRITE 00000111;

P_READ 00000100;

P_WRITE 00000011;

P_READ 00000010;

P_WRITE 00000001;

LOAD 0000, P_READ 00000001;

COPY, P_READ 10000000;

DONE, P_WRITE 01111111;

… … …
```

Since the instruction set is in the format of long-instruction-word (LIW), the data fetching and ROF computing can be executed in parallel. So, the generated instruction stream can pipeline

the ROF iterations, and the data fetching is hidden in each ROF latency. Fig. 11 shows the reservation table of the 1-D ROF example. As seen in the reservation table, the first iteration and the second iteration are overlapped at the seventeenth, eighteenth and nineteenth clock steps. At the seventeenth clock step, the second iteration starts with loading a new sample while the first iteration processes the LSB bit-slice. At the eighteenth clock step, the second iteration copies samples from the data field to the computing field, and reads the MSB bit-slice. At the same time, the first iteration prepares the first ROF result for OUTR. At the nineteenth clock step, the first iteration sends the result out while the second iteration performs the first polarization. Thus, the iteration period for each iteration is 15 cycles.

6 Application of the proposed ROF processor

In Section 5, we use 1-D non-recursive ROF as an example to show the programming of the proposed ROF processor. Due to the programmable design, the proposed ROF processor can implement a variety of ROF applications. The following subsections will illustrate the optimized programs for three examples: 1-D RMF, 2-D non-recursive ROF, and 2-D RMF.

6.1 1-D recursive median filter

The recursive median filtering (RMF) has been proposed for signal smoothing and impulsive noise elimination. It can effectively remove sharp discontinuities of small duration without blurring the original signal. The RMF recursively searches for the median results from the most recent median values and input samples. So, the input window of RMF can be denoted as $\{y_{i-k}, y_{i-k+1}, \ldots, y_{i-1}, x_i, \ldots, x_{i+l}\}$, where $y_{i-k}, y_{i-k+1}, \ldots, y_{i-1}$ are the most recent median values and $x_i, \ldots, x_{i+l}$ are the input samples, and the result y_i is the $[(l + k + 1)/2]$-th value of the input window.

Fig. 12 demonstrates the implementation of the 1-D RMF. To recursively perform RMF with previous median values, the *i*-th iteration of 1-D RMF loads two inputs to the DCRAM; one is x_{i+l} and the other is y_{i-1}. As shown in Fig. 12, the 2-to-1 multiplexer is used to switch the input stream to the data field, controlled by the instruction sequencer; the input stream is from either "**d_in**" or "**d_out**". When the proposed ROF processor receives the input stream, the program will arrange the data storage as shown in Fig. 12. The date storage shows the data reusability of the proposed ROF processor.

Given a 1-D RMF application with N =9 and r =5, the pseudo-code is written as follows:

```
SET 5;

i=0;

start loop; -- IS

set input_sel, 0; -- IS

LOAD i, CF_NULL;

DONE, CF_NULL;

set input_sel, 1; -- IS

LOAD ((i+4) mod 9), CF_NULL;

COPY, P_READ 10000000;
```

```
P_WRITE 01111111;

P_READ 01000000;

P_WRITE 00111111;

P_READ 00100000;

P_WRITE 00011111;

P_READ 00010000;

P_WRITE 00001111;

P_READ 00001000;

P_WRITE 00000111;

P_READ 00000100;

P_WRITE 00000011;

P_READ 00000010;

P_WRITE 00000001;

P_READ 00000001;

i++; -- IS

i=(i mod 9); -- IS

end loop; -- IS
```

As mentioned above, the input stream to the DCRAM comes from either "**d_in**"" or "**d_out**"". The statements of "`set input_sel, 0;`" and "`set input_sel, 1;`" can assert the signal "input_sel" to switch the input source accordingly. The statements of "`LOAD i, CF_NULL;` " and "`LOAD i, CF_NULL;` " is employed for the data stream, as per Fig. 13. As seen in Fig. 14, the throughput rate is limited by the recursive execution of the 1-D RMF; that is, the second iteration cannot load the newest median value until the first iteration generates the result to the output. However, we still optimized the throughput rate as much as possible. At the twentieth clock step, the program overlaps the first iteration and the second iteration so that the data fetching and result preparing can be run at the same time. As the result, the sample period is 18 cycles.

Fig. 15 illustrates the block diagram for the 2-D non-recursive ROF. From Fig. 16, each iteration needs to update three input samples (the pixels in the shadow region) for the 3 × 3 ROF; that is, only n input samples need to be updated in each iteration for the $n \times n$ ROF. To reuse the windowing data, the data storage is arranged as shown in Fig. 17. So, for the 2-D ROF, the data reusability of our process is high; each iteration updates only n input samples for an $n \times n$ window. Given a 2-D $n \times n$ ROF application with n=3 and r =5, the optimized reservation table can be scheduled as Fig. 18.

6.3 2-D recursive median filter

Similar to the 1-D RMF, the two-dimensional(2-D) *n-by-n* RMF finds the median value from the window formed by some previous-calculated median values and input values. Fig. 19(a) shows the content of the 3 × 3 window centered at (*i*, *j*). At the end of each iteration, the 2-D 3 × 3 RMF substitutes the central point of the current window with the median value. The renewed point will then be used in the next iteration. The windowing for 2-D RMF iterations is shown in Fig. 19(b), where the triangles represent the previous-calculated median values and the pixels in the shadow region are updated at the beginning of each iteration. According to the windowing, Fig. 20 illustrates the data storage for high degree of data reusability. Finally, we can implement the 2-D RMF as the block diagram illustrated in Fig. 21. Given a 2-D 3 × 3 RMF application, the optimized reservation table can be scheduled as Fig. 22.

7 The Fully-Pipelined DCRAM-based ROF Architecture

As seen in Section 6, the reservation tables are not tightly scheduled because the dependency of bit-slicing read, threshold decomposition, and polarization forms a cycle. The dependency cycle limits the schedulability of ROF tasks.

To increase the schedulability, we further extended the ROF architecture to a fully-pipelined version at the expense of area. The fully-pipelined ROF architecture interleaves three ROF iterations with triple computing fields. As shown in Fig. 23, there are three computing fields which process three tasks alternatively. To have the tightest schedule, we pipelined the Level-Quantizer into two stages, LQ1 and LQ2, so the loop (computing field, Level-Quantizer, Shift Register) has three pipeline stages for the highest degree of parallelism. The LQ1 is the FA/HA tree and the LQ2 is the carry generator.

Since there exists three iterations being processed simultaneously, a larger memory is required for two more iterations. Hence, we extended the DCRAM to an ($N + 2\delta$)-word memory, where N is the window size of ROF and δ is the number of updating samples for each iteration. The value of δ is 1 for 1-D ROF, and n for 2-D n-by-n ROF. To correctly access the right samples for each iteration, the signal "**cm**" is added to mask the unwanted samples during the copy operation. In each computing field, the unwanted samples are stored as all zeros. Doing so, the unwanted samples will not affect the rank-order results. Fig. 24 illustrates the modified computing cell for fully-pipelined ROF. The INV5 and INV6 are replaced with GATE1 and GATE2. When "cm[i]" is '0' the computing cell will store '0'; otherwise, the computing cell will have the copy of the selected sample from the data cell. Finally, we use "**cp**", "**w_cf**", and "**r_cf**" to selectively perform *read*, *write*, or *copy* on computing fields.

To efficiently program the fully-pipelined architecture, the instruction set is defined as shown in Fig. 25. The fields <*c_cf*> of `COPY`,<*w_cf*> of `P_WRITE`, and <*r_cf*> of `P_READ` are used to control "**cp**", "**w_cf**", and "**r_cf**". Given a 1-D non-recursive rank order filter application with N =9 and r =3, the reservation table can be optimized as shown in Fig. 26.

8 CHIP DESIGN AND SIMULATION RESULTS

To exercise the proposed architecture, we have implemented the ROF architecture, shown in Fig. 2, using TSMC 0.18 μm 1P6M technology. First, we verified the hardware in VHDL at the behavior level. The behavior VHDL model is cycle-accurate. As the result of simulation, the implementations of the above examples are valid. Fig. 27 and Fig. 28 demonstrate the results of VHDL simulations for the 2-D ROF and RMF, respectively. Fig. 27(a) is a noisy "Lena" image corrupted by 8% of impulsive noise. After being processed by 2-D ROFs with r=4, 5, and 6, the denoise results are shown in Fig. 27(b), (c), and (d), respectively. Fig. 28(a) is a noisy "Lena" image corrupted by 9% of impulsive noise. After being processed by the 2-

D 3×3 RMF, the denoise result is shown in Fig. 28(b). The results are the same as those of Matlab simulation.

Upon verifying the proposed ROF processor using the cycle-accurate behavior model, we then implemented the processor in the fully-custom design methodology. Because of high regularity of memory, the proposed memory-based architecture saves the routing area while comparing with the logic-based solutions. Fig. 29(a) shows the overall chip layout and the dash-lined region is the core. The die size is $1063.57 \times 1069.21 \mu m^2$ and the pinout count is 40. Fig. 29(b) illustrates the detail layout of the ROF core. The core size is $356.1 \times 427.7 \mu m^2$ and the total transistor count is 3942. Fig. 29(c) illustrates the floorplan and placement. The physical implementation has been verified by the post-layout simulation. Table 1 shows the result of timing analysis, obtained from NanoSim. As seen in the table, the critical path is the path 3 and the maximum clock rate can be 290 MHz at 3.3V and 256 MHz at 1.8V. As the result of post-layout simulation, the power dissipation of the proposed ROF is quite low. For the 1-D/2-D ROFs, the average power consumption of the core is 29mW at 290MHz or 7mW at 256MHz. The performance sufficiently satisfies the real-time requirement of video applications in the formats of QCIF, CIF, VGA, and SVGA. The chip is submitting to Chip Implementation Center (CIC), Taiwan for the fabrication.

Furthermore, We have successfully built a prototype which is composed of a FPGA board and DCRAM chips to validate the proposed architecture before fabricating the custom designed chip. The FPGA board is made by Altera and the FPGA type is APEX EP20K. The FPGA board can operate at 60 MHz at the maximum. The DCRAM chip was designed by full-custom CMOS technology. Fig. 30(a) shows the micrograph of the DCRAM chip. The chip implements a subword part of DCRAM and the Fig. 30(b) illustrates the chip layout. The fabricated DCRAM chip was layouted by full-custom design flow using TSMC 0.35 2P4M technology. As shown in Fig. 31, with the supply voltage of 3.3V, the DCRAM chip can operate at 25 MHz. Finally, we successfully integrated the FPGA board and the DCRAM chips into a prototype as shown in Fig. 32 The prototype was validated with ROF algorithms mentioned above.

9 Comparison of existing ROF architectures

Since this paper is the first one that uses SRAM-like memory as the kernel of the ROF processor, it is hard to quantitatively compare the proposed memory-based architecture with existing logic-based architecture. Therefore, we qualitatively make comparisons with the other ROF architectures. The follows will address on comparisons in terms of complexity, flexibility and regularity.

A variety of ROF architectures have been published in literature. We classified them into three types: 2-D sorting array, linear sorting array, and bit-serial logic network. The 2-D sorting array is fast, but costly. This type of ROF requires a large number of compare-swap units and registers. It has the hardware complexity of $O(BN^2)$, where N is the window size and B is the bitwidth of input samples. The latency of the 2-D sorting array is proportional to N. Comparing with the 2-D sorting array, the proposed ROF architecture has lower hardware complexity and latency complexity because the hardware complexity and latency complexity of the proposed ROF architecture are $O(BN)$ and $O(B)$, respectively.

The linear sorting array presents the linear structure to maintain samples in sorted order. For a sliding window of size N, the linear sorting array consists of N processing elements and repeatedly executes the delete-and-insert procedure, called DI. The DI contains three steps: finding the proper location for new coming sample, discarding the eldest one, and moving samples between the newest and eldest one position. Although the linear sorting array can reduce the hardware complexity to $O(N)$, it requires a large latency for DI steps and has the latency complexity of $O(N)$. Obviously, our architecture outperforms most of linear sorting

arrays. Paper [20] presents a shiftable memory, called SCAM, for reducing the the latency complexity, but the reduction is true only when applying their architecture for 1-D applications. For a window of size *n-by-n*, the SCAM processor needs *n* DI procedures for each filtering computation because each iteration of the 2-D ROF updates at least *n* samples. To have an efficient 2-D rank-order filter, papers [12,27] present the linear sorting arrays for 2-D rank-order filtering at the expense of area. Comparing with them, our approach has higher degree of flexibility for 1-D and 2-D applications while the hardware cost is low.

Based on the bit-sliced algorithm, the bit-serial logic network can bitwisely select the ranked candidates and generates the ranked result one bit at a time. The bit-serial logic network recursively executes two steps: majority calculation and polarization. This type of ROFs can reduce the latency complexity to O(*B*); however, many of them use complex logic networks to implement the majority calculation. Paper [21] uses an inverter as a voter for majority calculation. It significantly improves both hardware cost and processing speed; nevertheless, the noise margin will become narrow as the number of inputs increases.

The proposed architecture basically takes the advantages of the bit-sliced algorithm: (1) the latency is independent of the window size, and (2) the result can be obtained without exhaustive comparisons. Comparing with the bit-serial logic network, the proposed architecture has higher degree of flexibility and regularity because of the DCRAM structure. The DCRAM structure reduces not only the complexity of the majority calculation, but also the routing area between components.

Another major strength of the proposed ROF processor is the programmability. Paper [31] is one of the pioneer on programmable ROF processor; however their application is limited to median filtering. Thanks to the DCRAM structure, the proposed ROF processor is flexible with the variation of the rank order *r* and algorithms. As shown in Section 6, our ROF processor can be programmed for any rank order *r* and diverse applications. Furthermore, the proposed architecture can reuse input data as many as possible and hence reduce the power consumption on memory access.

10 CONCLUSION

In this paper, we have proposed an architecture based on a maskable memory for rank-order filtering. This paper is the first literature using maskable memory to realize ROF. Driving by the generic rank-order filtering algorithm, the memory-based architecture features high degree of flexibility and regularity while the cost is low and the performance is high. With the LIW instruction set, this architecture can be applied for arbitrary ranks and a variety of ROF applications, including recursive and non-recursive algorithms. As shown in the implementation results, the core of the processor has high performance and low cost. The post-layout simulation shows that the power consumption can be as low as 7 mW at 256 MHz. The processing speed can meet the real-time requirement of image applications in the QCIF, CIF, VGA, or SVGA formats.

Acknowledgments

This work was supported by the National Science Council, R.O.C., under the grant number NSC 94-2220-E-009-023. We gratefully acknowledge the implementation work made by Shih-Jay Huang.

References

1. Kang DH, Choi JH, Lee YH, Lee C. Applications of a DPCM system with median predictors for image coding. IEEE Trans Consumer Electronics Aug;1992 38(3):429–435.

2. Rantanen H, Karlsson M, Pohjala P, Kalli S. Color video signal processing with median filters. IEEE Trans Consumer Electron Apr;1992 38(3):157–161.

3. Viero T, Oistamo K, Neuvo Y. Three-dimensional median-related filters for color image sequence filtering. IEEE Trans Circuits Syst Video Technol Apr;1994 4(2):129–142.

4. Song, X.; Yin, L.; Neuvo, Y. Image sequence coding using adaptive weighted median prediction. Signal Processing VI, EUSIPCO-92; Brussels. Aug; 1992. p. 1307-1310.

5. Oistamo K, Neuvo Y. A motion insensitive method for scan rate conversion and cross error cancellation. IEEE Trans Consumer Electron Aug;1991 37:296–302.

6. Zamperoni, P. Variation on the rank-order filtering theme for grey-tone and binary image enhancement. IEEE Int. Conf. Acoust., Speech, Signal Processing; 1989. p. 1401-1404.

7. Chen, CT.; Chen, LG. A self-adjusting weighted median filter for removing impulse noise in images. Int. Conf. Image Processing; Sept; 1996. p. 16-19.

8. Yang D, Chen C. Data dependence analysis and bit-level systolic arrays of the median filter. IEEE Trans Circuits and Systems for Video Technology Dec;1998 8(8):1015–1024.

9. Ikenaga T, Ogura T. CAM2: A highly-parallel two-dimensional cellular automation architecture. IEEE Trans Computers July;1998 47(7):788–801.

10. Breveglieri L, Piuri V. Digital median filter. Journal of VLSI Signal Processing 2002;31:191–206.

11. Chakrabarti C. Sorting network based architectures for median filters. IEEE Trans Circuits ans Systems II: Analog and Digital Signal Processing Nov;1993 40:723–727.

12. Chakrabarti C. High sample rate architectures for median filters. IEEE Trans Signal Processing March;1994 42(3):707–712.

13. Chang L, Lin J. Bit-level systolic array for median filter. IEEE Trans Signal Processing Aug;1992 40(8):2079–2083.

14. Chen C, Chen L, Hsiao J. VLSI implementation of a selective median filter. IEEE Trans Consumer Electronics Feb;1996 42(1):33–42.

15. Hakami MR, Warter PJ, Boncelet CC Jr. A new VLSI architecture suitable for multidimensional order statistic filtering. IEEE Trans Signal Processing April;1994 42:991–993.

16. Hatirnaz, F.; Gurkaynak, K.; Leblebici, Y. A compact modular architecture for the realization of high-speed binary sorting engines based on rank ordering. IEEE Inter. Symp. Circuits and Syst; Geneva, Switzerland. May; 2000. p. 685-688.

17. Hiasat AA, Al-lbrahim MM, Gharailbeh KM. Design and implementation of a new efficient median filtering algorithm. IEE Proc Image Signal Processing Oct;1999 146(5):273–278.

18. Hoctor RT, Kassam SA. An algorithm and a pipelined architecture for order-statistic determination and L-filtering. IEEE Trans Circuits and Systems March;1989 36(3):344–352.

19. Karaman M, Onural L, Atalar A. Design and implementation of a general-purpose median filter unit in CMOS VLSI. IEEE Journal of Solid State Circuits April;1990 25(2):505–513.

20. Lee C, Hsieh P, Tsai J. High-speed median filter designs using shiftable content-addressable memory. IEEE Trans Circuits and Systems for Video Technology Dec;1994 4:544–549.

21. Lee CL, Jen C. Bit-sliced median filter design based on majority gate. IEE Proc-G Circuits, Devices and Systems Feb;1992 139(1):63–71.

22. Lucke LE, Parchi KK. Parallel processing architecture for rank order and stack filter. IEEE Trans Signal Processing May;1994 42(5):1178–1189.

23. Oazer K. Design and implementation of a single-chip 1-D median filter. IEEE Trans Acoust, Speech, Signal Processing Oct;1983 ASSP-31(4):1164–1168.

24. Richards DS. VLSI median filters. IEEE Trans Acoust, Speech, and Signal Processing January;1990 38:145–153.

25. Boncelet, GG, Jr. Recursive algorithm and VLSI implementation for median filtering. IEEE Int. Sym. on Circuits and Systems; June; 1988. p. 1745-1747.

26. Henning, C.; Noll, TG. Architecture and implementation of a bitserial sorter for weighted median filtering. IEEE Custom Integrated Circuits Conference; May; 1998. p. 189-192.

27. Lin, CC.; Kuo, CJ. Fast response 2-D rank order filter by using max-min sorting network. Int. Conf. Image Processing; Sept; 1996. p. 403-406.

28. Karaman, M.; Onural, L.; Atalar, A. Design and implementaion of a general purpose VLSI median filter unit and its application. IEEE Int. Conf. Acoustics, Speech, and Signal Processing; May; 1989. p. 2548-2551.
29. Hwang, J.; Jong, J. Systolic architecture for 2-D rank order filtering. Int. Conf. Application-Specific Array Processors; Sept; 1990. p. 90-99.
30. Pitas I. Fast algorithms for running ordering and max/min calculation. IEEE Trans Circuits and Systems June;1989 36(6):795–804.
31. Vainio O, Neuvo Y, Butner SE. A signal processor for median-based algorithm. IEEE Trans Acoustics, Speech, and Signal Processing Sept;1989 37(9):1406–1414.
32. Yu, H.; Lee, J.; Cho, J. A fast VLSI implementation of sorting algorithm for standard median filters. IEEE Int. ASIC/SOC Conference; Sptember; 1999. p. 387-390.
33. Fitch JP. Software and VLSI algorithm for generalized renked order filtering. IEEE Trans Circuits and Systems May;1987 CAS-34(5):553–559.
34. Karaman M, Onural L. New radix-2-based algorithm for fast median filtering. Electron Lett May; 1989 25:723–724.
35. Fitch JP. Software and VLSI Algorithms for Generalized Ranked Order Filtering. IEEE Trans Circuits and Syst May;1987 CAS-34(5):553–559.
36. Kar BK, Pradhan DK. A new algorithm for order statistic and sorting. IEEE Trans Signal Processing Aug;1993 41(8):2688–2694.
37. Pedroni VA. Compact hamming-comparator-based rank order filter for digital VLSI and FPGA implementations. IEEE Int Sym on Circuits and Systems May;2004 2:585–588.
38. Shobha, Singh; Shamsi, Azmi; Nutan, Agrawal; Penaka, Phani; Ansuman, Rout. Architecture and design of a high performance SRAM for SOC design. IEEE Int. Sym. on VLSI Design; Jan; 2002. p. 447-451.

11 Appendix I

Given a 2-D $n \times n$ ROF application with n=3 and r =5 and the following is the pseudo-code:

```
SET 5;

i=0; -- IS

start loop; -- IS

input_sel=0; -- IS

LOAD i, P_READ 00000010;

input_sel=1; -- IS

LOAD i+1, P_WRITE 00000001;

input_sel=2; -- IS

LOAD i+2, P_READ 00000001;

COPY, P_READ 10000000;

DONE, P_WRITE 01111111;

P_READ 01000000;

P_WRITE 00111111;
```

```
P_READ 00100000;

P_WRITE 00011111;

P_READ 00010000;

P_WRITE 00001111;

P_READ 00001000;

P_WRITE 00000111;

P_READ 00000100;

P_WRITE 00000011;

i++; -- IS

i=3*(i mod 3); -- IS

end loop; -- IS
```

Given a 2-D 3 × 3 RMF application, the pseudo-code is written as follows:

```
SET 5;

i=0; -- IS

start loop; -- IS

input_sel=0; -- IS

LOAD (i mod9), P_WRITE 00000001;

input_sel=1; -- IS

LOAD (i+1 mod 9), P_READ 00000001;

input_sel=2; -- IS

LOAD (i+2 mod 9), CF_NULL;

DONE, CF_NULL;

input_sel=3; -- IS

LOAD (i+4 mod 9), CF_NULL;

COPY, P_READ 10000000;

P_WRITE 01111111;

P_READ 01000000;

P_WRITE 00111111;
```

```
P_READ 00100000;

P_WRITE 00011111;

P_READ 00010000;

P_WRITE 00001111;

P_READ 00001000;

P_WRITE 00000111;

P_READ 00000100;

P_WRITE 00000011;

P_READ 00000010;

i=i+3; -- IS

end loop; -- IS
```

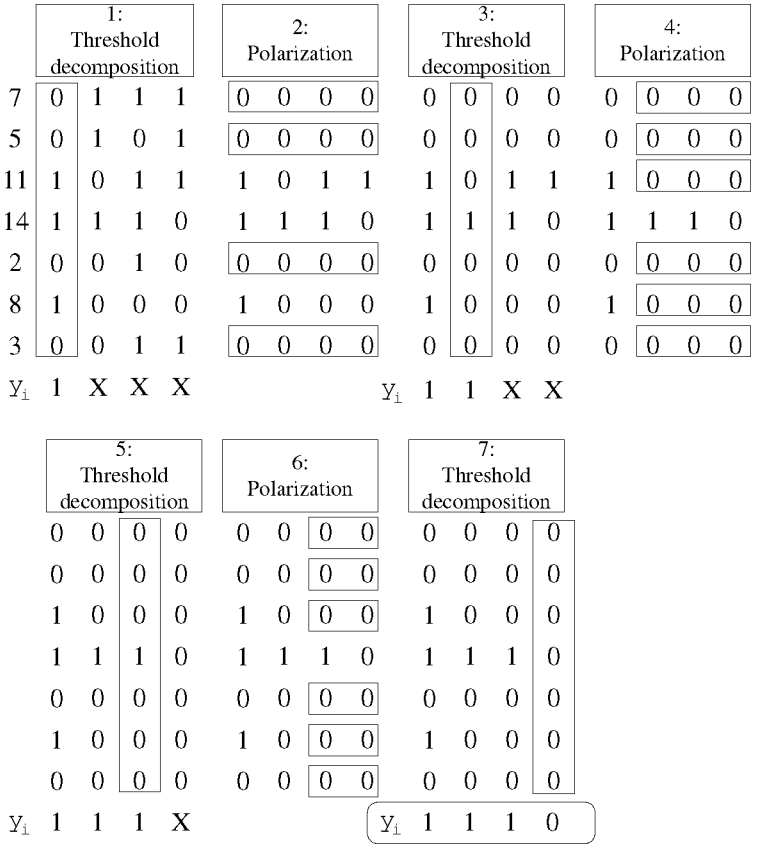


Fig. 1.
An example of the generic bit-sliced ROF algorithm for N =7, B=4, and r=1.

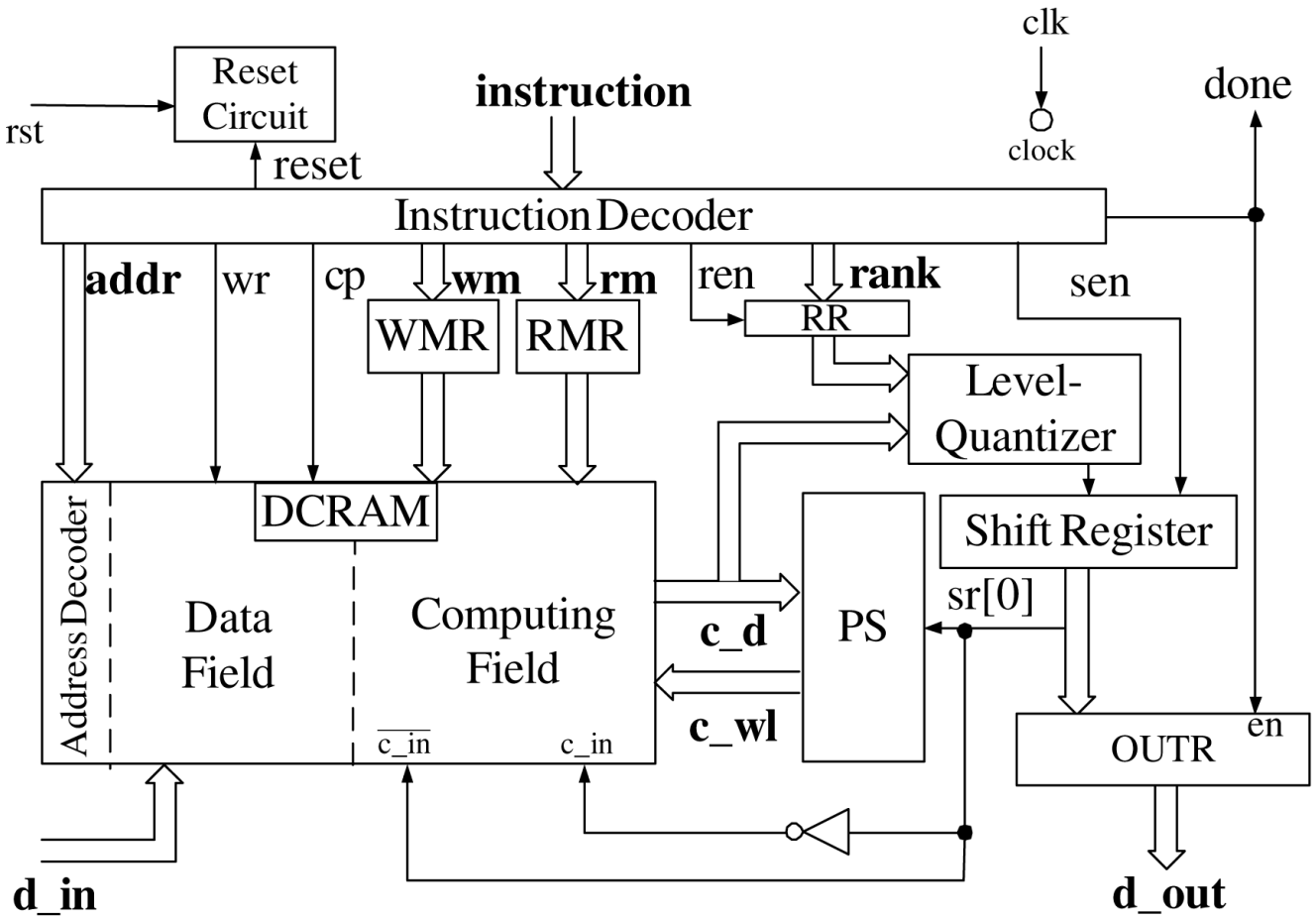


Fig. 2.
The proposed rank-order filtering architecture.

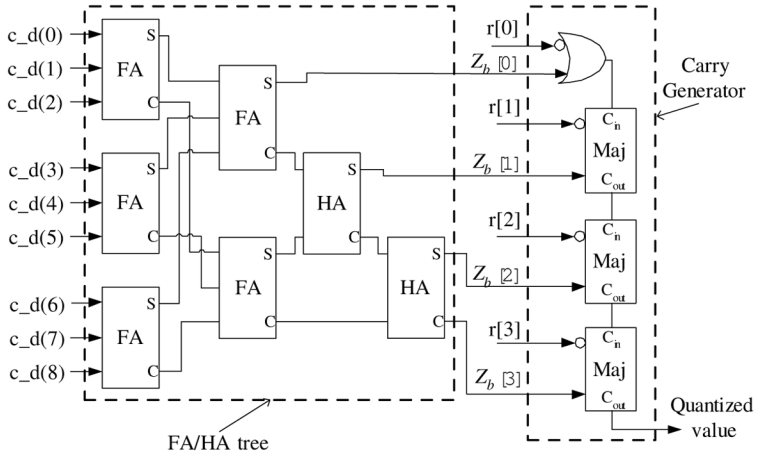


Fig. 3.
The block diagram of the Level-Quantizer.

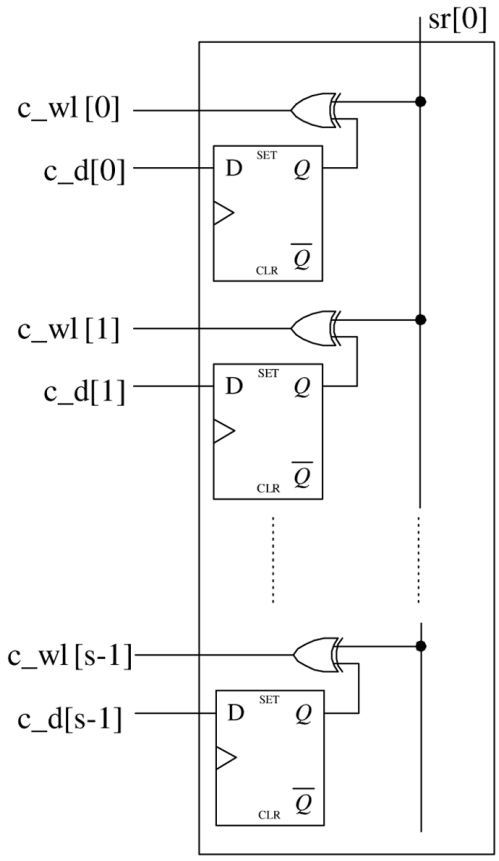


Fig. 4.
The polarization selector (PS).

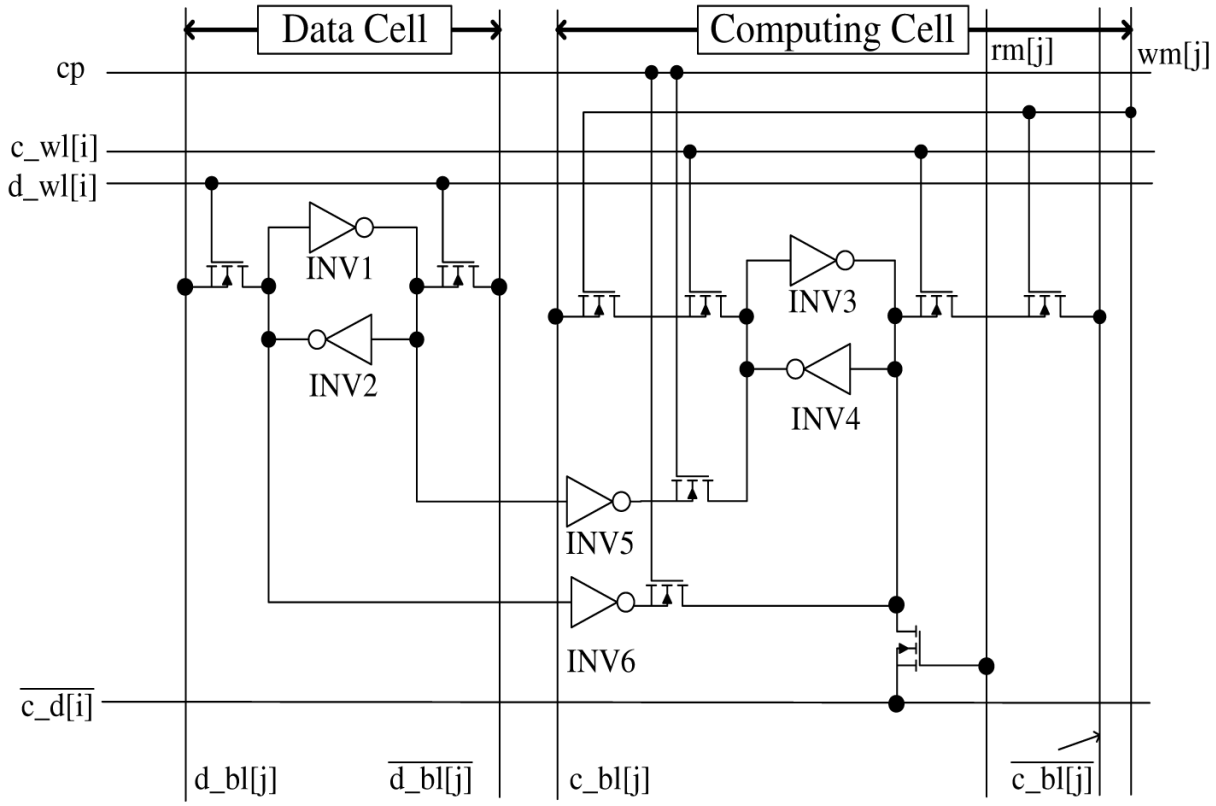


Fig. 5.
A basic element of DCRAM.

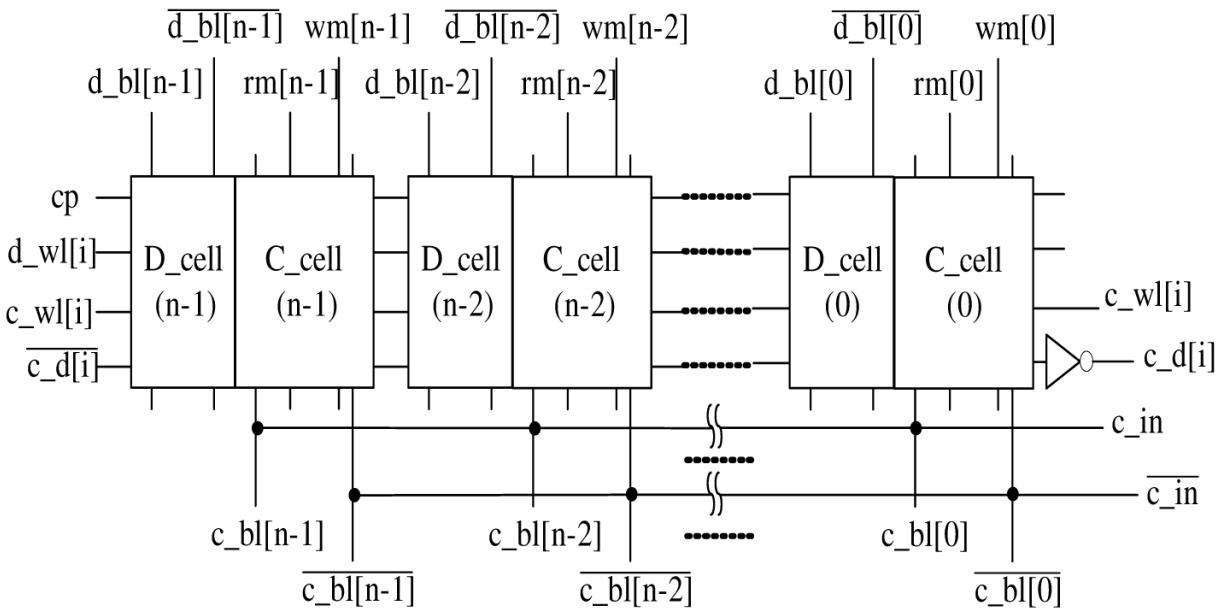


Fig. 6.
A DCRAM word mixing data field and computing field. D cell(*i*) denotes the data field of *i*-th bit and C cell(*i*) denotes the computing field of *i*-th bit.

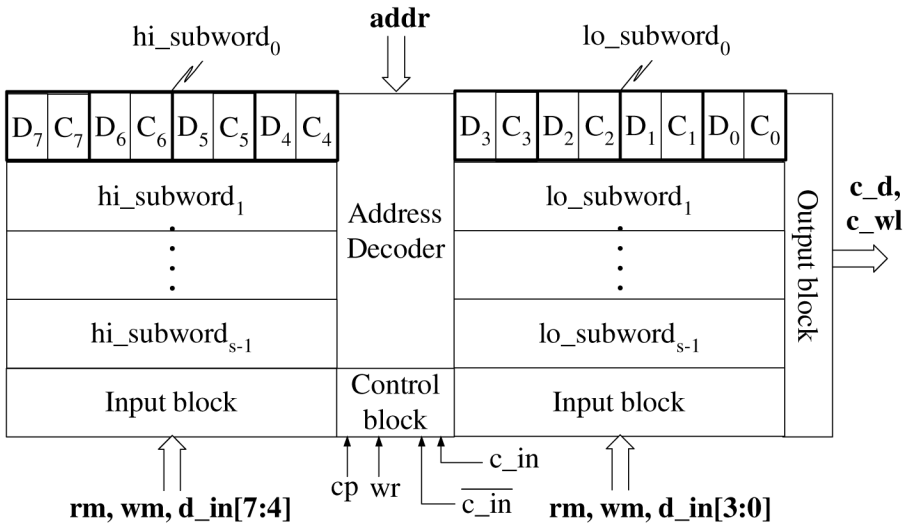


Fig. 7.
The floorplan of DCRAM.

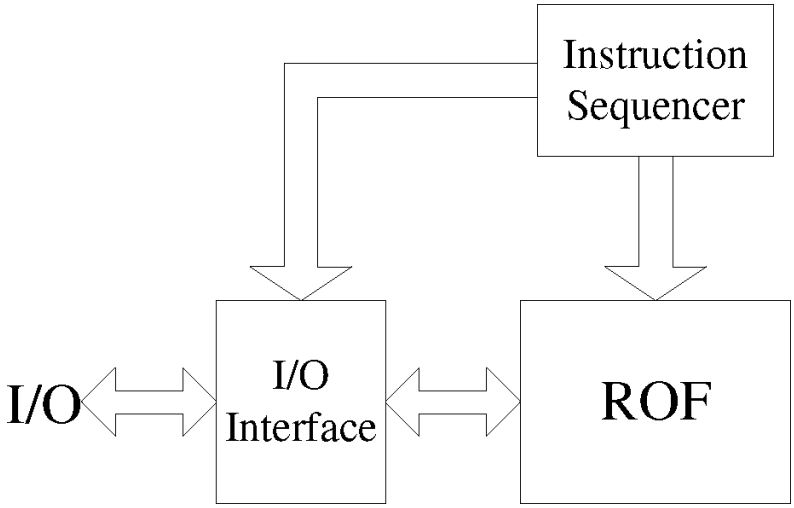


Fig. 8.
The conceptual diagram of the ROF processor.

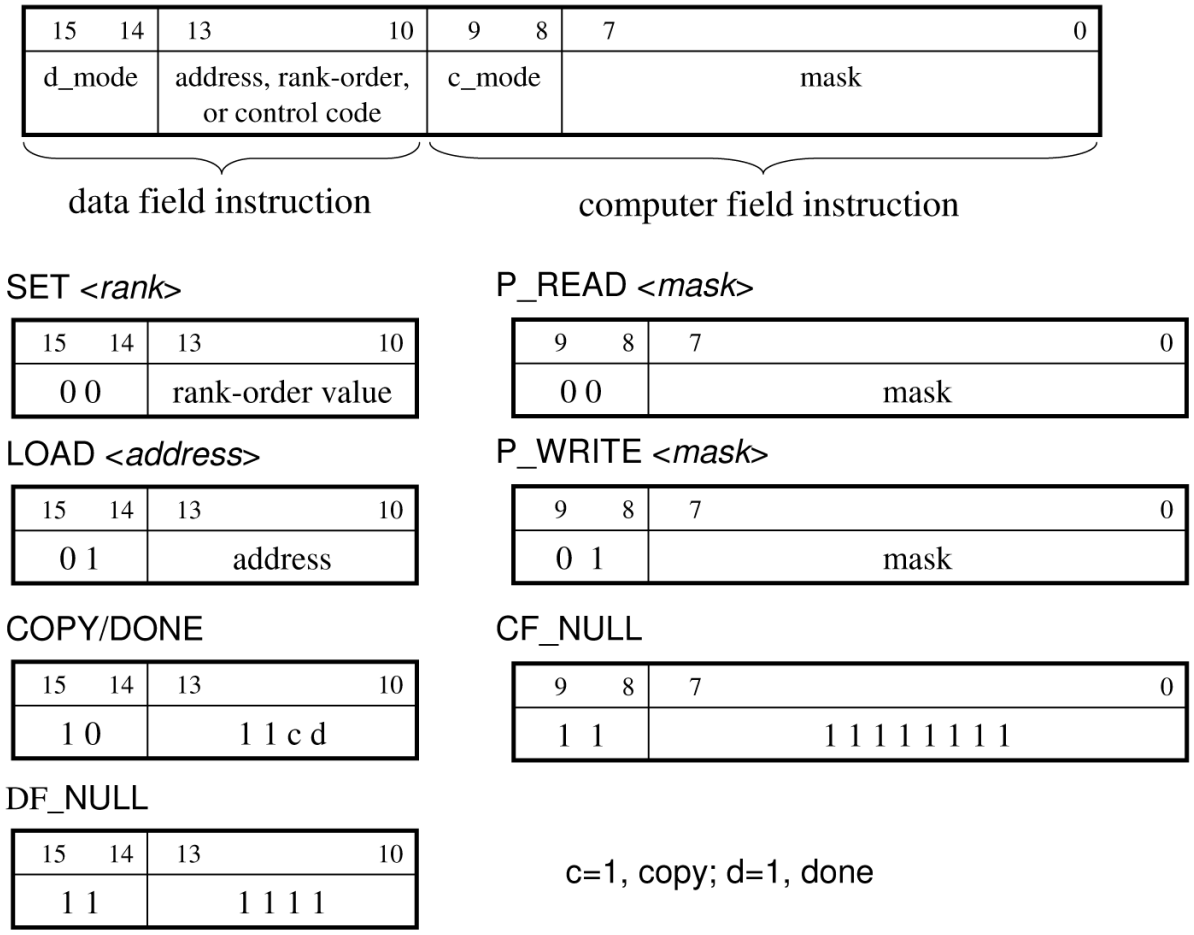


Fig. 9.
The format of the instruction set.

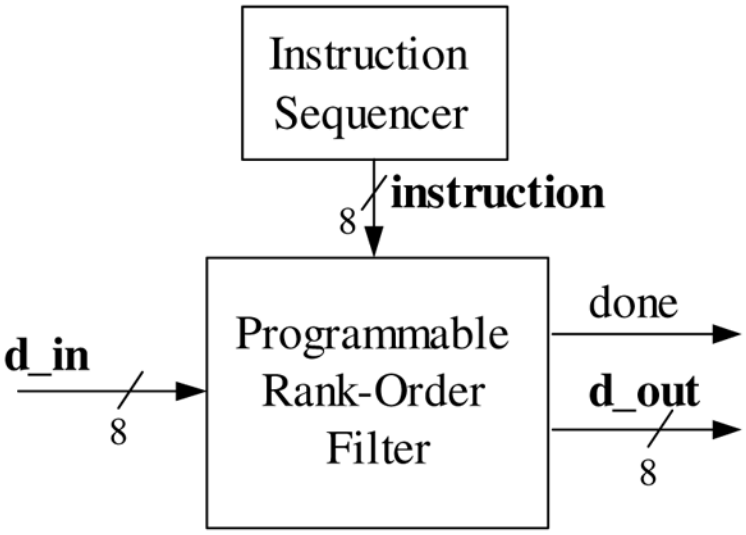


Fig. 10.
Block diagram of the 1-D non-recursive ROF.

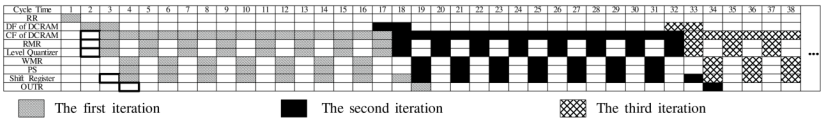


Fig. 11.
Reservation table of the 1-D non-recursive ROF.

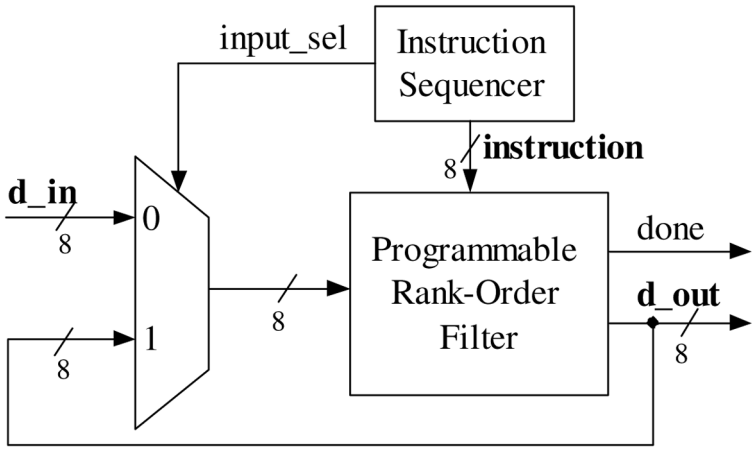


Fig. 12.
Block diagram of the 1-D RMF.

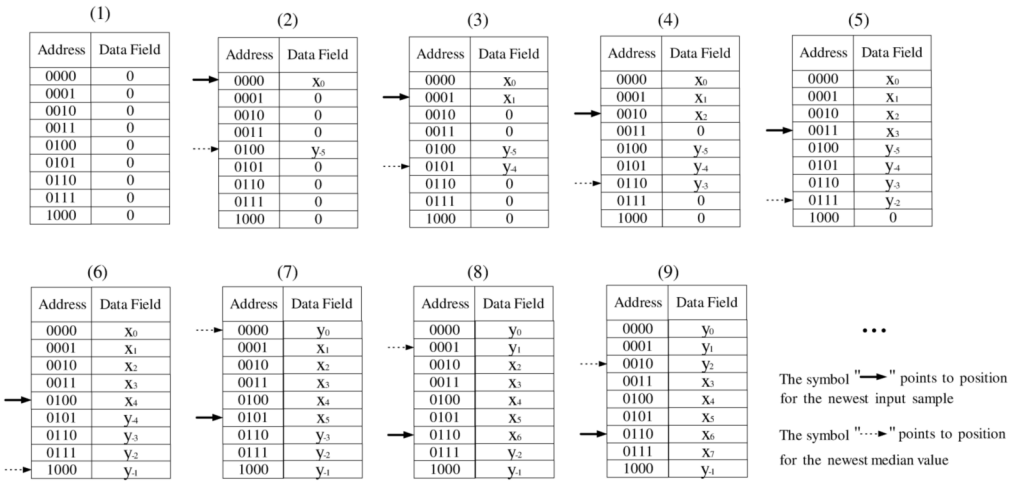


Fig. 13.
The flow for data storage of the 1-D RMF.

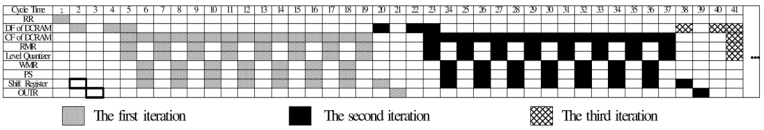


Fig. 14.
Reservation table of the 1-D RMF.

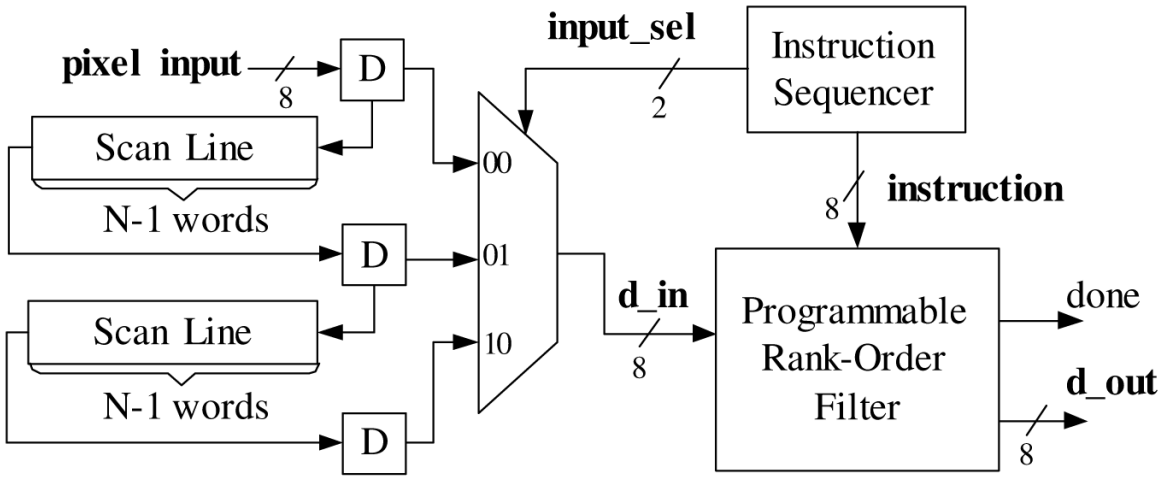


Fig. 15.
Block diagram of the 2-D non-recursive ROF with 3-by-3 window.

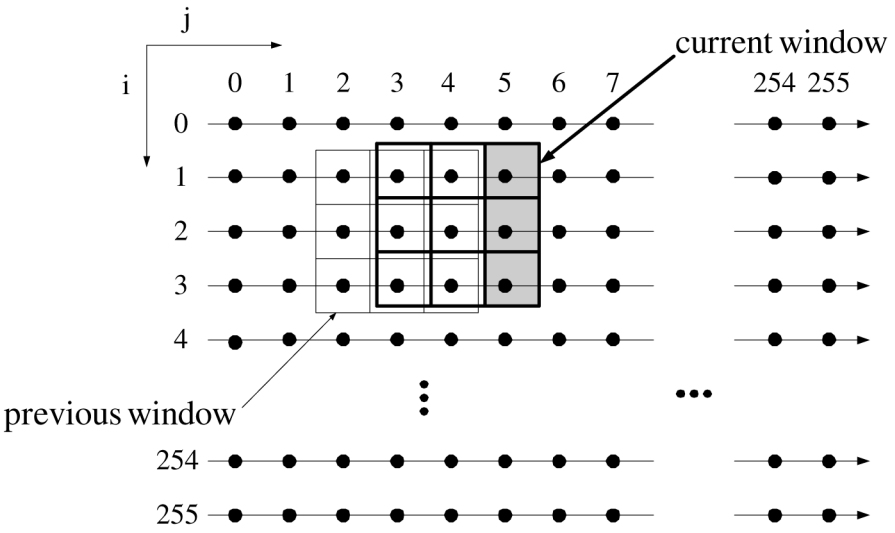


Fig. 16.
The windowing of the 3 × 3 non-recursive ROF.

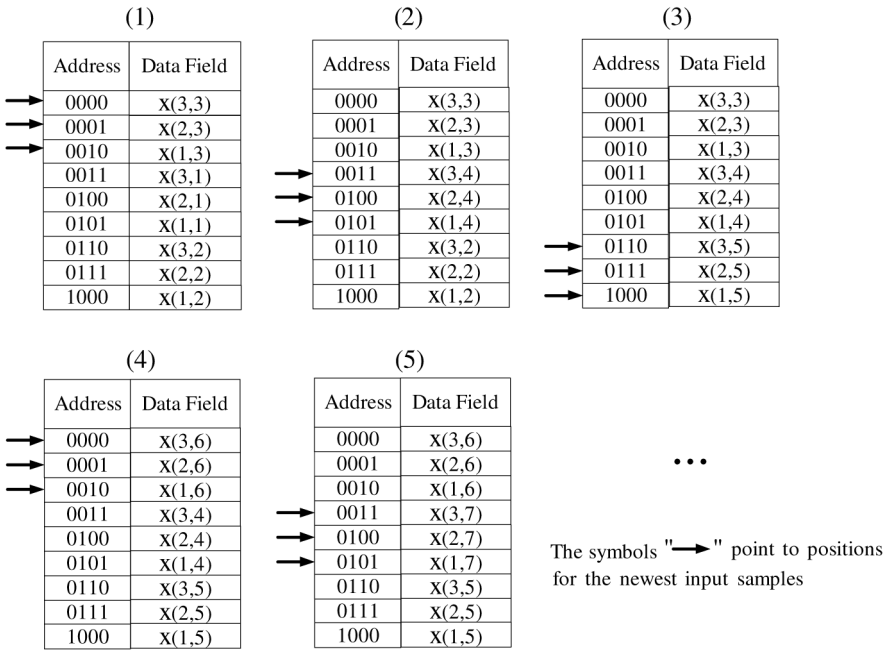


Fig. 17.
The data storage of the 2-D non-recursive ROF.

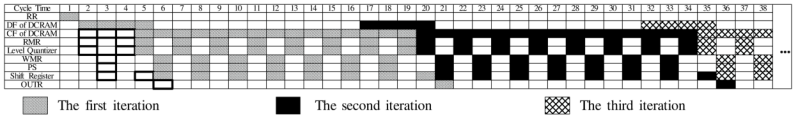


Fig. 18.
Reservation table of the 2-D ROF.

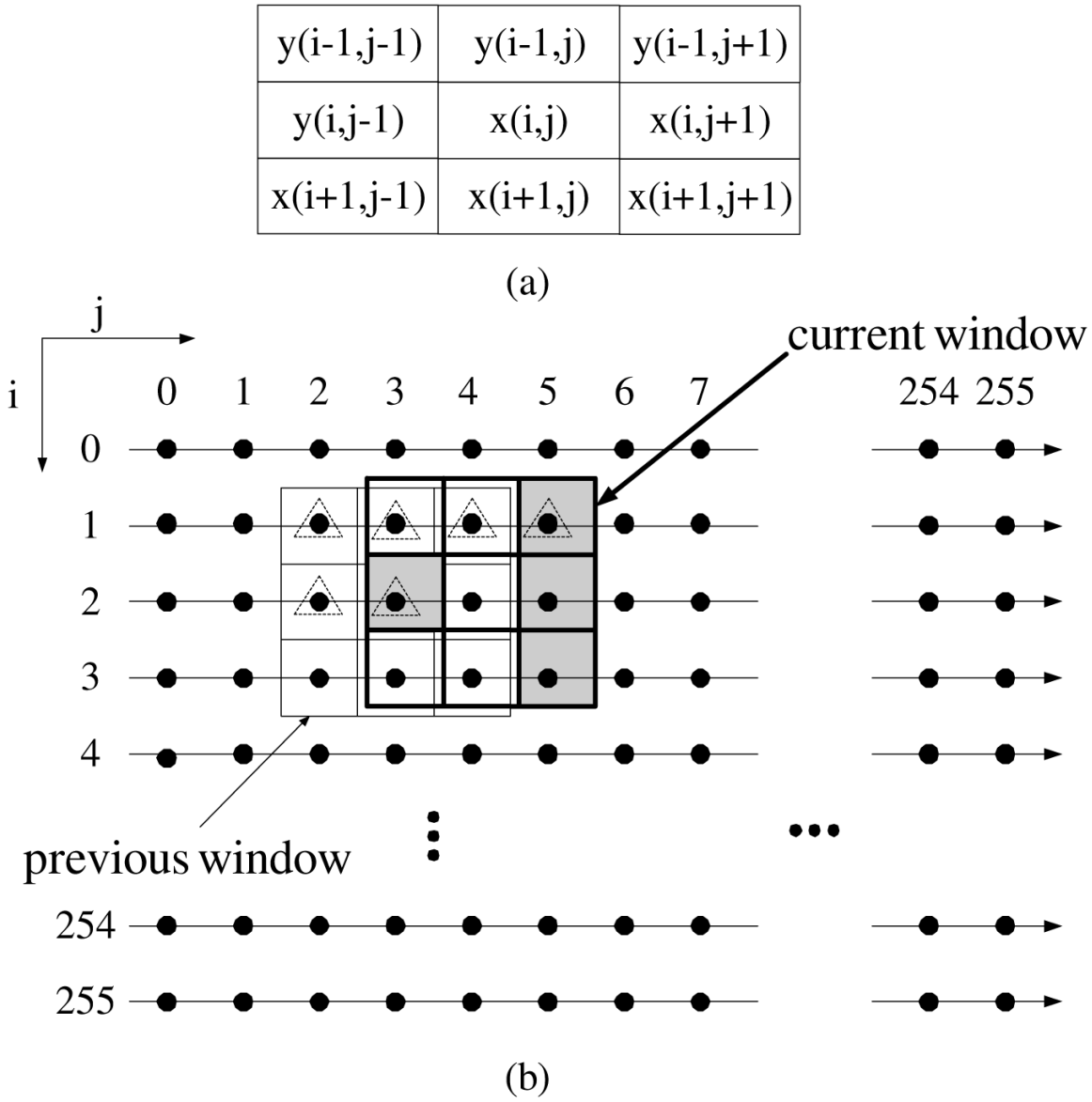


Fig. 19.
(a) The content of the 3 × 3 window centered at (*i*, *j*). (b) The windowing of the 2-D RMF.

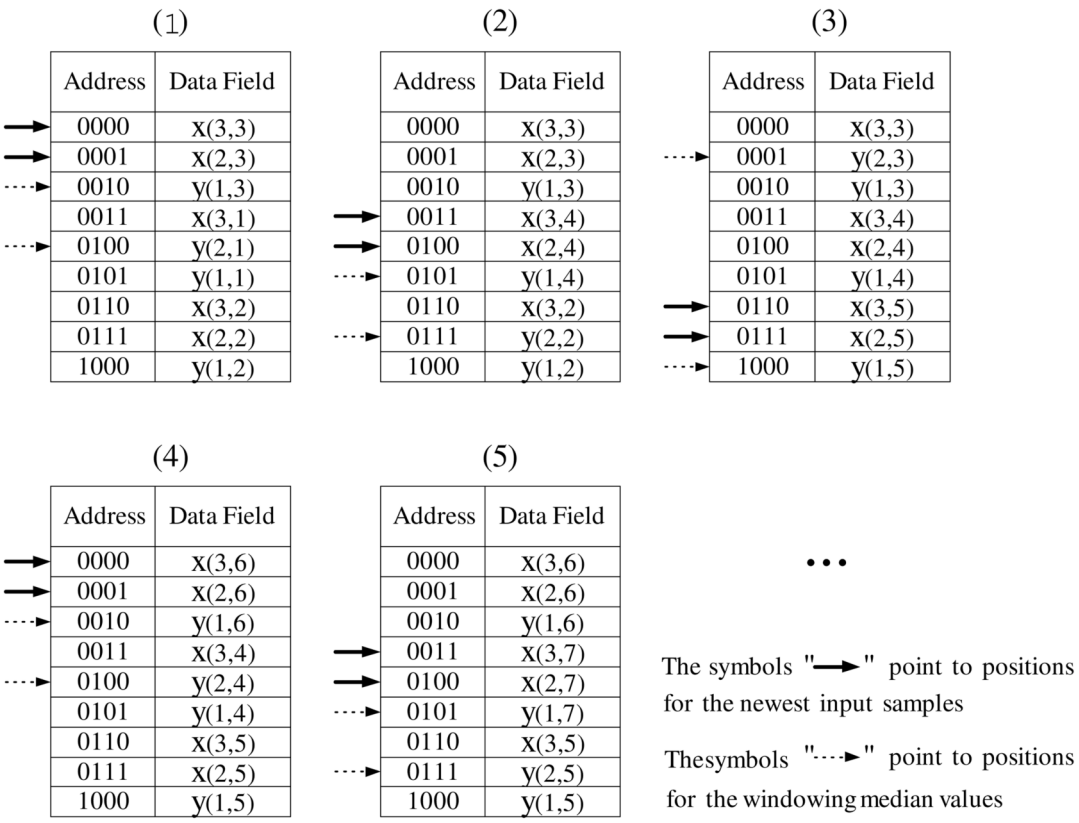


Fig. 20.
The data storage of 2-D RMF.

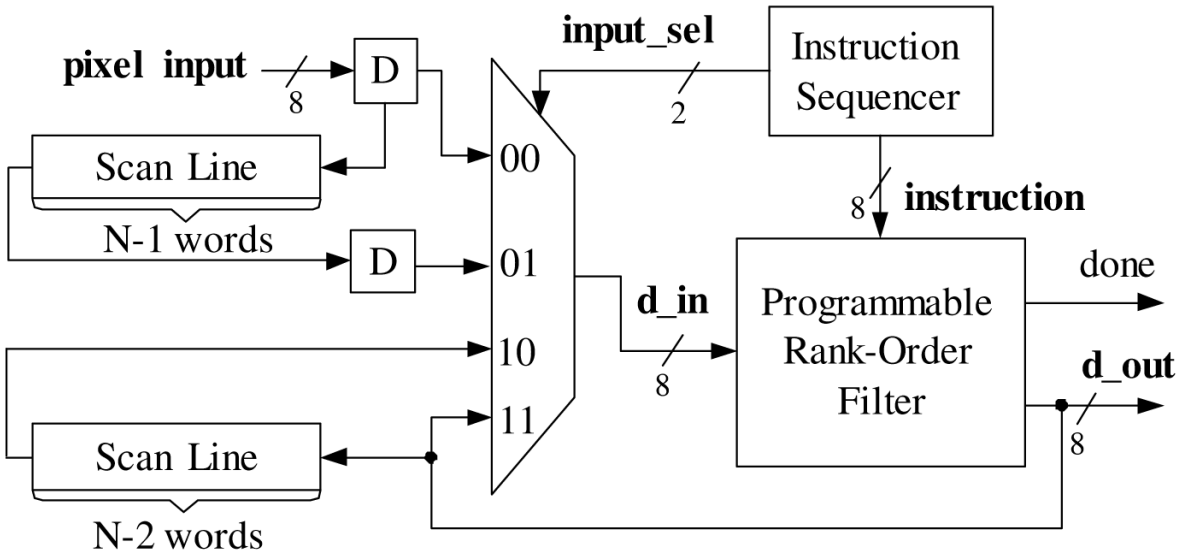


Fig. 21.
Block diagram of the 2-D RMF with 3-by-3 window.

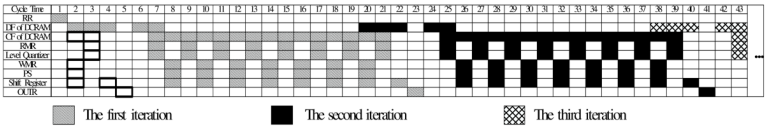


Fig. 22.
Reservation table of optimal pipeline for 2-D recursive median filter.

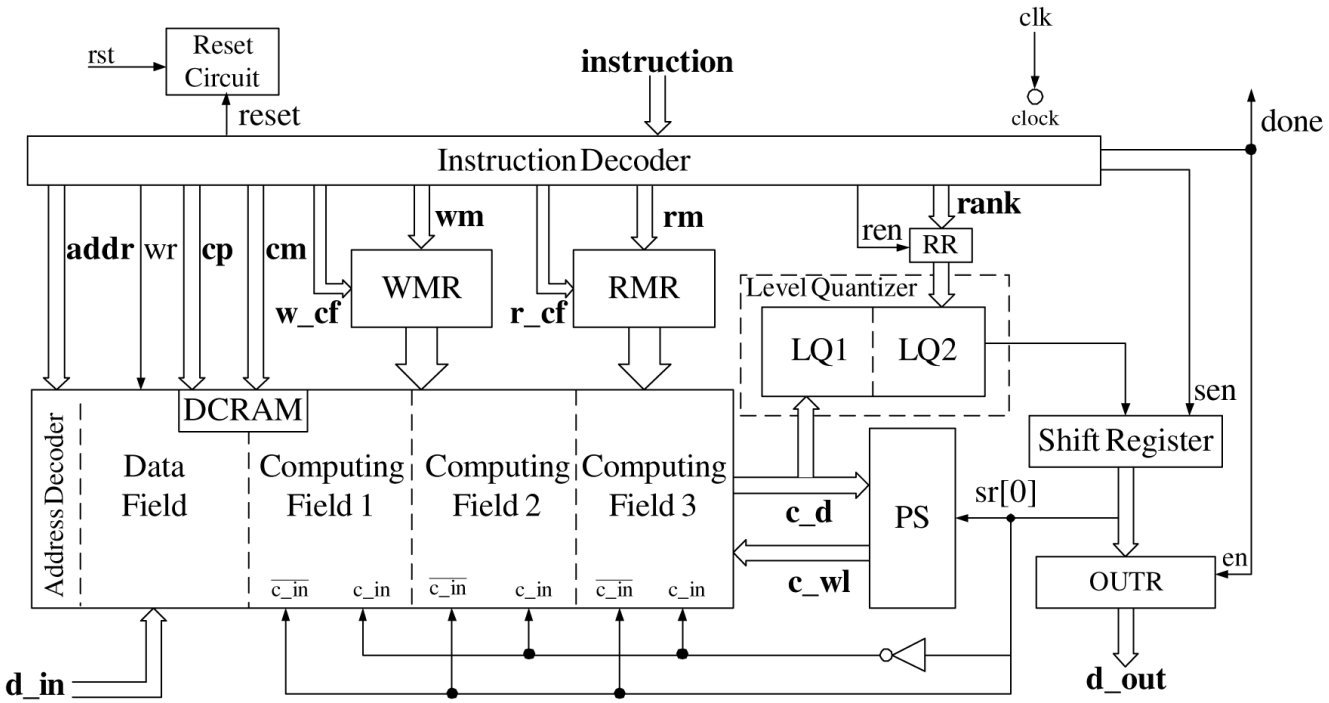


Fig. 23.
The fully-pipelined ROF architecture.

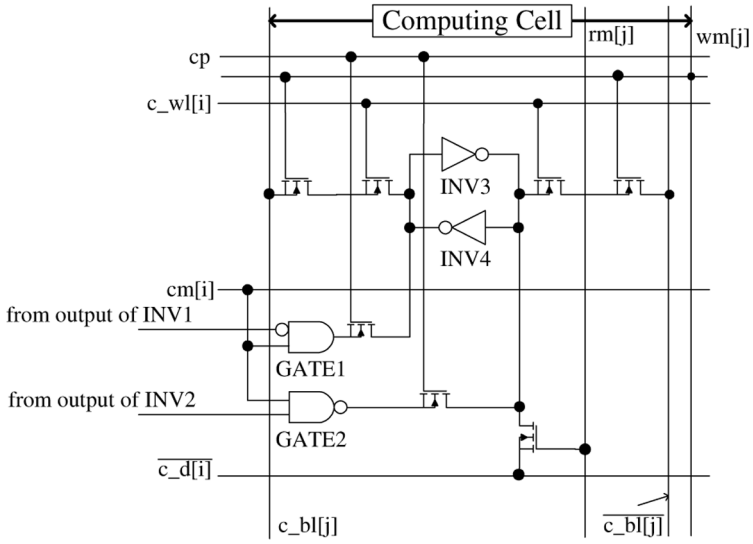


Fig. 24.
A modified circuit of computing cell for fully-pipelined ROF.

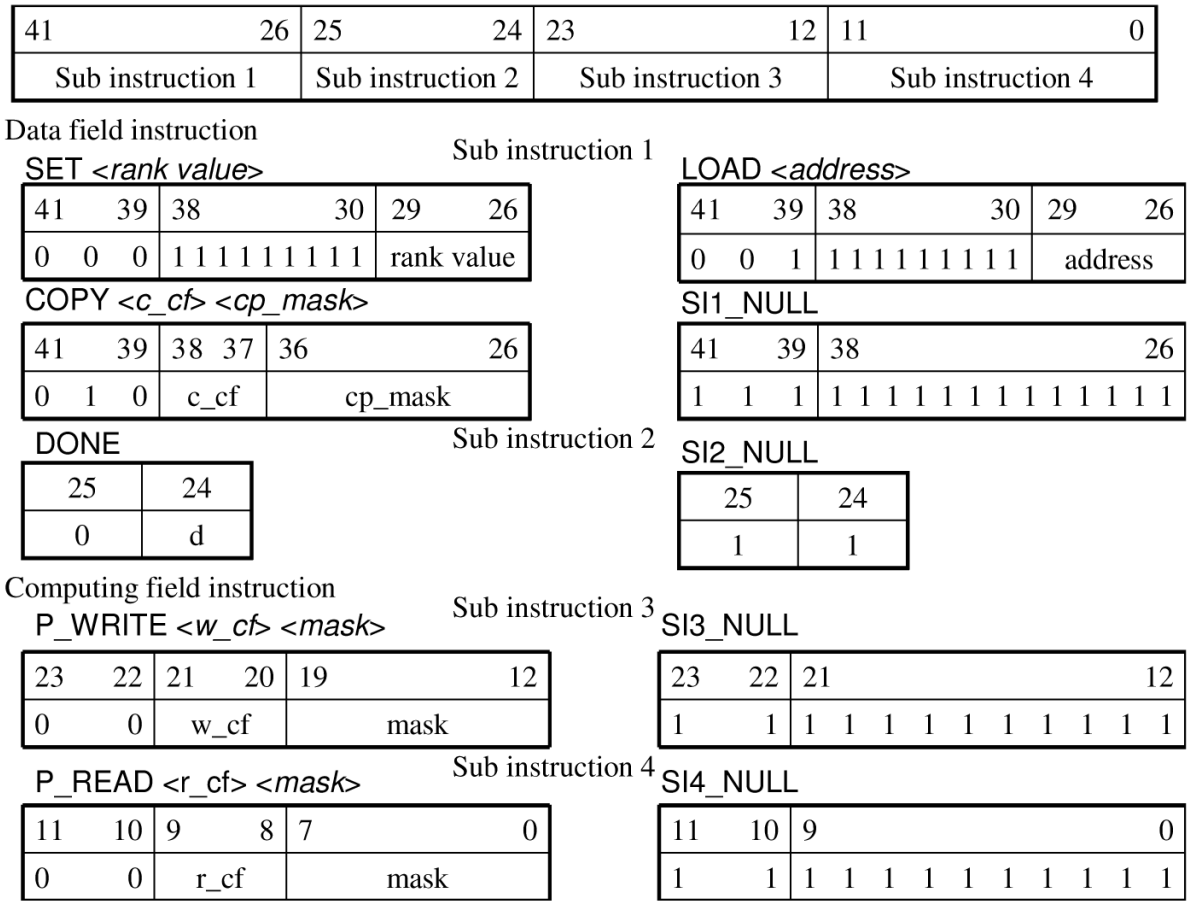


Fig. 25.
The format of the extended instruction set for the fully-pipelined ROF architecture.

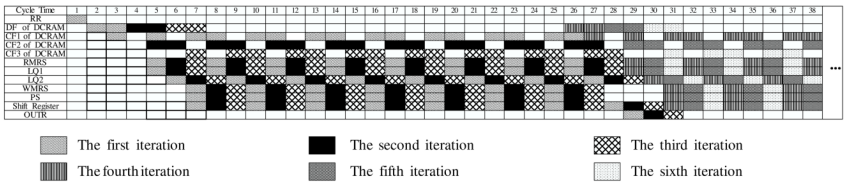
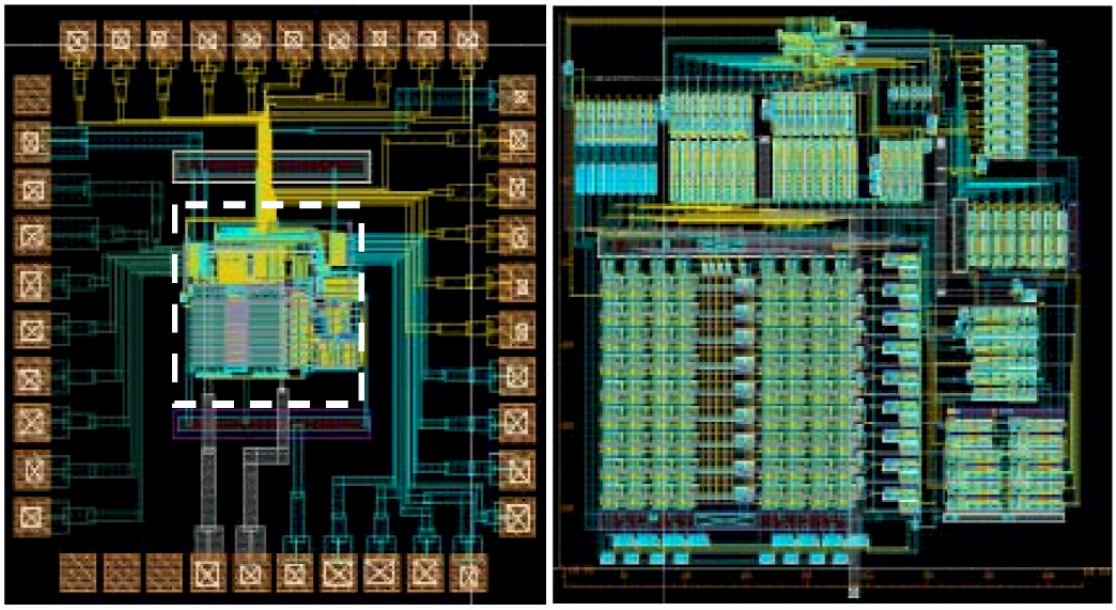


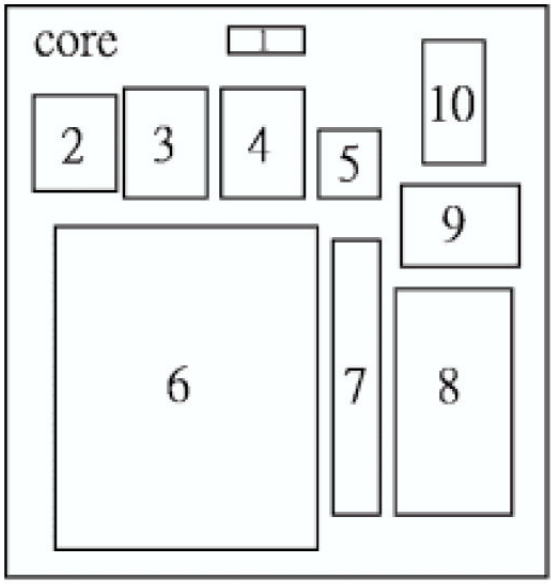
Fig. 26.
Reservation table of the 1-D non-recursive ROF for fully-pipelined ROF architecture.

Fig. 27.
Simulation results of a 2-D ROF application. (a) The noisy “Lena” image corrupted by 8% of impulsive noise. (b) The “Lena” image processed by the 3 × 3 4th-order filtering. (c) The “Lena” image processed by the 3 × 3 5th-order filtering. (d) The “Lena” image processed by the 3 × 3 6th-order filtering.

Fig. 28.
Simulation results of a 2-D RMF application. (a) The noisy “Lena” image corrupted by 9% of impulsive noise. (b) The “Lena” image processed by the 3 × 3 RMF.



(a) (b)



(c)

Fig. 29.

The result of chip design using TSMC 0.18um 1P6M technology. (a) The chip layout of proposed rank-order filter. (b) The core of the proposed ROF processor. (c) The floorplan and placement of (b). (1: Instruction decoder; 2: Reset circuit, 3: WMR, 4: RMR, 5: RR, 6: DCRAM, 7: PS; 8: Level Quantizer; 9: Shift Register; 10: OUTR.)

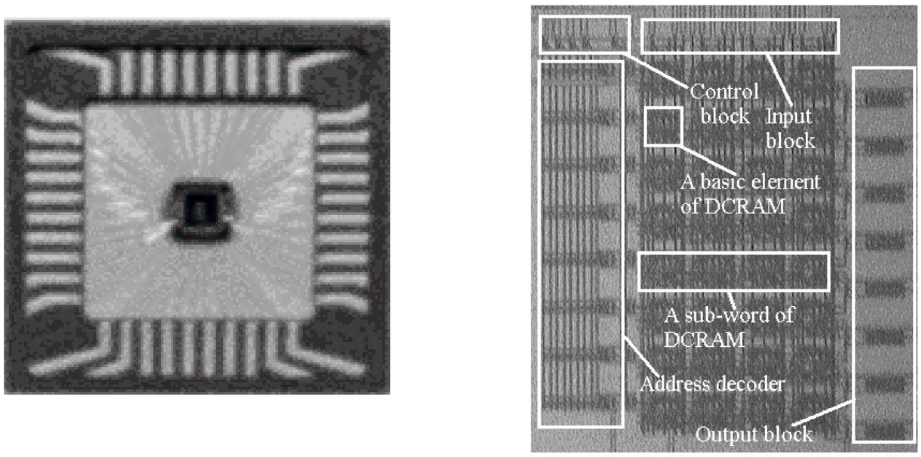


Fig. 30.
(a) The micrograph of DCRAM chip. (b) The layout of the DCRAM chip.

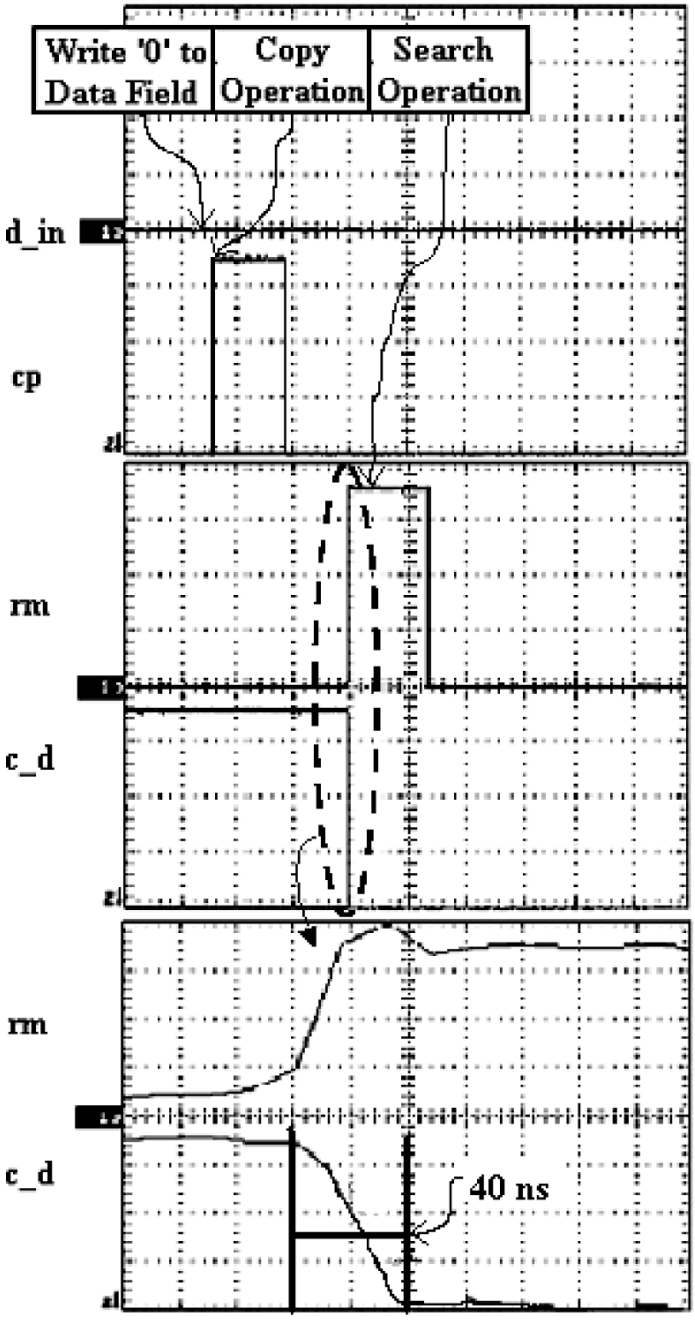


Fig. 31.
Measured waveform of the DCRAM chip.

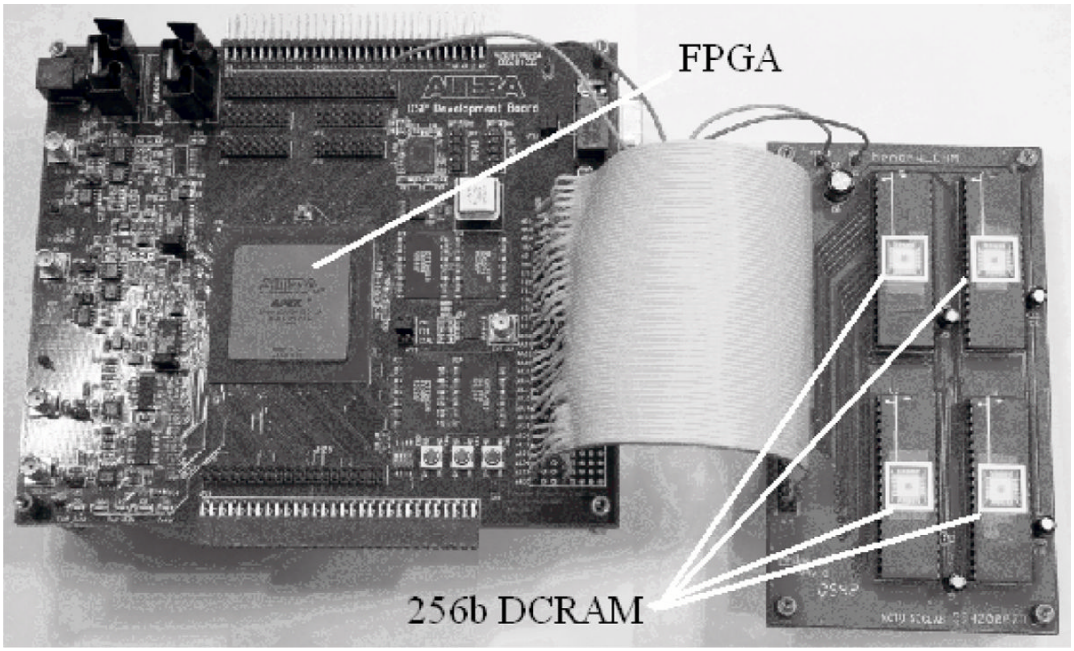


Fig. 32.
The system prototype of rank-order filtering processor.

Table 1

Timing analysis of the proposed ROF processor.

Path	Description	1.8V supply	3.3V supply
1	From the output of RR to the input of the shift register.	1.2 ns	0.78 ns
2	From the output of RMR, thru DCRAM to the input of the PS.	1.8 ns	1.1 ns
3	From the output of RMR, thru DCRAM and the Level-Quantizer, to the input of the shift register.	3.9 ns	3.44 ns
4	From the shift register, thru the inverter connected to "c_in", to the SRAM cell of the computing field.	3.02 ns	1.96 ns
5	From "**d_in**" to the SRAM cell of the data field.	3.05 ns	1.85 ns
6	From the SRAM cell of the data field to the SRAM cell of the computing field.	1.24 ns	1.09 ns