A Low Power and High Speed Pipeline Architecture using adaptive Median Filter for Noise Reduction in image Processing

S. Nirmal Raj[1], S. Ashok[2], P. Bala Vengateswarlu[3], G.Vishnu Vardhan Rao[4]
[1,2,3,4] Department of Electronics and Communication Engineering,
[1,2,3,4] Veltechmultitech Dr.Rangarajan Dr.Sakunthala Engineering College,
Avadi, Chennai, India
nirmal.vlsi2014@gmail.com[1], sashok@veltechmultitech.org[2], balavenkatesh@veltechmultitech.org[3],
vishnuvardhan@veltechmultitech.org[4]

ABSTRACT: *Low level data processing purposes are like FIR filtering, recognition of patterns or correlation, whereas the parallel implementation is upheld bythe design matched distinct intention arithmetic; elevated throughput FPGA routes facilely output waveform even for the most advanced DSP processors. In this paper the examination of a high-speed non-linear Adaptive median filter implementation is presented. Next the Adaptive Median Filter solves the dual intention of removing the impulse noise from the image and cutting to distortion in the image. Adaptive Median Filtering can be accomplish the filtering procedure of an image corrupted alongside impulse noise.*

Keywords: FIFO, Median Filter, Sorting, FIR Filter, Rank Calculation, Rank Selection, Median Selection

Received: 14 December 2015, Revised 20 January 2016, Accepted 28 January 2016

1. Introduction

Lowering the dynamic power of a very-large-scale integrated circuit is an experienced method to cut the total power consumption. One of the best methods to cut the dynamic power dissipation is to minimize the switching activities, i.e., the number of gesture transitions in the circuit .The token-ring architecture, adopted in the design of a mixed-timing first-in–firstout (FIFO) interface proposals the possible for low power consumption as data are immobile in the FIFO. In their designs, after data is queued, it will not be motivated and is plainly dequeued in place. A token, that is embodied as logic state 1 in the token list, is utilized to manipulation the dequeing of aged data and queuing of new data at the alike time. All the token lists form a token ring, that is a series of nodes interconnected in a circular manner. The token-ring design alongside precisely one token will be adopted in our design for lowering the power consumption of a one dimension (1-D) median filter.

A median filter is a nonlinear filter extensively utilized in digital signal and image processing for the flattening of signals, suppression of impulse noise, and frontier preservation The median filter replaces a example alongside the middle-ranked worth amid all the examples inside the example window, concentrated concerning the example in question. Reliant on the number of examples processed at the alike series, there are two kinds of architectures for hardware design, i.e., word-level architectures and bitlevel architecture. In the word-level architectures, the input examples are sequentially processed word by word, and the bits of the example are processed in parallel. On the contrary, the bit-level architectures process the examples in parallel and the bits of the incoming examples are sequentially processed .In this brief, the word-level architectures will be adopted in the design of a low-power median filter for useful use. The median of a set of examples in the word-level sorting web is frequently computed by early sorting the input examples and next selecting the middle value. In their methods, the input examples are sequentially processed word by word, and the incoming example is inserted into the correct locale in two steps. In the early pace, the oldest example is removed from the window by advancing a little of the stored examples to the left. In the subsequent pace, the incoming example is contrasted alongside the by now sorted examples and next inserted in the right locale by advancing a little of them to the right.

The distinction between the two architectures in is that these two steps are individually performed in two clock cycles,but in this it takes only one cycle. In both of their methods, though some of the stored examples have to be shifted left or right, depending on their values when a early input samples enters the window. For some applications that require a better example width, more signal transitions in the circuit are needed; i.e., more dynamic power will be obsessive. To overcome this trouble, a new median filter architecture targeting low power consumption is projected. as a substitute of sorting the samples physically in the window, the stored samples are kept immobile there. Only the rank of each sample, which uses fewer bits, has to be efficient at each latest cycle when an input sample enters the window. Since our architecture is implemented as a two-stage pipeline, the median output, which is the sample with median rank, will also be generated at each cycle. The development in power consumption is achieved by utilizing a token ring in our architecture. Since the stored samples in the window are stationary, our architecture is appropriate for low-power applications.

2. Existing System

2.1 Architecture

Figure 1 gives an overview of our low-power median filter design alongside window size *N*. It consists of a circular array of *N* identical cells and three auxiliary modules: rank calculation (Rank Cal), rank selection (Rank Sel), and median selection (Median Sel). All the cells are additionally related to a globe input list *X*, across that they accord the incoming example, and the median is stored in the output list *Y*. The design is requested as a two-stage pipeline, whereas the lists in all the cells assist as the inner pipeline registers. All the lists in the design are synchronized by the rising frontier of a globe clock. Every single cell block ci is composed of a rank creation (Rank Gen) module, a comparator module "==,"and three registers: an m-bit ($m = log2\ N$) locale list (*Pi*),a data list (*Ri*), and a 1-bit token list (*Ti*).

Register *Ri* stores the worth of the example in cell *ci*, list *Pi* keeps the rank of this example, and the enable signal (*en*) of *Ri* is stored in list *Ti*. All the examples in the window are ranked according to their benefits, even though of their physical locations in the window.

In our design, a cell alongside a larger example worth will be associated alongside a larger rank. Though, for two cells *ci* and *cj*, whose example benefits are equal, *ci* will be given a larger locale if *Ri* is newer than *Rj* (or *Rj* is older than *Ri*); i.e., the example in *cj* enters the window preceding than the example in *ci*. The locale is hence exceptional for every single cell. For a window alongside size *N*, the locale starts from 1 for a cell alongside the least example worth, and ends alongside *N* for a cell alongside the biggest example value. The median of the window can next be obtained from the example worth *Ri* of a cell *ci* whose locale *Pi* is equal to (N + 1)/2, assuming *N* is an odd number.

In the design, the input example enters the window in a FIFO manner. Later it is queued, it will not be advanced and is plainly dequeued in place. A token, that is embodied as logic state 1 in the token list of a little cell, is utilized to manipulation the dequeuing of aged example and queuing of new input example at the alike time. Later the token is utilized, it will be bypassed to the subsequent cell at a new cycle.

All the token lists form a token ring alongside precisely one token. Whenever an input example enters the window at a new series, the rank of every single cell has to be updated. It could have to be recalculated, or could be the aged rank decremented by 1, incremented by 1, or retained unchanged. The new rank of every single cell, that is denoted by signal *Qi* in the cell, is generated by the rank Gen module. Every single rank Gen module receives gesture A from the rank Cal module and signal *B* from the rank Sel module. Signal *A* is the recalculated rank of a cell *ci* that encompasses the token and signal *B* is the aged rank of *ci*. Moreover, signal *Yi* is the output of a comparator module "==," that assesses the worth of rank *Pi* alongside a steady worth (N + 1)/2 so that *Yi* = 1 if *Pi* is equal to (N + 1)/2, else *Yi* = 0. This signal issused to indicate if the corresponding cell ci encompasses the median in

Ri. Comparable to the Locale Sel module, the Median Sel module transfers the worth of Rito the output list *Y* if *Yi* = 1; i.e., if the median is stored in *Ri*. The figure 1 shows the low-power filter architecture.

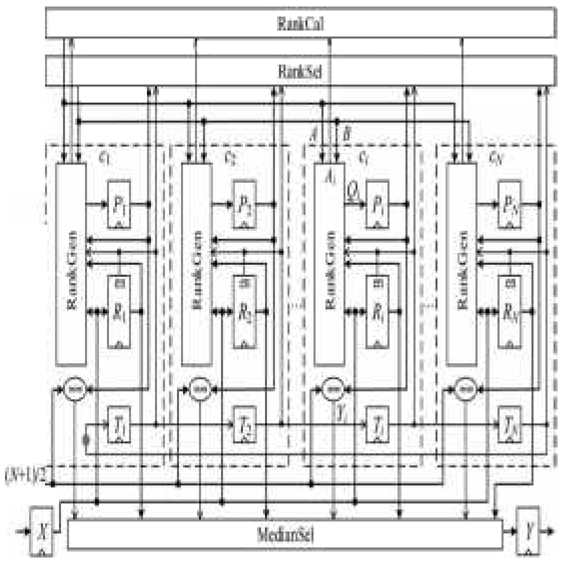


Figure 1. Low-power filter architecture

2.2 Circuit Behavior

At every single contraption series ti, the early period of our two-stage pipelined filter performs the pursuing procedures for the input example *X*: compute the new rank of every single cell, insert *X* in a cell that encompasses the token, and bypass the token to the subsequent cell . This way that for all *Pi*, *Ri*, and *Ti* lists, their new benefits will be computed and ambitious at this period so that they can be notified at the subsequent series *ti*+1. At the alike period, the subsequent pipeline period computes the median worth for the input example that enters the window at the preceding series *ti*-1. That is, for the output list *Y* , its new worth will be computed at this period so that it can additionally be notified at the subsequent series *ti*+1.

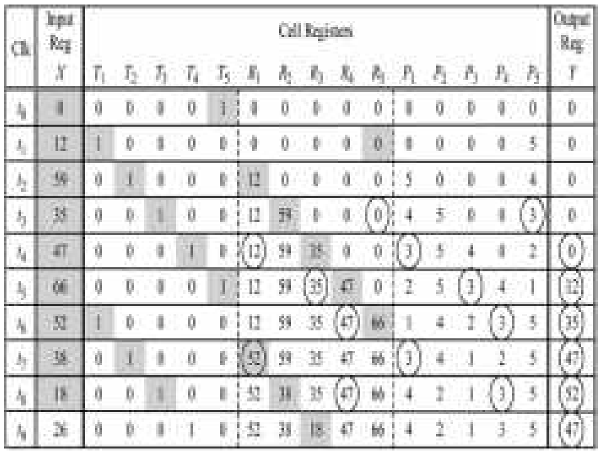
For every single cell ci, the benefits of its *Pi*, *Ri*, and *Ti* lists are all shown in the figure. Initially, at series t0, to make the early input example be stored in the early cell *c*1, the last cell *c*5 is projected to encompass the token (*T*5 = 1).The rank and example benefits (*Pi* and *Ri*) of every single cell, alongside alongside the benefits of the two input/output lists *X* and *Y* , are all reset to be zero.

When the early example 12 enters the window at series *t*1, the token has been advanced from *c*5 to *c*1 (*T*1 = 1 and *T*5 = 0). The worth of *P*5 has additionally been notified to be 5 as the early zero of *X* (now stored in *R*5) is indulged as a adjacent example at the early series *t*0. To design for the subsequent series *t*2, the new worth of *R*1 will be ambitious as 12 to store the input example as *c*1 encompasses the token. The new worth of *P*1 will be computed as 5 as example 12 (to be stored in *R*1) is larger than the example benefits of the supplementary four cells.

Finally, the new benefits of *T*1 and *T*2 will be ambitious as 0 and1, suitably, to indicate that the token will be advanced from c1to *c*2. All the benefits of *Pi*, *Ri*, and *Ti* will next be notified at the subsequent seriest 2. After the window is fully inhabited alongside valid data at series t6, cell c1 holds the token once more (*T*1 = 1).The new worth of the median output *Y* for the subsequent series *t*7will be ambitious as the worth of *R*4 (47) as rank *P*4 is equal to 3, i.e., (5 + 1)/2. As the counseled design is a two-stage pipeline, after *Y* is notified to be 47 at series *t*7 for the input example 66, the benefits of all *Pi*, *Ri*, and *Ti* will additionally be notified at this series for the subsequent input example 52. It can be perceived from this example that after an input example is inserted into the window, the aged example in every single cell will not be moved. Instead, the rank of every single cell is recalculated so that the new median can be obtained in a cell whose rank is equal to (*N* + 1)/2.

2.3 Rank Updating

This section explains how to determine the new rank for each cell. Two types of cells will be separately discussed: a cell with the token and a cell without the token.



Clk	Input Reg	Cell Registers															Output Reg
	X	T_1	T_2	T_3	T_4	T_5	R_1	R_2	R_3	R_4	R_5	P_1	P_2	P_3	P_4	P_5	Y
t_0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
t_1	12	1	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0
t_2	59	0	1	0	0	0	12	0	0	0	0	5	0	0	0	4	0
t_3	35	0	0	1	0	0	12	59	0	0	0	4	5	0	0	3	0
t_4	47	0	0	0	1	0	12	59	35	0	0	3	5	4	0	2	0
t_5	66	0	0	0	0	1	12	59	35	47	0	2	5	3	4	1	12
t_6	52	1	0	0	0	0	12	59	35	47	66	1	4	2	3	5	35
t_7	38	0	1	0	0	0	52	59	35	47	66	3	4	1	2	5	47
t_8	18	0	0	1	0	0	52	38	35	47	66	4	2	1	3	5	52
t_9	26	0	0	0	1	0	52	38	18	47	66	4	2	1	3	5	47

Table 1. Example illustrating the insertion of nine input samples into a window

2.4 Cell with the Token

For a cell *ci* with the token, its sample value *Ri* will be replaced by the input sample *X*, and its rank *Pi* has to be recalculated. The new value of *Pi* can be obtained by comparing *X* with the sample values of all the other *N* - 1 cells that do not contain the token. For these cells, if *K* is the number of cells whose sample value is less than or equal to *X*, the new value of *Pi* will be *K* + 1. At cycle t6 of Figure 2, for example, the new value of rank *P*1 will be calculated as 2 + 1 at the next cycle t7.

2.5 Cell without the Token

For a cell ci without the token, its sample value *Ri* will not be affected when an input sample *X* enters the window. However , its rank *Pi* may be affected by the sample value *Rj* of another cell *cj* that contains the token. Since the value of *Rj* will be replaced by *X*, the relation between *Ri* and *Rj* may change. However, the sample value Rk of any other cell *ck* that does not contain the token will not affect Pi since the value of *Rk* will not be changed. Depending on the relation between *Pi* and *Pj*, and the relation between *Ri* and *X*, the new value of *Pi* may be decremented by 1, incremented by 1, or kept unchanged when an input sample *X* is inserted into the window. The change of rank *Pi* can be explained as the following five cases, where cell *ci* does not contain the token, but cell *cj* does.

Case 1: (Decremented by 1) *Pi* > *Pj* and *Ri* <= *X*: If rank *Pi* is greater than rank *Pj*, *Ri* is greater than or equal to (but newer than) *Rj* at the current cycle. If *Ri* is less than or equal to the input sample *X*, *Ri* will be less than or equal to (but older than) the new value of *Rj* at the next cycle. In other words, *Rj* will change from a value that is less than or equal to (but older than) *Ri* to a value that is greater than or equal to (but newer than) *Ri*. Therefore, the number of cells whose sample value is less than or equal to (but older than) Ri will be decremented by1 at the next cycle; i.e., *Pi* has to be decremented by 1. Take rank *P*4 at cycle *t*6 of Figure 2, for example, its value will be decremented by 1 (from 3 to 2) at the next cycle t7.

Case 2: (Incremented by 1) *Pi* < *Pj* and *Ri*>*X*: Similar to Case 1, if rank Pi is less than rank *Pj*, *Ri* is less than or equal to(but older than) *Rj* at the current cycle. If *Ri* is greater than the input sample *X*, *Ri* will be greater than the new value of *Rj* at the next cycle. Therefore, the number of cells whose sample value is less than or equal to (but older than) *Ri* will be incremented by 1 at the next cycle; i.e., *Pi* has to be incremented by 1.

Case 3: (Kept Unchanged) Pi <Pj and Ri<= X: If *Pi* is less than *Pj*, *Ri* is less than or equal to (but older than) *Rj* at the current cycle. If *R*i is less than or equal to *X*, *Ri* will also be less than or equal to (but older than) the new value of *Rj* at the next cycle. Therefore, the number of cells whose sample value is less than or equal to (but older than) *Ri* at the current cycle will be equal to that at the next cycle; i.e., *Pi* has to be kept unchanged. For example, at cycle t7 of Figure 2, the value of rank *P*3 has to be kept unchanged at one at the next cycle *t*8.

Case 4: (Kept Unchanged) *Pi* > *Pj* and *Ri* > *X*: Similar to Case 3, if *Pi* is greater than *Pj* and *Ri* is greater than *X*, the number of cells whose sample value is less than or equal to (but older than) *Ri* at the current cycle will also be equal to that at the next cycle; i.e., *Pi* has to be kept unchanged.

Case 5: (Kept Unchanged) *Pi* = *Pj*:This case occurs when the window is not yet fully occupied with valid data. At the initial state, the rank of each cell is reset to be zero. After the window is fully occupied, each cell will be assigned a non zero and unique rank. If rank *Pi* is equal to rank *Pj*, the values of *Pi* and *Pj* are both zero, and neither cell *ci* nor cell *cj* contains valid data.

The new value of *Pi* at the next cycle will still be zero since *ci* does not contain the token; i.e., *Pi* has to be kept unchanged at zero. At cycle *t*3 of Figure 2, for example, the value of rank *P*4 will still be zero at the next cycle *t*4.It can be obtained from the above discussions that for a cell *ci* that contains the token, its rank *Pi* has to be recalculated at a new cycle. If *ci* does not contain the token, its rank *Pi* maybe decremented by 1, incremented by 1, or kept unchanged .Therefore, there are four sources for *ci* to update its rank.

2.6 Ranksel and Mediansel Modules

The RankSel module is accountable for transferring the rank *Pi* of a cell *ci* to its output *B* if *ci* encompasses the token; i.e., after *Ti* = 1. Figure 2(a) displays a easy implementation of this module employing AND/OR gates. It can additionally be requested by tri-state buffers in Figure 2(b), where as *B* is the output of a globe data bus that accumulates the output signals of all the tristate buffers. As there exists precisely one cell that encompasses the token at each time; i.e., there exists precisely one *Ti* signal whose worth is equal to 1, the worth of *B* will always be valid.

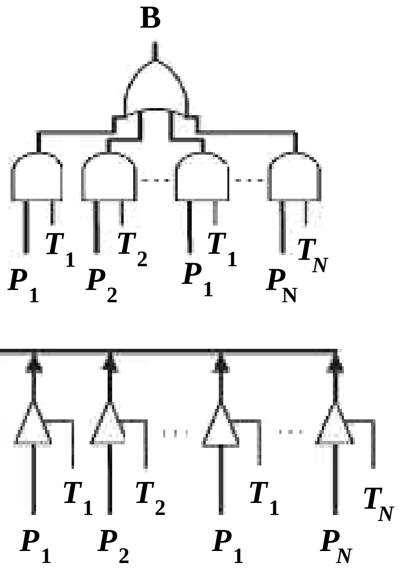


Figure 2. Implementation of the RankSel module

(a) Using AND/OR gates. (b) Using tristate buffers

The MedianSel module can additionally be requested in a comparable way (Figure 2). It transfers the worth of *Ri* to the output list *Y* if *Ri* is the median; i.e., after *Yi* = 1. Though, if it is requested by tristate buffers, the median will be valid merely after there exists at least (*N* + 1)/2samples in the window; or else, it will stay in a elevated impedance state.

2.7 Rankgen and Rankcal Modules

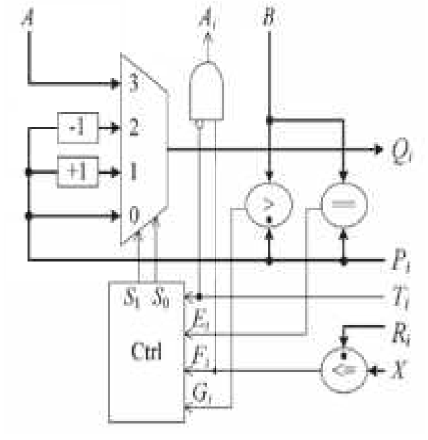
For the RankGen module in a cell ci, its implementation is given in Figure 4(a). signal *Fi* is the output of a comparator module“<=,” that assesses the worth of Ri alongside that of the input example *X* so that *Fi* = 1if *Ri* is less than or equal to *X* (*Ri*<=*X*), else *Fi* = 0 (*Ri* > *X*). signal *Ai* is the output of a logic AND gate so that *Ai* = 1 if *Ti* = 0 and *Fi* = 1; i.e., if cell *ci* does not encompass the token and Riis less than or equal to *X*, else *Ai* = 0. The Ai signal of every single cell is related to the RankCalmodule, that computes the new rank of a cell that encompasses the token. New rank is computed as *K* + 1, whereas *K* is the number of cells, that does not encompass the token and whose example worth is less than or equal to *X*; i.e., *K* is equal to the number of logic 1's on the *Ai* signals of all the cells.

The RankCal module can be requested by a multiinput adder that adds all the *Ai* signals and next increments the sum by 1. If cell ci encompasses the token, the output *A* of the RankCal module will be its new rank at the subsequent series. On the contrary, if cell ci does not encompass the token, its new rank will be ambitious by the supplementary signals. signal *Gi* is the output of a comparator module ">," that assesses the worth of rank *Pi* alongside that of signal *B* so that *Gi* = 1 if *Pi* is larger than *B*, else *Gi* = 0. As the worth of *B* is the rank *Pj* of one more cell *cj* that encompasses the token, the meaning of *Gi* can additionally be delineated as Gi= 1 if *Pi* is larger than *Pj*(*Pi* >*Pj*), else *Gi* = 0 (*Pi* <= *Pj*). signal *Ei* is the output of a compactor module "==," that additionally assesses the worth of *Pi* alongside that of *B* so that *Ei* = 1 if *Pi* is equal to B, else *Ei* = 0. Likewise, the meaning of *Ei* can be delineated as *Ei* = 1 if *Pi* is equal to *Pj(Pi* = *Pj*), else *Ei* = 0. Joining these two signals *Ei* and *Gi*, for the three relations amid *Pi* and *Pj*(*Pi* >*Pj*, *Pi* = *Pj*, and *Pi* < *Pj*), the corresponding worth of *Ei Gi* will be 01, 10, and 00, respectively.

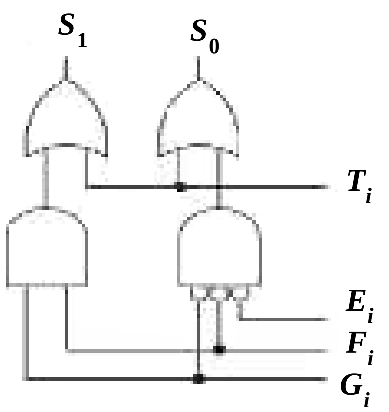
2.8 CTRL Module

For a cell *ci*, since there are four possible sources to update its rank, a 4-to-1 multiplexer is used to select one of these sources for signal Qi in Figure 3(a).

Then, the worth of rank *Pi* will be notified by the worth of *Qi* at every single cycle. The multiplexer is manipulated by two selection signals S1 and S0; and these two signals, that are generated by the Ctrl module, are ambitious by four signals *Ti*, *Ei*, *Fi*, and *Gi*. If cell ci encompasses the token (*Ti* = 1),its new rank is obtained from the output *A* of the RankCal module; i.e., the worth of *S*1*S*0 ought to be 11 when *Ti* = 1. If cell *ci* does not encompass the token (*Ti* = 0),there are five cases for this. In Case 1 after *Pi* >*Pj*(*EiGi*= 01) and *Ri* <= *X* (*Fi* = 1),the rank of *ci* will be decremented by 1; i.e., the worth of *S*1*S*0 should be 10 after *EiFiGi* = 011. Comparably in Case 2, after *Pi* <*Pj* (*EiGi* = 00) and *Ri*> *X* (*Fi* = 0),the rank of *ci* will be incremented by 1; i.e., the worth of *S*1*S*0 ought to be 01when *EiFiGi* = 000. Finally, after *Pi* <Pj(*EiGi*= 00) and *Ri*<= X (*Fi* = 1) in Case 3, *Pi* > *Pj*(*EiGi* = 01) and *Ri*>*X* (*Fi* = 0) in Case 4, and *Pi* = *Pj*(*EiGi* = 10) in Case 5, the rank of *ci* will be retained unchanged; i.e., after *EiFiGi* = 010,001, or 1 - 0, the worth of *S*1*S*0 ought to be 00. Figure 3 (b) depicts a easy implementation of the Ctrl module.



(a)



(b)

Figure 3. (a) Implementation of the Rank Gen module. (b) Implementation of the Ctrl module

3. Proposed System

3.1 Adaptive Median Filter

The Adaptive Median Filter is projected toremove the setbacks confronted alongside average median filter. The frank difference amid the two filters is that, in the Adaptive Median Filter, the size of window encircling every single pixel is variable. This variation depends on the median of the pixels in the present window. If the mediansof pixel worth is an impulse, next the size of the window is expanded. Otherwise, more processing is completed on portion of the image inside the present window specifications.

'Processing' the image basically entails the following: The center pixel window is assessed to confirm whether it is an impulse or not. If it is an impulse, next new worth of that pixel in filtered image will be the median worth of the pixels in that window. If, though, the center pixel is not an impulse, next the worth of the center pixel is retained in the filtered image. Thus, unless the pixel being believed is an impulse, the grayscale worth of the pixel in the filtered image is the alike as that of the input image. Thus, the Adaptive Median Filter solves the dual intention of removing the impulse noise from the image and cutting distortion in the image. Adaptive Median Filtering can grasp the filtering procedure of an picture corrupted alongside impulse noise of probability larger than 0.2. This filter additionally smoothens out supplementary kinds of noise, therefore, providing a far larger output image than the average median filter.

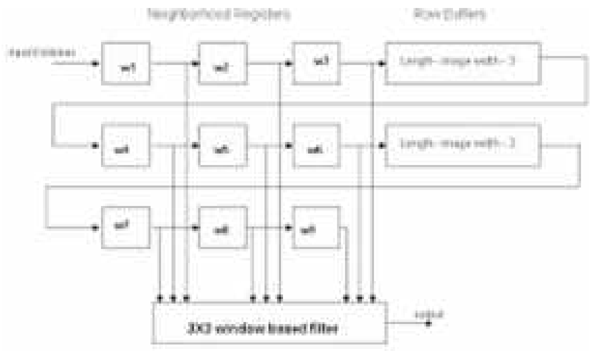


Figure 5. Implementation of a 3*3 filter window

3.2 Parallel Sorting Strategy

To make fair analogy of parallel sorting strategy opposing wave sorter strategy in words of the finished number of needed steps sort an array, it is vital to ponder the steps utilized to elucidate data from recollection and the steps needed to store sorted data back to memory. The counseled way is established on alike construction of lists array utilized in the wave sorter strategy. With this kind of array, data can be stored in the array by dispatching a datum to the early list and afterward, after subsequent datum is dispatched to the early list, the worth on the early array is advanced to subsequent register. Thus, for every single datum dispatched to the array to be stored, benefits in lists advanced to their corresponding adjacent registers. This procedure needs to n steps. The alike number of steps is needed to seize data out from the array. This way permits store a new set of data in the array as the preceding set is being dispatched back into the memory. As remarked in serving 2, suffix sorting could imply extra than one sorting iterations. If *k* sorts are needed, next parallel sorting needs (($n + n/2$) * $k + n$) to sort an array of n data.

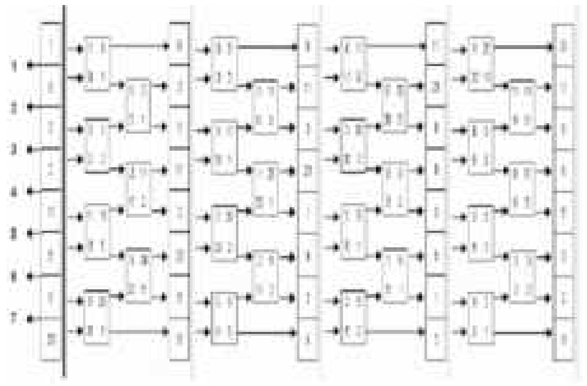
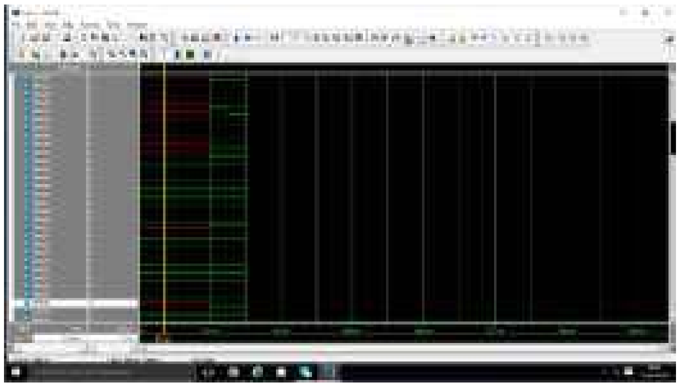
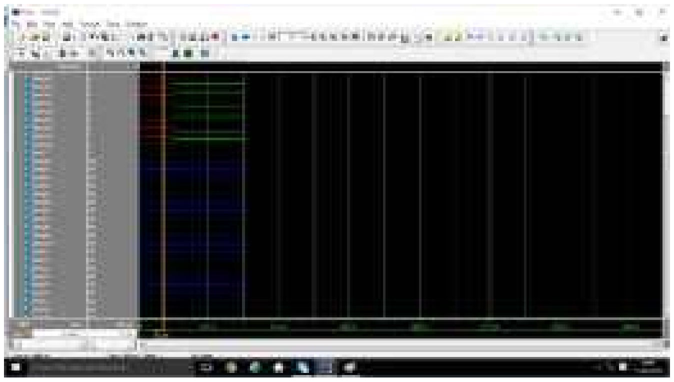


Figure 6. Parallel sorting

4. Result and Conclusion

Thus the paper investigated a elevated speed nonlinear adaptive median filter that is been requested in cutting the deblurring results in an image and bestowing a smoothening results on the edges.The picture is enhanced from one dimensional feature to two dimensional.The adaptive median filter solves the dual intention of removing the impulse noise from image and reducing the distortion in the image.





References

[1] Cadenas, J., Megson, G. M., Sherratt, R. S., Huerta, P. (2012). Fast median calculation method, *Electron. Lett.*, 48 (10), 558–560, May.

[2] Chelcea, T., Nowick, S. M. (2004). Robust interfaces for mixed-timing systems, *IEEE Trans. Very Large Scale Integrated. (VLSI) Syst.*, 12 (8), 857–873, August.

[3] Chen, O. T.-C., Wang, S., Wu, Y.-W. (2003). Minimization of switching activities of partial products for designing low-power multipliers, *IEEETrans. Very Large Scale Integrated. (VLSI) Syst.*, 11, 3, 418–433, June.

[4] Chen, R. -D., Chen, P. -Y., Yeh, C. -H. (2013). Design of an area-efficient one dimensional median filter, *IEEE Trans. Circuits Syst. II, Exp. Briefs*, 60 (10), 662–666, October.

[5] Choo, C., Verma, P. (2008). A real-time bit-serial rank filter implementation using Xilinx FPGA, *In: Proceedings SPIE Real-Time Image Process.*, 6811, 68110F-1–68110F-8.

[6] Fahmy, S. A., Cheung, P. Y. K., Luk, W. (2009). High-throughput one Dimensional median and weighted median filters on FPGA, *IET Comput.Digit. Tech.*, 3 (4), 384–394, July.

[7] Panades, I. M., Greiner, A. (2007). Bi-synchronous FIFO for synchronous circuit communication well suited for network-on-chip in GALS architectures, *In*: *Proceedings 1st Int. Symp. NOCS*, 83–94.

[8] Moshnyaga, V. G., Hashimoto, K. (2009). An efficient implementation of 1-D median filter, *In: Proceedings 52nd IEEE Int.*

MWSCAS, 451–454.

[9] Mottaghi-Dastjerdi, M., Afzali-Kusha, A., Pedram, M. (2009). BZ-FAD: A low-power low-area multiplier based on shift-and-add architecture, *IEEE Trans. Very Large Scale Integrated. (VLSI) Syst.*, 17 (2), 302–306.

[10] Prokin, D., Prokin, M. (2010). Low hardware complexity pipelined rank filter, *IEEE Trans. Circuits Syst. II*, *Exp. Briefs*, 57 (6), 446– 450, June.