5615

DESIGN AND IMPLEMENTATION OF A GENERAL PURPOSE MEDIAN FILTER UNIT IN VLSI

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCES

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Mustafa Karaman

October 1988

T. C.
Yükseköğretim Kurulu
Dokümantasyon Merkezi

To my family

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Asst. Prof. Dr. Levent Onural(Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Assoc. Prof. Dr. Abdullah Atalar (Second supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Asst. Prof. Dr. Mehmet Ali Tan

Approved for the Institute of Engineering and Sciences:

[illegible]

Prof. Dr. Mehmet Baray, Director of Institute of Engineering and Sciences

© Copyright 1988
by
Mustafa Karaman

ABSTRACT

DESIGN AND IMPLEMENTATION OF A GENERAL PURPOSE MEDIAN FILTER UNIT IN VLSI

Mustafa Karaman
M.S. in Electrical and Electronics Engineering
Supervisors: Asst. Prof. Dr. Levent Onural
and Assoc. Prof. Dr. Abdullah Atalar
October 1988

The median of a group, containing an odd number of elements, is defined as the middle element, when the elements of the group are sorted. A median filter finds the median of a number of elements at its inputs. Median filters are frequently used in many signal, and image processing applications for smoothing of the noisy signals and images while at the same time preserving the edge information.

The required window size, speed, and the word length of the median filters vary depending on the applications. In order to meet these changing demands a general purpose median filter unit configuration is proposed. The unit consists of two single-chip median filters. One of the chips is designed for unlimited word-length and extensibility to larger window sizes whereas the other one is for real-time video applications. The networks of the chips are based on the odd/even transposition sorting. The chips are implemented in 3-μm M^{2}CMOS by using full-custom VLSI design techniques. For physical testing of the chips, the test vectors and the corresponding outputs of the chips are generated by using software tools written for this purpose.

In this thesis, the algorithms, VLSI implementations, simulation results, testing, and applications of the chips are presented.

Keywords: Median filters, signal and image processing, signal processing hardware, VLSI implementation, VLSI testing.

ÖZET

GENEL AMAÇLI BİR ORTANCA SÜZGECİ BİRİMİNİN VLSI TASARIMI VE GERÇEKLEŞTİRİLMESİ

Mustafa Karaman
Elektrik ve Elektronik Mühendisliği Yüksek Lisans
Tez Yöneticileri: Yrd. Doç. Dr. Levent Onural
ve Doç. Dr. Abdullah Atalar
Ekim 1988

Tek sayıdaki verilerin oluşturduğu bir kümenin ortancası (medyanı), verilerin büyüklüklerine göre sıralanması ile elde edilen dizinin tam ortasındaki veri olarak tanımlanır. Bir ortanca süzgeci, girişindeki verilerin ortancasını bulur. Ortanca süzgeçleri, bir çok işaret ve görüntü işleme uygulamalarında kullanılmaktadır. Kenar keskinliğinin korunması ve *impulsive* gürültülerin iyi ayıklanması bu süzgeçlerin en önemli özelliklerindendir.

Ortanca süzgeçlerinin gereken pencere büyüklüğü, hızı ve bit uzunluğu uygulamaya bağlı olarak değişmektedir. Bu değişen gereksinimleri karşılayabilmek için genel amaçlı bir ortanca süzgeci birimi önerilmiştir. Bu birim, iki tek yonga ortanca süzgecinden oluşmaktadır. Yongalardan biri, sınırsız bit uzunluğu ve daha büyük pencerelere genişletilebilecek şekilde diğeri ise *on-line* uygulamalarında kullanılabilecek hızlara erişebilecek şekilde tasarlanmıştır. Yongalarda kullanılan ağlar *tek/çift değiştirimle sıralama* yöntemine dayalıdır. Yongalar, *full-custom* VLSI tasarım teknikleri kullanılarak 3-μm M^{2}CMOS'da gerçekleştirilmiştir. Yongaların üretiminden sonra yapılacak testlerinde kullanılacak olan giriş işaretleri ve bunlara ait çıkış işaretleri bu amaçlar için yazılmış bilgisayar programları yardımıyla elde edilmiştir.

Bu tezde, söz konusu yongaların algoritmaları, VLSI gerçekleştirimleri, simülasyon sonuçları, testleri ve uygulamaları sunulmaktadır.

Anahtar Kelimeler: Ortanca süzgeçleri, işaret ve görüntü işleme, işaret işleme donanımı, VLSI gerçekleştirim, VLSI test.

ACKNOWLEDGEMENT

I would like to express my deep thanks to my thesis supervisors Asst. Prof. Dr. Levent Onural and Assoc. Prof. Dr. Abdullah Atalar for their supervisions, guidances, and encouragements during this research.

I am grateful to my friend İ. Enis Ungan for his efforts to modify the VLSI tools and to develop new tools, and to my friend Satılmış Topcu for his valuable ideas on VLSI testing of the chips.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Median Filter

The median of a group, containing an odd number of elements, is defined as the middle most element, when the elements of the group are sorted. For example, given five elements, 9, 0, 23, 145, 67, to find the median, the numbers are sorted as 0, 9, 23, 67, 145 . Then, the median is 23 which is the middle most element of the sorted sequence. A median filter finds the median of a number of elements at its input.

In many signal and image processing applications, it is necessary to smooth the noisy signals while at the same time preserving the edge information. The most commonly used smoothing techniques are the linear filtering [1], the averaging filtering [2], and the median filtering [3,4]. The linear filters smooth the noisy signals but also the sharp edges. In addition, the impulsive noise components can not be suppressed sufficiently by the linear filtering and the digital implementation of the linear filters can be bulky and slow. On the other hand, the generalized mean filters have the disadvantages of allowing a small number of outlier points that distort the filtered output signal, and also deleting the edge information. The median filters have been better alternatives because of their some very interesting properties: they

- can smooth the transient changes in signal intensity (e.g., noise),
- are very effective for removing the impulsive noises from the signals,
- preserve the edge information in the filtered signal, and
- can be implemented by using very simple digital nonlinear operations.

Because of these properties of the median filters, they are frequently used in various signal and image processing applications, such as seismic signal

T. C.
Yükseköğretim Kurulu
Dokümantasyon Merkezi

processing, speech processing, computerized tomography, medical imaging, robotic vision, pattern recognition, peak detection, coding, and communication.

In the applications for standard median filtering of the signals and images, a window of size w, where w is odd, is moved along the sampled values of signal or image, the median ($(w+1)/2$'*nd* largest one) of the samples within the window computed and written as the output pixel corresponding to the location of the center of the window. The median computed at this operation is called the *running* or the *moving* median. Since the size of the window is constant, the number of incoming elements is equal to the number of outgoing elements. The detailed discussion of the theoretical analysis of the median filters and their properties can be found in [5,6,7,8,9,10,11]. The applications of the median filters in speech processing and image processing were studied in [12,13,14].

1.2 Motivation and Approach

The purpose of this study is to have an exercise in VLSI design by completing a part of a project which consists of the design and implementation of a flexible VLSI image processor. For this purpose, the VLSI design of a general purpose median filter unit is chosen since this unit can also be used in any signal or image processors.

Different two-dimensional applications require different window sizes and word-lengths for median filtering. The window size varies from 3×3 to quite larger ones. Also, the required speed of the filtering operation varies depending on the application. A median filter which is intended as a component of a general purpose signal or image processor must meet these changing demands. For this reason, the median filter that we design should satisfy the following constraints:

- the word-length must be unlimited,
- the filter should be extensible to larger window sizes, and
- the filter should be able to operate at real-time rate which is about 30 mega medians per second for a 1024×1024 frame with 25-30 frames per second.

We propose a solution to this problem in the form of two single-chip median filter networks which form a general purpose median filter unit. The networks are implemented in 3-micron CMOS using the full-custom VLSI design techniques. One of the chips is designed for unlimited word length and extensibility for larger window sizes whereas the other one is for the real-time median filtering applications. The chips can be selectively used in a processor environment.

1.3 On the Performance of Algorithms

The algorithms that are efficient for software implementation minimize the number of computations and hence the computation time. In VLSI, the cost is determined by the modularity and the internal communication between the modules. This is because the irregular structures and especially complex communication interconnections increase the chip area considerably. The computation is cheap in VLSI, because the recent developments offer designers very high potential for parallel operations. Hence, a good algorithm for VLSI implementation does not necessarily require the minimal computation but should possess the following properties: the algorithm

- can be implemented by using only a few different simple units,
- must have a simple and regular communication and control scheme, and
- should employ the parallelism and pipelining.

A class of algorithms, called the *systolic algorithms*, have these properties. These algorithms for various computational problems have been discussed in [15,16,17,18,19,20]. The design cost of the special-purpose chips can be brought down by a systolic algorithm [21]. Application of the systolic data-flow concepts at the low level implementation of circuits such as comparison and addition will result in pipelined regular structures, small propagation delays and high throughput independent of the circuit size.

In design and implementation of a median filter in VLSI which is a special-purpose design work, the selection of the algorithm determines the cost and performance. Hence, much effort to the choice of algorithm should be given. The optimizations at circuit or layout level will play rather minor role on the overall cost and performance.

1.4 Benefits

In this study, we have followed all the steps of VLSI chip design starting from the problem definition to the test pattern generation of the chips. As a result, we have gained experience on VLSI design, and a median filter unit is obtained as a versatile component of a general purpose digital signal or image processor. Furthermore, the designed chips have significant importance since they are two of the first three custom chips designed and implemented in Turkey (the third one is a correlator chip designed at this university [22,23]).

1.5 Organization of Thesis

In the next chapter, a review of the median filtering techniques and algorithms is given. Chapter 3 covers the algorithms of the extensible and the real-time median filters. The VLSI design approach, circuit technique, and the VLSI design tools are described in Chapter 4. In Chapter 5, the VLSI implementation of the chips, their simulations, testings and possible applications are presented. Finally, the results and conclusions of this study are discussed in Chapter 6.

2. A SURVEY OF MEDIAN FILTERING

In this chapter, we will go over some of the median filtering techniques used in different practical applications and pass to the survey of median filtering algorithms for hardware implementation.

2.1 Median Filtering Techniques

Median filters can be used to suppress the impulsive noises, and to preserve the edges on images. In some applications, suppression of the noise may be more important than preserving the edges, or vice verse. In addition, median filter can be used to enhance the edges. Furthermore, in some cases, the median filters may fail to provide sufficient smoothing of nonimpulsive noise components. In order to solve these problems, various median filtering techniques were developed to increase the performance of the median filter for a set of particular applications. Here, we summarize the most commonly used median filtering techniques.

2.1.1 Finding Root Signal

If a signal is not affected by passing through a median filter, this signal is termed a *root signal* of that filter. Any root of a median filter with a particular window size is also a root of any median filter with a smaller window size. In order to remove the nonimpulsive noise components quite well, it is sufficient to obtain the root signal [24].

One can use either an adaptive-length median filter [25] or pass the signal from a fixed sized median filter successive times for better nonimpulsive noise suppression. Any signal of length S_l samples is reduced to its root after at most $(S_l - 2)/2$ successive passes by any median filter [5]. The cascaded median filters can be used for this purpose. Arce and Stevenson has reported that the cascade technique is better than a larger single median filter [26].

The median filtering of an arbitrary discrete levels signal to its root is equivalent to decomposing the signal into binary signals, filtering each binary signal to a root with a median filter, and then reversing the decomposition [27]. For application of the decomposition technique, a threshold test is performed for each sample value such that if the sample value is greater than the threshold value, then a 1 assigned to that sample as the new value, and otherwise the new value is 0. The selection of the threshold value and other details can be found in [27].

2.1.2 Recursive Median Filtering

The operation of the recursive median filter is the same as that of the standard median filter except that, at each step, the leftmost $(w-1)/2$ sample values in the moving window are replaced with the previous $(w-1)/2$ output sample values [28,29]. The most interesting property of the recursive median filter is that any signal can be reduced to a root after the first pass. Due to this fact, the recursive median filters are much efficient in the applications that require the fast finding the root signals, such as peak detection and coding [12]. This class of median filters can also be used with other median filtering techniques such as the recursive median filtering with threshold decomposition [27] and the recursive separable median filtering [30].

2.1.3 Generalized Median Filtering

An alternative technique to increase the efficiency of the nonimpulsive noise reduction or/and the edge preserving is the generalized median filtering. A generalization using linear combinations of order statistics is studied in [31]. Another generalization as a combination of a linear and median filters, in particular a combination of L and M filters, is analyzed in [32]. In the applications of the generalized median filtering, there is a tradeoff between noise suppression and the edge preservation; the generalized median filters are frequently used for edge detection [33,34,35,36]

2.1.4 Separable Median Filtering

The median filtering in two-dimensions can be performed by using a two-dimensional median filter with a window of arbitrary shape. But the most common windows are the square and cross-shaped windows. An alternative technique is the separable median filter which consists of two one-dimensional median filters that operate in the horizontal and vertical direction [30]. This technique is efficient for finding the root signal in two-dimensions quickly by using two separable recursive median filters instead of two standard median filters [37].

2.1.5 Weighted Median Filtering

The weighted median of a group of the elements is defined as the middle element of the sequence obtained in such a way that, the elements of the group are first multiplied with proper weight coefficients and then sorted [38]. It is shown that the selection of the weight coefficients represents a trade-off between the noise reduction and the preservation of the edge information. The maximum noise reduction is achieved when all the weight coefficients are equal, that is the same as the case of standard median filtering, whereas when the weight coefficients decrease as we move away from the center of the window, the preservation of edges and small details improves at the expense of noise suppression.

Different applications require the selection of a proper median filtering technique. However, the standard median filtering technique is the fundamental one, since it is used as the basic component in the realizations of many median filtering techniques. Due to this fact, we have considered the design and implementation of a standard median filter rather than any other type of median filters. For this reason, starting from the next section, we will concentrate only on the standard median filter algorithms and implementations.

2.2 Median Filtering Algorithms

The median filtering was first proposed by Tukey in 1974 [3,4]. From that time, there have been much efforts to develop fast off-line and on-line algorithms.

In 1979, the *histogram method* was presented [39], [40]. In this method, the median is found by sorting the grey level histogram of the window elements, and updating it as the window moves. The computation time of the method is $O(\sqrt{w})$ for a square window. The histogram method is an off-line method and not practical for hardware implementation because of the hardware complexity.

A real-time median filtering algorithm used for finding the running medians was proposed by Ataman *et al.* [41]. This algorithm is based on the binary representation of the sample values of the the window elements. For finding the running median, the number of 1's are counted in the most significant bits of the binary values of the elements, if that number is greater than $(w+1)/2$, then the most significant bit of the median is found as 1, else as 0. To find the second bit of the median, only the set of data whose first bits are equal to that of the median is considered and this operation is continued to find the other bits of the median. In order to determine the sets which include the median at each step, an array, called the α-array, is computed and stored. The hardware implementation of this algorithm requires 2^L words of memory (L is the word-length of an element), and an arithmetic-logic unit for arithmetic and logic operations. The parallel implementation of the algorithm needs L logic and L memory units. The main disadvantage of this algorithm is that its hardware implementation for a window size can not be used for finding the *exact median* for a larger window size.

A variation of Ataman's algorithm is given in [42]. In this algorithm, an m-array is computed instead of the α-array of Ataman's algorithm. The *k'th* element of the array indicates the number of the samples in the window greater than or equal to $(k-1)$. It is reported that the computation time of this algorithm is less than that of the Ataman's algorithm since updating the m-array is faster than that of α-array.

The most commonly used median filtering algorithms are based on sorting. Many of the basic sorting algorithms, especially for software implementation, are collected and analyzed in [43]. The VLSI complexity of sorting is studied in [44].

A sorting network for VLSI implementation is proposed in [45]. The area and time evaluation shows a very high performance, but on the other hand the hardware complexity, especially in the internal communication structure, and also the extensibility to larger data groups are the remaining problems of this network.

Two well known networks for sorting and switching are the bitonic sorting and the shuffle exchange networks. An overview of two chips based on the bitonic and shuffle networks is presented in [46]: the bitonic sorting network consists of $\frac{w}{2} \times \frac{1}{2}(log_2 w)(log_2 w+1)$ switch elements ($\frac{w}{2}$ elements at each row), whereas the shuffle network has $\frac{w}{2} \times log_2 w$ switch elements. The chips are designed for sorting 64 elements with an unlimited word-length. Greater number of elements can be sorted by using more than one chips. The main disadvantages of this configuration are the low throughput (one output per L clocks) and the increase in the area and design time of the chips due to the internal communication. Also, since the structure is based on the binary trees, there is some waste of area in an implementation if the number of inputs is not an integer power of 2. For example, if $w = 9$, then a bitonic sorter of size 16 will be required. The configuration may be preferable for a large w.

A separable median filter was implemented in hardware as a component of an image processor using the odd/even transposition sorting networks [47]. The filter consists of two pair of cascaded sorters. One of the pairs operate in vertical direction while the other operating in the horizontal direction of the frame. The cascaded connection is preferred to increase the efficiency of the separable median filter. Each sorter has six 8-bit comparators to sort three pixels. This sorter can be implemented by using the bit level systolic approach to increase the performance.

A bit level pipelined systolic odd/even transposition sorting network with very high throughput was proposed by Oflazer [48] for median filtering. The network consists of L blocks of $w(w-1)/2$ bitwise compare-and-swap units. The network is a regular array of the compare-and-swap units, hence it has a very modular structure with a simple internal communication scheme. The network is implemented for $w = 5$ and $L = 8$ as a single-chip one-dimensional median filter. The throughput of the filter is one median per clock. The clock period is determined by the delay of a bitwise compare-and-swap unit; the clock rate of Oflazer's filter chip is 10 MHz. The chip is not extensible for larger window sizes and the word-length is fixed to 8. The odd/even transposition network is not preferable for large w because the area is proportional with w^2.

There are some other median filtering algorithms presented in the literature. These algorithms are for implementation in software using high level languages hence they are out of the focus in this study. For two of such algorithms, reader is addressed to [49] and [50].

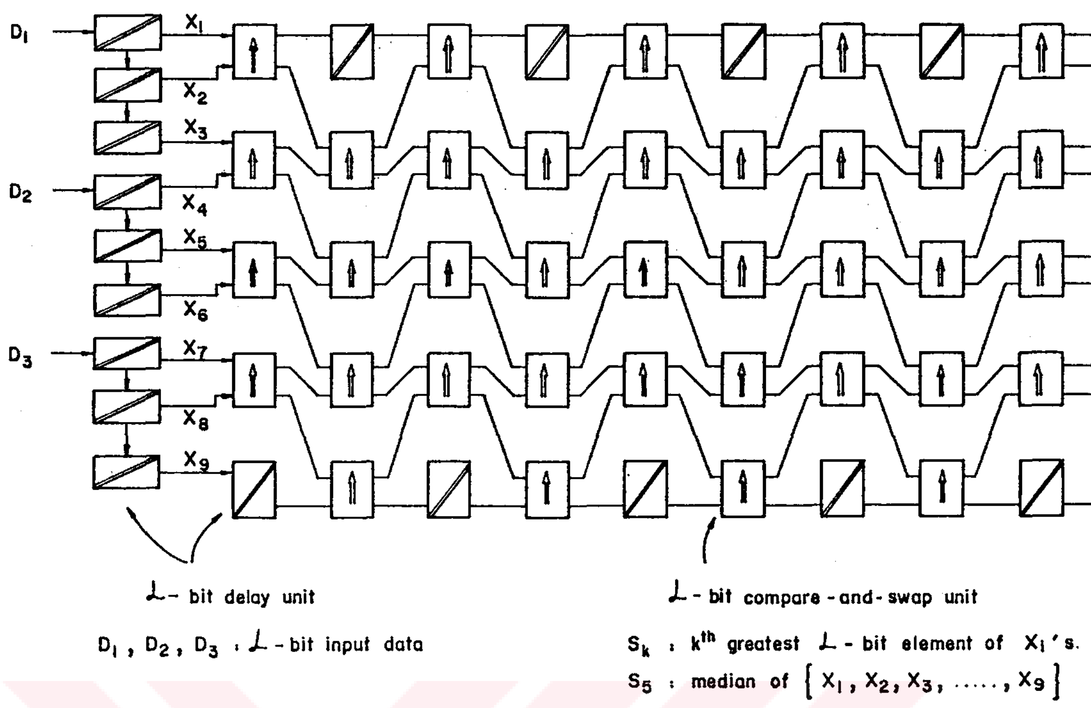
3. GENERAL PURPOSE MEDIAN FILTER

In this chapter, firstly, we will summarize the ordinary odd/even transposition sorting network. Then, the extensible and the real-time median filter networks which are based on the odd/even sorting are explained in detail. Finally, the feasibilities of the VLSI implementations of the networks are discussed.

3.1 Odd/Even Transposition Sorting Network

The odd/even transposition sorting network is a pipelined modular structure consisting of w *compare-and-swap* stages [43]. A compare-and-swap stage has $(w-1)/2$, w is odd, full-word (L-bit) parallel compare-and-swap units and an L-bit delay unit to delay the window element which is not compared at that stage. Each compare-and-swap unit operate on the odd and even pairs of the window elements alternately (Fig. 3.1). Odd pairs are in the form of (X_n, X_{n+1}) where n is odd, whereas n is even for even pairs. Each L-bit compare-and-swap unit compares the two L-bit elements at its inputs and interchanges them if necessary so that the larger one of the elements will be at the "top" and the smaller one will be at the "bottom" output. At the output of the last stage, the input elements will be sorted such that the largest will be at the "top" and the median will be at the middle.

If the odd/even transposition network described above is implemented in hardware as a single chip median filter for a fixed window size, w_0, then one can not use that chip to find the *exact* medians of the window elements that is larger than w_0. On the other hand, there is no flexibility on the word-length because the compare-and-swap unit can be implemented only for a fixed word-length in this configuration. Furthermore, the compare-and-swap unit can complete its job after an L-bit comparison resulting in a large propagation delay. Although, the throughput of the network is one L-bit median



D1
D2
D3
X1
X2
X3
X4
X5
X6
X7
X8
X9
S9
S8
S7
S6
S5
S4
S3
S2
S1
L - bit delay unit
D1, D2, D3 : L - bit input data
L - bit compare - and - swap unit
Sk : kth greatest L - bit element of Xi's.
S5 : median of {X1, X2, X3, ..., X9}

Figure 3.1: The ordinary odd/even transposition network ($w = 9$).

per clock (assuming a sequential logic implementation), it is not possible to clock this network with a frequency required for the real-time operation rate since the clock period will be at least the time required for a full-word comparison. All these results show that the odd/even transposition network in the configuration described above can not satisfy any of the requirements for a general purpose median filter (*extensibility* in window size, *unlimited* word-length, and *real-time* application capability).

However, the odd/even transposition network can be modified so that resultant structure(s) can satisfy our needs. Starting from this approach, two median filter networks are obtained. Both of these networks are bit-level, systolic and pipelined odd/even transposition sorting networks that employ all the advantages of the systolic algorithms and satisfy the listed requirements for a general purpose median filter. The networks serve different purposes: one of them designed for unlimited word-length and extensibility to larger window sizes whereas the other one is for real-time applications.

3.2 Extensible Median Filter Network

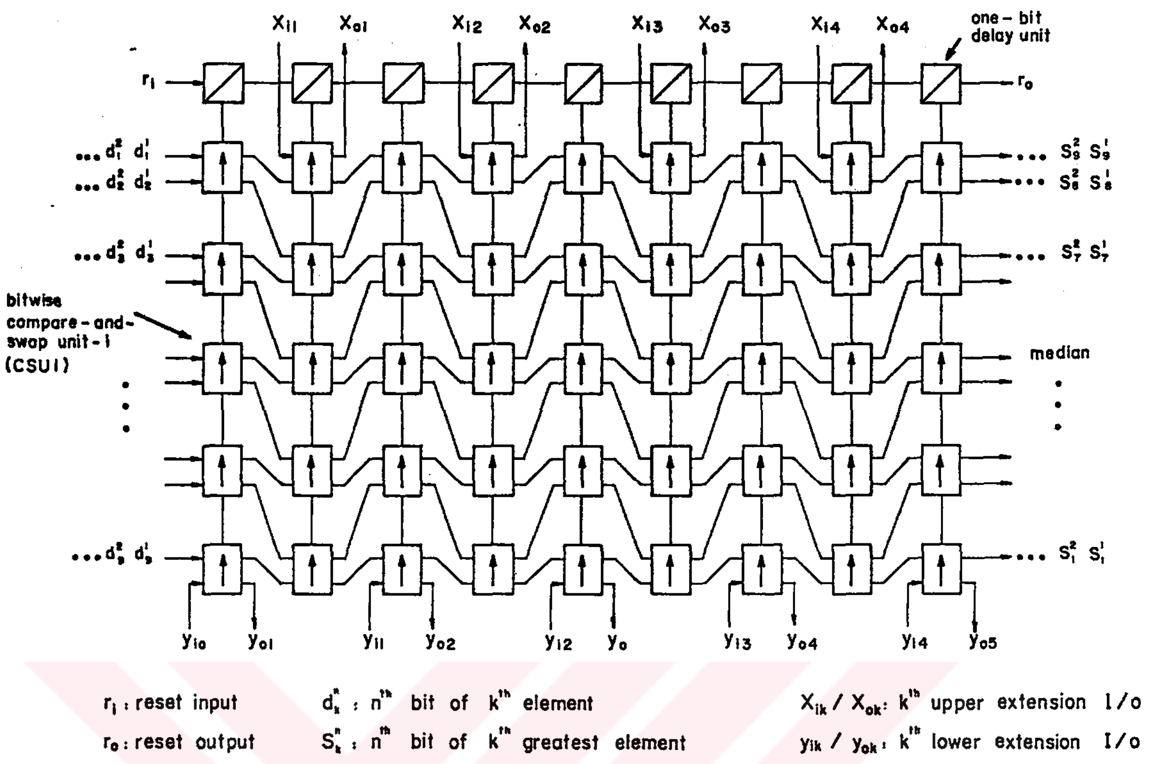
The odd/even transposition network shown in Fig. 3.1 is modified such that every operation is in the bit-serial form. We have used a bitwise compare-and-swap unit so that data flow is reduced to flowing of bit streams which eliminates the limit on the word-length and the large delay due to full-word comparison. In addition, the replacement of the delay units by the compare-and-swap units come up with extensibility in the window size. In the following paragraphs, the resultant extensible network is described in detail.

The extensible odd/even transposition sorting network is a pipelined systolic structure consisting of $w = 9$ *compare-and-swap* stages that operate on the consecutive pairs of window elements alternately. A compare-and-swap stage consists of $(W + 1)/2$ bitwise compare-and-swap units. Each of these units compares the two one-bit numbers at its inputs and interchanges them if necessary so that the larger one is at the "top". At the output of the last stage, the data will be sorted such that the largest will be at the "top", and the median will be at the middle.

At each clock, one bit from each word (total of w bits) enters the network and one bit of the median is obtained at the output. The flow is from the most significant bits toward the least significant bits both at the input and the output. Due to serial bitwise data flow, this structure allows arbitrary word-length, L. The network is given in Fig. 3.2 for $W = 9$.

The bitwise compare-and-swap unit, CSU1, has two one-bit data inputs A_i and B_i, a reset input R, and two one-bit data outputs A_o and B_o. It also has two internal parameters S (1/0: swap/pass) and E (1/0: equal/not equal) which are updated and stored in the CSU1 (Fig. 3.3). Because of the feedbacks S and E, the best structure for the CSU1 is a finite state machine to have a reliable timing [46]. The CSU1 compares A_i and B_i, and either passes them unaltered, or swaps them. It has three legal operation states: *equal state* , *pass state* and *swap state*. The present state values of S and E shows the subresult of the bitwise comparison of the more significant bits while the next state values indicate the new subresult.

CSU1 is set to the equal state ($S = 0$, $E = 1$) by the reset signal R at the beginning of each computation cycle. This is the instant after the least significant bit of the previous cycle leaves and before the most significant bit of the next cycle enters the CSU1. Thus the reset signal also flows through



x_{i1} x_{o1} x_{i2} x_{o2} x_{i3} x_{o3} x_{i4} x_{o4}
one-bit delay unit
r_i
r_o
...d_1^2 d_1^1
...d_2^2 d_2^1
...d_3^2 d_3^1
...d_9^2 d_9^1
bitwise compare-and-swap unit-1 (CSU1)
...s_9^2 s_9^1
...s_8^2 s_8^1
...s_7^2 s_7^1
median
...s_1^2 s_1^1
y_{i0} y_{o1} y_{i1} y_{o2} y_{i2} y_o y_{i3} y_{o4} y_{i4} y_{o5}
r_i : reset input
r_o : reset output
d_k^n : n^th bit of k^th element
S_k^n : n^th bit of k^th greatest element
X_{ik} / X_{ok}: k^th upper extension I/o
y_{ik} / y_{ok}: k^th lower extension I/o

Figure 3.2: The extensible median filter network ($w = 9$).

the stages of the network. For this purpose, a chain of the bitwise delay units are used to reset the stages at appropriate time instants (see Fig. 3.2).

During the computation, the CSU1 stays in the equal state ($S = 0$, $E = 1$) and passes the input data unaltered ($A_o = A_i$ and $B_o = B_i$) as long as the two input bits are equal as they flow in. However, it locks itself into the pass state when it first finds that $A_i > B_i$ and passes the inputs unaltered. On the other hand, it locks itself into the swap state when it first finds that $A_i < B_i$ and swaps the inputs, $A_o = B_i$ and $B_o = A_i$.

The extension I/O's ($x_{i/o}$'s and $y_{i/o}$'s, see Fig. 3.2) of the extensible median filter network are used to extend the filter to the window sizes larger than $w = 9$. If the *upper* and *lower extension* inputs (x_i's and y_i's) are connected to logic 1's and 0's, respectively, so the corresponding CSU1's operate as delay units. This is due to fact that the CSU1 will always pass the inputs unaltered if its inputs are 1X or X0 ($A_i = 1$,$B_i = X$ or $A_i = X$,$B_i = 0$), where X indicates *don't care* (i.e, X may be 1 or 0). On the other hand, if the network given in Fig. 3.2 is used as an array element to find the exact medians of window elements for $w > 9$ by connecting them as in Fig. 3.4.

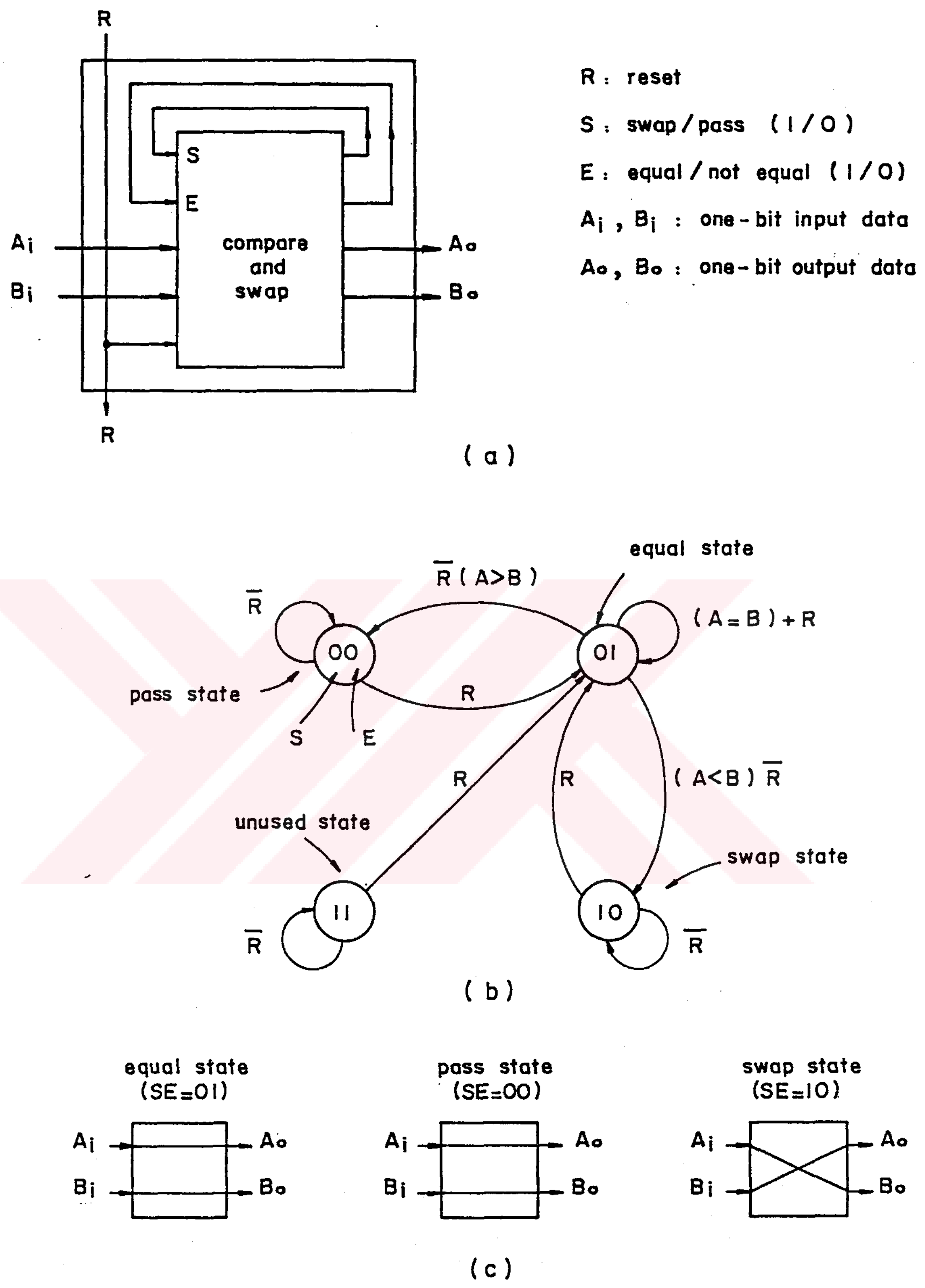


Figure 3.3: The compare-and-swap unit-1 (CSU1): a) block diagram, b) state diagram, c) operations.

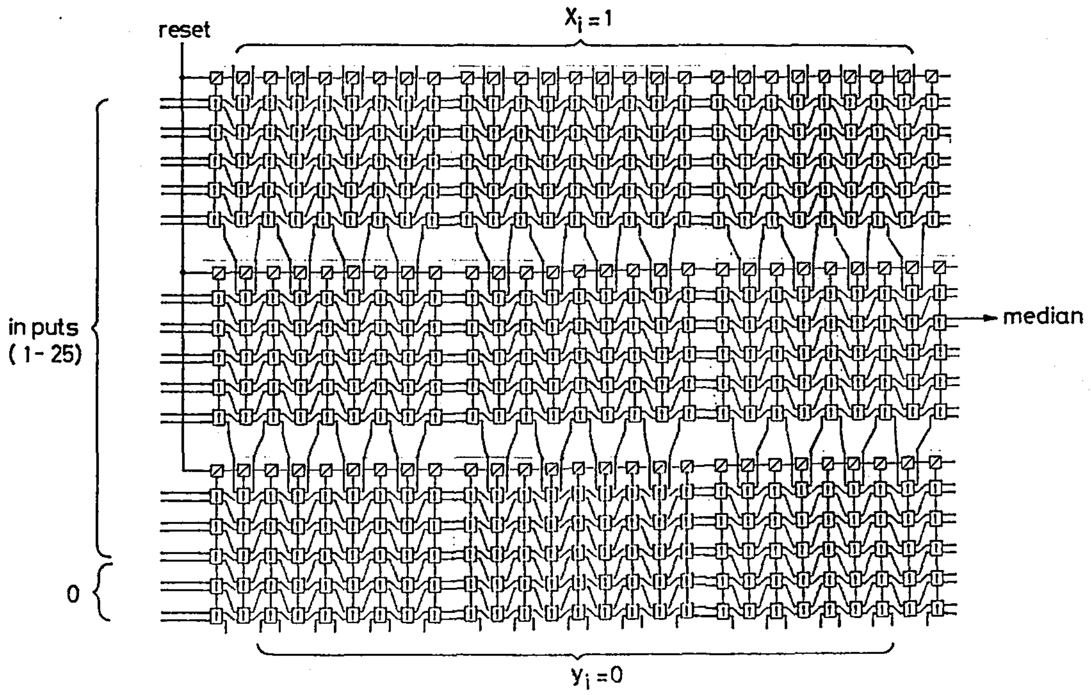


Figure 3.4: Interconnections of the extensible median filter networks for $w = 9$.

then these CSU1's operate as usual except those corresponding to the upper extension I/O's of the upper most network and corresponding to the lower extension I/O's of the lowest network. In this case, the CSU1's corresponding to the interconnected extension I/O's of the networks are not forced to lock in one state by means of extension signals, but they are free to enter any state depending on their inputs. As a result, the extensibility to indefinitely large windows is achieved: in the normal mode, it is possible to compute the exact median of a group of 9 elements using only one network, and for groups of w elements where $w > 9$, at most $[w/9]^2$ ([.] indicates the smallest greater integer) networks are needed with appropriate interconnections of the extension and data I/O's similar to the connection scheme in Fig. 3.4.

The extensible network described above generates its outputs with a pipeline delay of $w + L$ clocks and after the network is full, it finds one median per L clocks. The network can operate at a clock rate that is determined by the delay of one compare-and-swap unit which consists of, at least, the delay of a few cascaded bitwise comparisons and a 2×1 multiplexing. Although, the resulting speed can be sufficient for the real-time median filtering of the 512×512 frames with $L < 5$, it is less than the real-time operation rate for the 1024×1024 frames with $L > 1$.

3.3 Real-Time Median Filter Network

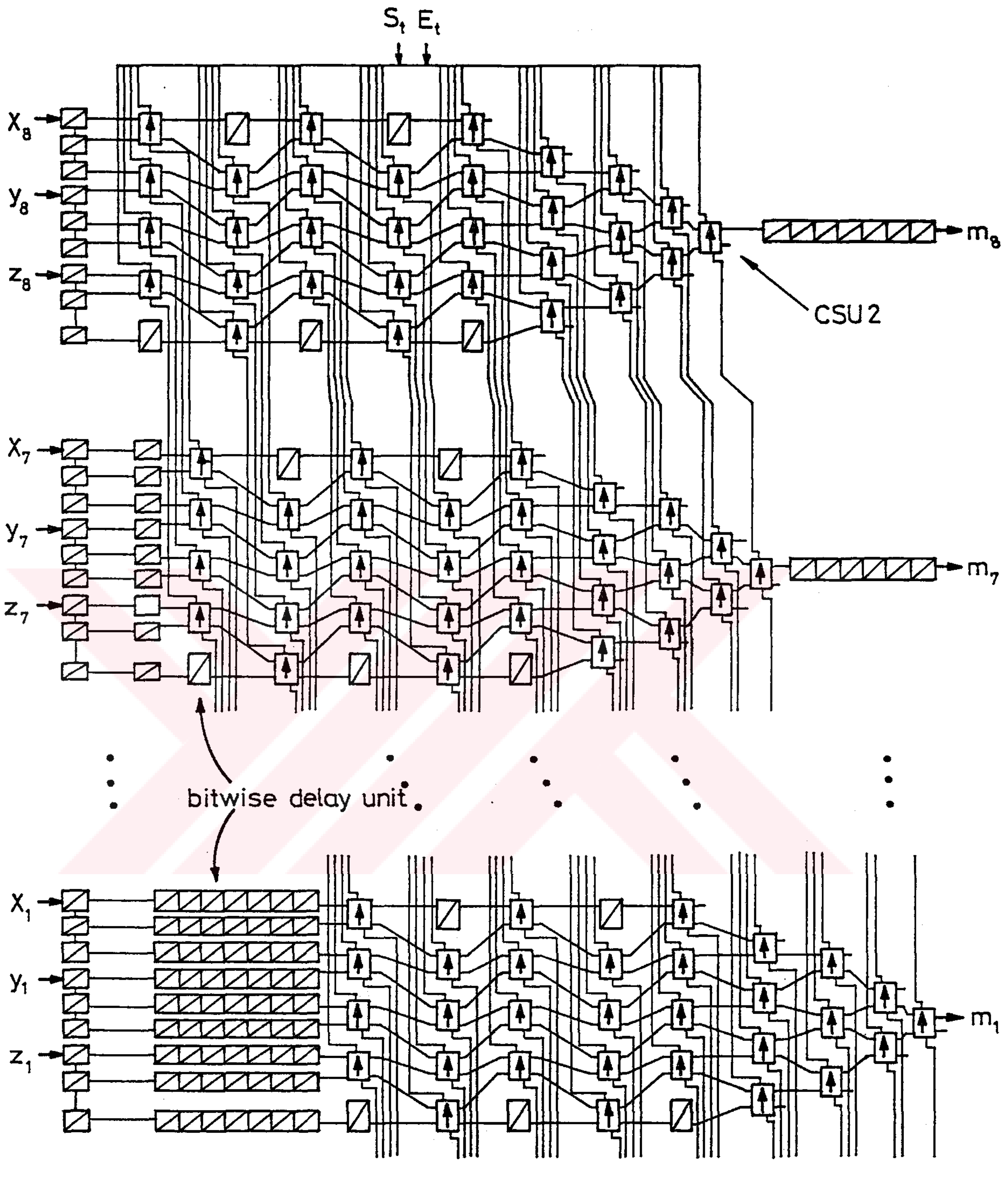
In order to achieve a real-time operating rate for the 1024×1024 resolution frames, the only way is to find a structure that has a very high throughput capability. The only solution we have found is the odd/even transposition sorting network configuration proposed by Oflazer [48]. In the following section, we will describe the real-time median filter network designed for $w = 9$ and $L = 8$ which is a variation of Oflazer's network that was designed for $w = 5$ and $L = 8$.

A median filter with very high throughput can be designed by interconnecting L number of the pipelined odd/even transposition sorter blocks in parallel and applying the systolic concepts at the bit-level. The block diagram of the overall network for $w = 9$ and $L = 8$ is shown in Fig. 3.5. In this network, the data enter in such a way that the most significant bits go to the first block, the second most significant bits to the second block and so on. There are nine 8-bit input data, and one 8-bit median is obtained per clock. At every clock, three new elements enter the network, corresponding to the new elements of a sliding 3×3 window.

The bitwise compare-and-swap unit, CSU2, used in this network is slightly different from CSU1 in such a way that the S and E parameters of CSU1 are external parameters in CSU2. These parameters are taken as inputs S_i and E_i (the cumulative subresult of the bitwise comparison of the more significant bits) from the upper block, used, updated and sent out as S_o and E_o (the new cumulative subresult) to the lower block. Hence, CSU2 does not have any storage or the reset input signal. The functional description of CSU2 is given in Fig. 3.6.

During the computation, CSU2 compares the inputs A_i and B_i and passes or swaps them conditionally depending on S_i and E_i. If $E_i = 1$ and $S_i = 0$, it passes or swaps the inputs so that larger one goes to A_o and the smaller to the B_o. On the other hand, if $E_i = 0$ and $S_i = 1$, CSU2 passes/swaps the inputs independent to the result of the comparison of the inputs. It also updates S_i and E_i and sends out as S_o and E_o for the comparison of the less significant bits.

The computation at each stage of a lower block starts after the corresponding stage of the upper neighborhood block completes its operation so that the S_i's and E_i's are ready at input of that stage. As a result of this,

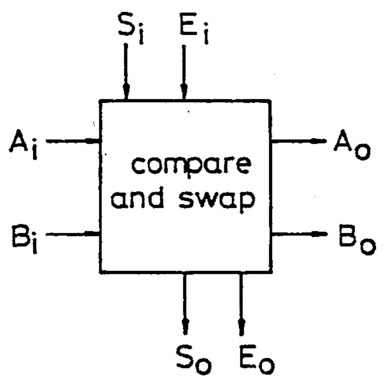


x_i, y_i, z_i : i th bits of the inputs corresponding the new elements in a 3x3 sliding window.

mi : i th bit of the median

S_t, E_t : Test inputs for testing of the blocks individually.

Figure 3.5: The real-time median filter network.



$$S_o = S_i + E_i \; (A_i < B_i)$$
$$E_o = E_i \; (A_i = B_i)$$
$$A_o = \text{if } S_o \text{ then } B_i \text{ else } A_i$$
$$B_o = \text{if } S_o \text{ then } A_i \text{ else } B_i$$

S_i and E_i indicate the cumulative subresult of the bitwise comparison of more significant bits while E_o and S_o indicate the new cumulative subresult.

Figure 3.6: The compare-and-swap unit-2 (CSU2).

each stage of a lower block must operate one clock later compared to the corresponding stage of the upper block. For the same reason, the outputs of a lower block are ready one clock later compared to the upper neighborhood block. In order to employ a proper timing, the pipeline delay units are used both at the input and the output of the network such that *k'th* block from top has $(k-1)$ delay units at its inputs and $(L-k)$ delay units at its output, where $k = 1, 2, ...L$ (see Fig. 3.5).

The throughput of the network is one full-word median per clock. The clock period is determined by the delay of one CSU2. Recent VLSI technology allows the implementation of CSU2 at a speed larger than the real-time rate for the 1024×1024 frames with $L = 8$.

3.4 Feasibility of VLSI Implementation

Both of the extensible and real-time median filter networks are regular arrays of bitwise compare-and-swap and delay units. Also, their internal communication schemes are simple and regular. This makes the VLSI implementations to be easy and straightforward. Furthermore, the testing of the VLSI chip may be easily accomplished by the *functional test* techniques [51], since the operations of the cells may be selectively probed by issuing proper test vectors. The testing operation includes the running of the chip with test vectors as inputs and the comparison of the outputs with expected results. The testing can be performed on-line or off-line, in any case, the comparison time is reasonably short. The generation of the expected results, which are the correct outputs, can be obtained by using a software sorter and a software median filter.

The extensible median filter chip can be tested by using a few hundreds test vectors. If we apply the same element to all of the inputs, then every CSU1 will act in equal state. On the other hand, the sorted inputs will cause all the CSU1's operating in swap state. Consequently, a set of inputs sorted in reverse order will make all the CSU1's operate in swap state. By this way the CSU1's will be tested in all the operation states.

The real-time median filter chip can be tested by using 12,288 ($8 \times 3 \times 2^9$) test vectors. The S_i's and E_i's at the top of the network are connected to external test control inputs, S_t and E_t respectively (see Fig. 3.5). These test control signals are used to isolate the operation of each block from each others, thus the number of test vectors reduced to 12,288. If these test inputs are not used then the number of test vectors will be a few millions. The *k'th* block can be tested individually with applying logic 1 to all of the upper blocks' inputs so that it receives the external S_t and E_t without any change. As a result, each block can be tested by using 1,536 (3×2^9) test vectors: all combinations of one-bit 9 inputs times 3, where 3 comes from the legal values of the pair S_iE_i (01, 00, 10).

The extensible network can be implemented with about 5 K transistors and packaged in 28 pins with 2×1 I/O multiplexing whereas the real-time median filter will consist of about 20 K transistors and has 40 pins.

One may choose to implement either the extensible median filter network or the real-time network for a large w, but this not preferable since the area is proportional to w^2. We have chosen $w = 9$, because it is the minimum and the most commonly used window size in two-dimensional median filtering applications.

In an attempt to increase the efficiency of the silicon area use, one may consider to design a network which utilizes the same compare-and-swap units (CSU1 and CSU2) in both modes by altering the interconnections via the external select signal. Examination of this possibility did not give the desired result: the chip area turned out to be larger because of the decreased regularity and increased interconnections.

We have examined a network with real-time and unlimited word-length operation capability which is obtained by modifying the Oflazer's network such that all of the S_i and E_i inputs of the upper most sorting block and S_o and E_o outputs of the lowest block are used as extension I/O's. But VLSI implementation of this network needs an extremely large number of I/O's and unfortunately there is no way to multiplex such a fast network I/O's without decreasing the speed. Hence, it is not feasible and we have not attempted to implement it.

We have also examined the implementation of the two networks in the same chip. The I/O's of the two networks need to be multiplexed. On the other hand, due to very long wiring paths from I/O pads to the networks' I/O's, the strong buffers are needed to drive the large wiring capacitive loads. All these increase the area and decrease the throughput of the networks resulting a reduced speed which makes the real-time operation critical. Also, while one of the networks operates the other one can not be used due to multiplexed I/O's, i.e. one of the networks will be idle at all times. However, the implementation of the networks as two single-chips will avoid these undesired results.

4. VLSI DESIGN METHODOLOGY

4.1 VLSI Design Approach

For the design of any kind of systems, a systematic approach is necessary; especially for the design of complex and large systems, the systematic approach becomes very important. For design of a chip, starting from the problem definition, a systematic way should be followed throughout the design steps. For system design in VLSI, there are some proposed approaches in the literature [21,46,52]. In fact, all approaches are based on dividing the task into four main subtasks which are the problem specification, behavioral description, circuit description, and layout description. The problem specification consists of the detailed definition of the problem and well defined constraints. The behavioral description is the algorithm found as the solution of the problem whereas the circuit and layout descriptions are the implementations of the algorithm in circuit and layout levels. Further divisions of those main subtasks, and their orders depend on the application and probably on the designer.

For our design, the two main subtasks, the problem specification and the behavioral description, are given in the previous chapters. Our problem is to design and implement a general purpose median filter which has the extensibility, unlimited word length and real-time operation capability. We have a solution to that problem in the form of two median filter algorithms that satisfy the constraints. The algorithms are described in terms of the network block diagram representations with functional descriptions. A testing strategy is also determined in that step. All these complete the two main subtasks. Now, the remaining parts are the circuit and layout implementations. Since our networks are modular and have regular and simple communication schemes, it is quite easy to divide the circuit and layout implementation in further subparts: we can design the circuits of the units (compare-and-swap unit, delay unit, and I/O cells) instead of designing the full network at a time. Obviously, the same partitioning can be performed in the layout level. All these point out the importance of the structured algorithms (e.g. systolic algorithms) since the complexity and difficulty of the tasks at circuit and layout levels are determined by the algorithm.

The flow chart given in Fig. 4.1 shows the all steps which we have followed throughout the design of the extensible and the real-time median filter chips. After completing the problem and behavioral description parts, it is essential to construct a floor plan for the chip. The floor plan at this step is not necessarily strict but it is rather flexible for modifications in layout steps. The floor plan strongly depends on the structure of the algorithm, especially on the data flow and control flow of the algorithm. The determination of the hierarchical levels is to classify the groups of the cells and combinations of them level by level such that the lowest level consists of probably the smallest cells and the second level has the units that are the combinations of a number of subcells and so on. Obviously, the top level in the hierarchy is the overall chip.

The next step in the approach is the logic design of cells. If the cells are standard gates or consisting of a few them, then the logic expressions of the cell parameters and their representations in terms of gates need to be determined so that, in the circuit design, the only task will be to replace the gates with their known equivalent circuits. However, if the cells consist of rather large number of gates, then the cell parameters must be expressed as logical functions but the gate representation is not essential because the circuit implementations in transistor level of such cells may be simpler than doing that in gate level. The designer must consider the testing of the chip at this step so that the extra logic circuits are included for testing if necessary.

For the circuit implementation of cells, the designer should choose a circuit design technique in a particular technology (e.g, NMOS or CMOS) which shows the best performance for his application. Each technology offers some circuit techniques employing different properties. The performance of the technologies and circuit techniques vary depending on the design criteria. For example, CMOS is preferable for the designs requiring a low power consumption. We will discuss the selection of a suitable circuit design technique in detail in the next section.

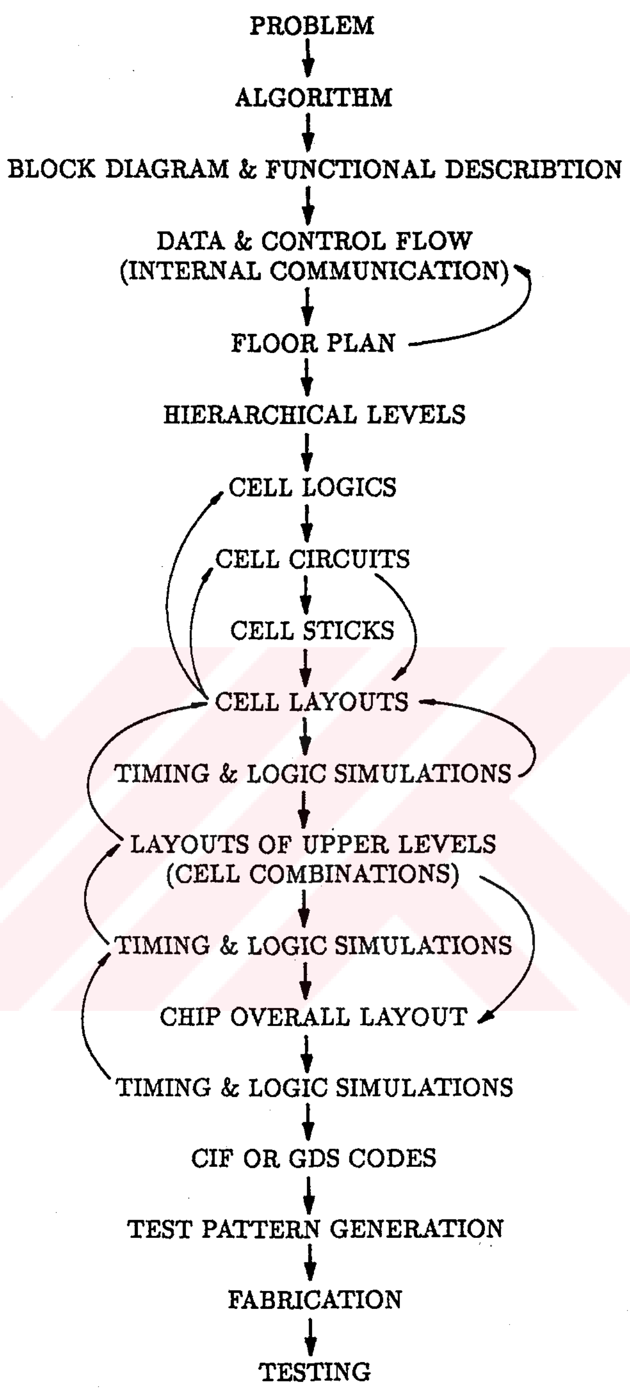


Figure 4.1: The VLSI design steps.

The cell circuits can be mapped into layout either directly or after representing them in stick forms. For a well organized layout, the sticks may be helpful especially to match the constructed floor plan. But in general, the circuits can be directly converted to layout forms by experience. In any way, during the generation of cell layouts, one should care about the physical behavior of the layout (transistor sizes, the resistive and capacitive effects, coupling, etc.). Furthermore, the cell boundaries for cell combinations, the data and control flow between them, and the floor plan must be also considered.

After the generation of cell layouts, the next step is the simulation. The circuits corresponding to the layouts are first extracted and then simulated by using software simulators. The simulation can be performed in two steps: timing analysis and logic analysis. But these two steps can be handled in a single step if there are a few cell parameters. For example, an OR gate can be simulated in a single step, since the parameters are two or three inputs and one output, so that the timing analysis can include also the logic test. If there are reasonably many cell parameters, timing and logic simulations must be performed separately by a timing simulator and a logic simulator, since the logic test with timing simulator will be costly in terms of simulator run-time. According to our experience, the mostly used feedback loop is the cell simulations to cell layouts because the most of the optimizations (speed, area, power, etc.) are performed at the cell level since it is the lowest hierarchical level.

After the cell layouts, the following design step is the layouts of upper hierarchical levels. The simulation at these levels are not essential because all subcells are simulated at the lowest (or lower) level. But if the designer worries about the operation of cells when they are combined, he must perform the necessary simulations. However, if the simulations at lower levels are performed with consideration of the combinations (e.g., some extra capacitances may be need to add to the output capacitances of the cell being simulated), simulations at upper levels are not necessary.

As a last step of the layout level, the overall chip layout is obtained including the I/O cells (buffers, pad drivers, pads, etc.) Finally, a timing and a logic simulation of the chip must be performed. If the simulator run-time is feasible, the logic simulation can be performed by using the test vectors which will be used at the physical testing of the chip. But in general, the number of test vectors is too large so that the simulation by using them is not possible. In fact, since all the voltages and currents within the chip are observable in the simulation, the simulation with test vectors are not so essential. Consequently, if the test vectors will be used in the simulations, they are generated at that step.

The last two tasks are the generation of the CIF or GDS-II codes and the test patterns (test vectors). The CIF or GDS-II codes are generated by the layout editor automatically. The test vectors can be obtained by either a test pattern generator or manually. In general, the test vectors are generated manually since a general purpose test pattern generator is a very expensive tool. However, one can write a software program to generate the test vectors for his own purpose. The chip testing can be performed either by the manufacturer or by the designer. If the chip will be tested by the manufacturer, the test vectors must be generated before the fabrication.

4.2 Circuit Technique

In integrated circuit design, especially for digital applications, NMOS technology has been used with a growing potential since early of 1960's. Due to the increase in density of devices on chips, the power dissipation and complexity of chips have become a serious problem. Since the late 1960's, CMOS technology is being used to overcome these problems by possessing a reasonable low power consumption, a rather easy design and a comparable speed with NMOS [55]. Also, the noise immunity of the CMOS is better than that of NMOS. For these reasons, we have implemented the median filter chips in CMOS technology. In particular, it is a 3-micron double metal double poly n-well process.

During the implementation of circuits in CMOS, one should consider the *latchup,* the *charge sharing,* the *body effect*, and the *noise margins.* [46,56,53,54]. Note that, these problems except the latchup are also present in circuits implemented in any technology (e.g. NMOS).

The latchup problem is caused by the improperly biased parasitic bipolar transistors formed within the CMOS structure. In some cases, these parasitic transistors are biased in such a way that Vdd and Vss (positive and negative power sources) are shorted resulting in destruction of the chip due to excessive current flow. In order to overcome this problem, one should bias the parasitic transistors so that they do not cause to the short circuit. In practice, one can reduce the chances of the latchup in n-well CMOS by a proper number of connections from the p-substrate to Vdd and from the n-well to Vss, and increasing the distance between n-well and n-diffusion.

If the output capacitance of a subcircuit, say a gate, is not comparable with the load capacitance (sum of input capacitances of the gates derived by that gate) then a charge sharing occurs between the capacitances resulting in that the output voltage of the gate oscillates before settling in its final value. On the other hand, the asynchronized inputs of a gate can cause the charge sharing at the output of the gate. In other words, the charge sharing creates some *glitches* which may cause wrong logical output. However, the charge sharing problem can be avoided by making the output capacitance and load capacitance to be comparable, and synchronizing the input signals by means of weak buffers.

If the source voltage of a p/n-transistor is not equal to Vdd/Vss, then the threshold voltage of that transistor changes depending on the source voltage since the threshold voltage is a function of the source to substrate voltage. In many circuits, some transistors are cascoded so that the source of every transistor is not connected to Vss. This causes the change of expected results especially in timing, due to the body effect. The only precaution for that problem is to increase the Vss connections, and to decrease the internal node capacitances as much as possible. Furthermore, in timing simulations, a worst case approach is essential.

The noise margin parameter is a measure of allowable noise voltage on the input of a circuit such that the output is not affected. In general the high and low noise margins are equalized by the equal low and high going transition times. Also, the selection of the logic style is important to maximize the noise margins.

In CMOS technology, there are some circuit implementation techniques called *logic styles.* These are complementary (standard) CMOS logic, pseudo NMOS logic, dynamic CMOS logic, clocked CMOS logic, CMOS domino logic, cascade voltage switch logic (CSVL), and pass transistor logic [46,53,54]. Each logic style possesses some different properties. In other words, the optimization of the cost function (speed, area, power, complexity, etc.) is different. For example, dynamic CMOS logic is the choice for fast circuits but its timing is critical. Hence, the designer must choose the best style for his particular application. In our design, the standard CMOS logic style is used since this logic style possesses a more reliable timing, a rather easier mapping in layout, a lower power consumption, and a better noise immunity compared to the other styles. Furthermore, its area and speed performances are comparable with the other styles.

4.3 VLSI Tools

At the layout step of a VLSI design, it is necessary to use some software tools in order to edit the layout of a chip which consists of thousands of transistors, and to perform its simulations. In our design, we have used the Berkeley CAD Tools of University of California and VLSI Tools of University of Washington [57,58] running under the Unix operating system on SUN workstations. For editing of layouts, we have used MAGIC which is an interactive editor that can make on-line design rule check by using information in the design rule file. Thus the designer must provide the design rules that are prescribed by the manufacturer. Unfortunately, MAGIC can not realign the symbolic layers if the design rule file is changed, so the layout must be reedited or modified according to the new design rules. MAGIC allows editing the layouts hierarchically, and extracting the circuit of the layout.

In order to perform simulation of a cell, first the circuit of the cell is extracted, and then it is further processed by some intermediate tools [57,58]. There are three simulators: SPICE, RNL, and ESIM. SPICE can be used for the timing simulations with a high accuracy. Before running SPICE, extracted circuit must be processed by EXT2SIM and SIM2SPICE tools to convert the extracted circuit to SPICE format. In addition, the model parameters of the transistors, and the timing commands should be entered into the SPICE input file. The results of SPICE simulations are either observed on the screen by using SPCVIEW and SPICE2SUMMARY tools or plotted by SPCPLOT tool. RNL is used for logic simulations with approximate timing. In order to use RNL, extracted circuit is processed by PRESIM. If RNL is used in interactive mode, the control and timing commands are entered interactively, otherwise RNL needs a control file consisting of control commands, and a time commands file consisting of input signals ordered in time. One can use GEN_TIME and GEN_CONTROL to generate the control and timing command files, respectively. ESIM is used for logic simulations of only combinational circuits, but it is very fast since it performs switch-level simulation. The results of simulations with RNL and ESIM are stored in ordinary files, thus user can read and analyze the results from these files.

After completing the layout editing and simulation tasks, the final layout is converted to CIF or GDS-II codes by using MAGIC. These codes are used to generate the masks of the layers for fabrication of the chip. If it is required, one can plot the masks from codes by using CIFPLOT tool, and convert the codes back to layout format by using MAGIC.

5. VLSI IMPLEMENTATION

5.1 Implementation of the Extensible Median Filter

In the description of the algorithm of the extensible median filter, it is proposed to multiplex the I/O's of the extensible median filter chip to reduce the pin number from 40 to 28. However, in the layout implementation of the chip, it is observed that there is no increase in the area when 40 pads are used. This is due to fact that there are empty spaces between the pads when the 28 pads are placed. For this reason, we have used 40 pads in the extensible median filter chip. Since there is no multiplexing in this case, the speed of the chip is increased by about 30 percent, and the data loading and reading operations are simplified. The only disadvantage is increased package size in the case of "dip" package. In this section, the implementation of the 40 pin extensible median filter chip with no multiplexing is described. Here, let's point that one can easily add multiplexers around the network and reduce the pin number to 28.

As the first step, the floor plan of the chip is designed (Fig. 5.1). Then, the power distribution scheme of the chip is determined. A clock distribution scheme is selected such that the lengths of the paths that the clock signals propagate are equal and small enough so that the clock skew is not a serious problem. Hence, a single clock buffer is used for each clock phase, instead of a distributed clock buffering scheme [59]. The power and clock distribution scheme is shown in Fig. 5.2.

In the following subsections, logic expression(s), the circuit and layout descriptions, and the timing and/or logic simulations will be described for each unit of the chip. The complete layout and the logic simulation results of the chip will be given. Finally, an overview of the chip and its testing will be presented.

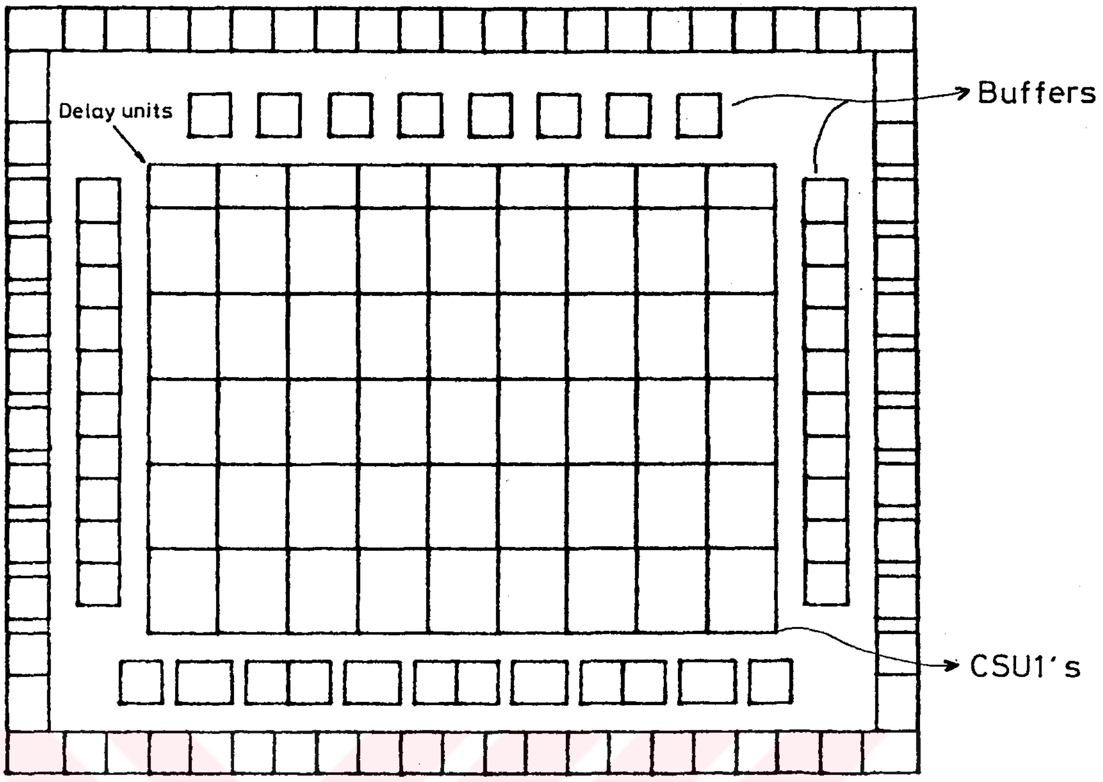


Figure 5.1: The floor plan.

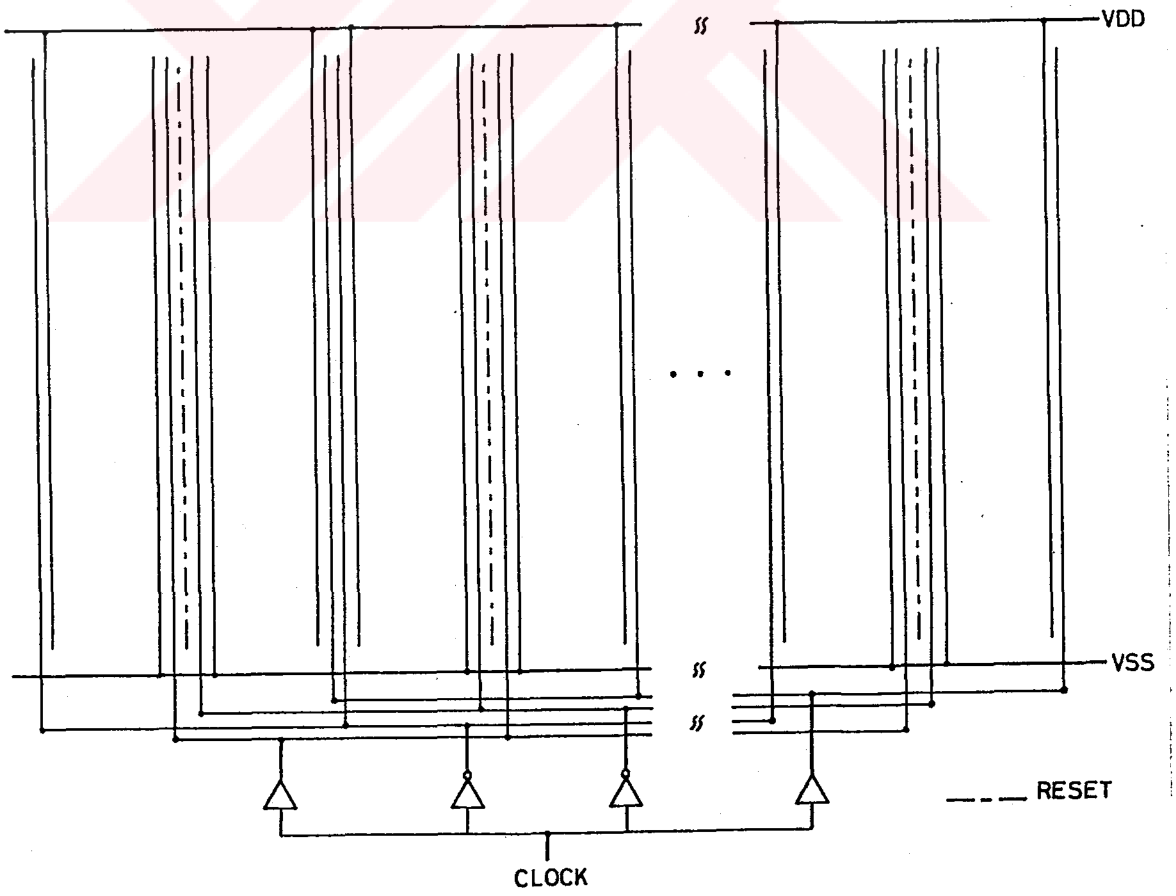


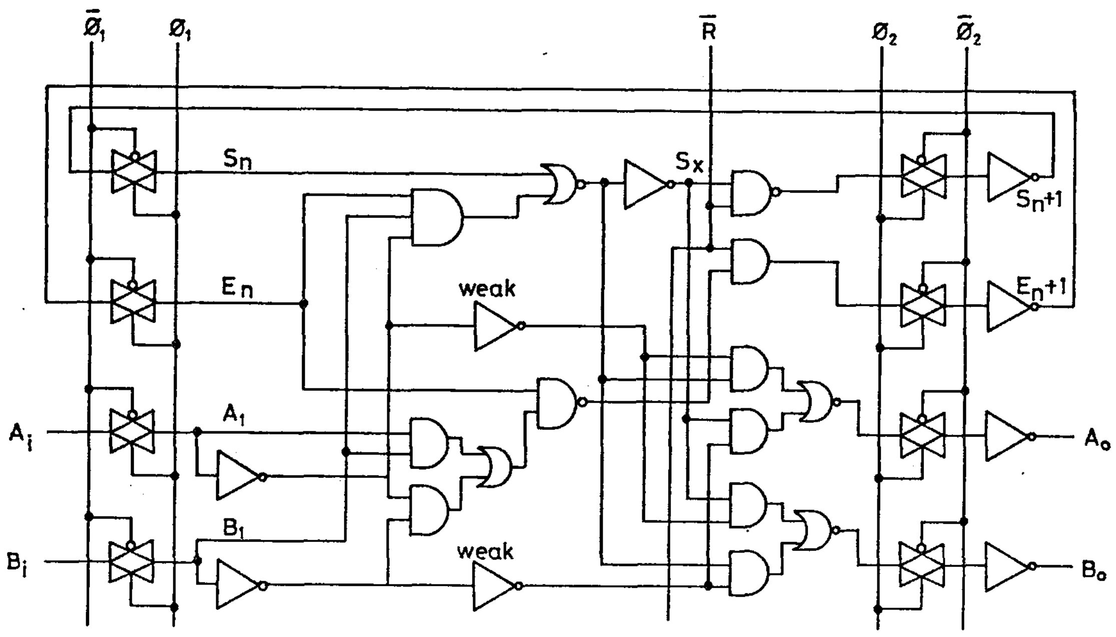
Figure 5.2: The power and clock distribution scheme.

5.1.1 Compare-and-Swap Unit-1 (CSU1)

The logic expressions of the CSU1 are obtained from the state diagram given in Fig. 3.3. They are minimized by using the *carnough-map* method [60]. The resultant logic expressions and the circuit diagram of the CSU1 are given in Fig. 5.3. In this circuit, inputs enter the cell during ϕ_1 ($\phi = 1$, $\phi^- = 0$). The computation of the outputs starts as soon as the inputs are latched. When the outputs are computed, they are latched at the outputs of the inverters during ϕ^-. In the circuit, the *weak buffers* are used to prevent the charge sharing. The inverters at the output of the circuit are necessary to latch the outputs. These inverters are dynamic latches so they can not hold data more than about 1 millisecond without refreshing [21]. This means that the clock frequency can not be less than 1 KHz.

The circuit of the CSU1 is converted to the layout by considering its geometry in the floor plan. After the equivalent circuit of the layout is extracted and simulated, the layout is modified to increase the speed, to prevent the charge sharing and to have equal rise and fall times by resizing the proper transistors. The maximum average current is about 0.7 mA. The power lines of the layout are wide enough so that the *metal migration* does not take place (the allowable maximum current density of each layer is determined by the manufacturer). The final form of the layout of the CSU1 is shown in Fig. 5.4. This layout consists of 39 p-channel and 39 n-channel transistors.

The timing simulations of the CSU1 are performed with different clock frequencies and with different rise and fall times by using SPICE. Here, we give the input and the output waveforms for 40 MHz clock which is the maximum clock frequency (Fig. 5.5). In these waveforms, the rise and fall times of the clock, inputs, and outputs are about 5 nsec. However, the rise and fall times may be larger than 5 nsec for low operation frequencies if it is desired. The ripples on the waveforms are less than 1 V, thus they are negligible. The logic simulation of the CSU1 is performed by using RNL for all possible inputs. As an example, the results of the logic simulation for three different 4-bit input words are given in Table. 5.1. For the sake of the clearness, the clocks and time steps are not shown. As a result, the timing and the logic simulations show that the CSU1 with the given layout functions correctly.



$S_{n+1} = (S_n + E_n\bar{A}_i B_i)\bar{R}$ $\quad A_o = \bar{S}_x A_i + S_x B_i$

$E_{n+1} = E_n(A_iB_i + \bar{A}_i\bar{B}_i) + R$ $\quad B_o = S_x A_i + \bar{S}_x B_i$ $\quad S_x = (S_n + E_n\bar{A}_i B_i)$

Figure 5.3: Circuit diagram of the CSU1.

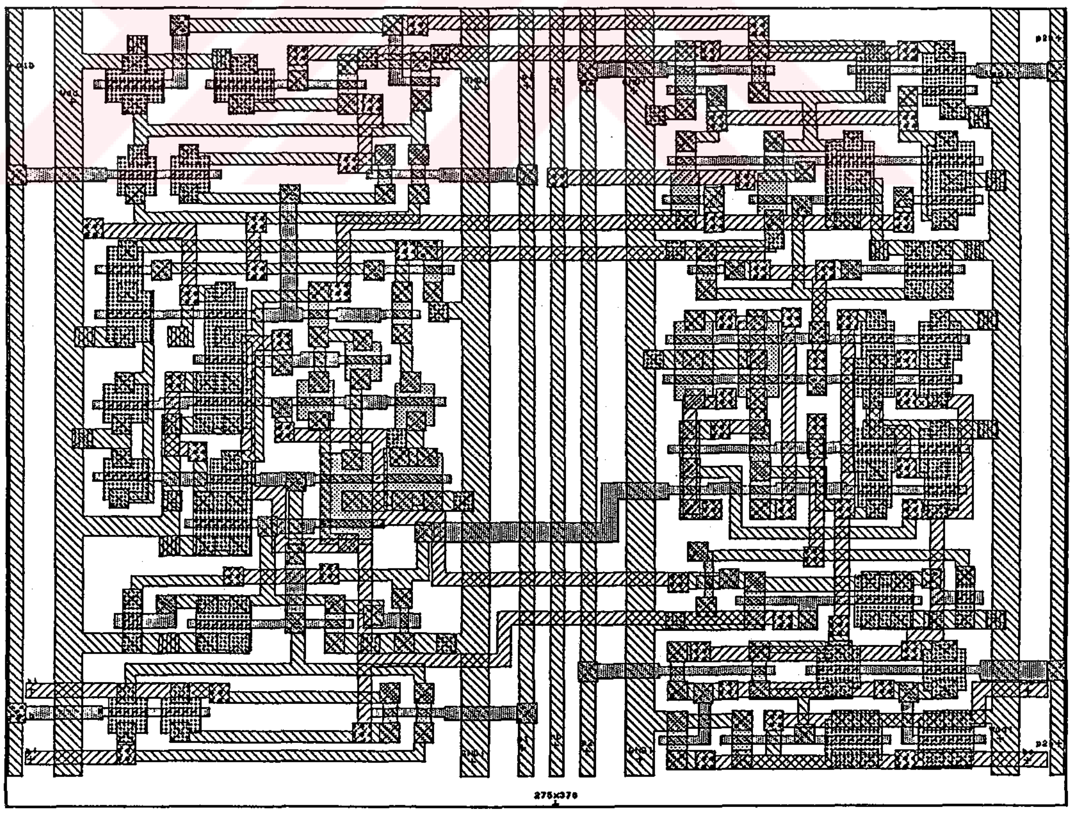


Figure 5.4: Layout of the CSU1.

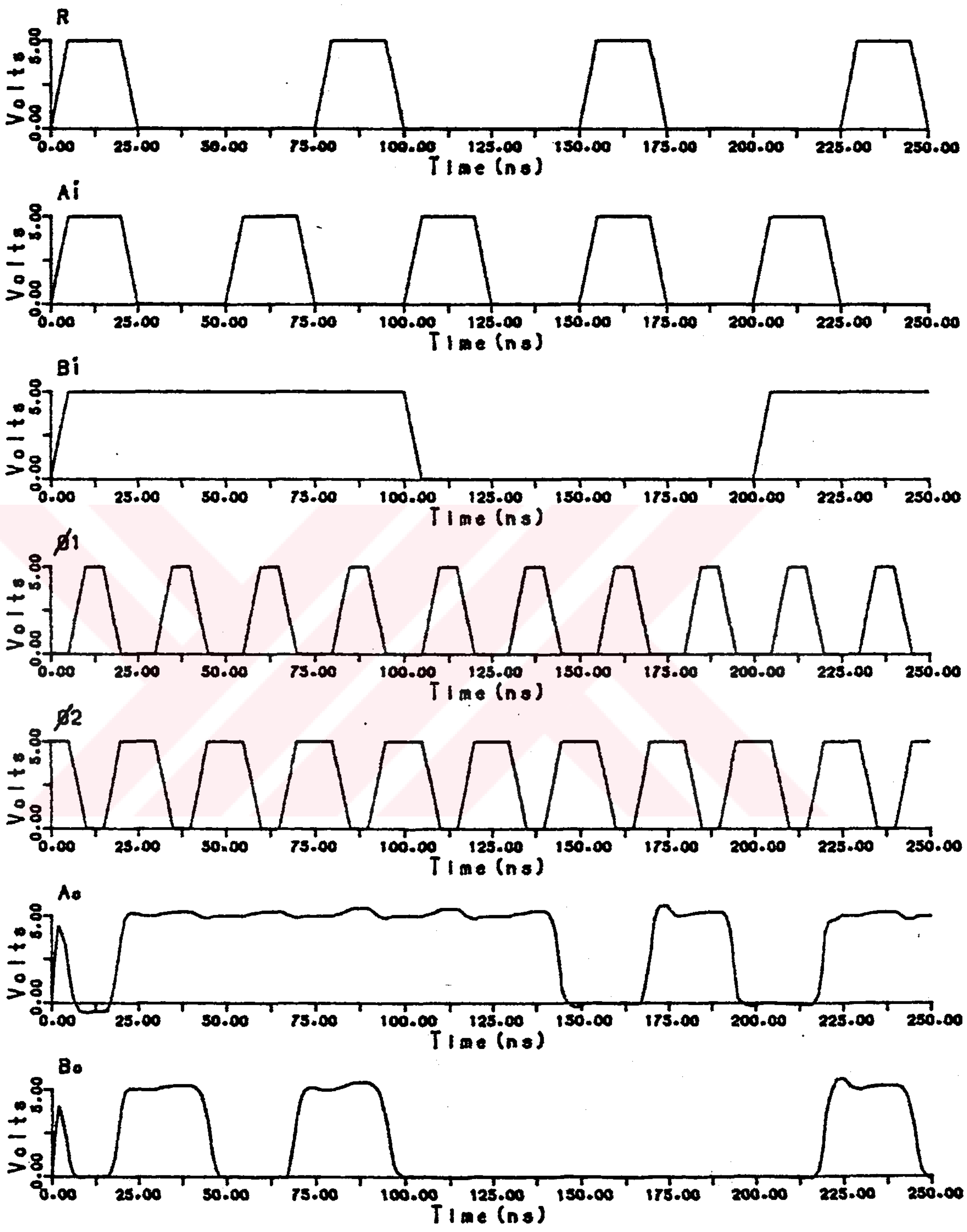


Figure 5.5: Timing simulation results of the CSU1.

R	A_i	B_i	A_o	B_o
0	0	0	0	0
0	1	1	1	1
0	0	0	0	0
1	1	1	1	1
0	1	0	1	0
0	0	1	0	1
0	1	0	1	0
1	0	1	0	1
0	1	1	1	1
0	0	1	1	0
0	1	0	0	1
1	0	1	1	0

Table 5.1: Logic simulation results of the CSU1.

5.1.2 Delay Unit

Since the reset signal must propagate through the delay units one stage per cycle, a shift register cell can be used as the delay unit. The static D-type master-slave flip-flop is chosen as the delay unit in order to have a reliable shift operation [60,53]. The circuit diagram and the layout of the shift register cell are shown in Fig. 5.6 and Fig. 5.7. In this circuit, the input is latched at the output of the master flip-flop during $\phi = 1$. The master flip-flop is disabled when ϕ is low. The input is latched at the output of the slave flip-flop during $\phi^- = 1$. The output of the master flip-flop is buffered and sent to the compare-and-swap stage as the complement of the reset signal. The results of the timing simulations of the cell are given in Fig. 5.8.

5.1.3 Clock Buffers

The total capacitive load of ϕ and ϕ^- lines are about 15 pF and 20 pF respectively. Four buffers are used to drive these loads with 100 percent tolerances: two oninverting buffers for ϕ, and two inverting buffers for ϕ^-. The noninverting buffer can drive a 15 pF load with the 2.5 nsec rise and fall times whereas the inverting buffer can drive a 20 pF load with the 2.5 nsec rise and fall times. In order to minimize the delay of the buffers, the number of stages and their ratios are calculated [61,20]: the noninverting buffer has 4 stages with 3.6 stage ratio while the inverting buffer has the 5 stages with 2.9

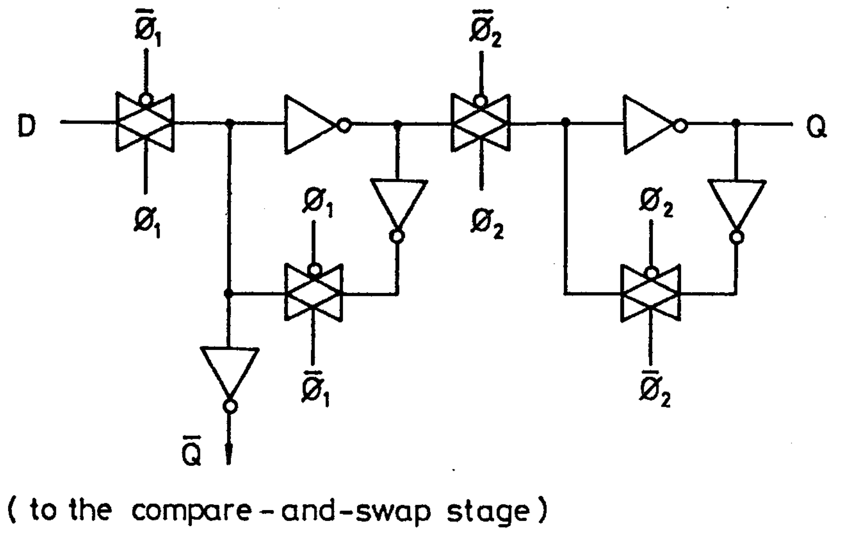


Figure 5.6: Circuit diagram of the shift register cell.

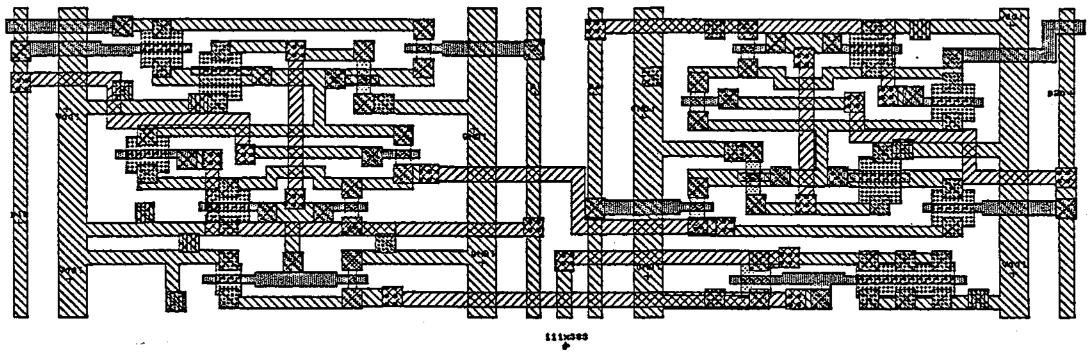


Figure 5.7: Layout of the shift register cell.

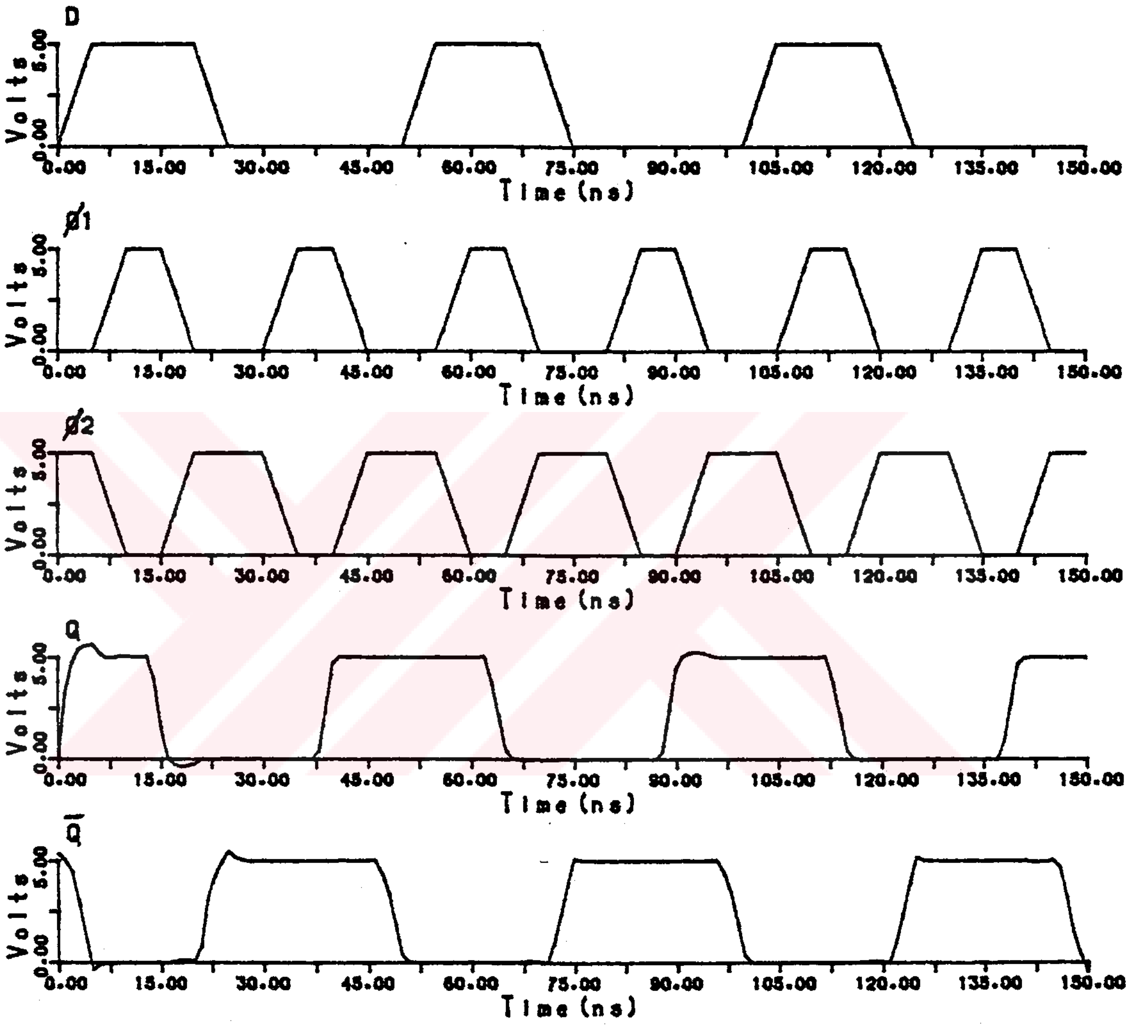
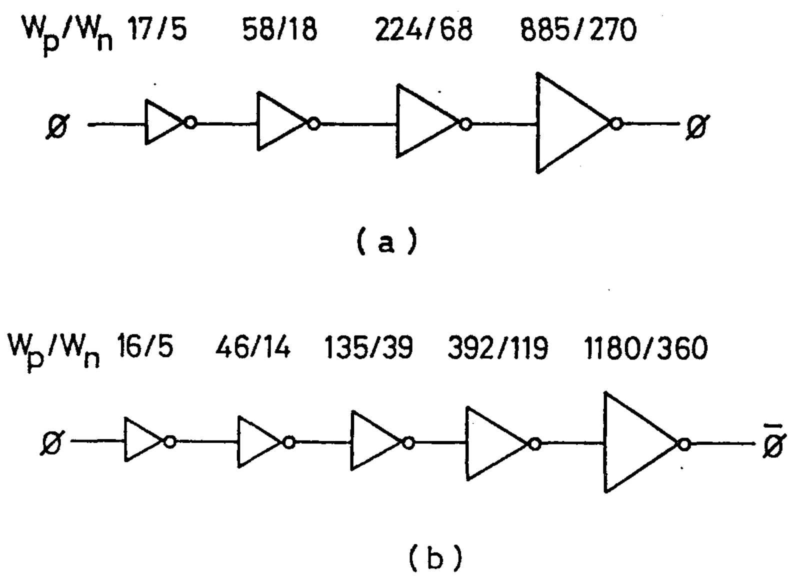


Figure 5.8: Timing simulation results of the shift register cell.



W_p/W_n 17/5 58/18 224/68 885/270
Ø
Ø
(a)
W_p/W_n 16/5 46/14 135/39 392/119 1180/360
Ø
$\bar{Ø}$
(b)

Figure 5.9: Circuit diagrams of the clock buffers.

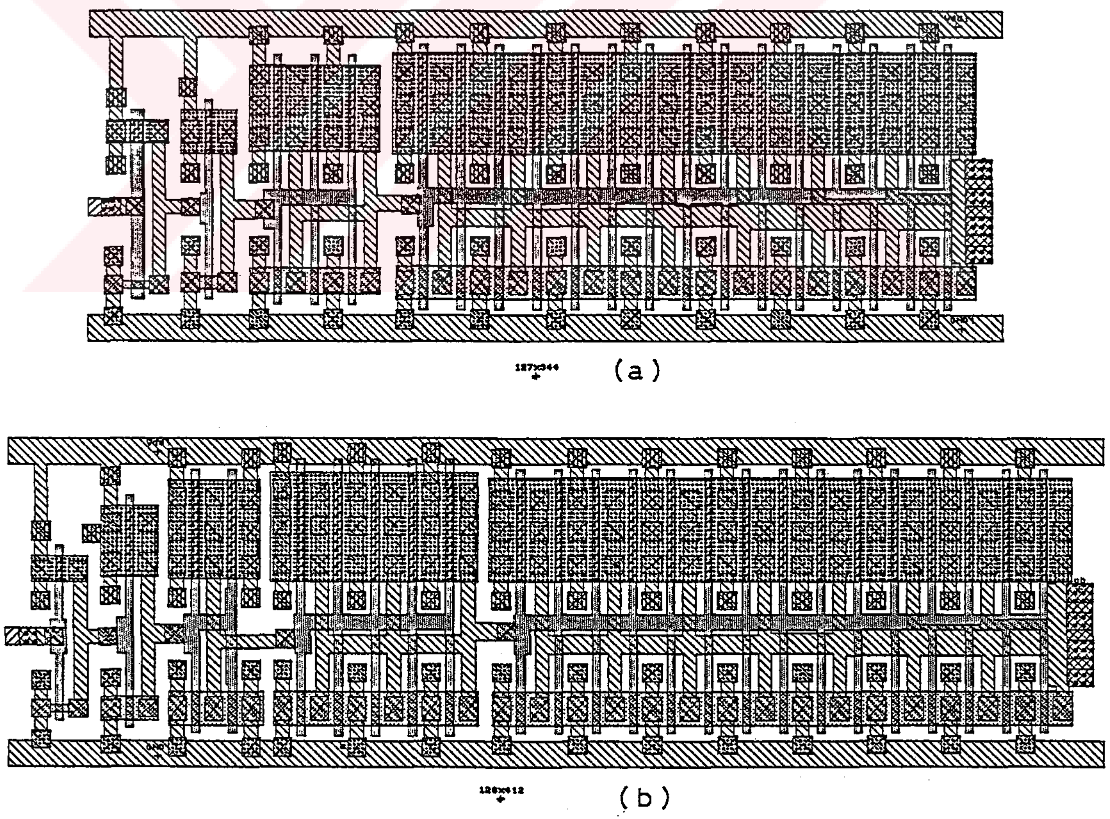


Figure 5.10: Layouts of the clock buffers: a) noninverting, b) inverting.

stage ratio. The circuit diagrams and the transistor ratios of each stage of the buffers are given in Fig. 5.9, and their layouts are shown in Fig. 5.10. The timing simulations of the buffers with their loads are performed for 50 MHz and 20 MHz (Fig. 5.11). According to the timing simulations, the delay of each buffer is about 7.5 nsec with a clock skew about 0.5 nsec between ϕ and ϕ^-.

5.1.4 Chip Overall Layout

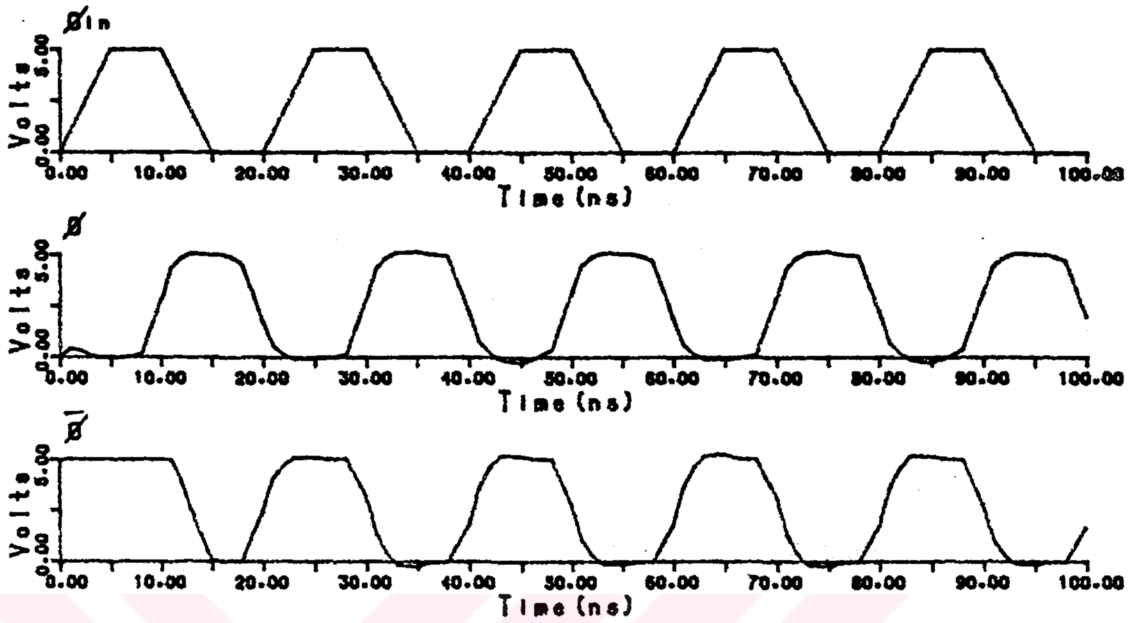
After completing the layouts and simulations of the cells (CSU1, one bit register, and clock buffers), the cells are combined hierarchically to form the overall layout of the network: the first and second stages of the network are obtained by combining five CSU1's for each. These two stages are used to form the sorter which consists of nine stages. The nine bit shift register is obtained as an array of one bit shift register cells. Finally, the clock buffers are placed at two sides of the network, and 40 pad cells are placed around the network. The pad cells are supported by the manufacturer as black box standard cells with given I/O connection schemes and parameters (size, delay, etc.). The complete layout of the chip is shown in Fig. 5.12.

The logic simulations of the chip are performed with the test vectors which will be also used for the physical testing of the chip after fabrication. The test vectors, which consists of the sorted, reverse sorted and equal set of elements, are applied to the inputs of the chip, and the generated outputs are found to be matching to the expected results. The outputs of the chip corresponding to the inputs which enter at k'*th* clock cycle are ready at the output pads at end of the $k + 8$'*th* clock cycle. The logic simulation results are given in Fig. 5.13 (also see Fig. 5.14 for signal labels).

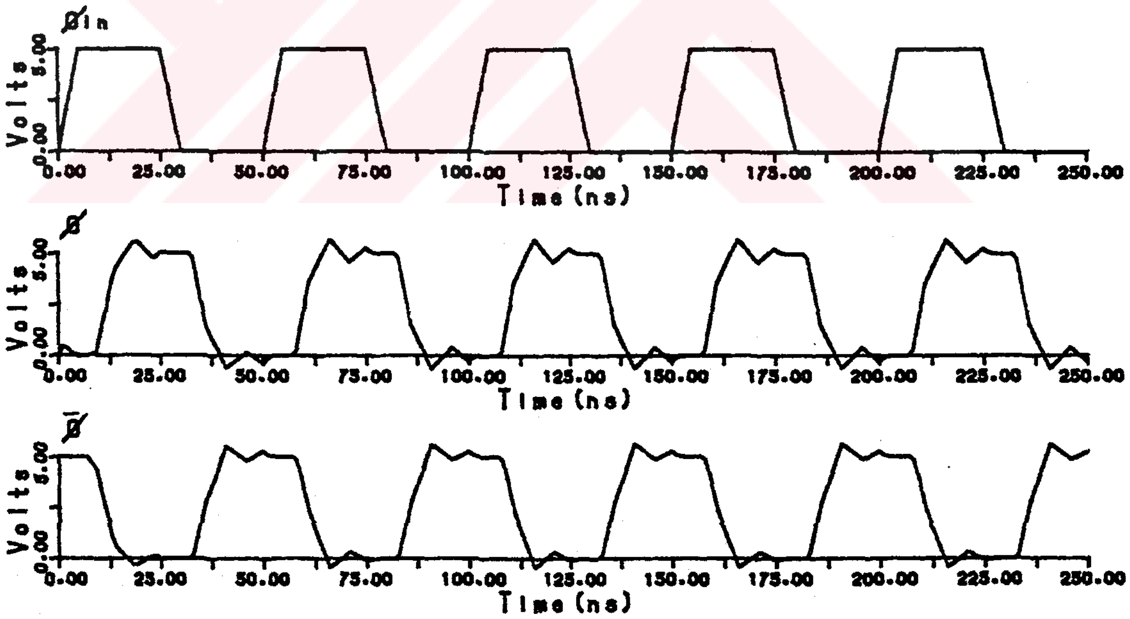
5.1.5 An Overview of the Extensible Median Filter Chip

In this section, we will give the information about how the chip can be used. The extensible median filter chip has 40 pins: the pin configuration is shown in Fig. 5.14. The pin labels are given in Table. 5.2.

The timing diagrams of the data load and read operations for the extensible median filter chip are given in Fig. 5.15. The input data must be ready



(a)



(b)

Figure 5.11: Timing simulation results of the clock buffers: a) for 50 MHz, b) for 20 MHz.

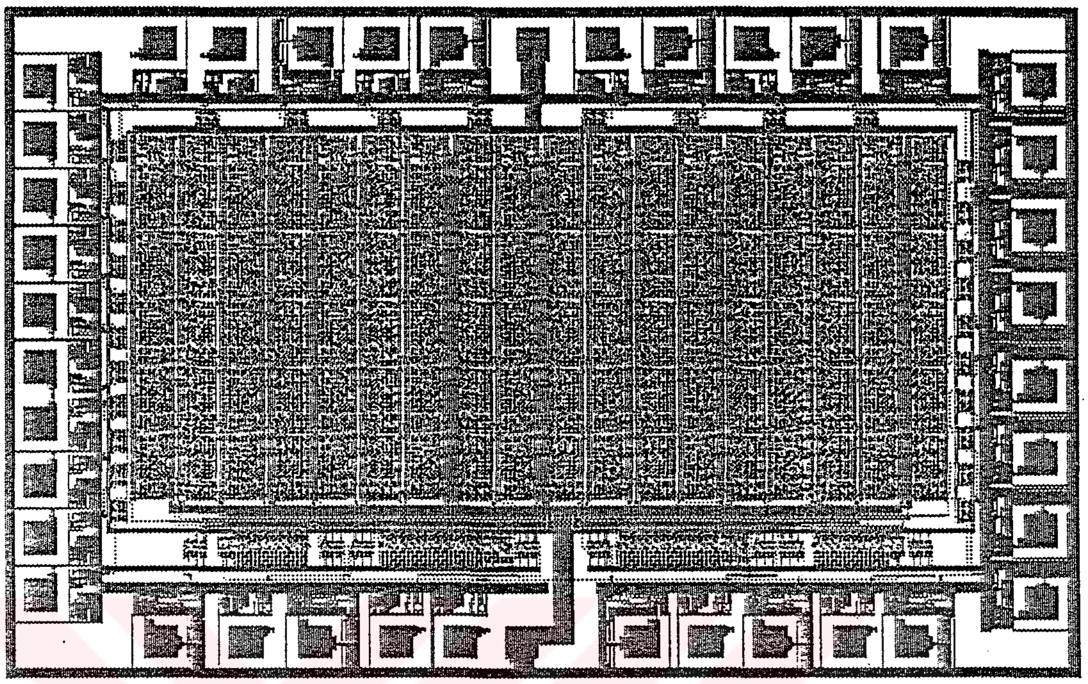


Figure 5.12: Complete layout of the extensible median filter chip.

LABEL	FUNCTION
RI	RESET INPUT
DI1-DI10	DATA INPUTS
DO1-DO10	DATA OUTPUTS
UI1-4	UPPER EXTENSION INPUTS
UO1-4	UPPER EXTENSION OUTPUTS
LI1-4	LOWER EXTENSION INPUTS
LO1-4	LOWER EXTENSION OUTPUTS
CLK	CLOCK INPUT
VSS	0 V POWER SUPPLY
VDD	5 V POWER SUPPLY

Table 5.2: Pin labels of the extensible median filter chip.

```
RI DI1-10 UI1-234 LI1-234 O1-10 UO1-234 LO1-234

1 0000000000 1111 0000 1111111101 1111 0111
0 0000000010 1111 0000 1111111101 0111 0111
0 0000000110 1111 0000 1111110101 1111 0011
0 0000001110 1111 0000 1111110101 1011 0011
0 0000011110 1111 0000 1111010101 1111 0001
0 0000111110 1111 0000 1111010101 1101 0001
0 0001111110 1111 0000 1101010101 1111 0000
0 0011111110 1111 0000 1101010101 1110 0000
0 0111111110 1111 0000 0000000000 1111 0000
0 1111111110 1111 0000 1000000000 1111 0000
1 0000000000 1111 0000 1100000000 1111 0000
0 1000000000 1111 0000 1110000000 1111 0000
0 1100000000 1111 0000 1111000000 1111 0000
0 1110000000 1111 0000 1111100000 1111 0000
0 1111000000 1111 0000 1111110000 1111 0000
0 1111100000 1111 0000 1111111000 1111 0000
0 1111110000 1111 0000 1111111100 1111 0000
0 1111111000 1111 0000 1111111110 1111 0000
0 1111111100 1111 0000 0000000000 1111 0000
0 1111111110 1111 0000 1000000000 1111 0000
1 0000000000 1111 0000 1100000000 1111 0000
0 0000000000 1111 0000 1110000000 1111 0000
0 1111111110 1111 0000 1111000000 1111 0000
0 1111111110 1111 0000 1111100000 1111 0000
0 0000000000 1111 0000 1111110000 1111 0000
0 1111111110 1111 0000 1111111000 1111 0000
0 0000000000 1111 0000 1111111100 1111 0000
0 0000000000 1111 0000 1111111110 1111 0000
0 1111111110 1111 0000 0000000000 1111 0000
0 1111111110 1111 0000 0000000000 1111 0000
1 0000000000 0000 0000 1111111110 0000 0000
0 0000000000 0000 0000 1111111110 0000 0000
0 0000000001 1111 1111 0000000001 1111 0111
0 0000000001 1111 1111 1111111111 1111 0011
0 0000000000 0000 0000 0000000000 0000 0000
0 1111111110 0000 0000 0000000000 0000 1000
0 1111111111 1111 1111 1111111111 1111 1001
0 1111111111 1111 1111 1111111111 1111 1100
0 1111111110 0000 0000 0000000000 0000 1100
0 1111111110 0000 0000 0010001000 0000 1110
1 0000000001 1111 1111 1000100010 1111 0110
0 0000000001 1111 1111 1000100010 1111 0111
0 0000000000 0000 0000 0010001000 0000 0011
0 0000000000 0000 0000 0111011101 0000 0011
0 0000000001 1111 1111 1101110111 1111 0001
0 1111111111 1111 1111 1101110111 1111 1001
0 1111111110 0000 0000 0111011101 0000 1000
0 1111111110 0000 0000 0111011101 0000 1100
0 1111111111 1111 1111 1000100010 1111 1100
0 1111111111 1111 1111 1000100010 1111 1110
```

Figure 5.13: Logic simulation results of the extensible median filter chip.

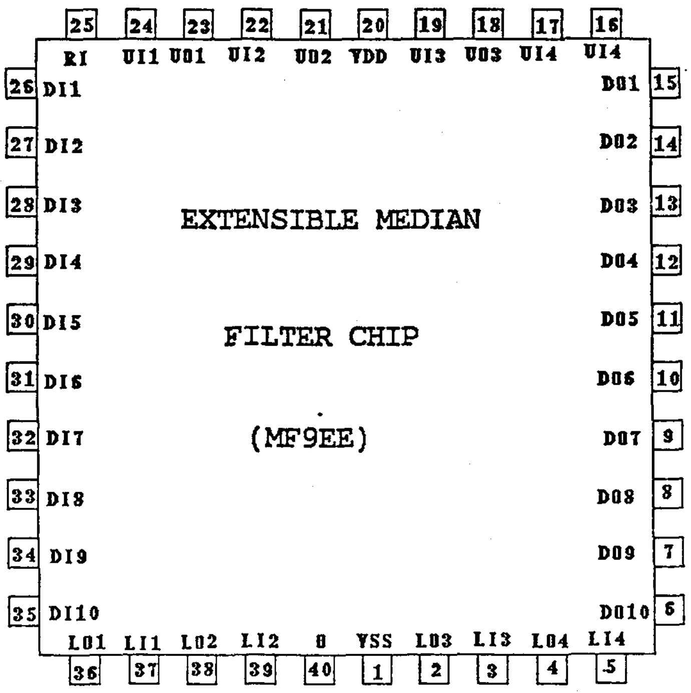


Figure 5.14: Pin configuration of the extensible median filter chip.

at the input pins while the clock is rising. Since there is an extra 7.5 nsec delay at clock signals due to delay of the clock buffers, input data signals may rise/fall while the clock is rising. But, the input signals must not change before the clock falls to logic low. The reset input signal, RI, must be at logic 1 during the last clock cycle of every word, and at logic 0 during the other clock cycles. For example, if the word length is 4-bit, then RI will be the sequence, 000100010.... At the beginning of running of the chip, the chip must be reset by entering logic 1 to RI for at least one clock cycle. The outputs are ready at the end of the clock cycle, and they do not change until the end of the next clock cycle.

The Normal Operation Mode

If the extensible median filter chip is used to find the medians of 9 elements, which is the normal operation mode, the input data are loaded from the first nine data inputs (DI1, DI2,..., DI9) (Table. 5.3). The upper extension inputs (UI1, UI2, UI3, and UI4) must be connected to logic 1's whereas the lower extension inputs (LI1, LI2, LI3, and LI4) and the lowest data input (DI10) must be connected to logic 0's. In this mode, the reset input signal (RI) is used such that logic 1 is entered at the end of the every word (together with

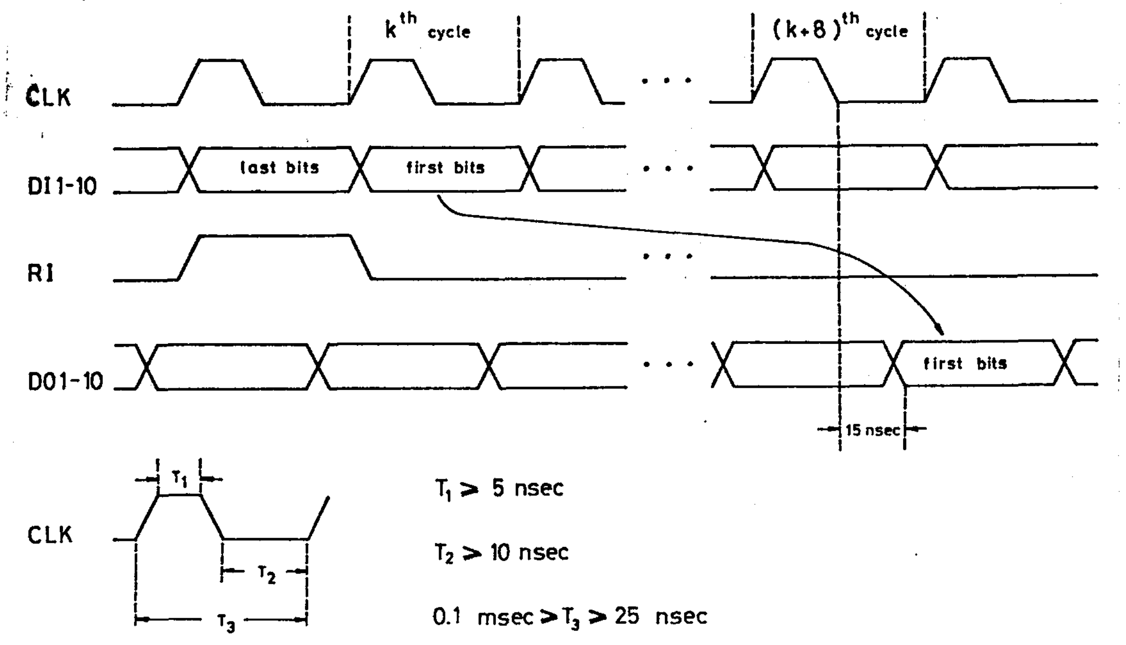


Figure 5.15: Timing diagrams for data load and read operations of the extensible median filter chip.

the last bit of the word). The input data are sorted at the data outputs in such a way that DO1 is the largest, DO5 is the median, and the DO9 is the smallest input. DO10 and the extension outputs have garbage data in this mode.

The Extension Mode

If the extensible median filter chip is used to find the exact median of w elements for $w > 9$, at most $[w/9]^2$ chips must be used. In this mode, the

DATA INPUTS	HIGH INPUTS	LOW INPUTS	MEDIAN
DI1	UI1	LI1	DO5
DI2	UI2	LI2	
...	UI3	LI3	
DI9	UI4	LI3	
	DI10		

Table 5.3: Functions of the I/O's in the normal mode of the extensible median filter chip.

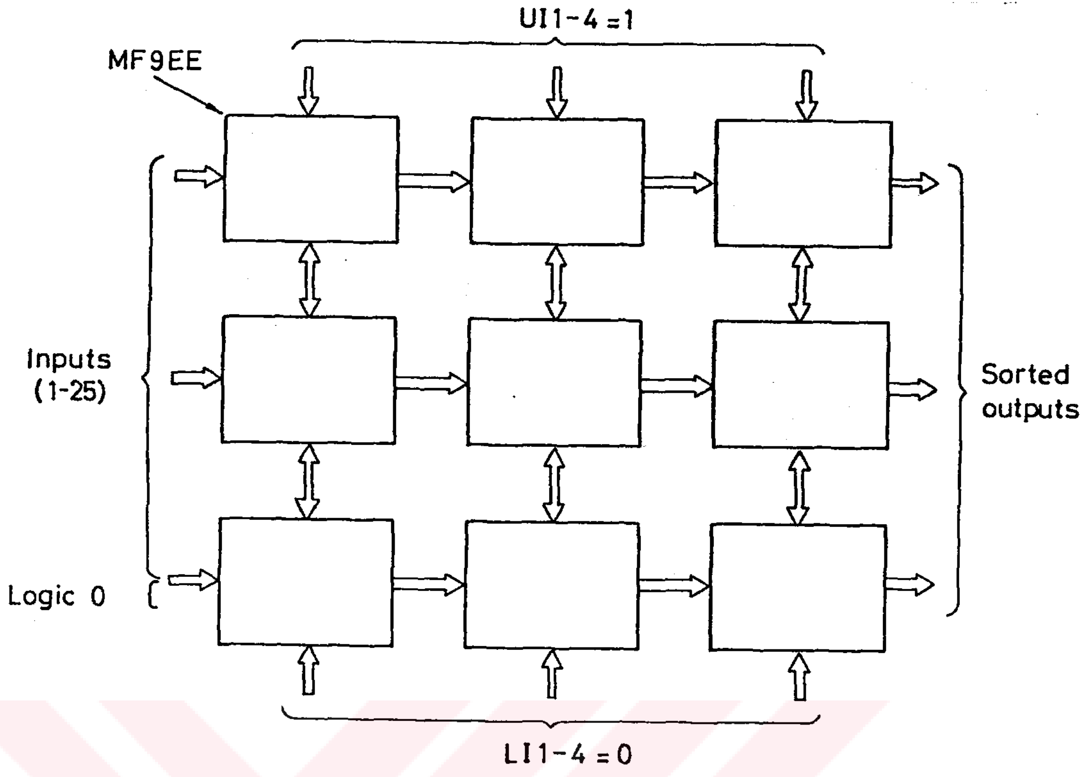


Figure 5.16: Interconnections of the extensible median filter chips for $w = 25$.

chips must be connected in such a way that they form an array of size $w \times w$ (i.e. the array has w chips in one row and w chips in one column). For the simplicity of description, we will show the connections of the chips for $w = 25$ which requires 9 chips (see Fig. 5.16). In this configuration, 20 ones of the the input data are applied to DI1-DI10's of the first chips at the first and second rows, and the remaining 5 data are applied to DI1-DI5 of the first chip at the third row. The upper extension inputs of the chips at the first row must be at logic 1's, and the lower extension inputs of the chips at the third row must be at logic 0's. The extension and the data I/O's of the chips must be interconnected as in Fig. 5.16. The reset input of the chips at the second and third columns must be delayed by 9 and 18 clock cycles from that of the reset input of the chips at the first column. The inputs are sorted at the outputs so that the median is DO3 of the last chip at the second row.

5.1.6 Testing of the Extensible Median Filter Chip.

The extensible median filter chip will be tested by using the *functional testing* method: the predetermined inputs will be applied to the chip and the outputs will be compared with the expected results. The test vectors are three set of input data. Each one of these input data sets causes all the CSU1's to operate at one of the *equal, pass*, and *swap* states: if inputs are

1. *equal* ($DI1 = DI2 = ... = DI9$) then all the CSU1's operate in *equal* state,
2. *sorted* ($DI1 > DI2 > ... > DI9$) then all the CSU1's operate in *pass* state, and
3. *reverse-sorted* ($DI1 < DI2 < ... < DI9$) then all the CSU1's operate in *swap* state.

Thus, the CSU1's are tested at all possible legal operation states. In addition, in order to test whether there is any stuck-at zero/one fault at the extension I/O's, two set of test vectors are generated. The logic simulations are performed by using all of the test vectors so that the test vectors and the expected outputs of the chip are the input and the output data given in Fig. 5.13. The total number of the test vectors is 500. As a result, the testing time of the extensible median filter chip is reasonably small.

5.2 Implementation of the Real-Time Median Filter

At the beginning of the implementation of the real-time median filter, a floor plan of the chip is designed, Fig. 5.17. The floor plan has an architecture similar to the block diagram of the network. The size and placement of the delay units at the inputs and outputs of the network are adjusted such that the unused space due to unequal numbers of the delay units at each block is as much as small. The clock distribution scheme has clock signal paths which are all equal. The power and clock distribution scheme of the chip is given in Fig. 5.18. A distributed clock buffering is preferred to avoid the clock skew and the large delay of the clock buffers driving large capacitive clock loads. Hence, 8 clock buffers are used for each clock phase.

In order to enable the outputs of the chip, one pin is required as the chip enable signal input. But there is no unused pin in the 40 pin package. For this reason, the test input signals S_t and E_t (see Fig. 3.5) are used for that purpose. In normal operation of the chip, S_tE_t will be 01 whereas S_tE_t will be 01,00, and 10 for testing operation. Note that S_tE_t pair does not take the value of 11 at these operations. This combination can be used as the chip enable signal. Hence, the test inputs, S_t and E_t are NAND'ed and buffered to be used as the chip enable signal; in order to disable the chip outputs, S_t and E_t are held at logic 1's.

In the following subsections, the circuits, layouts, and simulation results of the cells, CSU2 and delay unit, are described. The complete layout of the chip and logic simulations are given. Finally, an overview of the chip and its testing will be presented.

5.2.1 Compare-and-Swap Unit-2 (CSU2)

The functional description of the CSU2 given in Fig. 3.6 is transformed to the logic expressions. A sequential circuit design is performed. The logic expressions and the circuit of the CSU2 are shown in Fig. 5.19. This circuit consists of four main parts: input switch elements to latch the inputs during ϕ_1, logic gates to compute S_o and E_o, 2×1 multiplexers to pass or swap the inputs to the outputs, and the output switch elements and buffers to latch the computed outputs during ϕ_2. Weak buffers are used to delay the data inputs to prevent the charge sharing at the multiplexing stage. The latches at the input and output are dynamic that can not hold the data

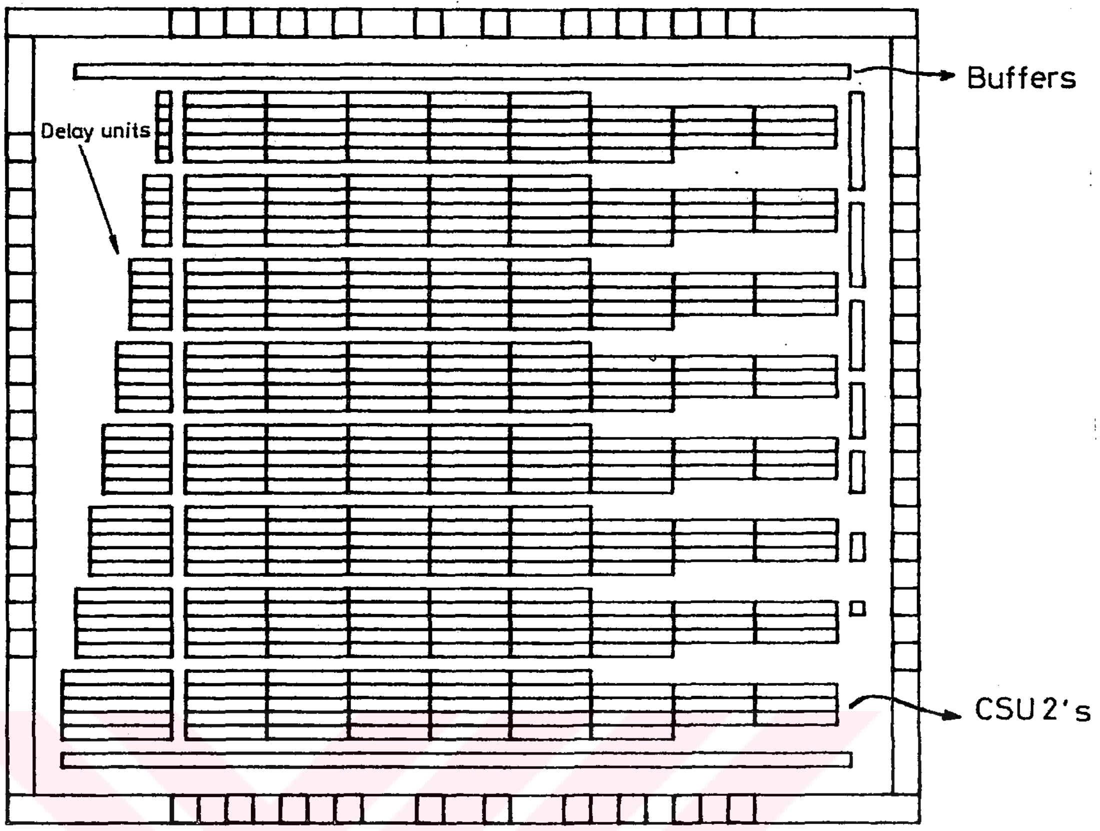


Figure 5.17: The floor plan.

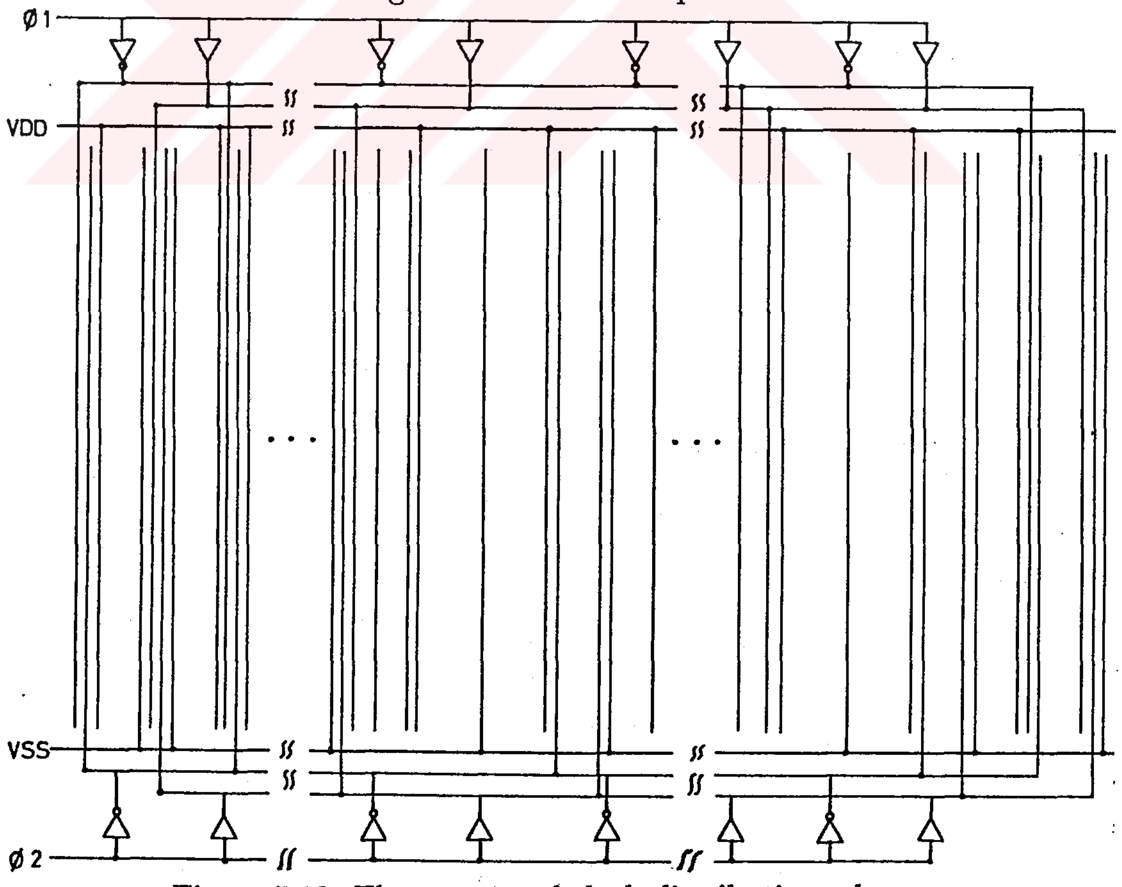


Figure 5.18: The power and clock distribution scheme.

S_i	E_i	A_i	B_i	S_o	E_o	A_o	B_o
0	1	1	1	0	1	1	1
0	1	0	1	1	0	1	0
0	1	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	0	1	1	1
0	1	0	1	1	0	1	0
0	1	1	0	0	0	1	0
0	1	0	0	0	1	0	0
1	1	1	1	1	1	1	1
1	1	0	1	1	0	1	0
1	1	1	0	1	0	0	1
1	1	0	0	1	1	0	0
1	1	1	1	1	1	1	1
1	1	0	1	1	0	1	0
1	1	1	0	1	0	0	1
1	1	0	0	1	1	0	0

Table 5.4: Logic simulation results of the CSU2.

longer than 0.1 msec. Hence, the minimum clock frequency is 10 KHz for correct operation.

In order to have a square shaped chip area, the geometry of the CSU2 need to be a thin rectangle. Keeping this in mind, the circuit diagram of the CSU2 is mapped to the layout by using standard CMOS logic style. The equivalent circuit of the layout is extracted and simulated by using the SPICE. The layout of the CSU2 is modified many times (i.e. the proper transistors are resized) in order to clock the CSU2 at least at the real time rate. The final layout of the CSU2 is shown in Fig. 5.20. This layout consists of 34 p-channel and 34 n-channel transistors.

The timing simulations of the CSU2 are performed for different clock frequencies with different rise and fall times. The CSU2 functions correctly up to 50 MHz clock frequency with 2.5 nsec rise and fall time of the clock signals at that frequency (Fig. 5.21. The maximum current for which all outputs make transition from either low to high or high to low is about 0.7 sec. The logic simulations of the CSU2 are performed for all combinations of the inputs, Table. 5.4.

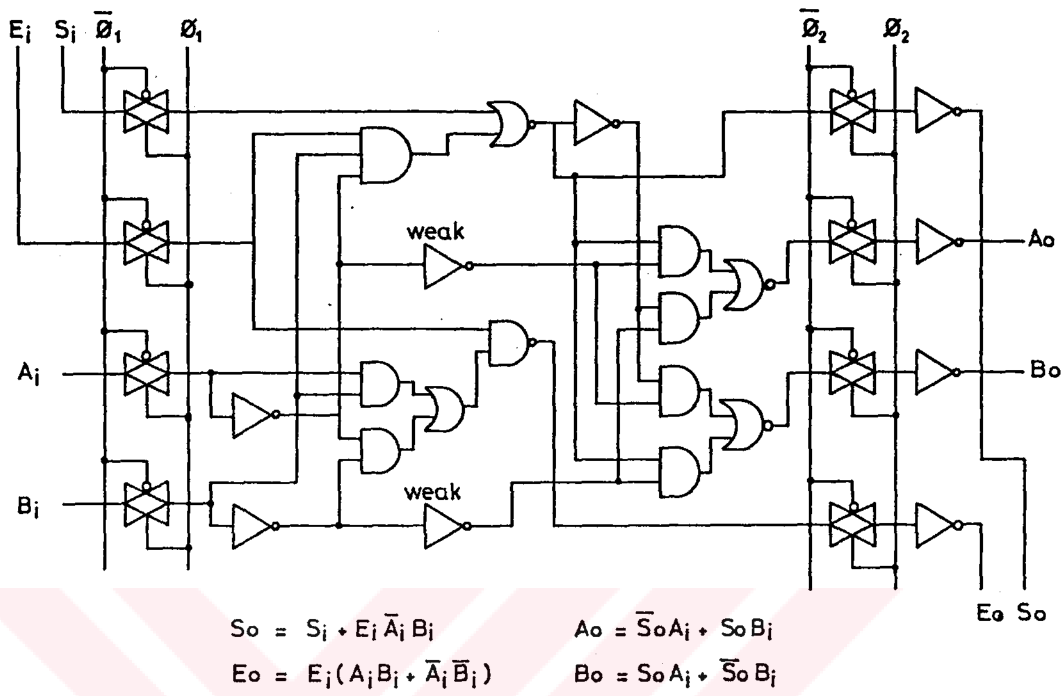

E_i
S_i
$\bar{\varnothing}_1$
$\varnothing_1$
$\bar{\varnothing}_2$
$\varnothing_2$
weak
weak
A_i
B_i
A_o
B_o
E_o
S_o
$S_o = S_i + E_i \bar{A}_i B_i$
$E_o = E_i (A_i B_i + \bar{A}_i \bar{B}_i)$
$A_o = \bar{S}_o A_i + S_o B_i$
$B_o = S_o A_i + \bar{S}_o B_i$

Figure 5.19: Circuit diagram of the CSU2.

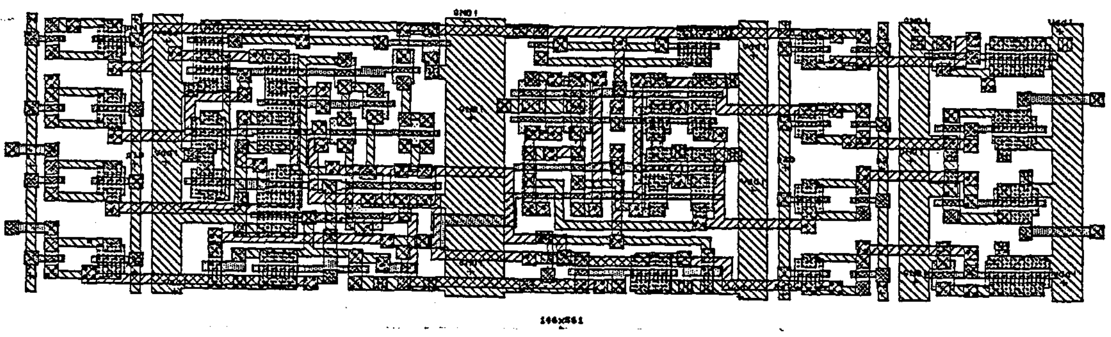


Figure 5.20: Layout of the CSU2.

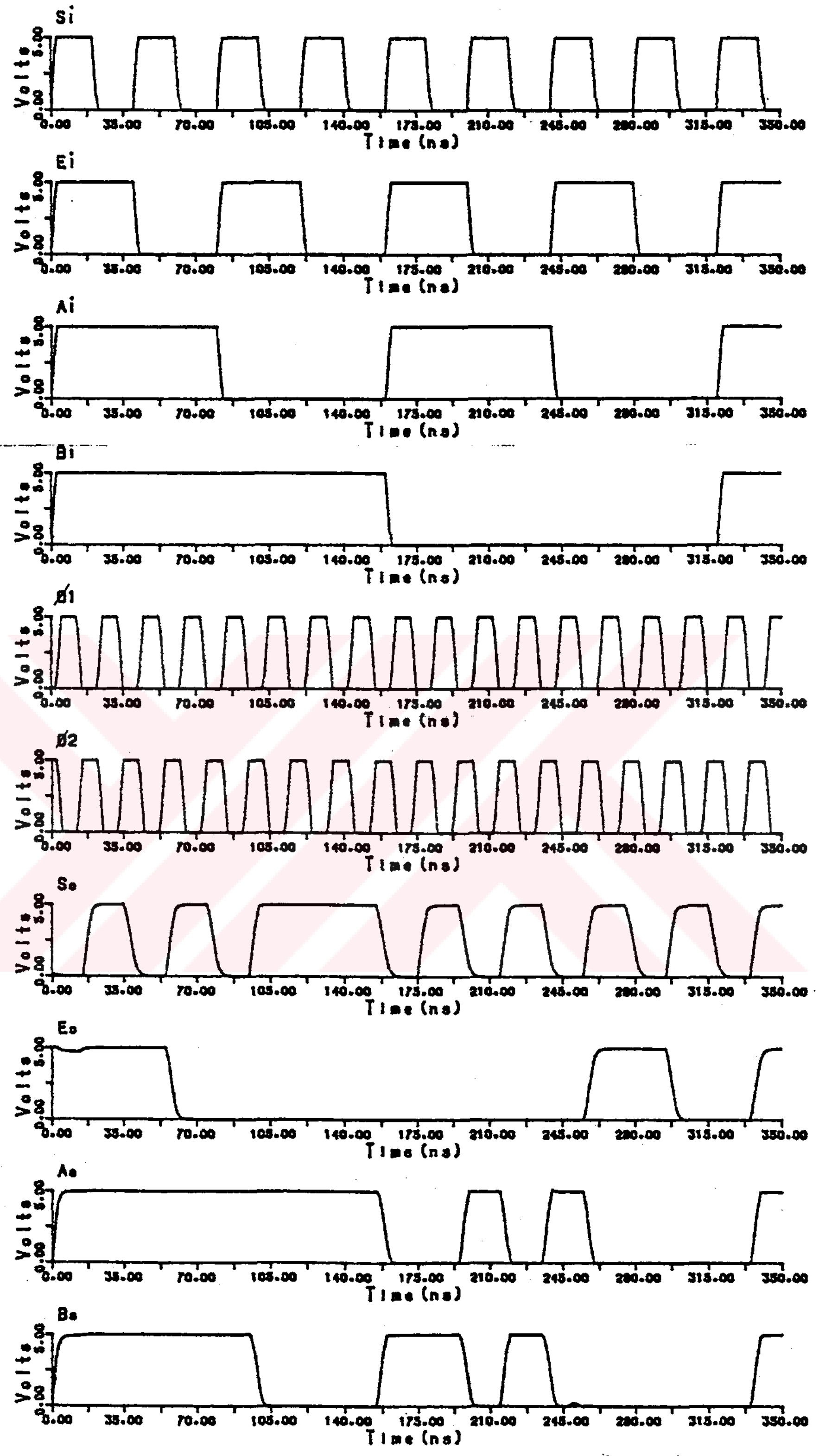


Figure 5.21: Timing simulation results of the CSU2.

The timing and the logic simulations show that the CSU2 operates correctly even if a single clock is used with its complement.

5.2.2 Delay Unit

The size of the delay units at the inputs of the network is desired to be a square shaped whereas that of the delay units used at the internal stages of the network is a thin rectangle to match the floor plan geometry to minimize the unused area. For that reason, two types of dynamic delay units are used. The delay unit shown Fig. 5.22.a is used at the inputs while the other one shown in Fig. 5.22.b is used at the internal stages of the network. Both of them are D-type dynamic master-slave shift register cells. They read the input during ϕ_1, and shift it out during ϕ_2. The layouts of the delay units are shown in Fig. 5.23. The timing simulation results of the delay units are shown in Fig. 5.24.

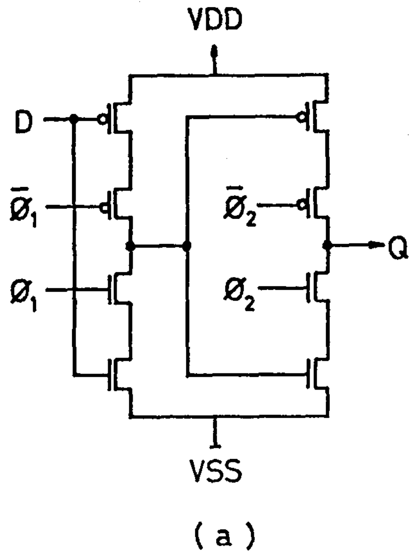
5.2.3 Clock Buffers

The total capacitive load of each of ϕ_1 and ϕ_2 is about 30 pF whereas that of each of their complements is about 40 pF. For each of the clocks and their phases, 4 clock buffers are used. These buffers are the same clock buffers which are used in the the extensible median filter chip; the inverting and noninverting clock buffers can drive 20 pF and 15 pF loads respectively with 2.5 nsec rise and fall times. Thus, there is 100 percent tolerance for the clock loads that allow the buffers to drive the loads with the rise and fall times faster than 2.5 nsec. Since the design and the simulation results of the clock buffers are given in the section of the implementation of the extensible median filter chip, they are not repeated here.

5.2.4 Chip Overall Layout

The basic cells, CSU2's and the delay units, are combined hierarchically to form the complete layout of the real-time median filter chip. The clock buffers and the pads are placed around the network. The final overall layout of the chip is shown in Fig. 5.25

The switch-level simulation of the chip is performed by using the *ESIM* to detect the stack-at 0 and stack-at 1 faults. It is not possible to use the test



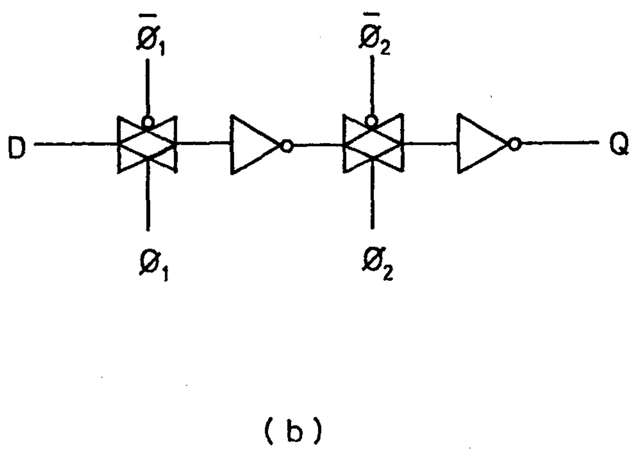
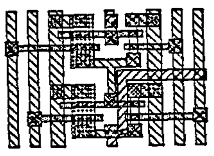


Figure 5.22: Circuit diagrams of the delay units used at the: a) inputs, b) internal stages of the the real-time median filter chip.



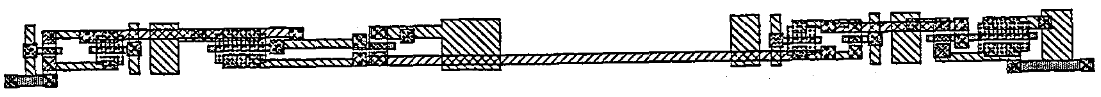
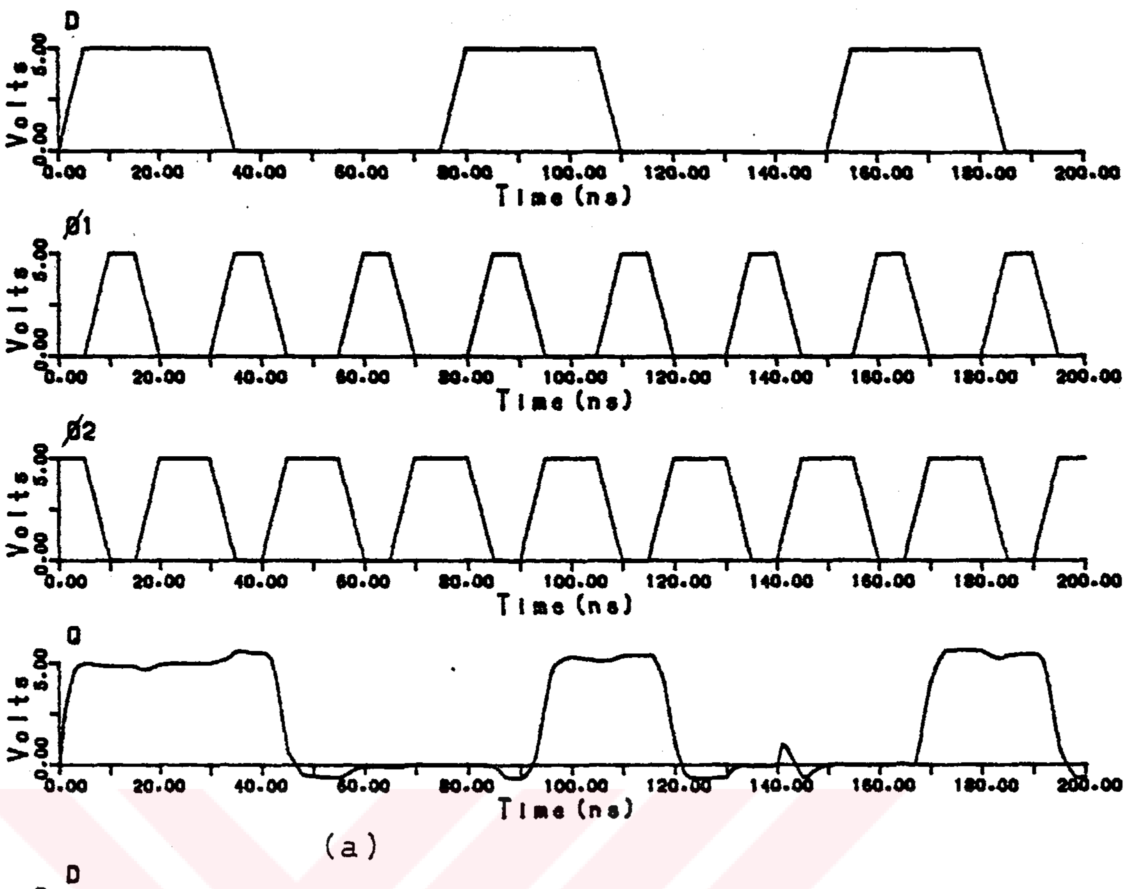


Figure 5.23: Layouts of the delay units used at the: a) inputs, b) internal stages of the the real-time median filter chip.



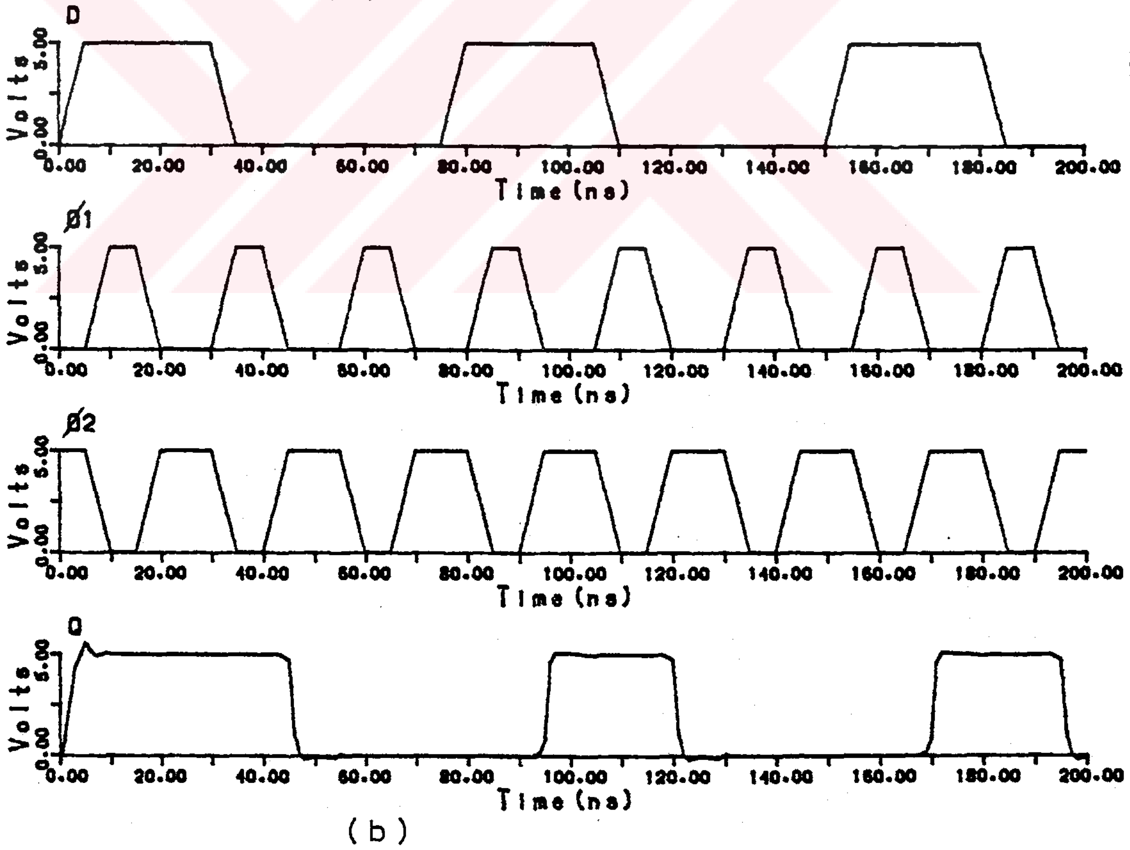


Figure 5.24: Timing simulations of the delay units used at the: a) inputs, b) internal stages of the the real-time median filter chip.

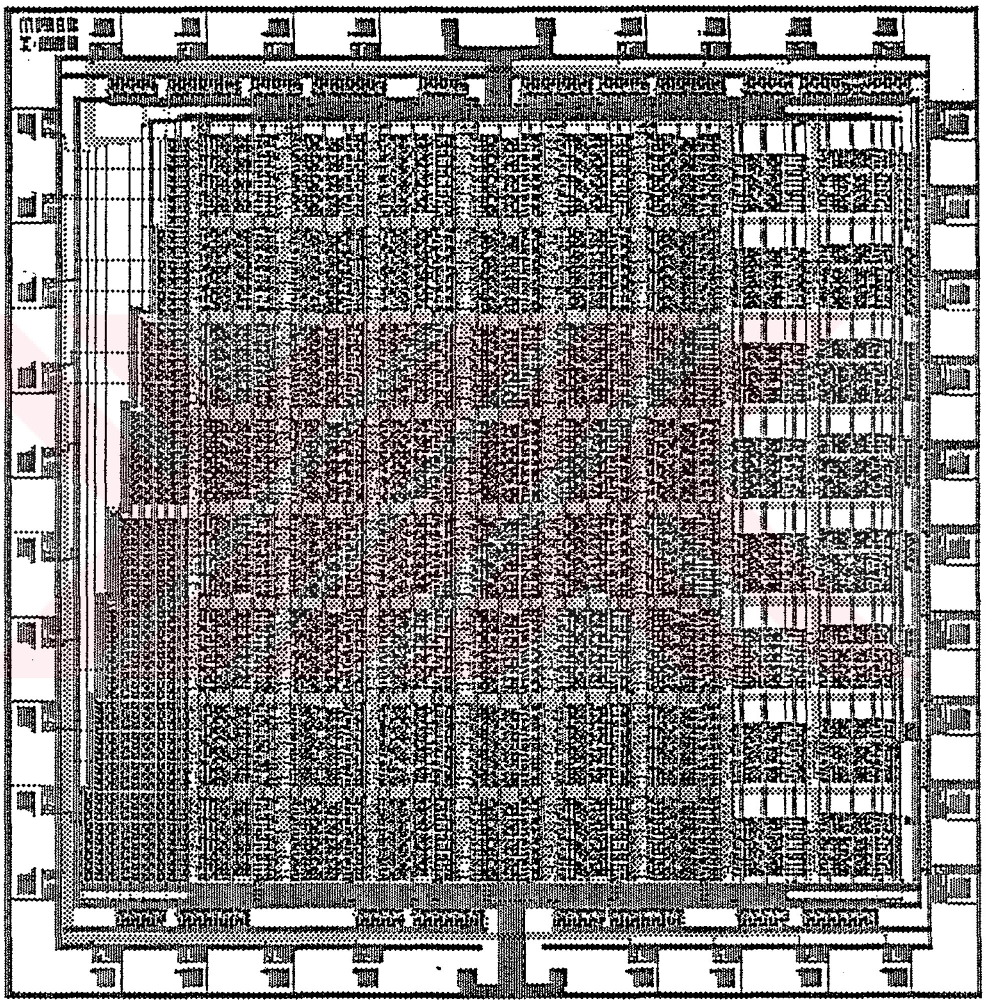


Figure 5.25: Overall layout of the real-time median filter chip.

vectors of the chip for logic simulation by *RNL* because the run time of only one clock cycle is about 5 minutes due to large number of transistors. Hence, the logic simulations are performed using some random inputs, Table 5.5. In this table, the median of the first 9 numbers which are 0, 0, 0, 5, 14, 11, 35, 24, and 22, is ready at the output 15 step (clock cycle) later. Similarly, median of any consecutive 9 inputs is ready at the output 15 clock later.

5.2.5 An Overview of the Real-Time Median Filter Chip

The real-time median filter chip has 40 pins; 24 pins are used for data inputs, 8 pins for data outputs, 2 pins for test control signals, 2 pins for two clock inputs, and 4 pins for the power supplies. The pin configuration is shown in Fig. 5.26. The pin labels are given in Table 5.6.

The the real-time median filter chip is used to find the median of the elements of a window which moves on a frame. At each position of the window, three 8-bit samples are discarded from the window and three new samples are taken in. Thus, at each clock cycle of the chip, three 8-bit samples enter the chip via the data inputs, and one 8-bit median is generated. The chip outputs 8-bit median of the samples in the window 15 clock cycles later. The input data are loaded to the chip while ϕ_1 is rising, and the medians are ready at the outputs of the chip during ϕ_2 is high.There is a delay due to pads for each input and output signal. In addition, there is an extra 7.5 nsec delay for the clock signals due to the clock buffers. The timings for data load and read operations are shown in Fig. 5.27.

The the real-time median filter chip functions properly even if the single clock is used such that ϕ_2 is the complement of ϕ_1. But for reliability, two nonoverlapping clocks are preferred. The maximum clock frequency is 50 MHz with 5 nsec rise and fall times. The clocks must stay at least 7.5 nsec at high level and 7.5 nsec at the low level. The test input signals, ST and ET, must be connected to 0 and 1 for normal operation. To disable the chip, ST and ET must be 1 and 1.

5.2.6 Testing of the Real-Time Median Filter Chip

The real-time median filter chip will be tested by using functional test method. Each sorter block of the network will be tested individually by isolating it

INPUTS			OUTPUTS
I11-I18	I21-I28	I31-I38	O1-O8
0	0	0	252
5	14	11	248
35	24	22	240
30	36	74	224
38	32	44	192
23	32	55	128
67	32	66	0
70	64	77	0
129	129	129	0
129	129	129	0
129	129	255	0
129	129	255	0
129	129	255	0
255	255	255	0
255	255	0	0
255	255	0	0
0	255	0	0
0	0	3	0
0	3	3	11
0	3	3	24
0	3	3	35
5	14	11	36
35	24	22	38
30	36	74	64
38	32	44	70
23	32	55	129
67	32	66	129
70	64	77	129
129	129	129	129
129	129	129	255
129	129	255	255
129	129	255	255
129	129	255	255
255	255	255	0
255	255	0	0
255	255	0	3
0	255	0	3
0	0	3	3
0	3	3	11
0	3	3	24
0	3	3	35

Table 5.5: Logic simulation results of the real-time median filter chip.

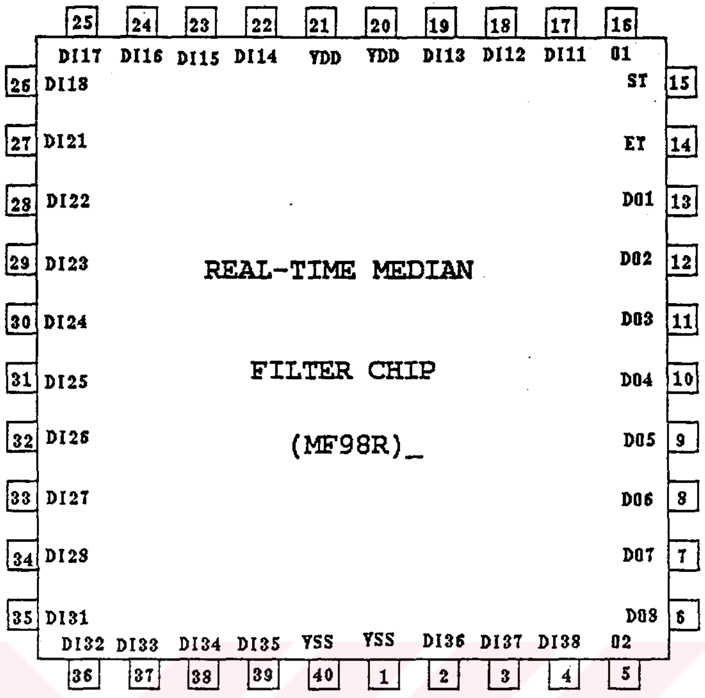


Figure 5.26: Pin configuration of the the real-time median filter chip.

LABEL	FUNCTION
I11-I18	DATA INPUTS FOR FIRST 8-BIT WORD (MSB TO I11)
I21-I28	DATA INPUTS FOR SECOND 8-BIT WORD (MSB TO I21)
I31-I38	DATA INPUTS FOR THIRD 8-BIT WORD (MSB TO I31)
O1-O8	MEDIAN (MSB IS O1)
ST ET	TEST INPUTS (01,00,10) CHIP DISABLE (11)
CLK1 CLK2	CLOCK INPUTS
VDD VSS	POWER SUPPLY 5 V 0 V

Table 5.6: Pin labels of the the real-time median filter chip.

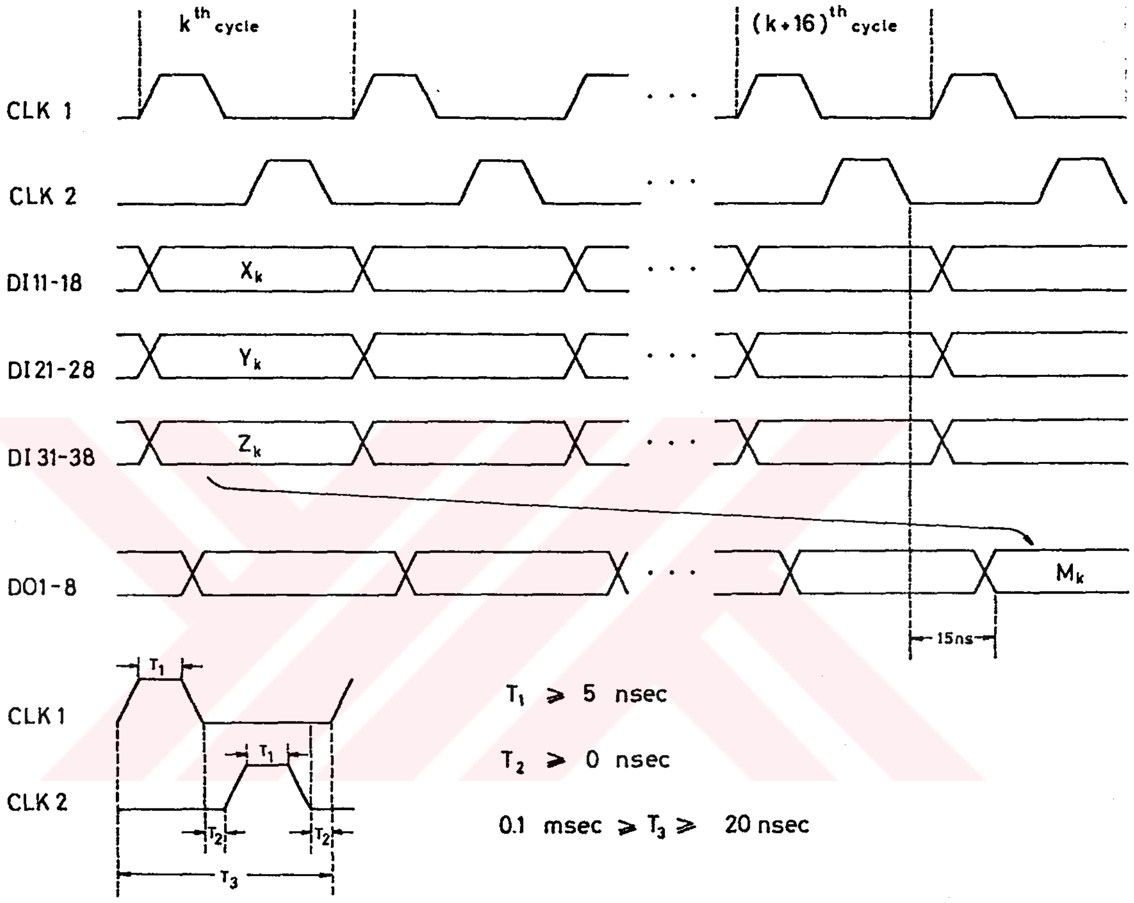


Figure 5.27: Timing diagrams for the data load and read operations of the real-time median filter chip.

from the other blocks by means of the test input signal ST and ET. When STET pair is 01, all the CSU2's in that sorter block pass the inputs which are equal or lower one is less, or swaps the inputs else. For STET=00, the CSU2's pass their inputs independent on the inputs. If the STET=10, then the CSU2's swaps the their inputs unconditionally. Thus, the CSU2's can be tested for all possible operations.

In order to test *k'th* block, the inputs of the upper blocks must all be 0 or all be 1, so that the ST and ET arrive this block without any change (the inputs of the lower blocks are *don't care*). For each values of the test control input signals (01,00,10), three 521-bit input sequence enter the inputs of that block successively one bit at a clock. This sequence generates all of the possible data flow combinations through that block. The output of that block is compared with the expected results. The same procedure is repeated for each of the 8-blocks. The test vectors and the corresponding outputs are generated by a program written in FORTRAN77 for this purpose. This program is given in the appendix. Since length of the sequence is 518 rows, here, we give a sample output which contains the first and the last parts of the sequences (Table 5.7). The total number of test inputs and testing clock cycles is $518 \times 3 \times 8 = 12,412$.

5.3 Applications of the Chips

The extensible and the real-time median filter chips can be selectively used in a general purpose digital image or signal processor environment by means of the chip enable signal that each chip has [47]. The exact medians of the elements, in a window size $w = 9$ with arbitrary word length L, can be found by using only one extensible median filter chip. For $w > 9$ with arbitrary L, at most $[[w/9]]^2$ chips are required to find the exact medians by interconnecting the chips similar to the configuration shown in Fig. 5.16. The real-time median filter chip can find the exact running medians of the elements in a window of a fixed size $w = 9$ with fixed word length $L = 8$ at the real-time rate. Furthermore, the extensible and/or real-time median filter can be used for the realizations of many median filtering techniques:

- The extensible median filter is a favorable choice to realize the *adaptive-length* median filters [25,62], since one can change the window size from 3 to indefinitely large ones by using the extensible median filter chip(s)

INPUTS			OUTPUTS		
IK1	IK2	IK3	OK(SE=01)	OK(SE=00)	OK(SE=10)
0	0	0	x	x	x
0	0	0	x	x	x
0	0	0	0	0	0
1	0	0	0	0	0
1	0	0	0	0	0
1	0	0	0	0	0
1	1	0	0	0	0
1	1	0	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1
0	1	1	1	1	1
1	1	1	1	1	1
1	1	1	1	1	1
1	0	1	1	1	1
1	1	1	1	0	0
1	1	1	1	1	1
.	.	.	.	.	.
0	0	0	0	1	1
1	0	0	0	0	0
0	0	1	0	0	0
0	0	0	0	0	0
0	1	0	0	0	0
0	0	0	0	1	1
0	0	0	0	0	0
0	0	1	0	0	0

Table 5.7: A sample of the test vectors and corresponding outputs. The IKJ/OKJ:J'*th* input/output of the K'*th* block of the real-time median filter sorting network.

by applying logic 0's or 1's to unused inputs of the chip(s) appropriately.

- For the realizations of the *weighted* median filters [38], the extensible median filter can be used with a pipelined multiplier to multiply the input data with the weight coefficients. Since all input data of the chip are entered to the chip directly at each move of the window, one can realize an adaptive weighted median filter by changing the weight coefficients at each position of the window on the frame.

- A pair of the extensible or the real-time median filter chips can be used as a *selective median* filter [36] together with an external control logic consisting of two full-word subtractors and a full-word comparator.

- Either the extensible or the real-time median filter chip can be used as a *line-recursive* median filter [25] by loading the window elements from the frame appropriately.

- The extensible median filter chips can be used for the realizations of the *multi-level* median filters [29] together with an reasonable external hardware.

- The chips can be used for the realizations of the *separable* median filters [30] without any external hardware.

6. RESULTS AND CONCLUSIONS

An extensible median filter and a real-time median filter are designed and implemented to form a general purpose median filter unit. The network of the extensible median filter is obtained by modifying the ordinary odd/even transposition sorting network. The real-time median filter network is a variation of the Oflazer's network [48]. Both of the networks are pipelined systolic structures that are modular and have simple communication schemes. This makes their VLSI implementation to be simple and straightforward. Both of the networks are not preferable to be implemented at larger window sizes since the area is proportional to the w^2. We have implemented the networks for $w = 9$ because this is the minimum and the most commonly used window size in two dimensional median filtering applications.

The networks are implemented in 3-micron double-metal standard CMOS by using full-custom VLSI design techniques. The timing and logic simulations of the chips are performed by using software tools. The test vectors and the corresponding outputs of the chips are generated. The testing times of the chips will be reasonably small since the numbers of test vectors are small.

The exact medians of the elements, in a window size $w = 9$ with arbitrary word length L, can be found by using only one extensible median filter chip. For $w > 9$ with arbitrary L, at most $[[w/9]]^2$ chips are required to find the exact medians. The real-time median filter chip can find the exact running medians of the elements in a window of a fixed size $w = 9$ with fixed word length $L = 8$ at the real-time rate. The main features of the chips are summarized in Table 6.1. These features of the chips are comparable with that of the chips in the literature, for example [63].

PARAMETER	THE EXTENSIBLE	THE REAL-TIME
Die size	$2.77 \times 4.59\text{mm}^2$	$7.19 \times 6.95\ \text{mm}^2$
Transistor count	5,000	22,000
Transistors/mm^2	400	440
Max. clock frequency	40 MHz	50 MHz
Max. throughput	$30/L$ mega medians/sec	50 mega medians/sec
Pin number	40	40
Max. power dissipation	250 mW	800 mW

Table 6.1: Main features of the chips.

The main contributions of this study are the architecture of the extensible median filter and its VLSI implementation. Another achievement of this study is the implementation of the real-time median filter which is the first single-chip median filter in this area that can operate at the real-time rate for the 1024×1024 resolution frames. The extensible and real-time median filter chips together with a reasonable external hardware can be used for the realizations of many median filtering techniques. As a result, after the fabrication of the chips, they can be used in various full-scale general purpose digital signal and image processors.

REFERENCES

[1] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Englewood Cliffs, N.J.: Prentice-Hall, 1975.

[2] A. Kundu, S. K. Mitra, and P. P. Vaidyanathan, "Application of two-dimensional generalized mean filtering for removal of impulse noises from images," *IEEE Trans. Acoustic, Speech, and Signal Processing*, vol. ASSP-32, NO. 3, pp. 600-609, Jun. 1984.

[3] J. W. Tukey, "Nonlinear (nonsuperposable) methods for smoothing data," in *Conf. Rec.*, p. 673, *EASCON* 1974.

[4] N. C. Gallagher, Jr., "Median filters: a tutorial," *Proc. IEEE Symp. on Cir. Sys.* pp. 1737-1744, Finland, 1988.

[5] N. C. Gallagher, Jr., and G. L. Wise, "A theoretical analysis of the properties of median filters," *IEEE Trans. Acoustic, Speech, and Signal Processing*, vol. ASSP-29, pp. 1136-1141, Dec. 1981.

[6] T. A. Nodes and N. C. Gallagher, JR., "Median filters: some modifications and their properties," *IEEE Trans. Acoustic, Speech, and Signal Processing*, vol. ASSP-30, NO. 5, pp. 739-746, Oct. 1082.

[7] E. Ataman, V. K. Aatre, and K. M. Wrong, "Some statistical properties of median filters," *IEEE Trans. Acoustic, Speech, and Signal Processing*, vol. ASSP-29, pp.1073-1075, Oct. 1981.

[8] T. A. Nodes and N. C. Gallagher, Jr., "The output distribution of median type filters," *IEEE Trans. Commun.*, vol. COM-32, pp. 532-541, May 1984.

[9] J. Neejarvi, P. Heinonen, and Y. Neuvo, "Sine wave responses of median filters," *Proc. IEEE Symp. on Cir. Sys.*, pp. 1503-1506, Finland, 1988.

[10] J. Astola, P. Haavisto, P. Heinonen, and Y. Neuvo, " Median type filters for color signals," *Proc. IEEE Symp. on Cir. Sys.*, pp. 1753, Finland, 1988.

[11] J. S. Jimmy Li and W. H. Holmes, "Analog implementation of median filters for real-time signal processing," *IEEE Trans. Circuits and Systems*, vol. 35, pp. 1032-1033, Aug. 1988.

[12] L. R. Rabiner, M. R. Sambur, and C. E. Schmidt, "Applications of a nonlinear smoothing algorithm to speech processing," *IEEE Trans. Acoustic, Speech, and Signal Processing*, vol. ASSP-23, pp. 552-557, Dec. 1975.

[13] E. Ataman and E. Alparslan, "Application of median filtering algorithm to images," Electronics Division, Marmara Research Institute, Gebze, Turkey, Tech. Rep. UI 78/10, Sep. 1978.

[14] A. C. Bovik, "Streaking in median filtered images," *IEEE Trans. Acoustic, Speech, and Signal Processing*, vol. ASSP-35, pp. 493-503, Apr.1987.

[15] H. T. Kung, "Let's design algorithms for VLSI systems," *Technical Report*, Dep. of Comp. Science, Carnegie-Mellon Unv., Jan. 1979.

[16] H. T. Kung, "Why systolic architectures?," *IEEE Computer*, pp. 37-46, Jan. 1982.

[17] S. Y. Kung, H. J. Whitehouse, and T. Kailath, Eds.,*VLSI and Modern Signal Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1985.

[18] D. I. Moldovan, "On the design of algorithms for VLSI systolic arrays," *IEEE Proc.*, vol. 71, pp. 113-120, Jan. 1983.

[19] G. Li and B. W. Wah, "The design of optimal systolic arrays," *IEEE Trans. Computers*, vol. C-34, pp. 66-77, Jan. 1985.

[20] C. Mead and L. Conway, *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980.

[21] M. J. Foster and H. T. Kung, "The design of special purpose VLSI chips, "*IEEE Computer*, pp. 26-40, Jan. 1980.

[22] S. Topcu, İ. E. Ungan, Ş. Toygar, and A. Atalar, “Design and testing of a microprocessor compatible 128-bit correlator,” in *Proc. of Third Inter. Symp. on Comp. Infor. Sci.*, Çesme, İzmir, Turkey, 1988.

[23] İ. E. Ungan, S. Topcu, and A. Atalar, “ VLSI implementation of a microprocessor compatible 128-bit programmable correlator,” in *Proc. of Third Inter. Symp. on Comp. Infor. Sci.*, Çesme, İzmir, Turkey, 1988.

[24] G. R. Arce and N. C. Gallagher, Jr., “State description for root-signal set of median filters,” *IEEE Trans. Acoustic, Speech, and Signal Processing*, vol. ASSP-30, pp. 894-902, Dec. 1982.

[25] H. M. Lin and N. Willson, Jr., “Adaptive-length median filters for image processing,” *Proc. IEEE Symp. on Cir. Sys.*, pp. 2557-2560, Finland, 1988.

[26] G. R. Arce and R. L. Stevenson, “ On the synthesis of median filter systems,” *IEEE Trans. Circuits and Systems*, vol. CAS-34, pp.420-429, Apr. 1987.

[27] J. P. Fitch, E. J. Coyle, and N. C. Gallagher, JR., “ Median filtering by threshold decomposition,” *IEEE Trans. Acoustic, Speech, and Signal Processing*, vol. ASSP-32, pp. 1183-1188, Dec. 1984.

[28] C. G. Boncelet, Jr., “Recursive algorithms and VLSI implementations for median filtering,” *Proc. IEEE Symp. on Cir. Sys.*, pp. 1745-1747, Finland, 1988.

[29] G. R. Arce, P. J. Warter, and R. E. Foster, “Theory and VLSI implementation of multilevel median filters,” *Proc. IEEE Symp. on Cir. Sys.*, pp. 2795-2798, Finland, 1988.

[30] M. P. McLoughlin and G. R. Arce, “Deterministic properties of the recursive separable median filter,” *IEEE Trans. Acoustic, Speech, and Signal Processing*, vol. ASSP-35, pp.98-106, Jan. 1987.

[31] A. C. Bovik, T. S. Huang, and D. C. Munson, JR., “ A generalization of median filtering using linear combinations of order statistics,” *IEEE Trans. Acoustic, Speech, and Signal Processing*, vol. ASSP-31, pp.1342-1349, Dec. 1983.

[32] Y. H. Lee and S. A. Kassam, "Generalized median filtering and related nonlinear filtering techniques," *IEEE Trans. Acoustic, Speech, and Signal Processing*, vol. ASSP-33, pp. 672-683, Jun. 1985.

[33] A. C. Bovik, T. S. Huang, and D. C. Munson, Jr., " The effect of median filtering on edge estimation and detection," *IEEE Trans. Pattern Analy. Mach. Intell.*, vol. PAMI-9, Mar. 1987.

[34] A. Niemien, P. Heinonen, and Y. Neuvo, "A new class of detail-preserving for image processing," *IEEE Trans. Pattern Analy. Mach. Intell.*, vol. PAMI-9, pp. 74-90, Jan. 1987.

[35] Y. Neuvo, P. Heinonen, and I. Defee, "Linear-median hybrid edge detectors," *IEEE Trans. Circuits and Systems*, vol. CAS-34, pp. 1337-1343, Nov. 1987.

[36] S. J. Ko, Y. H. Lee, and A. T. Fam, "Selective median filters," *Proc. IEEE Symp. on Cir. Sys.*, pp. 1495-1498, Finland, 1988.

[37] T. A. Nodes and N. C. Gallagher, Jr., "Two-dimensional root structures and convergence properties of the separable median filter," *IEEE Trans. Acoustic, Speech, and Signal Processing*, vol. ASSP-31, pp. 1350-1365, Dec. 1983.

[38] T. Loupas, W. N. McDicken, and P. L. Allan, " Noise reduction in ultrasonic images by digital filtering, " *The British Journal of Radiology*, vol. 60, pp.389-392, Apr. 1987.

[39] G. Garibotto and L. Lambarelli, "Fast on-line implementation of two-dimensional median filtering," *Electronics Letters*, vol. 15, pp.24-45, Jan. 1979.

[40] T. S. Huang, G. J. Yang and G. Y. Tang, "A fast two-dimensional median filtering algorithm," *IEEE Trans. Acoustic, Speech, and Signal Processing*, vol. ASSP-27, pp. 13-18, Feb. 1979.

[41] E. Ataman, V. K. Aatre, and K. M. Wong, "A fast method for real-time median filtering," *IEEE Trans. Acoustic, Speech, and Signal Processing*, vol. ASSP-28, pp. 415-421, Aug. 1980.

[42] V. V. B. Rao and K. S. Rao, “A new algorithm for real-time median filtering,” *IEEE Trans. Acoustic, Speech, and Signal Processing*, vol. ASSP-34, pp. 1674-1675, Dec. 1986

[43] D. L. Knuth, *The Art of Computer Programming- Searching and Sorting*, vol. 3. Reading MA: Addison-Wesley, 1973.

[44] C. D. Thompson, “The VLSI complexity of sorting,” *IEEE Trans. Computers*, vol. C-32, pp. 1171-1184, Dec. 1983.

[45] G. Bilardi and F. P. Preparata, “ A minimum area VLSI network for $O(log\ n)$ time sorting,” *IEEE Trans. Computers*, vol. C-34, pp. 336-343, Apr. 1985.

[46] N. Weste and K. Eshraghian,*Principles of CMOS VLSI Design*, Reading MA: Addison-Wesley, 1985.

[47] P. A. Ruetz and R. W. Brodersen, “Architectures and design techniques for real-time image-processing IC’s,” *IEEE J. Solid-State Cir.*, vol. SC-22, pp.233-250, Apr. 1987.

[48] K. Oflazer, “Design and implementation of a single-chip 1-D median filter,” *IEEE Trans. Acoustic, Speech and Signal Processing*, vol. ASSP-31, pp. 1164-1168, Oct. 1983.

[49] M. O. Ahmad and D. Sundararajan, “A fast algorithm for two- dimensional median filtering,” *IEEE Trans. Circuits and Systems*, vol.CAS-34, pp. 1364-1374, Nov. 1987.

[50] U. E. Ruttimann and R. L. Webber, “Fast computing median filters on general-purpose image processing systems,” *Optical Engineering*, vol. 25, Sep. 1986.

[51] J. A. Abraham and W. K. Fuchs, “Fault and error models for VLSI,” *IEEE Proc.*, vol. 74, pp. 639-654, May 1986.

[52] F. Guterl, “In pursuit of the one-month chip,” *IEEE Spectrum*, pp. 28-46, Sep. 1984.

[53] L. A. Glasser and D. W. Dobberpuhl, *The Design and Analysis of VLSI Circuits* Addison-Wesley, 1985.

[54] A. Mukherjee, *Introduction to nMOS and CMOS VLSI System Design*, Prentice-Hall, 1986.

[55] R. D. Davis, "The case for CMOS," *IEEE Spectrum*, pp.26-32, Oct. 1983.

[56] G. J. Hu, "A better understanding of CMOS latch-up," *IEEE Trans. Electron Devices*, vol. ED-31, pp.62-67, Jan. 1984.

[57] *Berkeley CAD Tools User's Manual*, EECS Dep., University of California at Berkeley, 1986.

[58] *VLSI Tools Reference Manual*, TR#87-02-01, Release 3.1, NW Lab. Int. Sys., Dep. Computer Sci., University of Washington, Feb.1987.

[59] M. Hatamian and G. L. Cash, "Parallel bit-level pipelined VLSI designs for high-speed signal processing," *Proc. IEEE*, vol. 75, pp. 1192-1202, Sep. 1987.

[60] M. M. Mano, *Digital Design*, Prentice-Hall, 1979.

[61] N. Hedenstierna and K. O. Jeppson, "CMOS circuit speed and buffer optimization," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 270-280, Mar. 1987.

[62] W. J.Song and W. A. Pearlman, "Edge-preserving noise filtering based on adaptive windowing," *IEEE Trans. Circuits and Systems*, vol. 35, pp. 1048-1054, Aug. 1988.

[63] T. Denayer, E. Vanzieleghem, and P. G. A. Jespers, " A class of multiprocessors for real-time image and multidimensional signal processing," *IEEE J. Solid-State Cir.*, vol. SC-23, pp.630-638, Jun. 1988.

APPENDIX

Test Pattern Generator for the Real-Time Median Filter Chip

```
* * *
* THIS PROGRAM GENERATES THREE SEQUENCES OF BINARY
* NUMBERS (1 OR 0). PROGRAM HAS NO INPUT ENTRY.
* IT MUST BE COMPILED IN FORTRAN77 (% f77 testgen.f testgen).
* THE OUTPUTS ARE SENT TO THE SCREEN. IF ONE NEEDS
* TO WRITE THE OUTPUTS IN A FILE, SAY INTO
* vec.out, THEN THE PROGRAM MUST BE EXECUTED AS:
* % testgen >vec.out.
   INTEGER W(9,513),R(9),S1(527),S2(527),S3(527),Z,NC,
INTEGER OUT01(527),OUT00(527),OUT10(527)
DO 11 I=1,513
DO 11 J=1,9
11 W(I,J)=8
* * * INITIALIZATION
DO 12 I=1,9
12 R(I)=0
NC=1
CALL LOAD(R,W,NC)
* * * ASSUME LOGIC 1
101 Z=1
CALL SHIFT(Z,R)
CALL COMP(R,W,NC,IE)
IF(IE.EQ.0) GOTO 13
IS=NC+9
S1(IS)=Z
```

```
NC=NC+1
CALL LOAD(R,W,NC)
IF(NC.EQ.512) GOTO 1234
CALL CONV(R,IH)
GOTO 101
* * * THE BIT IS LOGIC 0
13 Z=0
222 CALL LOADB(W,R,NC)
CALL SHIFT(Z,R)
CALL COMP(R,W,NC,IE)
IF(IE.EQ.0) GOTO 14
IS=NC+9
S1(IS)=Z
NC=NC+1
CALL LOAD(R,W,NC)
IF(NC.EQ.512) GOTO 1234
CALL CONV(R,IH)
GOTO 101
* * * SEARCH RECURSIVELY
14 NC=NC-1
PRINT *,'ERROR! LINEAR VECTORS !'
IF(NC.EQ.0) GOTO 15
CALL LOADB(R,W,NC)
IF(S1(NC).EQ.1) GOTO 13
Z=1
GOTO 222
15 PRINT *,'! ONLY 512 VECTORS POSSIBLE'
*** COMPUTE OUTPUTS: OUT01(I) (SE=01), OUT10(I) (SE=10), OUT10(I)
(SE=00)
1234 DO 55 J=1,512
OUT00(J+8)=W(5,J)
OUT10(J+8)=W(5,J)
IW=0
DO 44 I=1,9
44 IW=IW+W(I,J)
IF(IW.LT.5) OUT01(J+8)=0
IF(IW.GE.5) OUT01(J+8)=1
55 CONTINUE
* * * GENERATE THE SEQUENCES S2,S3
DO 1122 I=1,512
S2(I+3)=S1(I)
S3(I+6)=S1(I)
1122 CONTINUE
* * * PRINT OUTPUTS
PRINT *,'INPUTS OUTPUTS'
PRINT *,'IK1 IK2 IK3 OK(SE=01) OK(SE=00) OK(SE=10)'
DO 321 I=1,518
PRINT *,S1(I),S2(I),S3(I),' ',OUT01(I),OUT00(I),OUT10(I)
321 CONTINUE
STOP
END

   SUBROUTINE LOAD(N,M,K)
INTEGER M(9,513),N(9),K
DO 91 I=1,9
91 M(I,K)=N(I)
RETURN
END

   SUBROUTINE SHIFT(K,N)
INTEGER N(9),K
DO 92 I=0,8
IX=9-I
92 N(IX)=N(IX-1)
N(1)=K
RETURN
END

   SUBROUTINE COMP(N,M,K,IC)
INTEGER M(9,513),N(9),K
DO 93 J=1,K
DO 94 I=1,9
```

```
IF(N(I).NE.M(I,J)) GOTO 93
94 CONTINUE
IC=0
GOTO 95
93 CONTINUE
IC=1
95 RETURN
END

    SUBROUTINE LOADB(M,N,K)
INTEGER M(9,513),N(9),K
DO 96 I=1,9
96 N(I)=M(I,K)
RETURN
END

    SUBROUTINE CONV(N,IV)
INTEGER N(9)
IV=0
DO 99 I=1,9
99 IV=IV+N(I)*2**(9-I)
RETURN
END
```

T. C.
Yükseköğretim Kurulu
Dokümantasyon Merkezi