4

Image Enhancement for Gastrointestinal Endoscopy

In Chapter 3, the low-power image encoders have been successfully proposed for gastrointestinal(GI) endoscopes. GI images shot can be encoded and then transmitted to the storage device through the RF transmitter for the treatment of a patient. A encoded bitstream can be decoded and restored a color image using the color interpolation technique before a professional doctor begins to diagnose. All decoded GI images can be very suitable for practical diagnosis according to analysis results of Subsection 3.4.2; however, these decoded GI images could be further accomplished the assignment of image enhancement due to the interference of impulsive noise caused by a faulty sensor during image acquisition and the visual-quality degradation during the lossy image compression processing. the former can efficiently use the noise reduction technique to suppress the impulsive noise and the later can also use the sharpness technique to enhance a edge gradient of decoded GI image validly.

Standard linear filtering methods are conceptually pleasing and extremely useful in the field of image enhancement while uniformly smoothing images are little or no sharp grey level variation before the addition of corruptive influence. A linear smoother may indeed prove to be very effective. However, our world is nature and particularly contains objects,

lines, and boundaries whose two-dimensional projections onto photographic films and other recording media evokes sharp grey-level variations along boundaries, which may be regarded as edges. It is clear that edges are an important factor in the subjective evaluation of an image and the human visual system (HVS) has been noted to contain edge-sensitive elements. An unfortunate property of linear-type smoothers that effectively remove high-frequency noise components is the accompanying blurring of information-bearing edges and loss of detail due to the band-limiting nature of such filters. Conversely, a linear high-pass filter may preserve or even accentuate edge-type signals, but such filters are extremely noise-sensitive, and tend to suppress smooth features which correspond to real-world surfaces and regions of slowly varying reflectance.

Therefore, if we require a filtering method which will effectively remove high-frequency noise components while simultaneously preserving the edge structure of an image, we must resort to nonlinear techniques. In the class of nonlinear technique, rank-order filtering (ROF), or order-statistical filtering, has been widely applied for various speech and image processing applications [25]-[31]. Unlike linear filtering, the ROF can remove sharp discontinuities of small duration without blurring the original signal. Most of recent works in nonlinear rank-order filters have focused on the problem of noise suppression [32]-[44]. Some rank-order-based sharpeners have been proposed [45]-[48]. However, these rank-order filters mentioned above do not have abilities of noise suppression and sharpness simultaneously. To overcome these problems, paper [49] has successfully proposed a new class of rank-order-based filter, lower-upper-middle (LUM) filter and the LUM filter has two parameters, one for noise suppression and the other for sharpness. Varying the parameter of noise suppression can change the level of smoothing from no smoothing to median filtering and varying the parameter of sharpness also have excellent detail-preserving characteristics to avoid shortcomings of conventional linear filters, the ringing effect and the amplification of background noise. Different values of parameters will cause different degree of smoothing and sharpening for the LUM filter, hence a flexible and configurable hardware of the LUM filter should be developed to satisfy the demands of users.

Next, Section 4.1 introduces definitions of LUM filters in detail and also respectively analyze the performance using the LUM smoother and the LUM sharpener for GI images.

Section 4.2 proposes a novel rank-order filtering design, the main core of the LUM filter and this novel design first uses the dual-cell random-access memory (DCRAM) to implement the generic bit-sliced rank-order filtering algorithm. Section 4.3 shows the basic non-recursive/recursive rank-order-filtering for 1-D/2-D applications. In addition, Section 4.4 further extends the ROF architecture to a fully-pipelined version at the expense of area in order to increase the schedulability. Section 4.5 implements the chip design. Finally, Section 4.6 compares the existing ROF architecture with our proposed design.

4.1 The LUM Filters

In this section, the LUM filters are defined. First a subclass of LUM filter that referred to as the LUM smoother is defined in Subsection 4.1.1 and then a second subclass of LUM filter, the LUM sharpener, which can enhance edges is also defined in Subsection 4.1.2. Finally, the general class of LUM filter that includes the LUM smoother and the LUM sharpener as a special case is introduced in Subsection 4.1.3 and the LUM filter can simultaneously enhance edges and suppress impulsive noise itself. In the beginning, we consider a window function (W) containing a set of N samples centered about the sample x^* before defining the LUM filters. This set of observations will be denoted $W = \{x_{(1)},\ x_{(2)},\ ...,\ x_{(n)}\}$, in which, N is assumed to be odd. For a 2-D image, let the window be a simple $(2m+1) \times (2m+1) = N$ square. The rank-ordered set can be written as $x_{(1)} \leq x_{(2)} \ldots x_{(N)}$.

4.1.1 The Definition of LUM Smoother

The LUM smoother is defined below. The output of the LUM smoother with parameter k is given by

$$y^* = med\{x_{(k)},\ x^*,\ x_{(N-k+1)}\} \tag{4.1.1}$$

in which, $1 \leq k \leq (N+1)/2$. Thus, the output of the LUM smoother is $x_{(k)}$ if $x^* < x_{(k)}$. If $x^* > x_{(N-k+1)}$, then the output of the LUM is $x_{(N-k+1)}$. Otherwise the output of the LUM smoother is simply x^*.

The reasoning behind comparing the middle sample x^* to the lower- and upper-order statistics is that these order statistics form a range of normal-valued samples. If x^* lies in this range it is not modified. If x^* lies outside this range it is replaced by a sample that lies closer to the median. This creates a smoothing function. For example, if x^* is an impulse it is likely to fall outside the range of the upper- and lower-order statistics. It would then be replaced with a value closer to that of the median, and the outlier would be removed. The tuning parameter k controls the smoothing characteristics of the filter and can be adjusted to best balance the tradeoffs between noise smoothing and signal-detail preservation. Note that for $k = (N+1)/2$ the output of the LUM smoother is the median of W, and this is the maximum amount of smoothing that can be performed. As k is decreased the filter exhibits improved detail-preserving characteristics; when $k = 1$, the LUM smoother becomes an identity filter (i.e., $y^* = x^*$).

4.1.2 The Definition of LUM Sharpener

The LUM smoother and other rank-order smoothers obtain smoothing characteristics by shifting samples toward the median. To obtain sharpening characteristics, samples must be moved away from the median to more extreme order statistics. This is how the LUM sharpener operates. Before defining the LUM sharpener, let us first define a value centered between the lower- and upper-order statistics, $x_{(l)}$ and $x_{(N-l+1)}$. This midpoint or average, denoted t_l , is given by

$$t_l = (x_{(l)} + x_{(N-l+1)})/2 \tag{4.1.2}$$

We define the LUM sharpener as follows. The output of the LUM sharpener with parameter l is given by

$$y^*(n) = \begin{cases} x_{(l)}, & if \quad x_{(l)} < x^* \leq t_{(l)} \\ x_{(N-l+1)}, & if \quad t_{(l)} < x^* < x_{(N-l+1)} \\ x^*, & otherwise. \end{cases} \tag{4.1.3}$$

Thus, if $x_{(l)} < x^* < x_{(N-l+1)}$ then x^* is shifted outward to $x_{(l)}$ or $x_{(N-l+1)}$ according to which is closest to x^*. Otherwise the sample x^* is unmodified. The reasoning behind this sharpening operation is that if $x_{(l)} < x^* < x_{(N-l+1)}$ then x^* is interpreted as being

a transition sample in a slope region. By shifting it to an extreme-order statistic we are removing the transition points and creating a steeper slope and in many cases an ideal step edge. By changing the parameter l, various levels of sharpening can be achieved. In the case where $x_{(l)} = (N+1)/2$, no sharpening occurs and the LUM sharpener is simply an identity filter. In the case where $x_{(l)} = 1$, a maximum amount of sharpening is achieved since x^* is being shifted to one of the extreme-order statistics $x_{(1)}$ or $x_{(N)}$.

4.1.3 The Definition of LUM Filter

To obtain an enhancing filter that is robust and can reject outliers, the philosophies of the LUM smoother and LUM sharpener must be combined. This leads us to the general LUM filter. Before we define this filter, let us first define a lower and upper statistic as follows

$$x^L = med\{x_{(k)},\ x^*,\ x_{(l)}\} \tag{4.1.4}$$

$$x^U = med\{x_{(N-k+1)},\ x^*,\ x_{(N-l+1)}\} \tag{4.1.5}$$

where $1 \le k \le l \le (N+1)/2$. Note that $x^L \le x^U$. The output of the general LUM filter is given by the statistic x^L or x^U that is closest to the center sample x^*. The parameters k and l can be considered tuning parameters that allow the LUM filter to have a variety of characteristics. More will be said below about these parameters. Two mathematically equivalent, but slightly different, definitions of the LUM filter are given below. Eq.4.1.7 is perhaps the most direct.

$$y^*(n) = \begin{cases} x^L, & if \quad x^* \le (x^L + x^U)/2 \\ x^U, & otherwise. \end{cases} \tag{4.1.6}$$

$$y^*(n) = \begin{cases} x_{(k)}, & if \quad x^* < x_{(k)} \\ x_{(l)}, & if \quad x_{(l)} < x^* \le t_{(l)} \\ x_{(N-l+1)}, & if \quad t_{(l)} < x^* < x_{(N-l+1)} \\ x_{(N-k+1)}, & if \quad x_{(N-l+1)} < x^* \\ x^*, & otherwise. \end{cases} \tag{4.1.7}$$

where $t_{(l)}$ is the midpoint between $x_{(l)}$ and $x_{(N-l+1)}$ defined in Eq.4.1.2

Regardless of how it is implemented, the LUM filter performs a relatively simple function. The LUM outputs the sample $x_{(k)}$ if $x^* < x_k$. Similarly, if $x^* > x_{(N-k+1)}$ the output of the LUM filter is $x_{(N-k+1)}$. Thus, just like in the LUM smoother, the samples $x_{(k)}$ and $x_{(N-k+1)}$ form bounds on the LUM-filter output. On the other hand, if $x_{(l)} < x^* < x_{(N-l+1)}$ then the center sample is shifted outward to $x_{(l)}$ or $x_{(N-l+1)}$ according to which of these lies closest to x^*, as in the LUM sharpener. If x^* lies such that $x_{(k)} \leq x^* \leq x_{(l)}$ or $x_{(N-l+1)} \leq x^* \leq x_{(N-k+1)}$ then the sample is unmodified by the filtering operation. By manipulating the parameters k and l, the LUM filter takes on a variety of characteristics. In addition, the LUM filter can become the LUM smoother while $l = (N+1)/2$ and k is varied and the LUM filter can become the LUM sharpener while $k = 1$ and l is varied.

4.1.4 Impulsive Noise Reduction and Image Sharpness for Gastrointestinal Images

In this subsection, we firstly illustrate the impulsive noise reduction characteristic of the LUM filter. The impulsive noises are caused by a faulty sensor during image acquisition. Fig.4.1(a) and Fig.4.1(b) show original mouth cavity pictures of a pick and each picture has a resolution of 256×256. Fig.4.1(c) shows the output of the LUM smoother with a 3×3 window and parameter k=2 and l=5. Comparing Fig.4.1(c) with Fig.4.1(e), the LUM smoother subjectively has better image quality than 3×3 median filtering under the same ability of impulsive noise reduction. Similarly, Comparing Fig.4.1(d) with Fig.4.1(f), the LUM smoother has also better image quality than 3×3 median filtering with the same smoothing. According to simulation results of Fig.4.1(c) and Fig.4.1(d), different parts of gastrointestinal images need to use different parameters of the LUM smoother to eliminate the impulsive noise.

In addition to the impulsive noise reduction characteristic of the LUM filter, we next illustrate the sharpening characteristic of the LUM filter. The decoded GI images need to be sharpened due to the visual-quality degradation during the lossy image compression processing. Fig.4.2 (a) and Fig.4.2 (c) are respectively the second and the fourth decoded GI images and their image quality suffers tiny quality degradation. Fig.4.2 (b) shows the

output of the LUM sharpener with a 3×3 window and parameter k=1 and l=1. Comparing Fig.4.2(a) with Fig.4.2(b), the LUM sharpener can efficiently emphasize the image sharpness to improve the image quality. Similarly, Comparing Fig.4.2(c) with Fig.4.2(d), the LUM sharpener can also efficiently emphasize the image sharpness to improve the image quality. According to simulation results mentioned above, different parts of gastrointestinal images also need to use different parameter of the LUM sharpener to accomplish the sharpening.

To sum up, different parameters of the LUM filter can significantly cause different degree of smoothing and sharpening according to the demand of a user. Therefore, a flexible and configurable hardware of the LUM filter will be proposed in the next subsection.

4.1.5 The Proposed LUM Filtering Processor

Fig.4.3 illustrates the conceptual diagram of the LUM filtering processor. The instruction sequencer is used for the generation of instruction codes and the control of input/output streams. Two parameters of LUM filter, l and k, can be set to generate desired ranks through their rank generators by users and then these ranks sequentially enter the circuit of rank-order filtering (ROF). The ROF is the mainly operational core in the LUM filtering processor and it can generate the correct rank value according to the driving of instruction codes and the current rank. In order to let the LUM filtering itself can use different kind of rank values to adaptively accomplish tasks of smoothing and/or sharpening, we need a highly flexible ROF hardware to select suitable rank values into the comparison circuit and the output of the LUM filtering can be generated after the comparison rule shown in Eq.4.1.7.

In order to design the hardware of ROF that has highly flexible and configurable abilities, Subsection 4.2 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. The design procedure of the proposed ROF

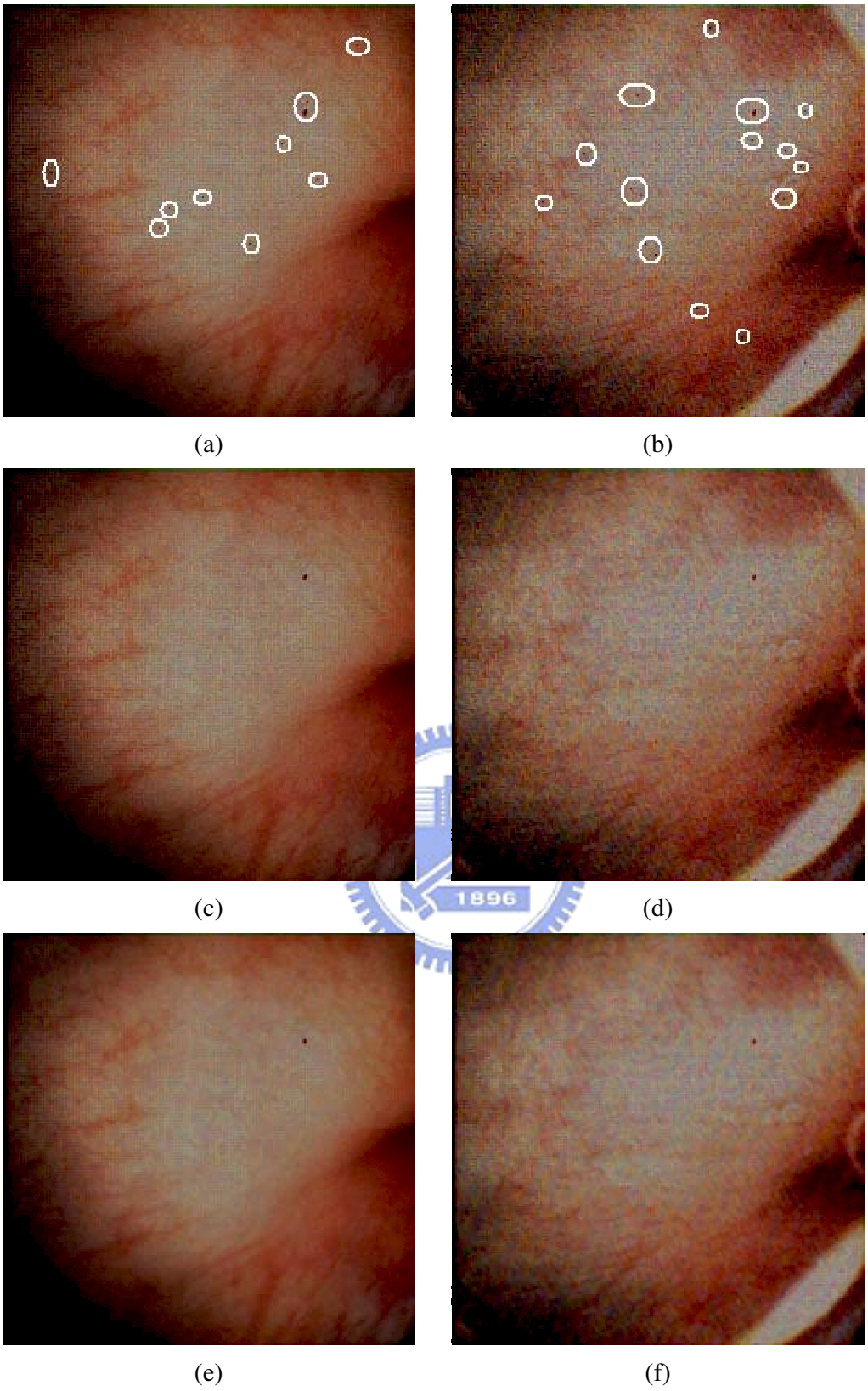


Figure 4.1. (a) The 1st picture of mouth cavity for a pick. (b) The 2nd picture of mouth cavity for a pick. (c) Output of the 3x3 LUM smoother where k=2 and l=5. (d) Output of the 3x3 LUM smoother where k=3 and l=5. (e) Output of the 3x3 median filter. (f) Output of the 3x3 median filter.

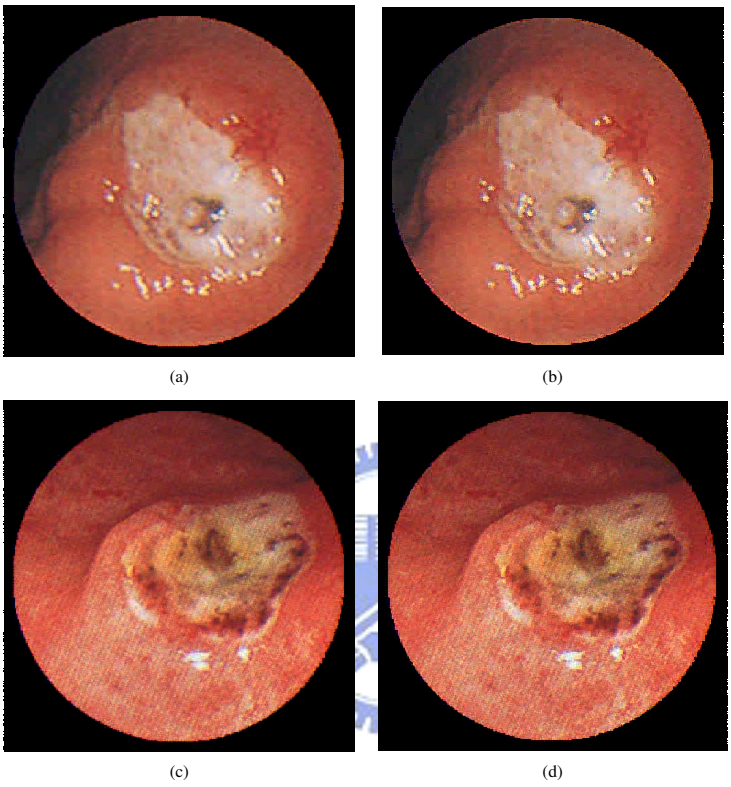


Figure 4.2. (a) The 2nd decoded GI image. (b) Output of the 3x3 LUM sharpener where k=1 and l=1. (c) The 4th decoded GI image. (d) Output of the 3x3 LUM smoother where k=1 and l=2.

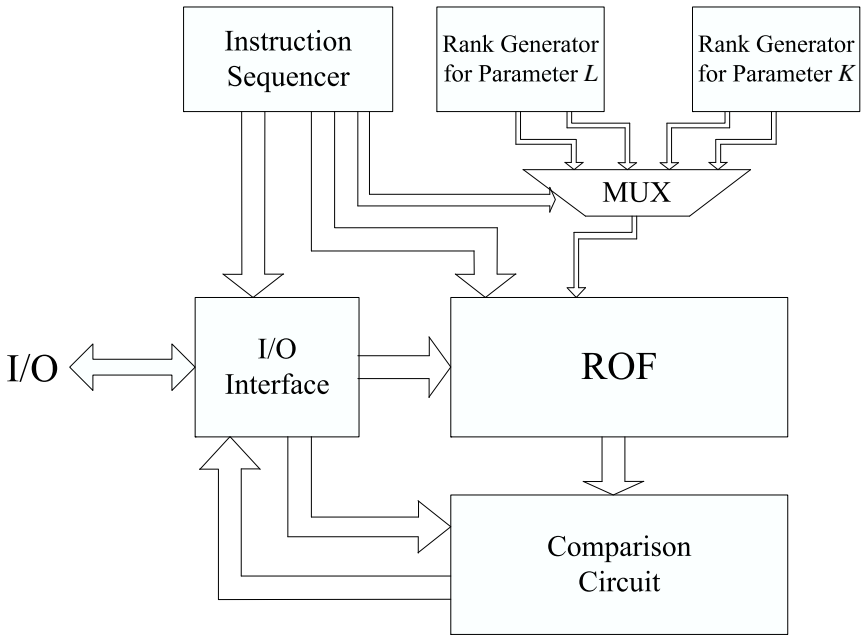


Figure 4.3. The conceptual diagram of LUM filtering processor.

hardware will be described in Subsection 4.2 in detail.

4.2 Design of Rank-Order Filtering Using The Dual-Cell RAM Architecture

Given a sequence of input samples $\{x_{i-k}, x_{i-k+1}, ..., x_i, ..., x_{i+l}\}$, 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 $\{y_{i-k}, y_{i-k+1}, ..., y_{i-1}, x_i, ..., x_{i+l}\}$. 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.

Many approaches for hardware implementation of rank-order filtering have been pre-

sented in the past decades [50]-[65]. Many of them are based on sorting algorithm [52], [63], [64], [66]-[69]. They considered the operation of rank-order filtering as two steps: sorting and choosing. Papers [51], [60] 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 [50], [53], [55], [56], [70]-[73] 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 [72] further presents a micro-programmable processor for the implementations of the median-type filters. Paper [61] 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 [53], [68] present solutions for 2-D rank-order filtering at the expense of area.

In addition to the sorting algorithm, the paper [76] 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 [51], [54], [57]-[59], [62], [74], [75] employ the bit-sliced majority algorithm for median filtering, the most popular type of rank-order filtering. The bit-sliced algorithm [77], [78] 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 [51], [58] and [78] 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 [72]. Paper [62] 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 section 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 section 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. Except efficiently eliminate annoying impulsive noises and spot-points caused by a sensor

for a noisy GI image, the performance of the proposed processor also can process video clips of SVGA format in real-time image processing.

4.2.1 The Generic Bit-Sliced Rank-Order Filtering Algorithm

Let W_i=$\{x_{i-k}, x_{i-k+1}, ..., x_i, ..., x_{i+l}\}$ 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.4.4 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).

4.2.2 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.4.5. 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.

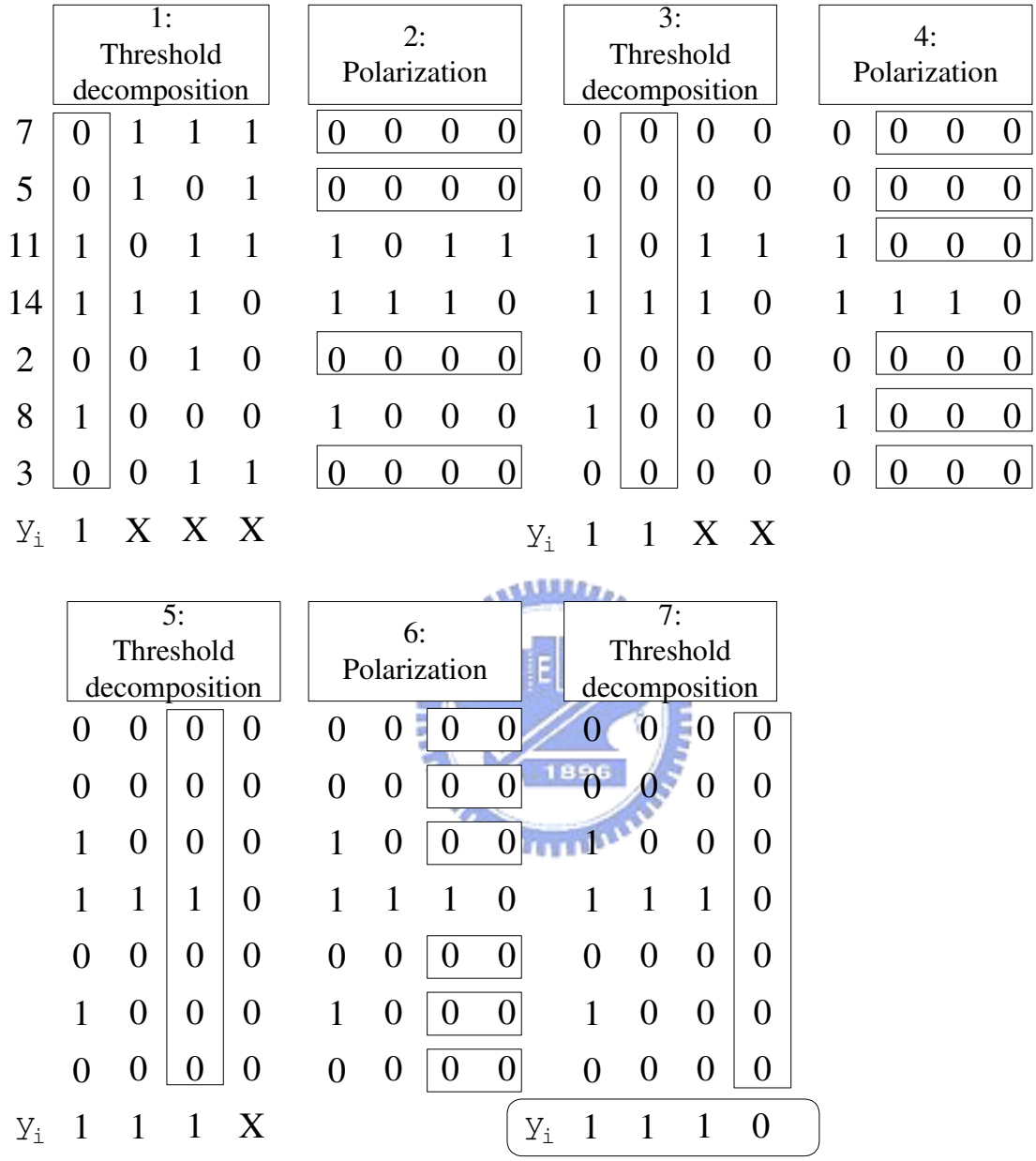


Figure 4.4. An example of the generic bit-sliced ROF algorithm for N=7, B=4, and r=1.

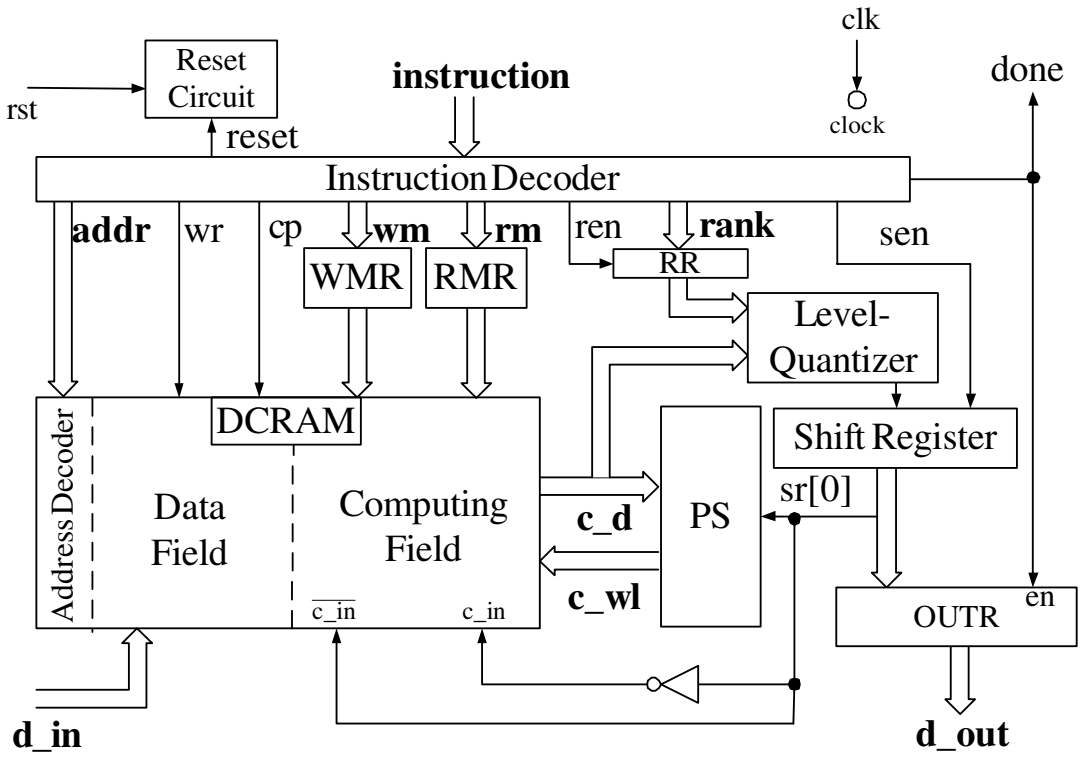


Figure 4.5. The proposed rank-order filtering architecture.

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 $\lceil log_2^N \rceil$. Fig.4.6 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 reformatted 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.4.2.8 shows the inequation of the comparison, where $\overline{r_S}$ denotes the one's complement of r_S and $\mathbf{1}$ denotes $(00\ldots01)_2$. Because the MSB of $Z_{b,S}$ is '0' and the MSB of $\overline{r_S}$ is '1', to satisfy Eq.4.2.8, 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

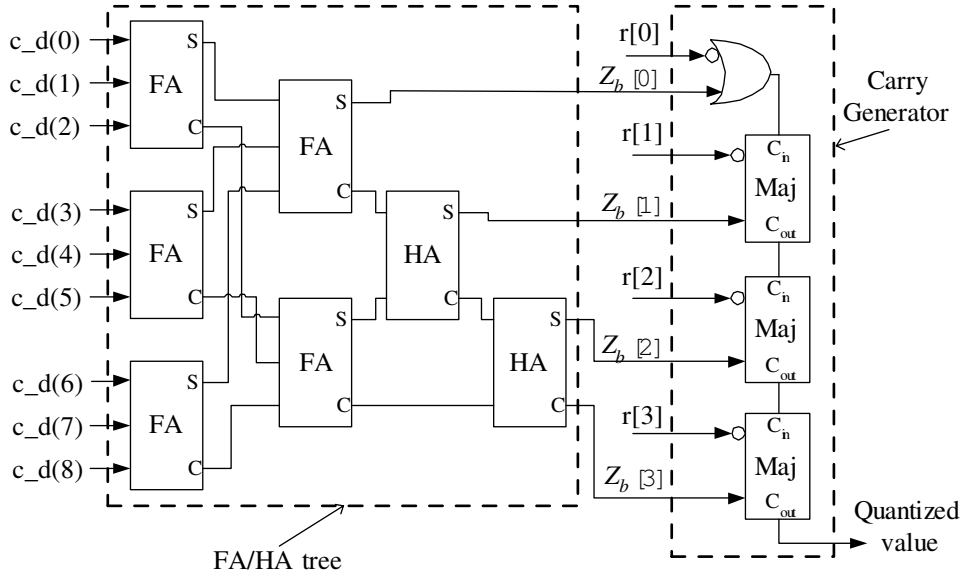


Figure 4.6. The block diagram of the Level-Quantizer.

the boolean function shown in Eq.4.2.9. Furthermore, we use an OR gate at the LSB stage because of Eq.4.2.10. 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.

$$\Delta = Z_{b,S} + \overline{r_S} + \mathbf{1} < 0. \tag{4.2.8}$$

$$Maj(A, B, C) = AB + BC + AC. \tag{4.2.9}$$

$$Z_b^0 \cdot \overline{r^0} + Z_b^0 \cdot 1 + 1 \cdot \overline{r^0} = Z_b^0 + \overline{r^0}. \tag{4.2.10}$$

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.4.7. Obviously, the XORs can examine the condition of $u_j^b \neq v_i^b$ 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_j^b which

is the inversion of v_i^b. Since v_i^b is the value of sr[0], we inverse the value of "sr[0]" to "c_in", as shown in Fig.4.5.

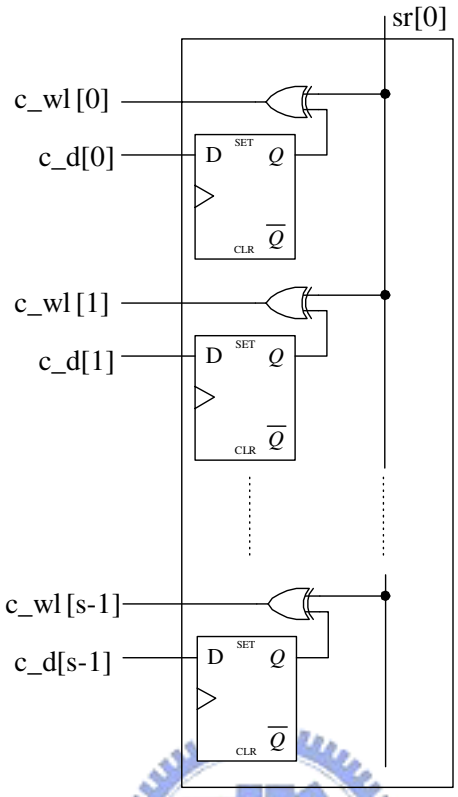


Figure 4.7. The polarization selector (PS).

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.

4.2.3 Implementation of Dual-Cell Random-Access Memory

Fig.4.8 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.4.9, 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 "$\overline{\text{c\_in}}$".

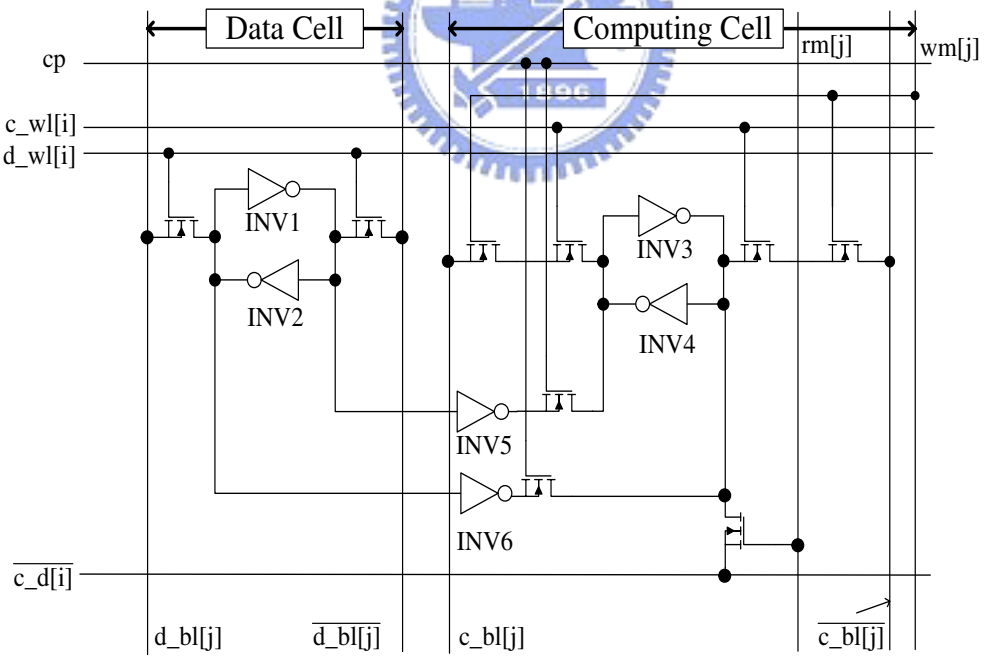


Figure 4.8. A basic element of DCRAM.

Fig.4.10 illustrates the implementation of DCRAM with the floorplan. Each $D_i - C_i$

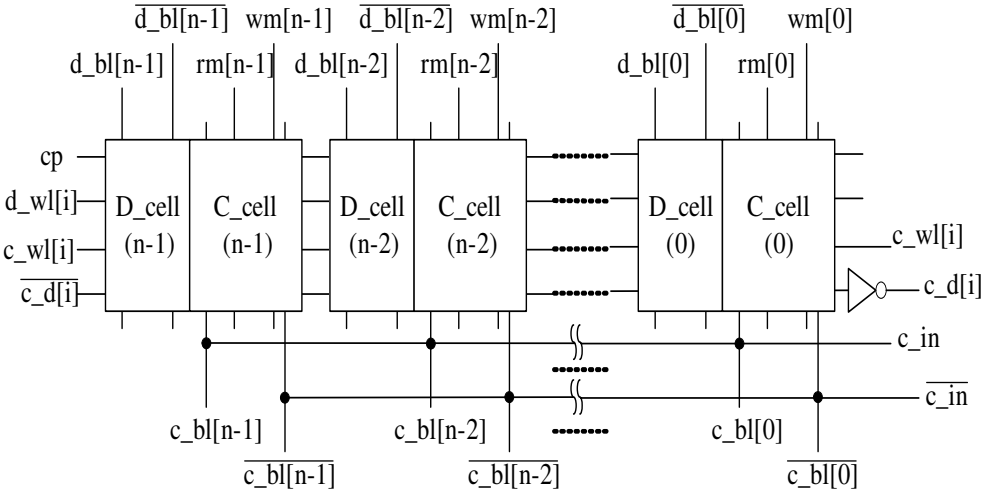


Figure 4.9. A DCRAM word mixing data field and computing field. D_cell(i) denotes the data field of i-th bit and C_cell(i) denotes the computing field of i-th bit.

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 [79]. 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.

4.2.4 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.4.11 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.4.12 lists the format of the instruction set. An instruction word contains two sub-words: 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

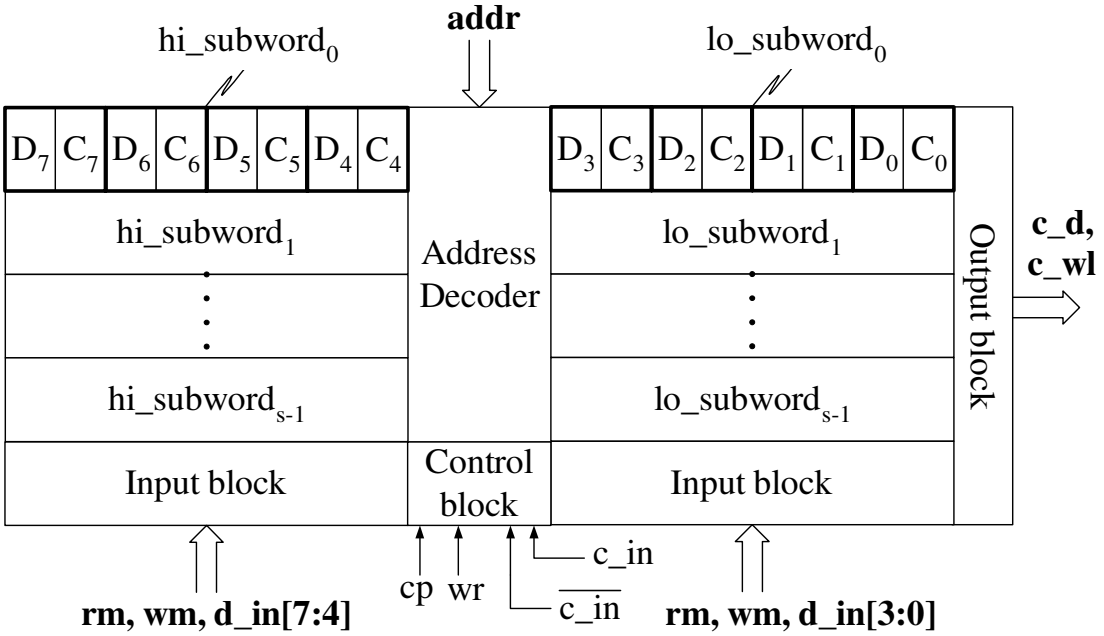


Figure 4.10. The floorplan of DCRAM.

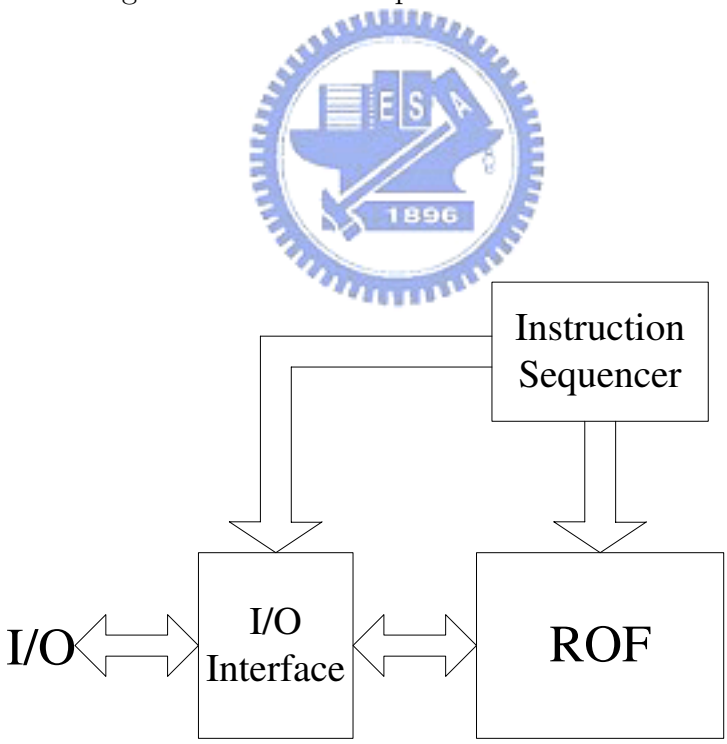


Figure 4.11. The conceptual diagram of the ROF processor.

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.

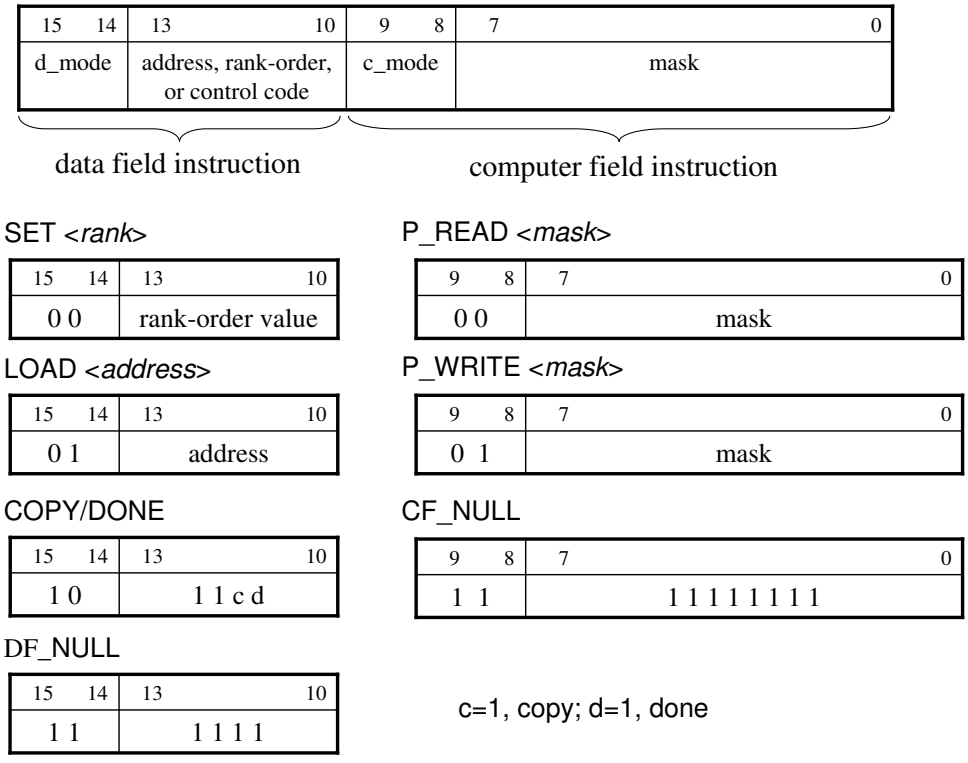


Figure 4.12. The format of the instruction set.

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.4.13. 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.

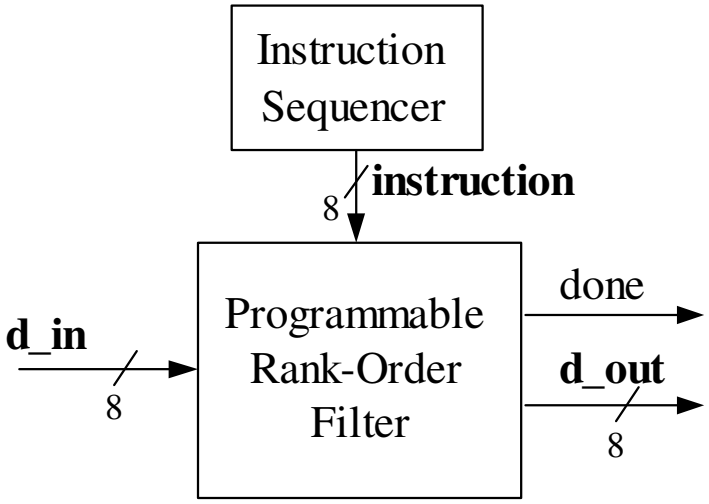


Figure 4.13. Block diagram of the 1-D non-recursive ROF.

```
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.4.14 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.

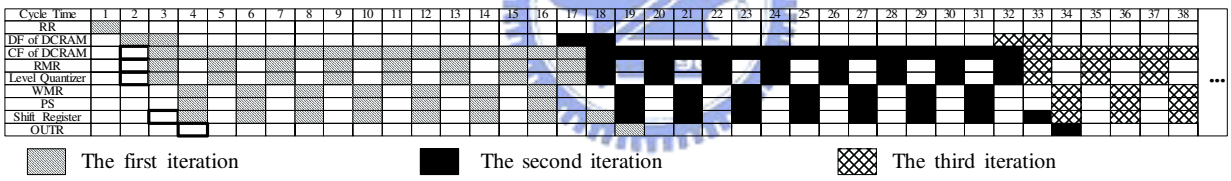


Figure 4.14. Reservation table of the 1-D non-recursive ROF.

4.3 Application of The Proposed ROF Processor

In Section 4.2.4, 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.

4.3.1 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 $\{y_{i-k}, y_{i-k+1}, ..., y_{i-1}, x_i, ..., x_{i+l}\}$, where $y_{i-k}, y_{i-k+1}, ..., y_{i-1}$ are the most recent median values and $x_i, ..., x_{i+l}$ are the input samples, and the result y_i is the $\lceil (l+k+1)/2 \rceil$-th value of the input window.

Fig.4.15 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.4.15, 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.4.15. The date storage shows the data reusability of the proposed ROF processor.

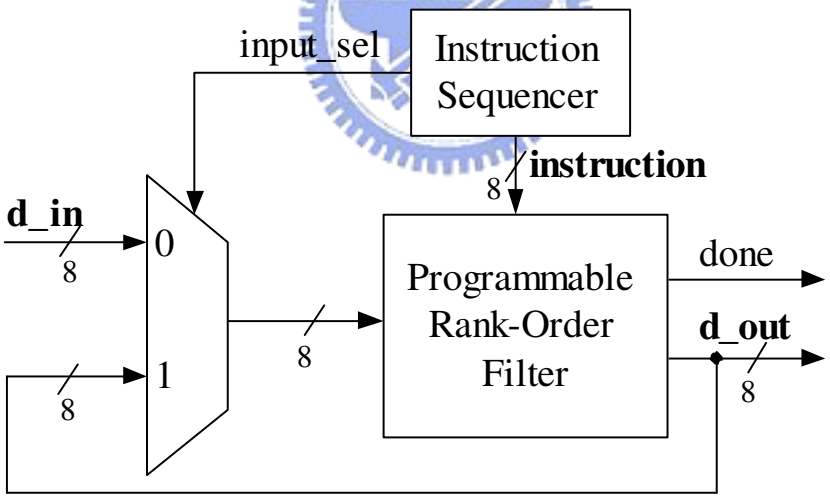


Figure 4.15. Block diagram of the 1-D RMF.

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

```
SET 5;
```

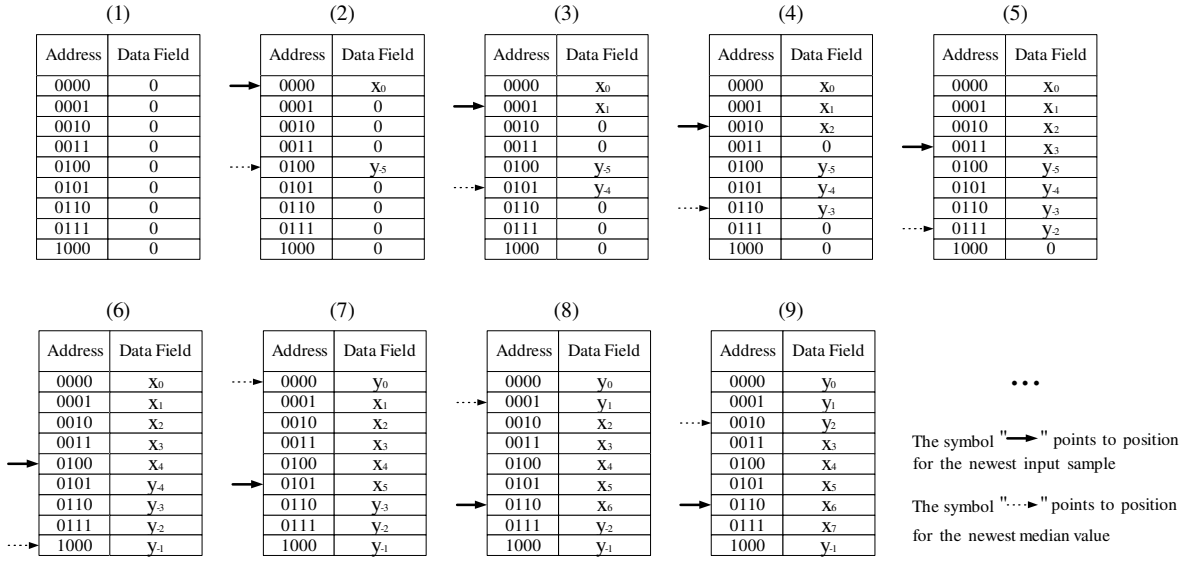


Figure 4.16. The flow for data storage of the 1-D RMF.

```
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.4.16. As seen in Fig.4.17, 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.

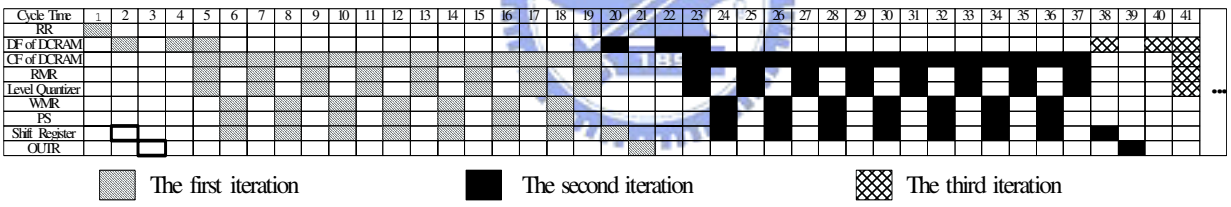


Figure 4.17. Reservation table of the 1-D RMF.

4.3.2 2-D Non-Recursive Rank-Order Filter

Fig.4.18 illustrates the block diagram for the 2-D non-recursive ROF. From Fig.4.19, 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.4.20. 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.4.21.

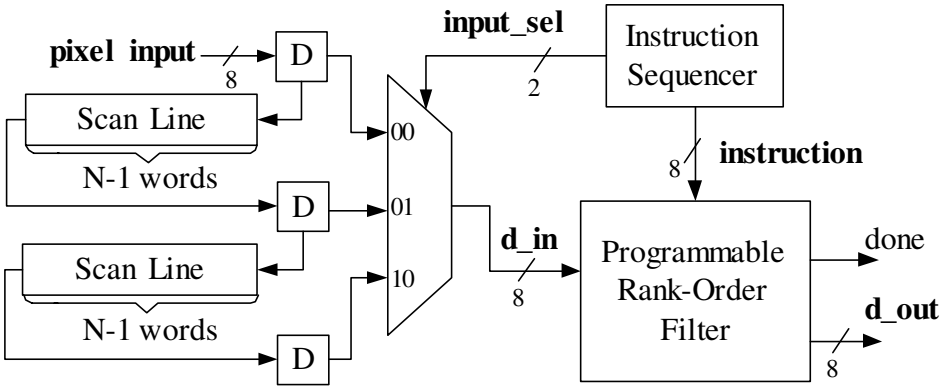


Figure 4.18. Block diagram of the 2-D non-recursive ROF with 3-by-3 window.

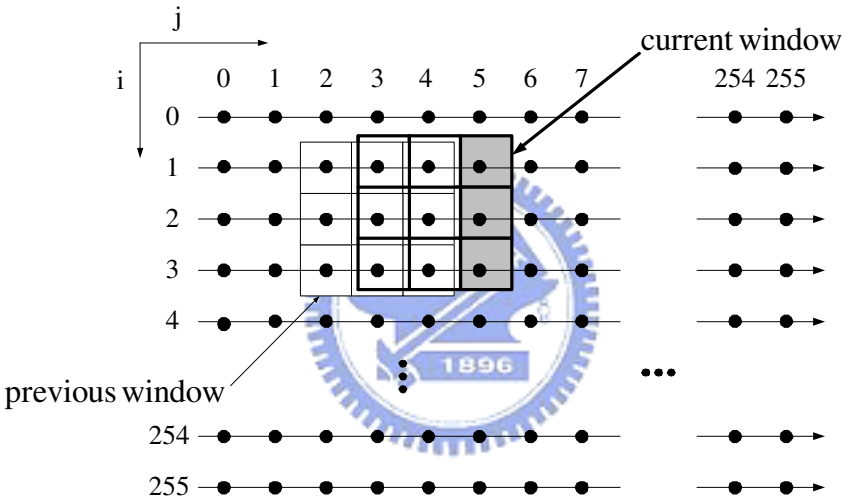


Figure 4.19. The windowing of the 3×3 non-recursive ROF.

Given a 2-D $n \times n$ ROF application with n=3 and r=5, the optimized reservation table can be scheduled as Fig.4.21 and the following is the pseudo-code:

```
SET 5;
i=0; -- IS
start loop; -- IS
input_sel=0; -- IS
LOAD i, P_READ 00000010;
```

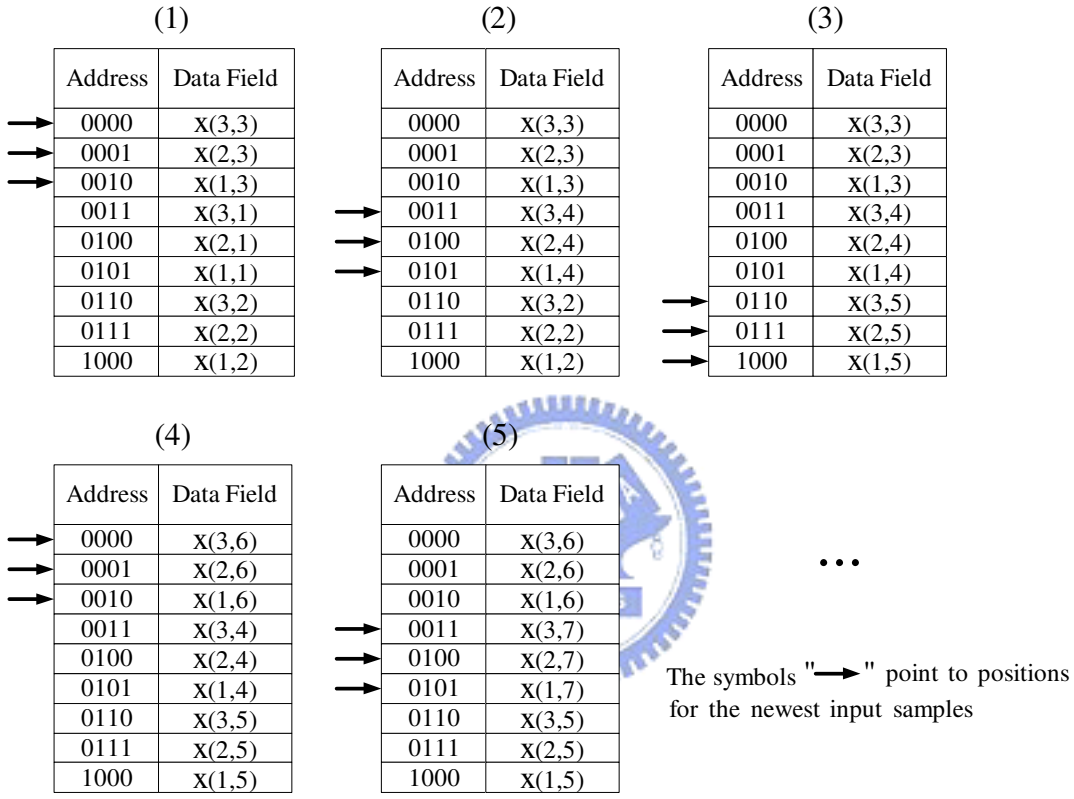


Figure 4.20. The data storage of the 2-D non-recursive ROF.

```
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
```

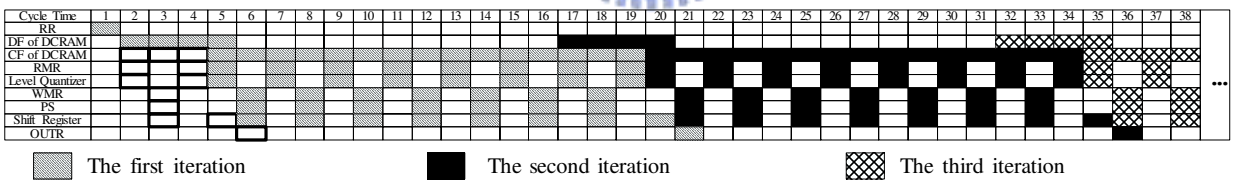


Figure 4.21. Reservation table of the 2-D ROF.

4.3.3 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.4.22(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.4.22(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.4.23 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.4.24. Given a 2-D 3×3 RMF application, the optimized reservation table can be scheduled as Fig.4.25.

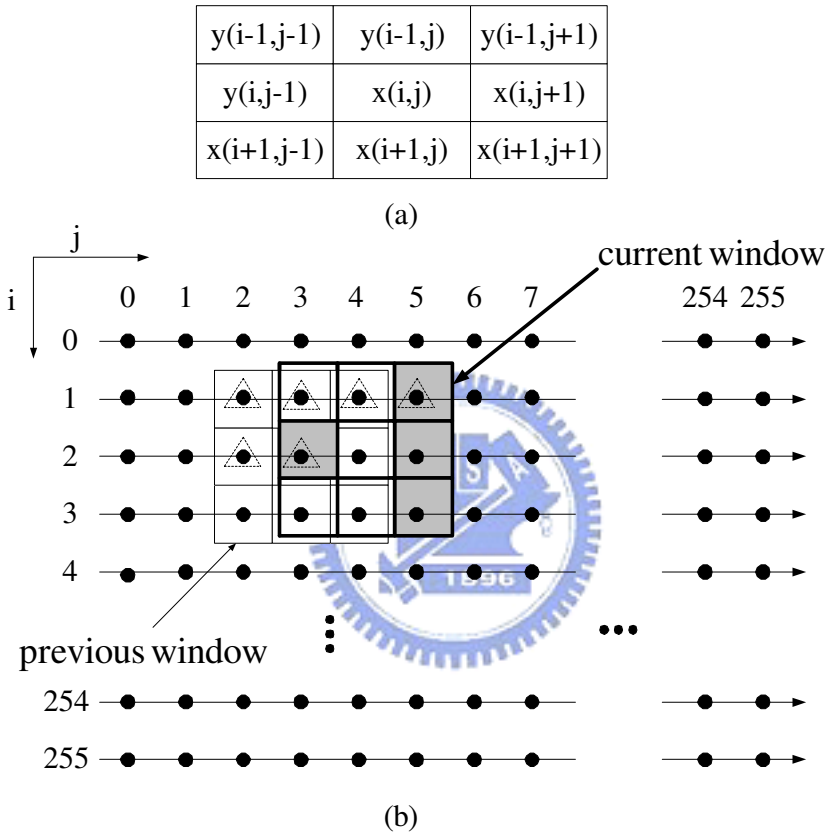


Figure 4.22. (a) The content of the 3×3 window centered at (i, j). (b) The windowing of the 2-D RMF.

Given a 2-D 3×3 RMF application, the optimized reservation table can be scheduled as Fig.4.25 and the pseudo-code is written as follows:

```
SET 5;
i=0; -- IS
```

(1)

Address	Data Field
0000	x(3,3)
0001	x(2,3)
0010	y(1,3)
0011	x(3,1)
0100	y(2,1)
0101	y(1,1)
0110	x(3,2)
0111	x(2,2)
1000	y(1,2)

(2)

Address	Data Field
0000	x(3,3)
0001	x(2,3)
0010	y(1,3)
0011	x(3,4)
0100	x(2,4)
0101	y(1,4)
0110	x(3,2)
0111	y(2,2)
1000	y(1,2)

(3)

Address	Data Field
0000	x(3,3)
0001	y(2,3)
0010	y(1,3)
0011	x(3,4)
0100	x(2,4)
0101	y(1,4)
0110	x(3,5)
0111	x(2,5)
1000	y(1,5)

(4)

Address	Data Field
0000	x(3,6)
0001	x(2,6)
0010	y(1,6)
0011	x(3,4)
0100	y(2,4)
0101	y(1,4)
0110	x(3,5)
0111	x(2,5)
1000	y(1,5)

(5)

Address	Data Field
0000	x(3,6)
0001	x(2,6)
0010	y(1,6)
0011	x(3,7)
0100	x(2,7)
0101	y(1,7)
0110	x(3,5)
0111	y(2,5)
1000	y(1,5)

• • •

The symbols "→" point to positions for the newest input samples

Thesymbols "⇢" point to positions for the windowing median values

Figure 4.23. The data storage of 2-D RMF.

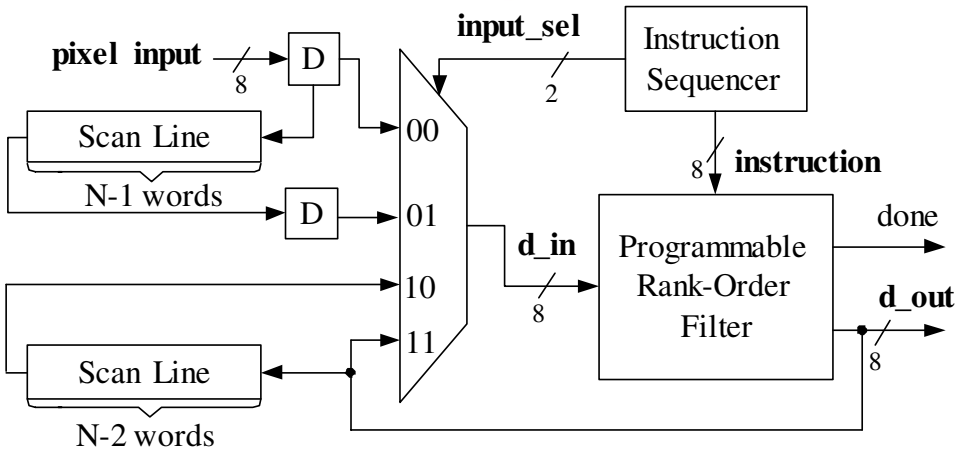


Figure 4.24. Block diagram of the 2-D RMF with 3-by-3 window.

```
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
```

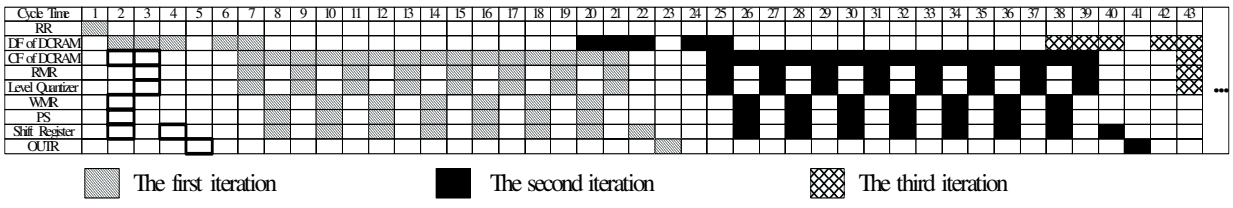


Figure 4.25. Reservation table of optimal pipeline for 2-D recursive median filter.

4.4 The Fully-Pipelined DCRAM-based ROF Architecture

As seen in Section 4.3, 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. 4.26, 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.

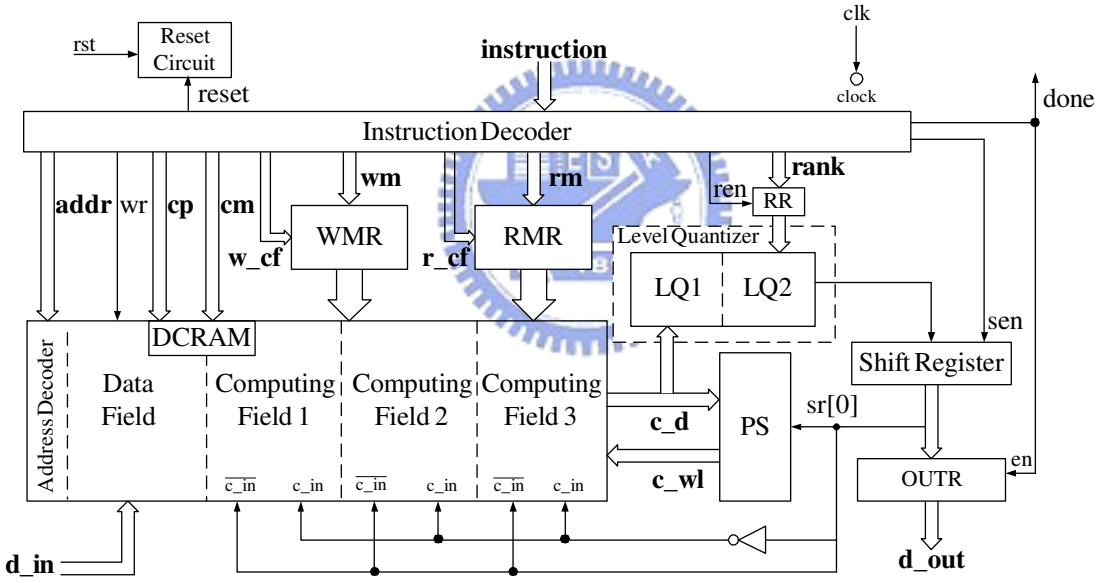


Figure 4.26. The fully-pipelined ROF architecture.

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.4.27 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.4.28. 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.4.29.

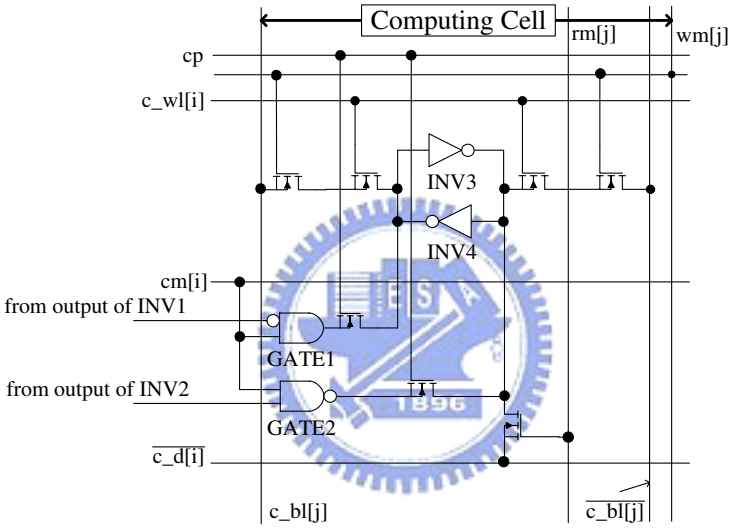


Figure 4.27. A modified circuit of computing cell for fully-pipelined ROF.

4.5 Chip Design and Simulation Results

To exercise the proposed architecture, we have implemented the ROF architecture, shown in Fig.4.5, 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. Although the ROF is the mainly operational core in the LUM filter, the ROF itself also has good performances in fields of smoothing and sharpening. As the result of simulation, the imple-

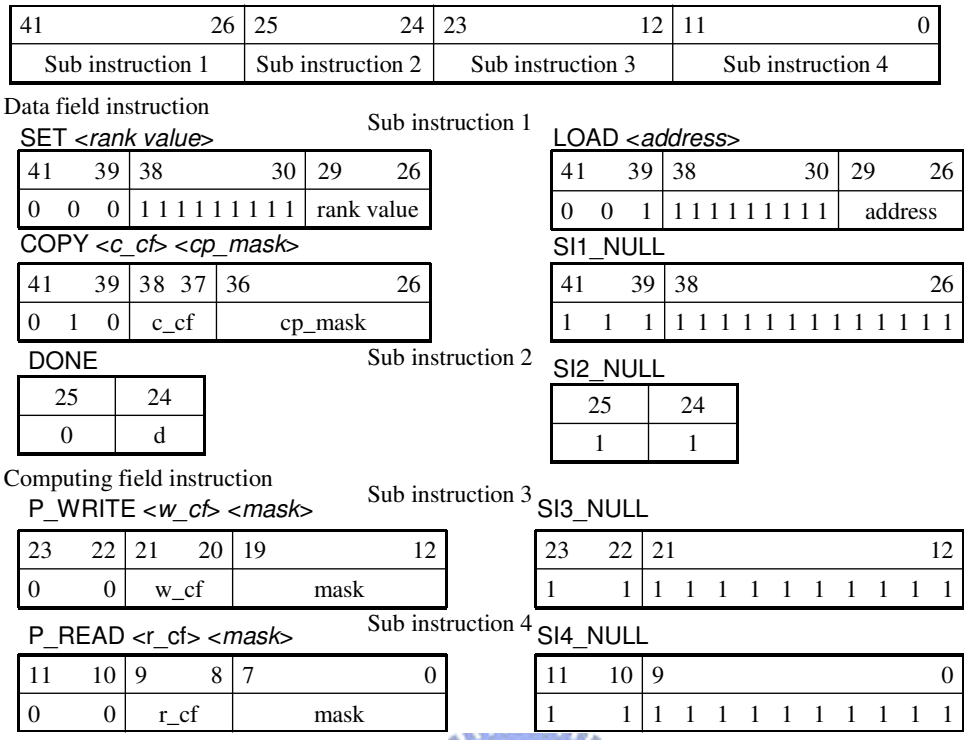


Figure 4.28. The format of the extended instruction set for the fully-pipelined ROF architecture.

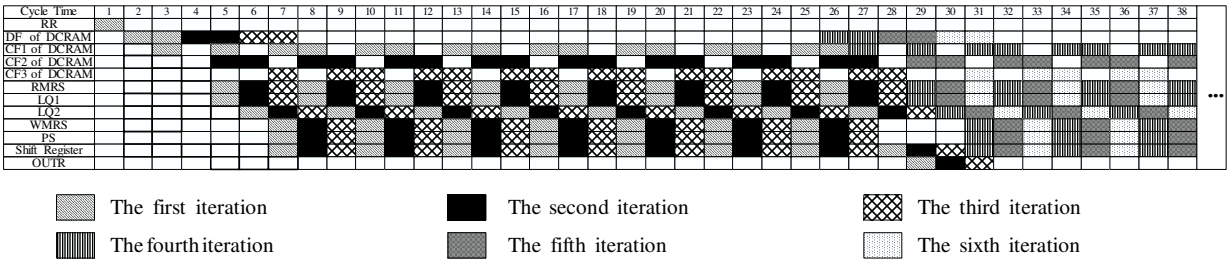


Figure 4.29. Reservation table of the 1-D non-recursive ROF for fully-pipelined ROF architecture.

mentations of the above examples are valid. Fig.4.30 and Fig.4.31 demonstrate the results of VHDL simulations for the 2-D ROF and RMF, respectively. Fig.4.30(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.4.30(b), (c), and (d), respectively. Fig.4.31(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.4.31(b). The results are the same as those of Matlab simulation.

Figure 4.30. 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.

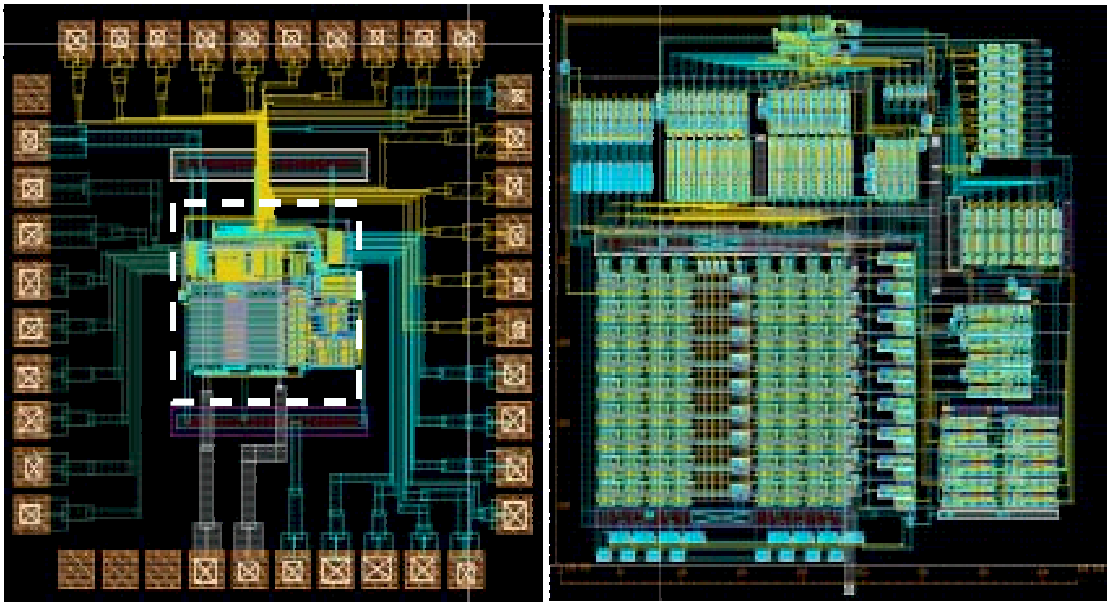
Upon verifying the proposed ROF processor using the cycle-accurate behavior model, we then implemented the processor in the fully-custom design methodology. Because of high regularity of memory, the proposed memory-based architecture saves the routing area

(a) (b)

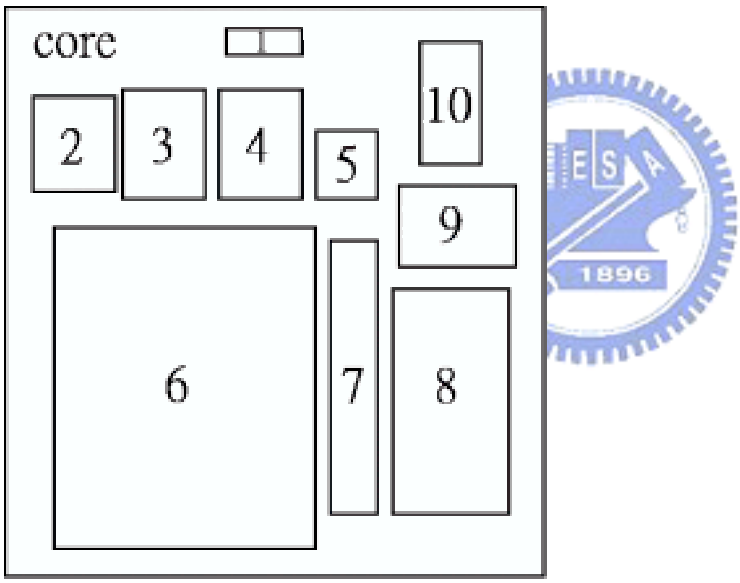
Figure 4.31. Simulation results of a 2-D RMF application. (a) The noisy "Lena" image corrupted by 9% of impulsive noise. (b) The "Lena" image processed by the 3×3 RMF.

while comparing with the logic-based solutions. Fig.4.32 (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.4.32 (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.4.32 (c) illustrates the floorplan and placement. The physical implementation has been verified by the post-layout simulation. Table 4.1 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.4.33(a) shows the micrograph of the DCRAM chip. The chip implements a subword part of DCRAM and the Fig.4.33(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.4.34, with the supply voltage



(a) (b)



(c)

Figure 4.32. 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.)

Table 4.1. Timing analysis of the proposed ROF processor.

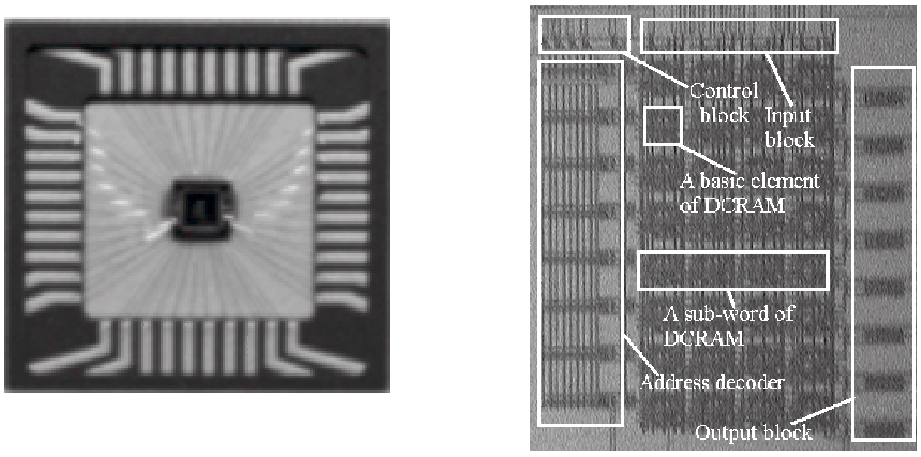
Path	Description	1.8V supply	3.3V supply
1	From the output of RR to the input of the shift register.	1.2 ns	0.78 ns
2	From the output of RMR, thru DCRAM to the input of the PS.	1.8 ns	1.1 ns
3	From the output of RMR, thru DCRAM and the Level-Quantizer, to the input of the shift register.	3.9 ns	3.44 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

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.4.35 The prototype was validated with ROF algorithms mentioned above.

4.6 Comparison of Existing ROF Architectures

Since this design 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



(a) (b)

Figure 4.33. (a) The micrograph of DCRAM chip. (b) The layout of the DCRAM chip.

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 [61] 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 [53, 68] 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.

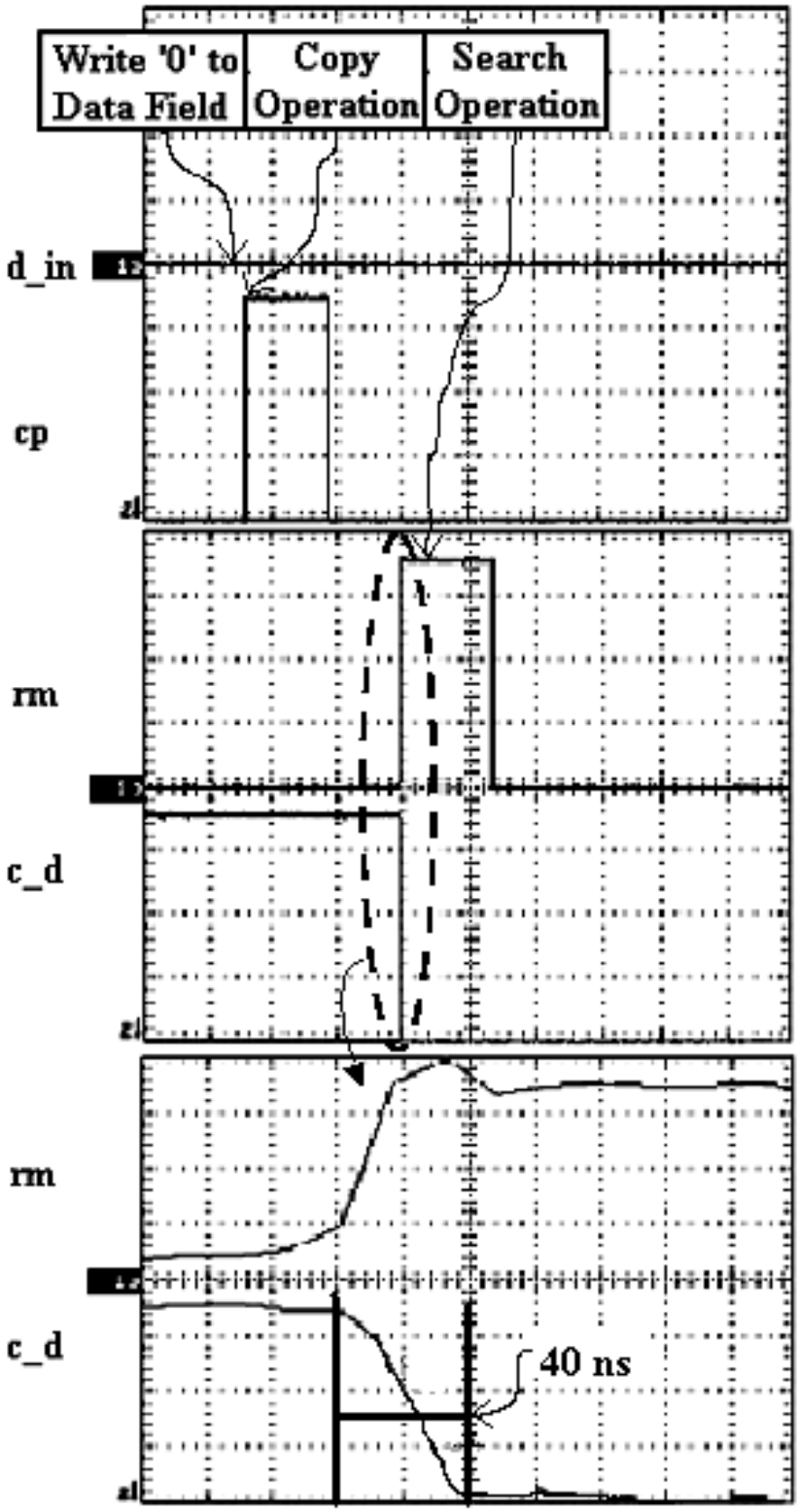


Figure 4.34. Measured waveform of the DCRAM chip.

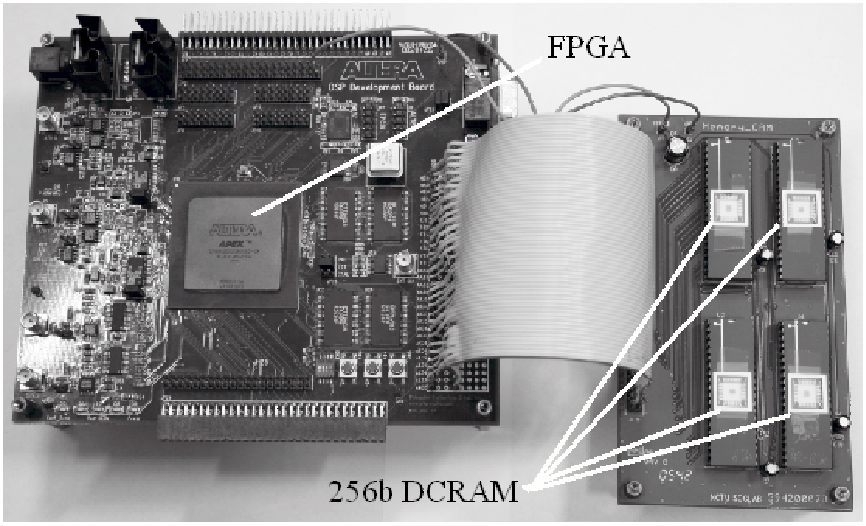


Figure 4.35. The system prototype of rank-order filtering processor.

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 [62] 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 [72] 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 4.3, 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.

5

Conclusions and Future Works

In this dissertation, we have successfully developed two kinds of ultra-low-power image compression processors for capsule endoscope or swallowable imaging capsules. In applications of capsule endoscopy, it is imperative to consider battery life/performance trade-offs. Instead of applying state-of-the-art image compression techniques, we first propose an RGB-based compression algorithm, called GICam-I. In which the memory size and computational load can be significantly reduced. We first simplified traditional image compression algorithms by removing the demosaicking technique and the color-space transformation. As shown in the result, the GICam-I only costs 31K gates at 2 frames per second, consumes 14.92 mW, and reduces the video size by 75 percents at least. In addition, to further extend the battery life of capsule endoscope, we herein present an improved ultra-low-power subsample-based GICam image compression processor, called GICam-II for capsule endoscope or swallowable imaging capsules. By using the subsample technique to improve computational loads in the GICam-I, we successfully make use of the subsample technique to reduce the memory requirements of G1, G2 and B components according to the analysis results of color sensitivity. As shown in the simulation result, the GICam-II can efficiently save the power dissipation than GICam-I by 38.5% under the same compression performance.

Except using novel ultra-low-power compression techniques to save the power dissipation of RF transmitter in the high-resolution capsule endoscope. How to efficiently eliminate annoying impulsive noise caused by a fault sensor and enhance the sharpness is necessary for a GI image and LUM filter is the most suitable candidate because it simultaneously has the characteristics of smoothing and sharpening. In the operational procedure of LUM filter, the mainly operational core is the rank-order filtering (ROF) and the LUM filter itself needs to use different kind of rank values to accomplish the task of smoothing or sharpening. Therefore, we need a flexible ROF hardware to arbitrarily select wanted rank values into the operation procedure of LUM filter. In this dissertation, we have proposed an architecture based on a maskable memory for rank-order filtering. This dissertation 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. Except efficiently eliminating annoying impulsive noises and enhance the sharpness for GI images, the processing speed of ROF can also meet the real-time image applications in the QCIF, CIF, VGA, or SVGA formats.

Next, we will begin to develop and realize the GICam endoscope system and then practically use this novel wireless endoscope system in the clinical diagnosis to benefit patients.

Bibliography

[1] S.J. Phee, W.S. Ng, I.M. Chen, F. Seow-Choen, and B.L. Davies, "Locomotion and steering aspects in automation of colonoscopy," IEEE Engineering in Medicine and Biology Magazine, vol. 16, pp. 85-96, Nov./Dec. 1997.

[2] I. Kassin, W.S. Ng, G. Feng, and S.J. Phee, "Review of locomotion techniques for robotic colonoscopy," in Proc. of the 2003 IEEE Intl Conf on Robotics & Automation, Taipei, Taiwan, pp. 1086-1091, Sept. 2003.

[3] B. Kim, Y. Jeong, H.-Y. Lim, J.-O. Park, A. Menciassi, and P. Dario, "Functional colonoscope robot system," in Proc. of the 2003 IEEE Intl Conf on Robotics & Automation, Taipei, Taiwan, pp. 1092-1097, Sept. 2003.

[4] C.A. Mosse, T.N. Mills, M.N. Appleyard, S.S. Kadirkamanathan, and C.P. Swain, "Electrical stimulation for propelling endoscopes," Gastrointestinal Endoscopy, vol. 54, no. 1, pp. 79-83, July 2001.

[5] L. Phee, D. Accoto, A. Menciassi, C. Stefanini, M.C. Carrozza, and Paolo Dario, "Analysis and development of locomotion devices for the Gastrointestinal Tract," IEEE Transactions on Biomedical Engineering, vol. 49, no. 6, pp. 613-616, June 2002.

[6] A.B. Slatkin and J. Burdick, "The development of a robotic endoscope," in Proc. of the 1995 IEEE/RSI Int'l Conf on Intelligent Robots & Systems, Pittsburgh, USA, pp. 162-171, Aug. 1995.

[7] G. Iddan, G. Meron, A. Glukhovsky, and P. Swain, "Wireless capsule endoscopy," Nature, vol. 405, pp. 417-418, May 2000.

[8] G.G. Ginsberg, A.N. Barkun, J.J. Bosco, G.A. Isenberg, C.C.Nguyen, B.T. Petersen, W.B. Silverman, A, Slivka, and G. Taitelbaum, "Wireless capsule endoscopy," Gastrointestinal Endoscopy, vol. 56, no. 5, pp. 621-624, Nov. 2002.

[9] Shinya Itoha, Shoji Kawahitob, Tomoyuki Akahoric, and Susumu Terakawad, "Design and implementation of a one-chip wireless camera device for a capsule endoscope," SPIE, vol. 5677, pp. 108-118, 2005.

[10] F. Gong. P. Swain, and T. Mills, "Wireless endoscopy," Gastrointestinal Endoscopy, vol.51, no. 6, pp. 725-729, June 2000.

[11] H. J.Park, H.W. Nam, B.S. Song, J.L. Choi, H.C. Choi, J.C. Park, M.N. Kim, J.T. Lee, and J.H. Cho, "Design of bi-directional and multi-channel miniaturized telemetry module for wireless endoscopy," in Proc. of the 2nd Annual Intl IEEE-EMBS Special Topic Conference on Microtechnologies in Medicine and Biology, Madison, USA, pp. 273-276, May 2002.

[12] http://www.givenimaging.com/Cultures/en-US/given/english

[13] http://www.rfsystemlab.com/

[14] Louis Phee, Dino Accoto, Arianna Menciassi*, Cesare Stefanini, Maria Chiara Carrozza, and Paolo Dario, "Analysis and development of locomotion devices for the gastrointestinal tract," IEEE Trans. On Biomedical Engineering, vol. 49, no. 6, June 2002.

[15] M. Sendoh, k. Ishiyama, and K.-I. Arai, "Fabrication of magnetic actuator for use in a capsule endoscope," IEEE Trans. On Magnetics, vol. 39, no. 5, pp. 3232-3234, Sept. 2003.

[16] Chao HU, Max Q.-H Meng, and Mrinal Mandal,"Efficient linear algorithm for magnetic localization and orientation in capsule endoscopy," in Proc. of the 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference, pp. 7143-7146, Sept. 2005.

[17] Chao HU, Max Qinghu Meng, and Mrinal Mandal,"Efficient magnetic localization and orientation technique for capsule endoscopy," IEEE/RSJ International Conference on Intellignet Robots and Systems, pp. 628-633, Aug. 2005.

[18] Shaou-Gang Miaou, Shih-Tse Chen, and Fu-Sheng Ke, "Capsule endoscopy image coding using wavelet-based adaptive vector quantization without codebook training," International Conference on Information Technology and Applications (ICITA), vol. 2, pp. 634-637, July 2005.

[19] Shaou-Gang Miaou, Shih-Tse Chen, and Chih-Hong Hsiao, "A wavelet-based compression method with fast quality controlling capability for long sequence of capsule endoscopy images," IEEE-EURASIP Workshop on Nonlinear Signal and Image Processing (NSIP), pp. 34-34, 2005.

[20] M. Lin, L. Dung, and P. Weng, "An ultra low power image compressor for capsule Endoscope," BioMedical Engineering Online 2006, vol. 5:14.

[21] H.A. Peterson, H. Peng, J. H. Morgan, and W. B. Pennebaker, "Quantization of color image components in the DCT domain" SPIE , Human Vision, Visual Processing, and Digital Display II, vol.1453, 1991.

[22] Dr. R.W.G. Hunt, "Measuring Colour," Fountain Press, 1998.

[23] Henry R. Kang "Color technology for electronic imaging devices," SPIE Optical Engineering Press, 1997.

[24] J.Ziv and A. Lempel, "A universal algorithm for sequential data compression," IEEE Trans. On Inform. Theory, vol. 23, pp. 337-343, May 1977.

[25] D.H. Kang, J.H. Choi, Y.H. Lee, and C. Lee, "Applications of a DPCM system with median predictors for image coding," IEEE Trans. Consumer Electronics, vol.38, no.3, pp.429-435, Aug. 1992.

[26] H. Rantanen, M. Karlsson, P. Pohjala, and S. Kalli, "Color video signal processing with median filters," IEEE Trans. Consumer Electron., vol.38, no.3, pp.157-161, Apr. 1992.

[27] T. Viero, K. Oistamo, and Y. Neuvo, "Three-dimensional median-related filters for color image sequence filtering," IEEE Trans. Circuits Syst. Video Technol., vol.4, no.2, pp.129-142, Apr. 1994.

[28] X. Song, L. Yin, and Y. Neuvo, "Image sequence coding using adaptive weighted median prediction," Signal Processing VI, EUSIPCO-92, Brussels, pp.1307-1310, Aug. 1992.

[29] K. Oistamo and Y. Neuvo, "A motion insensitive method for scan rate conversion and cross error cancellation," IEEE Trans. Consumer Electron., vol.37, pp.296-302, Aug. 1991.

[30] P. Zamperoni, "Variation on the rank-order filtering theme for grey-tone and binary image enhancement," IEEE Int. Conf. Acoust., Speech, Signal Processing, pp.1401-1404, 1989.

[31] C.T. Chen and L.G. Chen, "A self-adjusting weighted median filter for removing impulse noise in images," Int. Conf. Image Processing, pp.16-19, Sept. 1996.

[32] G. R. Arce and R. Foster, "Detail-preserving ranked-order based filters for image processing," IEEE Trans. Acoust., Speech, Signal Processing, vol.37. no.1 , Jan. 1989.

[33] G. R. Arce, N. C. Gallagher, and T. A. Nodes, "Median filters: Theory for one and two dimensional filters," in Image Enhancement and Restoration, Advances in Computer Vision and Image Processing, vol. 2 . T. S . Huang. Ed. Greenwich, CT: JAI Press. ch.3, pp. 90-166. 1986,

[34] G. R. Arce and M. P. Mcloughlin, "Theoretical analysis of maxi median filters," IEEE Trans. Acoust., Speech, Signal Processing. vol. ASSP-35, no. I , Jan. 1987.

[35] N. C. Gallagher and G. L. Wise. "A theoretical analysis of the properties of median filters," IEEE Truns. Acoust., Speech, Signal Processing, vol.29, no.6, Dec. 1981.

[36] P. Heinonon and Y. Neuvo, "FIR-median hybrid filters," IEEE Trans. Acoust. Speech, Signal Processing, vol.35. pp.832-838, June 1987.

[37] B. I. Justusson, "Median filtering: Statistical properties," in TMO-Dimensional Digital Signal Processing, II: Transforms and Median Filters. vol.42, Topics in Applied Physics, T. S. Huang, Ed. New York: Springer, ch.4, pp.161-196, 1981.

[38] J. S. Jimmy Li, "A class of multi-shell minimax median filters." in Proc. IEEE Int. Symp. Circuits Syst., (Oregon), May 1989.

[39] P. A. Maragos and R. W. Schafer. "Morphological filters-Part I: Their set theoretic analysis and relations to linear shift invariant tilters, and Morphological filters-Part II: Their relations to median. order statistic and stack filters," IEEE Trans. Acoust., Speech, Signal Processing, vol.35. no.8, pp.1153-1184, Aug. 1987.

[40] A. Nieniinen, P. Heinonen, and Y . Neuvo, "A new class of detailpreserving filters for image processing," IEEE Trans. Pattern Analysis Mach. Intelligence,, vol. PAMI-9, no.1 , pp.74-90, Jan. 1987.

[41] F. Palmieri and C. Boncelet, "LI-filters: A new class of order statistic filters," IEEE Trans. Acoust., Speech, Signal Processing, vol.31, no.5, May 1989.

[42] I. Pitas and A. N. Venetsanopoulos. "Nonlinear mean filters in image processing," IEEE Trans. Acoust., Speech, Signal Processing, vol.34, no.3, June 1986.

[43] R. L. Stevenson and G. R. Arce, "Morphological filters: Statistics and further syntactical properties," IEEE Trans. Circuits Syst., vol. CAS-34, no.11, Nov. 1987.

[44] I. W. Tukey, "Nonseparable (nonsuperposable) methods for smoothing data." in Cong. Rec. EASCON. 1974, p. 673; also in Exploratory Datcr Analysis. Reading MA: Addision-Wesley, 1977.

[45] Y. H. Lee and A. T. Fam, "An edge gradient enhancing adaptive order statistic filter," IEEE Trans. Acoust., Speech, Signal Processing, vol.35, May 1987.

[46] Y. H. Lee and S. Tantaratana, "An adaptive edge enhancing order statistic filter," in Proc. Inf. Conf. Acoust., Speech, Signal Processing (ICASSP), Apr. 1988.

[47] H. G. Longbotham and N. Barsalou. "The LMS. an adaptive optimal order statistic filter," in Proc. SPIE/SPSE Sytnp. Electron. Imaging Sci. Technol., Feb. 1990.

[48] H. G. Longbotham and N. Barsalou. "A class of gcneralired order statistic filters. similar to the median. that provide enhancement," in Proc. SPIE/SPSE Sytnp. Electron. Imaging Sci. Technol., Feb. 1991.

[49] Russell C. Hardie and Charles G. Boncelet,"LUM Filters: A class of rank-order-based filters for smoothing and sharpening," IEEE Trans. Signal Processing, vol.41, no.3, pp.1061-1076, March 1993.

[50] D. Yang and C. Chen, "Data dependence analysis and bit-level systolic arrays of the median filter," IEEE Trans. Circuits and Systems for Video Technology, vol.8, no.8, pp.1015-1024, Dec. 1998.

[51] L. Breveglieri and V. Piuri, "Digital median filter," Journal of VLSI Signal Processing, vol.31, pp.191-206, 2002.

[52] C. Chakrabarti, "Sorting network based architectures for median filters," IEEE Trans. Circuits ans Systems II: Analog and Digital Signal Processing, vol.40, pp.723-727, Nov. 1993.

[53] C. Chakrabarti, "High sample rate architectures for median filters," IEEE Trans. Signal Processing, vol.42, no.3, pp.707-712, March 1994.

[54] L. Chang and J. Lin, "Bit-level systolic array for median filter," IEEE Trans. Signal Processing, vol.40, no.8, pp.2079-2083, Aug. 1992.

[55] C. Chen, L. Chen, and J. Hsiao, "VLSI implementation of a selective median filter," IEEE Trans. Consumer Electronics, vol.42, no.1, pp.33-42, Feb. 1996.

[56] M.R. Hakami, P.J. Warter, and C.C. Boncelet, Jr., "A new VLSI architecture suitable for multidimensional order statistic filtering," IEEE Trans. Signal Processing, vol.42, pp.991-993, April 1994.

[57] Hatirnaz, F.K. Gurkaynak, and Y. Leblebici, "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, pp.685-688, May 2000.

[58] A.A. Hiasat, M.M. Al-lbrahim, and K.M. Gharailbeh, "Design and implementation of a new efficient median filtering algorithm," IEE Proc. Image Signal Processing, vol.146, no.5, pp.273-278, Oct. 1999.

[59] R.T. Hoctor and S.A. Kassam, "An algorithm and a pipelined architecture for order-statistic determination and L-filtering," IEEE Trans. Circuits and Systems, vol.36, no.3, pp.344-352, March 1989.

[60] M. Karaman, L. Onural, and A. Atalar, "Design and implementation of a general-purpose median filter unit in CMOS VLSI," IEEE Journal of Solid State Circuits, vol.25, no.2, pp.505-513, April 1990.

[61] C. Lee, P. Hsieh, and J, Tsai, "High-speed median filter designs using shiftable content-addressable memory," IEEE Trans. Circuits and Systems for Video Technology, vol.4, pp.544-549, Dec. 1994.

[62] C.L. Lee and C. Jen, "Bit-sliced median filter design based on majority gate," IEE Proc.-G Circuits, Devices and Systems, vol.139, no.1, pp.63-71, Feb. 1992.

[63] L.E. Lucke and K.K. Parchi, "Parallel processing architecture for rank order and stack filter," IEEE Trans. Signal Processing, vol.42, no.5, pp.1178-1189, May 1994.

[64] K. Oazer, "Design and implementation of a single-chip 1-D median filter," IEEE Trans. Acoust., Speech, Signal Processing, vol.ASSP-31, no.4, pp.1164-1168, Oct. 1983.

[65] D.S. Richards, "VLSI median filters," IEEE Trans. Acoust., Speech, and Signal Processing, vol.38, pp.145-153, January 1990.

[66] G.G. Boncelet, Jr., "Recursive algorithm and VLSI implementation for median filtering," IEEE Int. Sym. on Circuits and Systems, pp.1745-1747, June 1988.

[67] C. Henning and T.G. Noll, "Architecture and implementation of a bitserial sorter for weighted median filtering," IEEE Custom Integrated Circuits Conference, pp.189-192, May 1998.

[68] C.C Lin and C.J. Kuo, "Fast response 2-D rank order filter by using max-min sorting network," Int. Conf. Image Processing, pp.403-406, Sept. 1996.

[69] M. Karaman, L. Onural, and A. Atalar, "Design and implementaion of a general purpose VLSI median filter unit and its application," IEEE Int. Conf. Acoustics, Speech, and Signal Processing, pp.2548-2551, May 1989.

[70] J. Hwang and J. Jong, "Systolic architecture for 2-D rank order filtering," Int. Conf. Application-Specific Array Processors, pp.90-99, Sept. 1990.

[71] I. Pitas, "Fast algorithms for running ordering and max/min calculation," IEEE Trans. Circuits and Systems, vol.36, no.6, pp.795-804, June 1989.

[72] O. Vainio, Y. Neuvo, and S.E. Butner, "A signal processor for median-based algorithm," IEEE Trans. Acoustics, Speech, and Signal Processing, vol.37, no.9, pp.1406-1414, Sept. 1989.

[73] H. Yu, J. Lee, and J, Cho, "A fast VLSI implementation of sorting algorithm for standard median filters," IEEE Int. ASIC/SOC Conference, pp.387-390, Sptember 1999.

[74] J.P. Fitch, "Software and VLSI algorithm for generalized renked order filtering," IEEE Trans. Circuits and Systems, vol.CAS-34, no.5, pp.553-559, May 1987.

[75] M.Karaman and L. Onural, "New radix-2-based algorithm for fast median filtering," Electron. Lett., vol.25, pp.723-724, May 1989.

[76] J.P. Fitch, "Software and VLSI Algorithms for Generalized Ranked Order Filtering," IEEE Trans. Circuits and Syst., vol.CAS-34, no.5, pp.553-559, May 1987.

[77] B.K. Kar and D.K. Pradhan, "A new algorithm for order statistic and sorting," IEEE Trans. Signal Processing, vol.41, no.8, pp.2688-2694, Aug. 1993.

[78] V.A. Pedroni, "Compact hamming-comparator-based rank order filter for digital VLSI and FPGA implementations," IEEE Int. Sym. on Circuits and Systems, vol.2, pp.585-588, May 2004.

[79] Shobha Singh, Shamsi Azmi, Nutan Agrawal, Penaka Phani and Ansuman Rout, "Architecture and design of a high performance SRAM for SOC design," IEEE Int. Sym. on VLSI Design, pp.447-451, Jan 2002.

Vita and Publication List

Meng-Chun Lin was born in 1978. He received the B.S. degree and the Best Student Award in Electronic Engineering from Fu Jen Catholic University, Taipei, Taiwan,R.O.C. in 2001, and the M.S. degree in the Electrical and Control Engineering, National Chiao Tung University Hsinchu, Taiwan, R.O.C. in 2003. Since September 2003, he entered the National Chiao Tung University to pursue his Ph.D. degree in the department of electrical and control engineering under the supervision of Prof. Lan-Rong Dung. He expects to receive his Ph.D. degree in 2007. His research interests are image processing, video processing, VLSI architecture and memory circuit design. His publications are shown as follows.

Journal Paper

International

[**1**] Lan-Rong Dung and **Meng-Chun Lin**, "A Maskable Memory Architecture for Rank-Order Filtering," IEEE transactions on Consumer Electronic, Vol. 50, No. 2, pp. 558-564, May 2004. [SCI, EI, Full paper]

[**2**] **Meng-Chun Lin**, Lan-Rong Dung, and Ping-Kuo Weng,"An ultra-low-power image compressor for capsule endoscope," Biomedical Engineering Online 2006. Vol. 5:14. [SCI, Full paper]

[**3**] **Meng-Chun Lin** and Lan Rong Dung, "Development and Implementation of an Improved Diamond Search Algorithm Based on Adaptively Thresholding," WSEAS Transactions on Circuits and Systems, Issue 4, Vol. 6, pp.440-445, April, 2007. [EI, Full paper]

[4] **Meng-Chun Lin** and Lan-Rong Dung, "On VLSI Design of Rank-Order Filtering using DCRAM Architecture," accepted for publication in *Integration, the VLSI Journal*, 2007. [SCI, EI, Full paper]

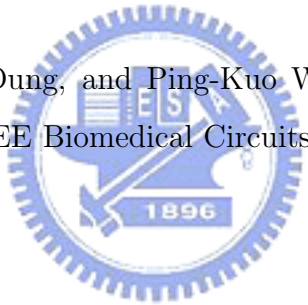
Conference Paper

International

[1] **Meng-Chun Lin** and Lan-Rong Dung, "A maskable memory architecture for rank-order filtering," IEEE Asia-Pacific Conference on Circuits and Systems, Vol.1, pp.453 -456, Dec. 2004.

[2] **Meng-Chun Lin**, Lan-Rong Dung, and Hsuan-Po Lin, "A Subsample-based Motion Estimation for Quality-Stationary Video Coding," IEEE Asia-Pacific Conference on Circuits and Systems, pp.1256-1259, 4-7 Dec. 2006.

[3] **Meng-Chun Lin**, Lan-Rong Dung, and Ping-Kuo Weng,"A Cardinal Image Compressor for Capsule Endoscope," IEEE Biomedical Circuits and Systems Conference, Dec. 2006.



Domestic

[1] **Meng-Chun Lin** and Lan-Rong Dung, "A Maskable Memory Based Rank-Order Filter Design," Proceedings of the 15th VLSI Design/CAD Symposium, Taiwan, August, 2004.

[2] Lan-Rong Dung, **Meng-Chun Lin**, Tai-You Chen, Yue-Zhang Song, Shih-Jay Huang, and Ping-Kuo Weng,"IMPLEMENTATION OF ULTRA-LOW-POWER IMAGE COMPRESSOR FOR CAPSULE ENDOSCOPE," Proceedings of the 16th VLSI Design/CAD Symposium, Taiwan, August, 2005.

[3] **Meng-Chun Lin**, Lan-Rong Dung, Chih-Wei Huang, Jun-Way Lyu, and Hsin-Cheng

Lai,"A Subsample-based Motion Estimation for Quality-Stationary Video Coding," Proceedings of the 17th VLSI Design/CAD Symposium, Taiwan, August, 2006.

[4] **Meng-Chun Lin**, Lan-Rong Dung and Ping-Kuo Weng,"An Improved Ultra-Low-Power Subsample-based Image Compressor for Capsule Endoscope,"Medical Informatics Symposium in Taiwan (MIST), Oct. 2006.