IEEE-International Conference on Signal processing, Communications and Networking
Madras Institute of Technology, Anna University Chennai India, Jan 4-6, 2008. Pp339-344

High Throughput VLSI Architecture for One Dimensional Median Filter

V. V. Ravi Teja[1], K. C. Ray, I. Chakrabarti[2], A. S. Dhar[3] , [1 2 3] Department of E&ECE,
Indian Institute of Technology, Kharagpur, India-721302,
teja.iitkgp@gmail.com, kcr,asd,indrajit}@ece.iitkgp.ernet.in

Abstract **- An attempt has been made to design a high throughput VLSI architecture for one dimensional median filter to suppress the impulse noise in real time signal and image processing applications. The proposed architecture is based on parallel and pipelined techniques. It takes 8-bit data serially and computes the median value in parallel and pipelined fashion out of a window having size of nine samples. This architecture is described in VerilogHDL and synthesized using commercially available 0.18μm CMOS technology at 1.8V power supply. The synthesis result gives an approximate core area and power of 1.2mm^2 and 92.5mW respectively at 700MHz clock frequency leading to a latency of thirteen clock cycles only.**

I. INTRODUCTION

In many signal and image processing applications, it is essential to suppress the noisy signals while preserving the required necessary information i.e., without losing edge information in this process. The techniques such as linear filtering, average filtering, and median filtering have been used to smoothen the noisy signals but the linear filtering smoothens noisy signals as well as edges i.e., high frequency information. Median filtering techniques have been used to smoothen the impulsive noise without losing high frequency signals i.e., preserving edge information. Some properties of median filtering are that (a) it smoothens the transient signals, (b) removes impulse noises from the signals and (c) preserves the edge information in the filtered signals (images). The concept of median filtering was first proposed by Tukey [1]. Median filtering techniques have been widely used in various signal and image processing applications mentioned in [2,3]. Since decades, implementation of median filtering has been attempted in software and commercially available DSP processor environment. But the main constraint of aforesaid implementations is speed. To overcome this constraint, some attempts have been made to implement median filtering in hardware for real time applications [3, 4]. Since most of the median value computations are based on sorting algorithm [5-8], Fast median filter architecture therefore depends on the availability of an efficient structure to perform sorting.

The rest of this paper is organized as follows: Section-II describes the algorithm for median filtering with an example, Section-III presents and describes proposed high throughput VLSI architecture for 9-sample window. The simulation and synthesis results have been presented and discussed in Section-IV and finally Section-V concludes the paper.

II.MEDIAN FILTERING ALGORITHM

This section presents steps for median filtering based on fast sorting algorithm with example for clarity. The basic theory of median filter is not discussed here, however readers may refer to [9, 10] for the same.

The fastest method of sorting is to perform multiple actions (addition and deletion of elements) at the same time [11,12]. We describe these operations as given here, when values are input in a serial order, into the median filter, we can maintain two arrays, one called window array which contains input elements and second one is called sorted array, which contains window elements in sorted fashion(ascending/descending). To relate both the arrays, we maintain a third array called the aging array. Here importance of the ageing array is that the i^{th} element of the aging array indicates the time to be spent by the i^{th} element of sorted array in the window array of the median filter. In a median filter of window length of 9, an element spends 9 clock cycles in the window array of the filter before leaving. So the aging array contains values from 0 to 8. An example of the above described arrays is shown below.

An Example:
Window Array W=[66 25 81 15 255 150 174 111 181]
Sorted Array S= [255 181 174 150 111 81 66 25 15]
Aging Array A = [4 0 2 3 1 6 8 7 5]

In an instant, we can see that the value 181 is at the right extreme of the window array, so when the next value gets input from the left, 181 has to leave the array. Hence its age is 0. The value 66 which is at the left extreme has 8 in its aging array register. For every clocking, the values in the aging array decrease by 1 indicating that the value bas moved 1 place closer to its exit. When value in aging array becomes zero, no more time to be spent by that element in the window and hence should then be deleted with the next clock pulse.
As soon as a new value enters the window array, the oldest value in that array gets deleted. The same oldest value must be deleted from the sorted array while adding the new element. The oldest value in the sorted array is that element which has its corresponding element in aging array with value = 0.

Here the steps are as follows:

1. The new element in window array W, is compared with remaining 8 elements of the window array to computes its position in sorted array S.

2. The oldest value in the sorted array is identified as explained before.

3. Necessary signals are generated to right shift or left shift certain elements in the sorted array such that oldest value is deleted from the sorted array while accommodating the latest value without disturbing the sorted order.

4. The elements in aging array A, are shifted in the same way as the corresponding elements in the sorted array. At the same time their values are decremented by 1 for every clock cycle so that the aging phenomenon is implemented.

5. The value 8 is introduced in the aging array in the place corresponding to the newly introduced value in the sorted array.

6. The middle value in the sorted array is taken as the median.

III. PROPOSED ARCHITECTURE

This section describes the proposed fast VLSI architecture for fixed window size of 9 and word size of 8-bits. Proposed fast median filter architecture as shown in Fig.1 has been explained in following stages briefly and subsequently details of these stages given.

Stage1. The comparison stage that involves 8 comparators with 8 bit comparison.

Stage2. Adders stage to add the output of the comparators to compute the position of the new arrival in the sorted array.

Stage3. The shifting signals generation stage where right or left shift signals are computed to input to the 9 registers of the sorted array and 9 registers of aging array. Aging array has a decrementor to decrement values in its registers by 1 in every clock cycles.

Stage4. Output stage which is just a median collector connected to 5th register in the sorted array registers.

The control logic path as in Fig.1, is an important circuitry to generate control signals for median filtering. And this is a parallel and pipelined version to reduce the critical delay.

The rest of this section explores the architecture at the gate level of each block described above and shown in Fig.1. Here we have designed the circuits for each block and integrated to a final architecture using parallel and pipelined techniques to have lesser critical delay and thus to achieve high throughput.

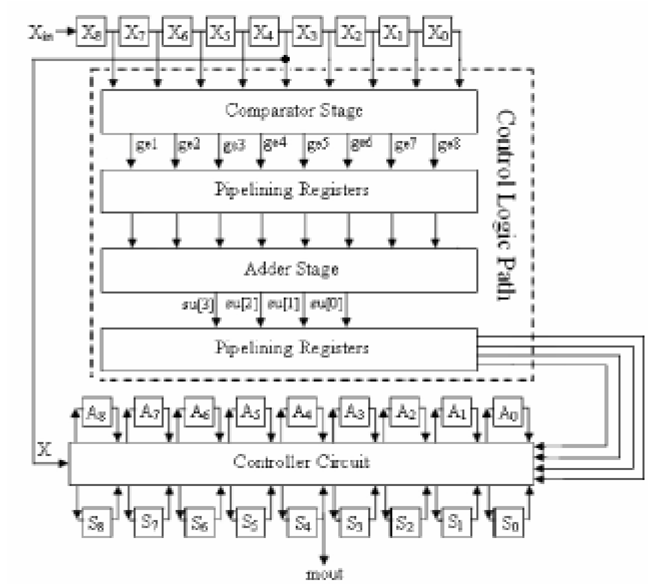

Fig.1: Proposed Median Filter Architecture

A. Comparator Stage:

The comparator stage shown in Fig.2, happens to be the critical path in [11] with an 8bit comparator taking 15gate delays. So to get the fastest filter, we need optimization of the circuit of this stage. We used a basic 2 bit comparator as shown in Fig3. This 2- bit comparator outputs b>a signal. We used similar version to get a>b signal. Every 2-bit pairs of the two numbers to be compared are compared in parallel using this idea. Two such blocks are merged to make a 4-bit comparison block as shown in Fig 4(a) using the merging circuit in Fig4 (b). Using same merging circuit, we can use two 4-bit comparison blocks to make an 8-bit comparison block. Hence the critical path of the 8-bit comparator is one 2-bit comparator circuit of Fig.3, which is 3 gates plus one inverter and two merging circuit of Fig.4(b) with two gates plus one inverter each. Thus total of seven 2-input gates plus three inverters delay as critical delay in comparator. The beauty of this comparator design is that whenever number of bit are doubled, the critical path increases by just one merging circuit which is 2 gate plus one inverter delays as in Fig 4(b). This design is much better than the one used in [11].

B. Addition Stage:

This stage is just a cluster of half adders and full adders that add 8 single bits. For speed constraint, we have used parallel addition of bits, as shown in structure given in Fig5. The ge bits indicate the a >= b output bits from the comparators shown in Fig2. The addition of all eight comparator outputs (ge) gives the number of elements in window array, which are less than the new element. Thus the output (su) of the adder stage gives position of the new value in the sorted array which is a 4 bit number ranging form 0000 to 1000. In our implementation we split this stage into two pipelined stages to reduce the delay.

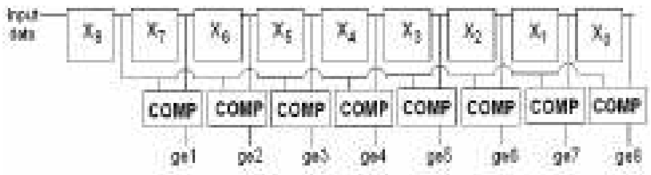

Fig.2: Window registers with Comparators

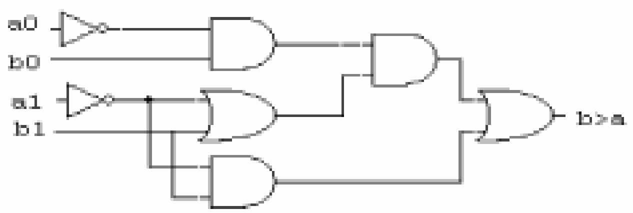

Fig.3: 2-bit comparator circuit

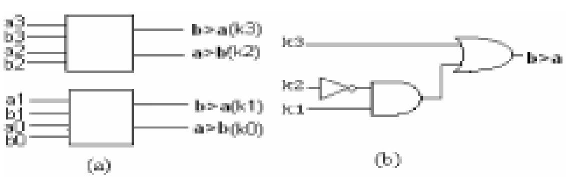

Fig.4: (a) Two 2-bit comparator blocks (b)Integration circuit for 4-bit comparator using (a)

C. Shifting Stage (Sorting and Aging):

This is one of the key stage in the median filtering process and thus has been described here in detail for the sake of clarity. This stage contains 18 registers in all and some combinational logic gates. Nine 8-bit registers for the sorted array (in descending order) and nine 4-bit registers for the aging array which can have nine possible values from 0000 to 1000. The oldest value in the sorted array is the one with its corresponding aging array value equal to 0000. The output of the addition stage 'su' indicates the position of the newly arrived value in the sorted array after the oldest value gets removed. 'su' can have nine possible values from 0000 to 1000. Also there are nine possible places for the oldest value, to be present in the sorted array. So with every clock pulse, right and left shift operations must be performed in sorted and aging arrays keeping 9×9 = 81 possible cases in view as shown in Table-I. These operations can be done as explained below: To each sorted register and the corresponding aging register, we assign some signals that control the data that has to enter the register. So for any sorted register S_i, if data has to shift right into it i.e., from S_{i+1} to S_i, then a signal $r_i = 1$ is generated. Similarly if data has to left shift from S_{i-1} into S_i, then a signal l_i=1 is generated. Since the register S_8 has no register to its left, data cannot right shift into it. So there is no signal called r_8. in table-1. Similarly data cannot left shift into register S_0 as there is no register to its right. So there is no signal called l_0 in table-1

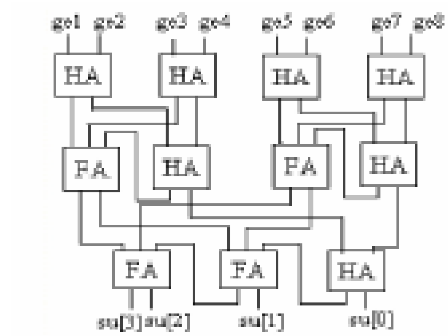

Fig.5: Adder Stage Circuit

If su=i , then the newly arrived value (X) as in Fig.6, has to enter S_i, with an assigned signal c_i equal to logic “1”. In case su ≠ i, c_i=0. In case of aging registers, the effect of the signals r_i, l_i, and c_i remains the same as their corresponding sorted registers. But when the newly arrived value(X) enters S_i, then a 4-bit value of 8 (i.e. V_8=1000), is assigned to its corresponding aging register A_i. In addition to this, the values in all the aging registers except the register in which V_8 enters have to decrement with every clock pulse. The decrementors are used at the output of all aging registers, and connected to the corresponding aging registers with control signals of r_i or l_i to get right or left shift respectively. D_i is the decremented value of the content A_i. D_{i-1} and D_{i+1} are the decremented values of right and left neighbors of A_i. Since there is no shifting operation in ageing registers in case of signals r_i and l_i, equal to logic “0” and c_i equal to logic “0”, the ageing registers have to update by feed back from their decrementors. In this case another signal n_i is derived from logic of l_i and r_i and c_i as shown in Fig.8 to update the aging registers.

These signals are used during shifting operation in sorted registers as shown in Fig.6. When the system is reset, the aging registers should get reset to values from 0 to 8. The implementation of this design is as shown in Fig.7. If the oldest value in the sorted array is in i^{th} register, then content in aging array A_i would be zero and an associated signal a_i becomes 1. The computation of r's, l's and c's, is based on Table-I, which is shown in Fig.8 in logical form and its analysis is presented in preceding paragraph for a particular case.

For example; the oldest value is in the 4^{th} sorted array register, then a_4=1. Now say the newly arrived value has to come to the least position i.e., su=0000, then all the values in the sorted array between 3^{rd} register to 0^{th} register need to left shift by one place so that the value in 4^{th} register gets deleted while the newly arrived value can be accommodated in the $zero^{th}$ place. So l_4=l_3=l_2=l_1=1 and c_0=1. This way for each combination of A_i and su, there is some combination of shifting operation (right shift/left shift) as shown in Table-1. Here reader may refer to Table-1 for other cases for clarity.

Table-I

Shifting Logic

	su	$r_7r_6r_5r_4r_3r_2r_1r_0$	$l_8l_7l_6l_5l_4l_3l_2$
$a_0=1$	0000	00000000	00000000
	0001	00000001	00000000
	0010	00000011	00000000
	0011	00000111	00000000
	0100	00001111	00000000
	0101	00011111	00000000
	0110	00111111	00000000
	0111	01111111	00000000
	1000	11111111	00000000
$a_1=1$	0000	00000000	00000001
	0001	00000000	00000000
	0010	00000010	00000000
	0011	00000110	00000000
	0100	00001110	00000000
	0101	00011110	00000000
	0110	00111110	00000000
	0111	01111110	00000000
	1000	11111110	00000000
$a_2=1$	0000	00000000	00000011
	0001	00000000	00000001
	0010	00000000	00000000
	0011	00000100	00000000
	0100	00001100	00000000
	0101	00011100	00000000
	0110	00111100	00000000
	0111	01111100	00000000
	1000	11111100	00000000
$a_3=1$	0000	00000000	00000111
	0001	00000000	00000110
	0010	00000000	00000100
	0011	00000000	00000000
	0100	00001000	00000000
	0101	00011000	00000000
	0110	00111000	00000000
	0111	01111000	00000000
	1000	11111000	00000000
$a_4=1$	0000	00000000	00001111
	0001	00000000	00001110
	0010	00000000	00001100
	0011	00000000	00001000
	0100	00000000	00000000
	0101	00010000	00000000
	0110	00110000	00000000
	0111	01110000	00000000
	1000	11110000	00000000
$a_5=1$	0000	00000000	00011111
	0001	00000000	00011110
	0010	00000000	00011100
	0011	00000000	00011000
	0100	00000000	00010000
	0101	00000000	00000000
	0110	00100000	00000000
	0111	01100000	00000000
	1000	11100000	00000000
$a_6=1$	0000	00000000	00111111
	0001	00000000	00111110
	0010	00000000	00111100
	0011	00000000	00111000
	0100	00000000	00110000
	0101	00000000	00100000
	0110	00000000	00000000
	0111	01000000	00000000
	1000	11000000	00000000
$a_7=1$	0000	00000000	01111111
	0001	00000000	01111110
	0010	00000000	01111100
	0011	00000000	01111000
	0100	00000000	01110000
	0101	00000000	01100000
	0110	00000000	01000000
	0111	00000000	00000000
	1000	10000000	00000000
$a_8=1$	0000	00000000	11111111
	0001	00000000	11111110
	0010	00000000	11111100
	0011	00000000	11111000
	0100	00000000	11110000
	0101	00000000	11100000
	0110	00000000	11000000
	0111	00000000	10000000
	1000	00000000	00000000

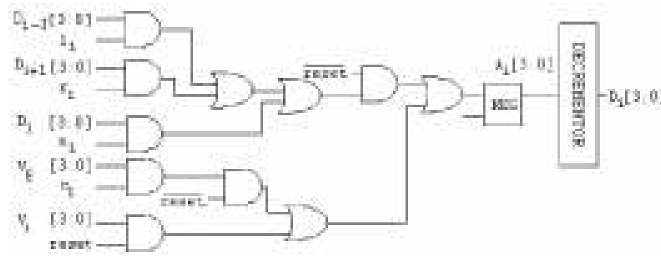

Fig.6: Sorting Array Register with control signals

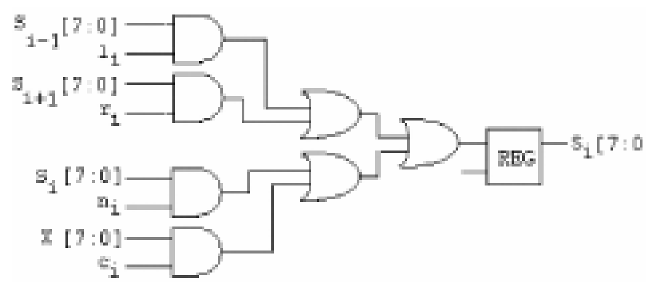

Fig.7: Aging Array Register with control signals

IV. SYNTHESIS RESULTS & DISCUSSION

The technology independent results of proposed architecture is presented in terms of gate delay in Table-II which is critical delay and vital for high throughputs. The critical delay, in terms of number of 2-input gate delays is much better than the results presented in [11]. The critical path delay in terms of gate delay increases by two with doubling of word size. This can be realized from Fig.4(b). The latency of our proposed parallel and pipelined architecture is only four clock cycles. But the window array requires initial cycles known as input latency to get window elements. Thus the input latency is equal to the window size (in this case 9) which can not be avoided. Hence total latency is sum of input latency plus architectural latency, which is equal to thirteen clock cycles in this architecture

$l_1 = (\sim(a_0)) \,\&\, (\sim(su[3] \mid su[2] \mid su[1] \mid su[0]))$ $l_2 = (\sim(a_1 \mid a_0)) \,\&\, (\sim(su[2] \mid su[1] \mid su[0]))$

$l_3 = (\sim(a_2 \mid a_1 \mid a_0)) \,\&\, (\sim(su[3] \mid su[2]\ (su[1] \,\&\, su[0])))$ $l_4 = (\sim(a_3 \mid a_2 \mid a_1 \mid a_0)) \,\&\, (\sim(su[3] \mid su[2]))$

$l_5 = (a_8 \mid a_7 \mid a_6 \mid a_5) \,\&\, (\sim(su[3] \mid (su[2] \,\&\, (su[1] \mid su[0]))))$ $l_6 = (a_8 \mid a_7 \mid a_6) \,\&\, (\sim(su[3] \mid (su[2] \,\&\, su[1])))$

$l_7 = (a_8 \mid a_7) \,\&\, (\sim(su[3] \mid (su[2] \,\&\, su[1] \,\&\, su[0])))$

$l_8 = (a_8) \,\&\, (\sim su[3])$

$r_0 = (a_0) \,\&\, (su[3] \mid su[2] \mid su[1] \mid su[0])$ $r_1 = (a_1 \mid a_0) \,\&\, (su[3] \mid su[2] \mid su[1])$

$r_2 = (a_2 \mid a_1 \mid a_0) \,\&\, (su[3] \mid su[2] \mid (su[1] \,\&\, su[0]))$

$r_3 = (a_3 \mid a_2 \mid a_1 \mid a_0) \,\&\, (su[3] \mid su[2])$

$r_4 = (\sim(a_8 \mid a_7 \mid a_6 \mid a_5)) \,\&\, (su[3] \mid (su[2] \,\&\, (su[1] \mid su[0])))$ $r_5 = (\sim(a_8 \mid a_7 \mid a_6)) \,\&\, (su[3] \mid (su[2] \,\&\, su[1]))$

$r_6 = (\sim(a_8 \mid a_7)) \,\&\, (su[3] \mid (su[2] \,\&\, su[1] \,\&\, su[0]))$

$r_7 = (\sim(a_8)) \,\&\, (su[3])$

$n_0 = (\sim(r_0 \mid c_0))$

$n_i = (\sim(r_i \mid l_i \mid c_i))$; for i = 1 to 7 $n_8 = (\sim(l_8 \mid c_8))$

Table-II
Comparator Critical Delay

Word Size	Critical Delay [11]	Critical Delay [our proposed architecture]
8-bits	15 Gates	7 Gates + 3 Inverters
16-bits	----------	9 Gates + 4 Inverters

Table-III
Characteristic Parameters

Parameter	[12]Real-Time Median Filter Chip	Our Proposed Architecture
	3µ CMOS Technology	0.18µ CMOS Technology
Die Size (mm^2)	49.7	1.2
Max.Clock Frequency(*MHz*)	50	700
Max.ThroughPut (*Mega-medians/sec*)	50	700
Max.Power Dissipation(*mW*)	800	92.5

Fig.8. Controller circuit signals (r_i,l_i, n_i and c_i) (Where &, | and ~ stands for AND, OR and INVERTER function respectively)

.

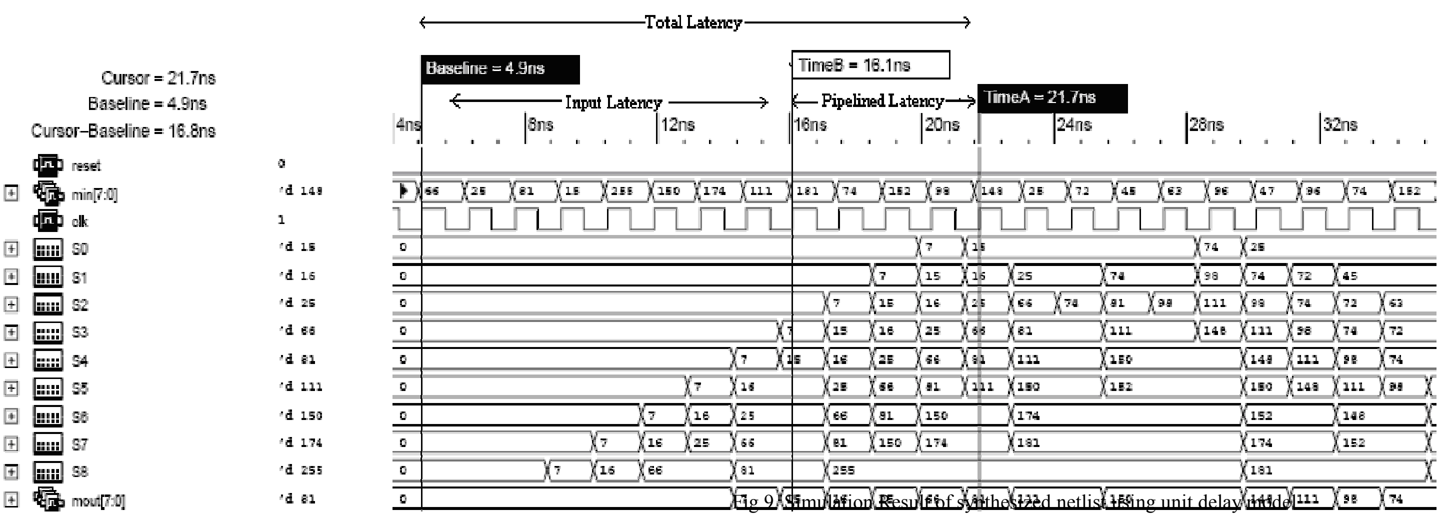

Fig 9. Simulation Result of synthesized netlist using unit delay mode

The proposed architecture for median filter has been coded using verilog HDL and synthesized using design compiler from Synopsys with commercially available $0.18\mu m$ CMOS technology and finally the synthesized net-list has been verified using random test vectors. The snapshot of the simulated output of synthesized netlist using unit delay models has been shown as in Fig9. Readers can observe that the median output is the middle value (S4=111) in sorted array as in example given in Section-II. The resultant latency of the proposed architecture has been also marked in Fig.9. The synthesized result after allowing 20% extra area for physical implementation and positive slacks to avoid any timing violation shows that the proposed VLSI architecture gives core area and power of $1.2mm^2$ and $92.5mW$ respectively at clock frequency of $700MHz$. This results are compared with the results in [12] as shown in Table-III.

V. CONCLUSION

The present paper describes a novel VLSI architecture to implement a one dimensional real time median filter. The proposed architecture is described at each level of hardware design to optimize for critical delay. Parallel and pipelined technique has been used to enhance the throughput. The technology independent HDL codes can be used for either ASIC implementation or advanced FPGA implementation. Although this architecture is proposed for one dimensional signals to suppress impulse noise, this can be used with some extra circuitry for median filtering in images. This architecture is very regular and can be extended and modified for different window size as well as different sampled word size so as to adopt it for specific application of real time signal and image processing.

REFERENCES

[1] J. W. Tukey, “Nonlinear (nonsuperposable) methods for smoothing data,” in Conf Rec., EASCON, pp. 673-674. I974

[2] Digital Image Processing ,second edition,by R C.Gonzalez , R E. Woods, Princeton publisher, 2005.

[3] K. Oflazer, "Design and Implementation of a Single-Chip 1-D Median Filter," IEEE Trans on Acoustics, Speech, and Signal Process, vol.31, no. 5, pp. 1164- 1168 October 1983

[4] R.L.Swenson and K.R.Dimond, "A Hardware FPGA Implementation of 2-D Median Filter using a Novel Rank Adjustment Technique," 7th International conference on Image Processing and its Applications, vol.1. pp. 103-106, July 1999

[5] G.R.Arce and P.J.Warter, “A median filter architecture suitable for VLSI implementation”, in proc.23rd Annu. Allerton conference on commu., contr., comput., pp.172-181, oct. 1984.

[6] A.L.Fisher, “Systolic algorithms for running order statistics in signal and image processing,” J. Digital Syst., vol.4, No.2/3, pp. 251-264, 1982.

[7] D S. Richards, "VLSI Median Filters," IEEE Trans. Acoust. Speech Signal Process, vol.38, pp. 145–153, January 1990

[8] L. Breveglieri, V. Piuri , “Pipelined Median Filters,” in Proc. IEEE Instrumentaion and Measurement Technology Conference (IMTC94), vol.3, pp. 1455-1458, Hamamatsu, Japan, May 1994.

[9] Sung-Jea Ko, Yong Hoon LE, and Adly T. Fam, “Efficient Implementation of One-Dimensional Recursive Median Filters,” IEEE Trans on Circuits Syst, Vol.37, pp. 1447-1450, November 1990

[10] S. B. Leeb, A. Ortiz, R. F. Lepard, S. R. Shaw, and J. L. Kirtley, “Applications of Real-Time Median Filtering with Fast Digital and Analog Sorters,” IEEE Trans on Mechatronics, Vol.2, No.2, pp. 136-143, June 1997

[11] C.J.Tsai, E.H.Lu, C.H.Chen, J.Y.Lee, and I.C.Jou, "A New Architecture of Median Filters With Linear Hardware Complexity," Circuits and Systems, IEEE International Symposium, vol.1, pp. 101-103, June 1991

[12] M. Karaman, L. Onural, and A. Atalar , "Design and implementation of a general purpose Median Filter Unit in CMOS VLSI," IEEE Journal of Solid state circuits, vol.25, No2, pp. 505-513, April, 1990.