NOVEL FPGA-BASED IMPLEMENTATION OF MEDIAN AND WEIGHTED MEDIAN FILTERS FOR IMAGE PROCESSING

Suhaib A. Fahmy[§]*, Peter Y. K. Cheung*[§] *and Wayne Luk*[‡]

[§]Department of Electrical and Electronic Engineering, Imperial College London, UK
{s.fahmy, p.cheung}@imperial.ac.uk
[‡]Department of Computing, Imperial College London, UK
wl@doc.ic.ac.uk

ABSTRACT

An efficient hardware implementation of a median filter is presented. Input samples are used to construct a cumulative histogram, which is then used to find the median. The resource usage of the design is independent of window size, but rather, dependent on the number of bits in each input sample. This offers a realisable way of efficiently implementing large-windowed median filtering, as required by transforms such as the Trace Transform. The method is then extended to weighted median filtering. The designs are synthesised for a Xilinx Virtex II FPGA and the performance and area compared to another implementation for different sized windows. Intentional use of the heterogeneous resources on the FPGA in the design allows for a reduction in slice usage, and high throughput.

1. INTRODUCTION

The median filter is a non-linear filter that has been used successfully in a variety of domains. Its strength lies in its ability to filter out impulsive noise without destroying the properties of the underlying signal. In the image processing domain, this is manifested in edges remaining intact, while using linear filters (such as Gaussian filters) would cause edges to become blurred. Given an input sequence $x_1, x_2, x_3, \quad$, of l-bit numbers, we define a window of size $2N + 1$ centred on the ith value as $W_i = \{x_{i-N}, x_{i-N+1}, \quad, x_i, \quad, x_{i+N-1}, x_{i+N}\}$. The output of the median filter, y_i, is thus the median of W_i; the middle value in the sorted list.

The weighted median is an extension of the normal median, where each input sample is given a weight, to determine how much that sample is able to determine the result. Weights can be fractional but are most usually integer values, as this simplifies the calculations. The input sequence becomes $(x_1, w_1), (x_2, w_2), (x_3, w_3), \quad$, where x_i are the input samples and w_i are the corresponding weights. An integer weight would simply correspond to having w_i copies of sample x_i taken into account in the median calculation. As an example the sequence $(1, 3), (2, 1), (3, 5), (4, 2), (7, 2)$ is equivalent to the sequence $1, 1, 1, 2, 3, 3, 3, 3, 3, 4, 4, 7, 7$. It is important to note that for the weighted median, the size of the window is actually the sum of weights rather than the number of tuples received. So for the above sequence it is 13 and not 5.

1.1. The Trace Transform

The motivation for designing an efficient implementation of median filters for large windows comes mainly from our research on hardware implementation of the Trace Transform [1, 2]. This is a recently developed transform in which functionals are calculated on lines crossing an image at differing angles and distances from the origin. The Trace Transform is used to give an alternative representation of the image, useful for recognition and authentication systems. In each case, the number of sample points can be over 100, depending on the size of the image in question. Furthermore, the number of samples of interest can vary as the lines cross the image at varying angles. Hence a scalable method of median calculation is needed, which must also cope with variable window size.

2. RELATED WORK

Median filters have been implemented in a variety of ways. [3] provides a very good review of the area. There are two main methods, the first is to maintain the input sample list in its original order, then pass it through a sorting network. The median value is then simply extracted from the correct output. The other method involves sorting the samples as they enter the system. Of the first approach, the simplest implementation is the bubble sorting grid, where a grid of dual input sorters each swap their inputs to propagate the higher valued samples upwards, and lower valued samples downwards (or vice-versa). The median is simply the middle sample of the grid output. An example of this architecture is shown in Figure 1. This method is regular yet its

0-7803-9362-7/05/$20.00 ©2005 IEEE

hardware requirements increase in proportion to the square of the window size and hence it is not scalable to larger windows. For a window of size $2N+1$, we require $N(2N+1)$ dual input sorters and $2N+1$ registers.

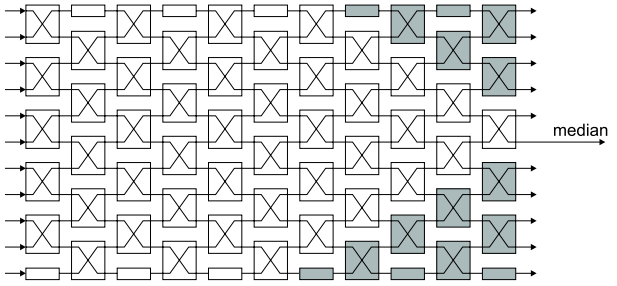

Fig. 1. A simple 11-sample bubble-sorting circuit layout. The large blocks are compare-swap units that swap their inputs if necessary to propagate the larger values upwards and the smaller ones downwards. The small blocks are registers. Note that the shaded blocks are not required for median calculation.

For small windows, simplifications can be made[4], where the columns, then rows are each sorted using a triple-input sorter. Then only one diagonal needs to be sorted to give the median. This saves on hardware requirements. Karaman et al.[5] propose a change to the standard sorting network by dealing with samples in a bitwise manner, needing only single bit sorters, however their implementation is still proportional to N^2 in area. The strength of regular array architectures is that they can be pipelined down to a single compare-swap stage. This means high throughput and frequency. Other methods that use fewer building blocks of higher complexity are described in [6, 7, 8]. Another method is that of threshold decomposition, as used in [9], however the architecture proposed relies on the window being of size 3 3 and uses 3-input adders and so is not scalable to large windows.

The principle of histogram-based median filtering is well established and known, having been mentioned in the basic textbooks on image processing. We have found two references to this method in the literature, [10] and [11], however both deal with software implementations running on general-purpose processors. We present here, what we believe to be the first implementation and analysis in hardware of the proposed method. The high degree of parallelism that can be had in hardware is what makes this method so attractive as compared to a sorting structure.

3. PROPOSED ARCHITECTURE

3.1. General Overview

The proposed architecture works by constructing a cumulative histogram of the input data. This is done by maintaining a count for occurrences of each possible input value. Since the application domain is video processing, we have assumed 8-bit unsigned numbers ($l = 8$). This means there are $2^8 = 256$ possible input values, and so a rank of 256 bins is used. To construct a histogram, when an input value is received, the corresponding bin is incremented. For a cumulative histogram, the corresponding bin and all subsequent bins must be incremented. Hence, the value stored in the final bin will always be equal to the number of input samples received. For example, if the median is to be calculated over a window of 101 elements, i.e. $2N+1 = 101$, $N = 50$, then the 51st (or generally the $(N+1)$th) element in the ordered list is the result. Using the histogram, it is only required to find which bin this 51st value lies in. This gives the median of the input samples, since the 51st ordered element must lie in the bin whose count is the first to reach 51 or surpass it.

To implement this, the content of each bin is compared to the median index (in this example, 51), giving a 0 if the bin count is smaller, and a 1 if it is equal or larger. Hence the result for all bins before the one containing the median will be 0, and all the others will be 1. A priority encoder can then be used to isolate the index of the first bin in the series of 1's. This gives the median of the input samples.

To implement this in hardware requires a register to keep a count for each possible input value. Hence we need 256 registers to store the counts. For all the registers to be updated in parallel, each register also needs an incrementer and a multiplexor (to decide whether or not to increment the stored value). This gives us the design for a bin node processor as shown in Figure 2. 256 (or in the general case 2^l) of these are required in the proposed system.

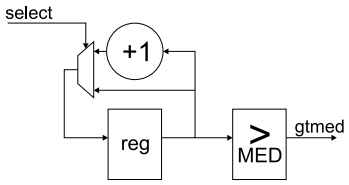

Fig. 2. A bin node processor

We then require a circuit to decide which of these 256 bins to increment for any given input value. One possible approach is to instantiate a comparator for each bin, and to compare the input sample value to the index of each bin. However, this would be costly in terms of hardware. Another approach would be to let each bin check the previous one, and if that is being incremented, then it should increment too. However this would slow the system down massively, since that signal would need to propagate through 256 stages in the worst case, all in one clock cycle.

A more efficient method is to use embedded block-RAMs on the FPGA to store the patterns in a ROM. We would need a 256 256-bit ROM to decode the 8-bit number to

Address	Contents
0	0xFFFFFF...FFFFF
1	0x7FFFFF...FFFFF
2	0x3FFFFF...FFFFF
3	0x1FFFFF...FFFFF
4	0x0FFFFF...FFFFF
5	0x07FFFF...FFFFF
⋮	⋮
253	0x0000000...00007
254	0x0000000...00003
255	0x0000000...00001

Table 1. ROM contents

a 256-bit signal, where each bit represents the select input shown in Figure 2, to the corresponding bin. Each bit of the output addresses a single bin node processor. The access patterns stored in the rom, ensure that the correct processors are enabled for any given input sample. This decoding method saved over 500 slices in the basic implementation. The circuit overview is shown in Figure 3. The contents of the ROM are shown in Table 1.

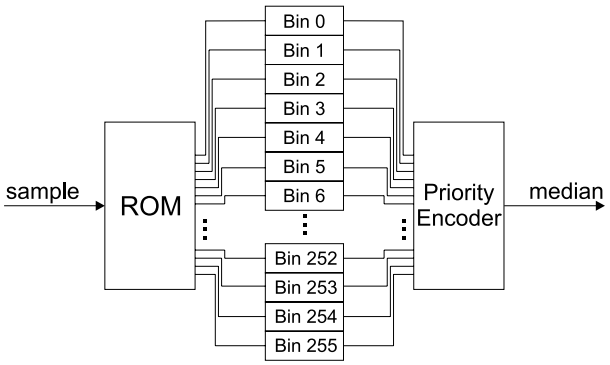

Fig. 3. Histogram-based median filter architecture

3.2. Sliding Window Implementation

To implement this algorithm for sliding windows, a small addition is made. A FIFO buffer must be used to store the samples for the window over which the median must be found. When a new sample is received and the window is full, the oldest sample is removed from the FIFO. Updating the histogram involves decrementing the cumulative count of the bin corresponding to this sample and all subsequent bins. At the same time, the bin corresponding to the new input sample and all subsequent bins must be incremented. This can all be done in one cycle, by simply leaving any bins that are included in both sets alone, since they must be incremented and decremented at the same time. Bins that are only enabled by the access pattern of the new sample are incremented, while bins enabled only by the access pattern of the removed sample are decremented. The new node design is shown in Figure 4.

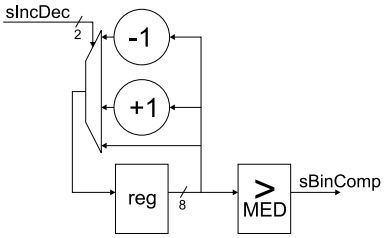

Fig. 4. A bin node for the sliding window implementation. sIncDec is simply a concatenation of the two bits from the ROM lookup

On-chip Block-RAMs are particularly useful for this architecture. Since these RAMs are dual-ported on our target architecture, we are able to extract the enable signals for both the new and oldest samples in parallel. These can then be processed to determine which bin is incremented. This is illustrated in Figure 5.

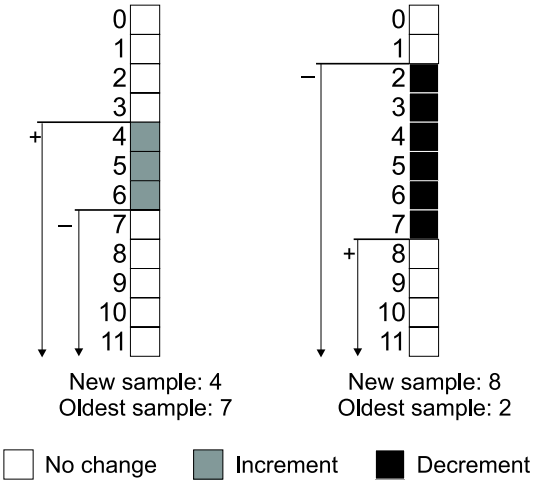

Fig. 5. Application to sliding windows. The arrows beside the bins show the access patterns for the oldest (-) sample and new (+) sample. The leftmost example shows a new sample value of 4 arriving while the oldest sample is of value 7. Only bins 4 to 7 need to be incremented, all others keep their current values. The rightmost example shows a new sample of value 8 arriving, while the oldest sample is of value 2. Only bins 2 to 8 need to be decremented; the others are left alone.

To implement this, all we require is a simple 2-input, 2-output lookup-table to determine the resultant action. This is shown in Table 2. This small logic function must be implemented for each bin. There is however one caveat; that is, that as the window is filling with values the first time, we do

OldEn	NewEn	Inc	Dec
0	0	0	0
0	1	1	0
1	0	0	1
1	1	0	0

Table 2. Extra sliding window logic. The signals OldEn and NewEn are the enable signals for the bin resulting from the ROM lookup of the oldest sample and new sample respectively. Inc and Dec are signals instructing the bin to increment or decrement respectively.

not require any subtractions to take place, since this would mean that our histogram will never fill up with values. As such, we append a single valid bit to each input sample. This propagates through the FIFO mentioned above and emerges at the final stage of the FIFO only when one full window of values has been received. This bit is ANDed with the bin subtraction control signal, so no subtraction can take place until it emerges. The architecture is shown in Figure 6.

3.3. Extension to Weighted-Median

To implement weighted median in the proposed architecture, small changes to the architecture in Figure 6 are needed. Firstly, another input signal is introduced to provide the weights. Instead of a simple incrementer, each bin processor must now add the weight value. For bins enabled by the access pattern corresponding to sample falling outside the window, we simply subtract the weight of that sample. For those bins enabled by both access patterns, we simply add the difference of the two weights (while being careful to maintain the sign). For those bins enabled only by the access pattern for the new sample, we simply add the new sample's corresponding weight. This architecture is illustrated in Figure 7. The three signals fed into each of the bins are the weight of the new input sample, the difference in weights and the weight corresponding to the sample falling outside the window.

The rank of the median is not known in advance for weighted median. Consider the expansion of the sequence shown in Section 1, and it becomes clear that the number of 'real' samples received is equal to the sum of the weights. Hence, the index of the median must be half of that, which is simply a right shift in hardware. In the proposed architecture, the difference of the two weights is simply added to a register on each clock cycle. This maintains the current weight sum. This is right shifted to divide by two and fed into each of the bins, and used for the comparison.

The word-length of each bin register must be wide enough to accommodate the maximum sum of the weights to prevent overflow. There must be some constraint put on the input weights to prevent overflow in any case.

4. IMPLEMENTATION RESULTS

All designs were initially implemented using the Celoxica Handel-C compiler. This allowed for rapid design and testing of these closely related and parameterised designs. The target device was a Xilinx Virtex II 6000, as found on the Celoxica RC300 development board. For comparison, an alternative implementation of the median filter based on the sorting grid mentioned in Section 2 was synthesised. Subsequently, both designs were synthesised in VHDL using Synplicity Synplify and Mapped and Placed-and-Routed using Xilinx ISE. We found that using Handel-C was acceptable for the sorting-grid architecture. However, due to the extra control signals that Handel-C inserts into a design, and the high level of parallelism in our architecture, routing delays were causing our circuit to have a high clock period. For our architecture we saved over half the area and reduced our clock period by over 60% by using VHDL. Part of the reason for the large area saving, is that Synplify inferred simple up-down counters for the histogram nodes for the non-weighted median filter, which can be implemented very efficiently in the Virtex II FPGA.

4.1. Synthesis Results

The proposed system was synthesised for various window sizes, as was the sorting-grid architecture. The synthesis results are shown in Table 3 and the graph in Figure 8. The circuit is able to run at 72MHz, computing one median per clock cycle. The area usage of the sorting grid architecture increases with N^2 while the proposed architecture has a constant area requirement. The slight variation in the number of slices used in the implementations of the proposed architecture, is due to the change in the length of the FIFO that stores the input samples. Not all window sizes were implemented on our architecture because the slice count is almost independent of window size. The crossover point, where the proposed architecture becomes more area-efficient than the grid sorter is at a window size of approximately 27. It is important to note that even though other architectures are more efficient than the sorting grid architecture, their complexities are $O(N)$ or higher. This contrasts with our architecture which has a constant complexity independent of N. Considering that we may require window sizes of 100 samples or more for the Trace Transform, it is clear that the proposed architecture is highly efficient.

Furthermore, the proposed architecture deals elegantly with varying window sizes. Recall that the count stored in the final bin is equal to the number of samples received by the system. This number can simply be right shifted to divide by 2, then used to find the median.

The weighted median architecture was also synthesised and used only 4,548 slices for a 51 sample window, an increase of 50%. Recall that weighted median is a much more

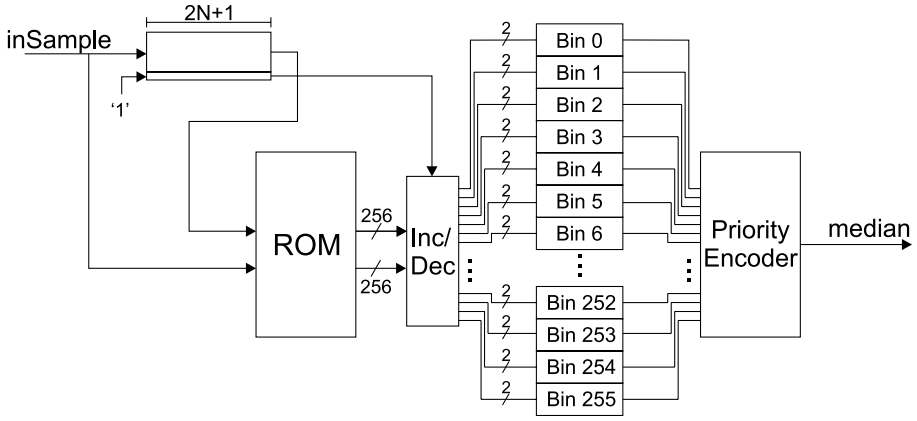

Fig. 6. Architecture of the sliding window median filter

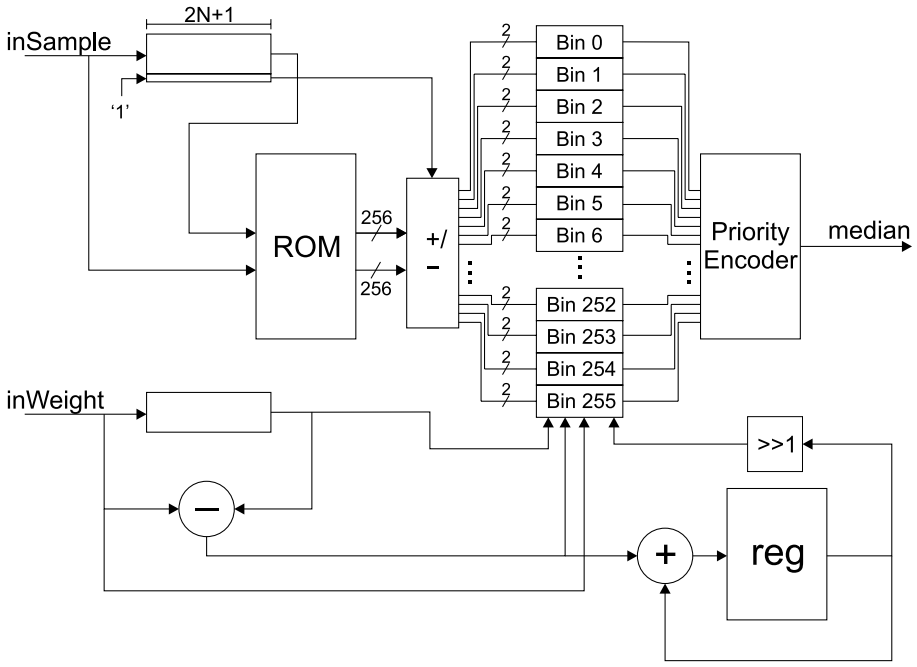

Fig. 7. Architecture of the weighted median filter

Window Size	Sorting Grid	Proposed
9	328	2774
13	708	–
17	1232	–
21	1900	3038
25	2712	–
29	3688	–
33	4768	3035
37	6012	–
41	7400	3042
45	8932	–
51	–	3040

Table 3. Synthesis results for area in Virtex II Slices.

complex function, yet the area impact using this architecture is not excessive.

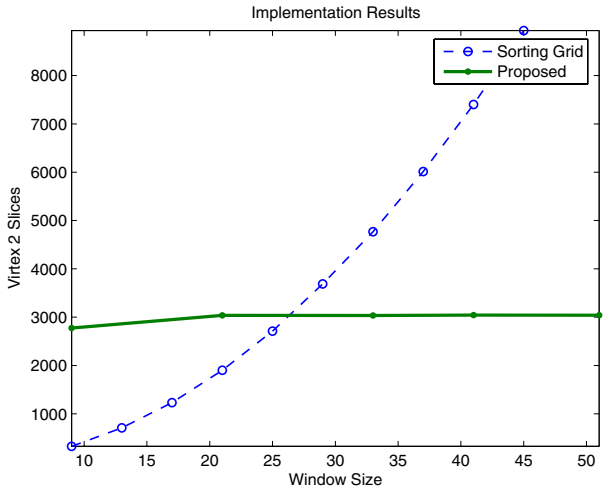

Fig. 8. Graph of synthesis results

5. CONCLUSION

We have presented an alternative implementation of median filtering for arbitrarily large windows. The architecture is immune to changes in window size, the area being determined solely by the bit width. This allows for a flexible window-size that can change from one calculation to the next. Use of heterogeneous FPGA resources allow the circuitry to be simplified. The area requirement was compared to that of another architecture, showing the efficiency of this method for larger windows. An extension to weighted median calculation was also shown, that had modest impact on area. The presented method is elegant in its flexibility with regards to window size. Of course for very small windows, like a 9-sample 3 3 window, other techniques may be faster, however for large windows, or systems where flexibility is needed, or for weighted median calculation, our method is both fast and compact.

6. ACKNOWLEDGEMENT

This work was partially supported by the Research Councils UK Basic Technology Research Programme "Reverse Engineering the Human Visual System" GR/R87642/02.

7. REFERENCES

[1] A. Kadyrov and M. Petrou, "The Trace Transform and its applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 8, pp. 811–828, 2001.

[2] M. Petrou and A. Kadyrov, "Affine invariant features from the Trace Transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence.*, vol. 26, no. 1, pp. 30–44, 2004.

[3] D. Richards, "VLSI median filters," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 38, no. 1, pp. 145–53, 1990.

[4] G. Bates and S. Nooshabadi, "FPGA implementation of a median filter," in *Proceedings of IEEE TENCON '97 IEEE Region 10 Annual Conference. Speech and Image Technologies for Computing and Telecommunications, 2-4 Dec. 1997*, vol. vol.2. Brisbane, Qld., Australia: IEEE, 1997, pp. 437–40.

[5] M. Karaman, L. Onural, and A. Atalar, "Design and implementation of a general-purpose median filter unit in CMOS VLSI," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 2, pp. 505–13, 1990.

[6] H.-S. Yu, J.-Y. Lee, and J.-D. Cho, "A fast VLSI implementation of sorting algorithm for standard median filters," in *Twelfth Annual IEEE International ASIC/SOC Conference, 15-18 Sept. 1999*. Washington, DC, USA: IEEE, 1999, pp. 387–90.

[7] C.-T. Chen, L.-G. Chen, and J.-H. Hsiao, "VLSI implementation of a selective median filter," *IEEE Transactions on Consumer Electronics*, vol. 42, no. 1, pp. 33–42, 1996.

[8] L. Breveglieri and V. Piuri, "Digital median filters," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 31, no. 3, pp. 191–206, 2002.

[9] A. Burian and J. Takala, "VLSI-efficient implementation of full adder-based median filter," in *2004 IEEE International Symposium on Circuits and Systems, 23-26 May 2004*, vol. Vol.2. Vancouver, BC, Canada: IEEE, 2004, pp. 817–20.

[10] G. Angelopoulos and I. Pitas, "A fast implementation of two-dimensional weighted median filters," in *Proceedings of 12th International Conference on Pattern Recognition, 9-13 Oct. 1994*, vol. vol.3. Jerusalem, Israel: IEEE Comput. Soc. Press, 1994, pp. 140–2.

[11] L. Hayat, M. Fleury, and A. Clark, "Two-dimensional median filter algorithm for parallel reconfigurable computers," *IEE Proc. Vision, Image and Signal Processing*, vol. 142, no. 6, pp. 345–50, 1995.