REAL-TIME MEDIAN FILTERING WITH A FAST HARDWARE SORTER

Steven B. Leeb Alfredo Ortiz James L. Kirtley, Jr.

Laboratory for Electromagnetic and Electronic Systems
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

Abstract

This work develops a hardware realization of a median filter suitable for use in real-time control and monitoring applications in power electronic circuits. Median filters have the ability to suppress impulse noise in signals while preserving underlying edges. The performance of the filter is demonstrated with results from a prototype.

I. Introduction

This paper describes a fast hardware implementation of a median filter with emphasis on possible applications to power electronics. The median filter is a discrete, nonlinear filter which can remove spike noise from a signal while accurately preserving underlying edges. It is one example of a group of filters which make use of a sorting or ranking operation to filter a waveform. This work focuses on the median filter as a relatively easily understood and implemented example which richly illustrates the potential of nonlinear filtering in power electronics.

Median and median-type filters have been popular for the past twenty years in signal and image processing applications, e.g., [1, 2]. Until recently, however, these filters have received little attention from the power electronics community [3]. In situations where the frequency spectra of the noise and the underlying signal overlap and the noise is highly impulsive, the median filter can have remarkable advantages over linear filters. Such situations are reasonably common in the implementation of power electronic circuits and machine drive systems.

Many specialized hardware architectures for implementing median-type filters have appeared in the signal processing literature [4, 5, 6]. Most proposed architectures for hardware median filters are not optimal for use in real-time control applications because their latency is too large or increases unreasonably with filter size. Such designs could introduce intolerable delay. This paper presents a filter design which is intended for use in control and monitoring loops in power electronic circuits. Note that this architecture is designed for use in filtering "signal strength" sources, not "power strength" inputs and outputs.

The next section of this report describes the median filter and some of its properties. To show the power of the median filter, a review of off-line results first presented in [3] compares output from the median and linear filters for a common input waveform: a switch voltage transient in a flyback converter. New off-line results in this section show the result of filtering a noisy motor tachometer signal with the median filter, suggesting one potential real-time application. Section III details some of the problems and trade-offs associated with several traditional median filtering architectures. Section IV describes the proposed architecture for a real-time median filter and discusses the benefits of the design. In Section V, results from an actual prototype of the design are presented. Section VI summarizes our results and anticipated future experiments.

II. Median Filter Basics

The median filter operates on a windowed section containing an odd number of evenly spaced, discrete samples from an input stream of data, X. The window slides across the input stream, advancing one point at a time. At any instant, the output of the filter, Y, is the median of the data in the window. The schematic in Fig. 1 illustrates this process for a window of $2N + 1$ points. This filter is referred to as a filter of size N. In Fig. 1, the median value of the data window is determined by a sorting operation; this is not the only method for determining the median, but an adaptation of this approach provides one of the fastest possible hardware implementations. In a real-time setting, we have found it convenient to initialize the data window with zeros, although other values are possible [3, 7].

There are several ways to understand the behavior of the median filter. Studying the frequency properties of the filter is *not* typically one of the best ways, since the

CH2992-6/91/0000-0254 $1.00 © 1991 IEEE

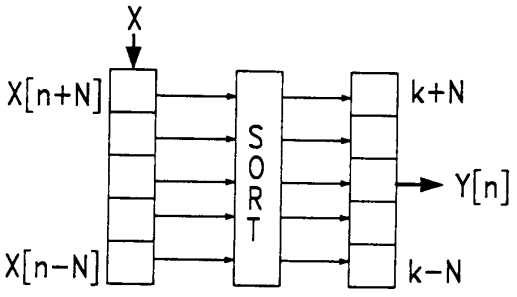

Figure 1: Schematic Diagram of a Median Filter

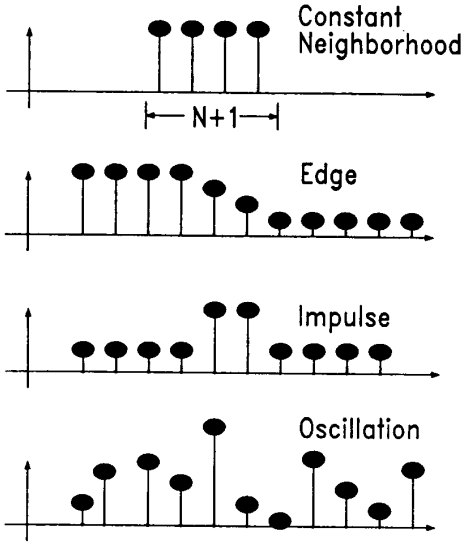

Figure 2: Signal Building Blocks

median filter is nonlinear. One successful approach is to consider how the median filter alters the local *geometry* or shape of a waveform. We will avoid extensive formalism here; the interested reader is directed to [3, 7, 8] for detailed explications of median filter behavior in geometric terms.

For a filter with a window size of $2N+1$, consider the four fundamental signal shapes shown in Fig. 2, following the development in [3, 7]. Note that any discrete waveform may be described as a sequential collection of constant neighborhoods, edges, impulses, and oscillations. We state that passing a signal once through a median filter will eliminate impulses and reduce oscillations. It has been shown that repetitive median filtering of a finite length signal will produce, after a finite number of repetitions, a *root* signal which is invariant to further applications of the filter [7]. Such a root signal consists only of edges and constant neighborhoods. Hence, if a waveform consists of an underlying signal of edges and constant neighborhoods corrupted with noise which consists of impulses and oscillations, the median filter will remove or reduce the noise without modifying the underlying signal.

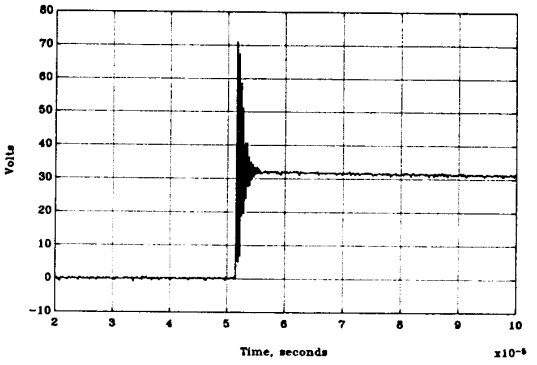

Figure 3: Switch Voltage

The median filter has proven to be an invaluable off-line tool for smoothing experimental curves for comparison to simulated or theoretical data [3]. For example, Fig. 3 shows the switch voltage in a flyback converter when the controllable switch turns off. The "spike" on the rising edge of the step is a high frequency ringing created by the MOSFET body capacitance and transformer leakage inductance. Figure 4 shows the results of median filtering this waveform with a filter of size $N = 8$. Figure 5 shows the results of filtering the waveform in Fig. 3 with three different fourth order Butterworth filters, whose cutoff frequencies span a range of values with respect to the sample frequency. The median filter is uniquely capable of removing the spike while preserving the edge.

Figure 6 shows an unfiltered plot of speed versus time during the free acceleration of a 3 hp, 4 pole induction motor. This plot was developed by sampling the voltage across a small DC tachometer connected to the motor. Commutator brush noise created the spikes in the waveform. Attempts to remove these spikes with a linear filter are complicated by the fact that the rate of occurrence of the spikes is a function of the motor speed, which is rapidly changing during the transient. Hence, it may be difficult or impossible in many situations to select a (non-adaptive) linear filter cut-off frequency which removes the spikes without distorting the underlying edge. In [9], for example, the authors were forced to rely on an

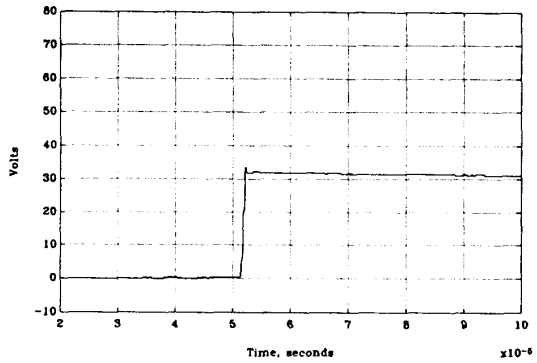

Figure 4: Median Filtered Switch Voltage

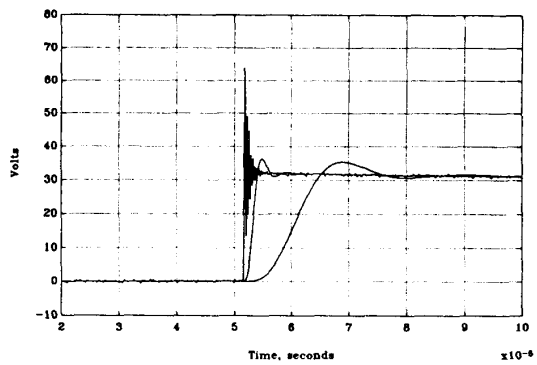

Figure 5: Low-Pass Filtered Switch Voltage

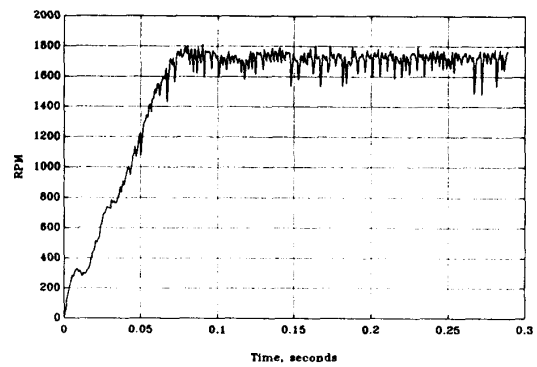

Figure 6: Induction Motor Free-Acceleration

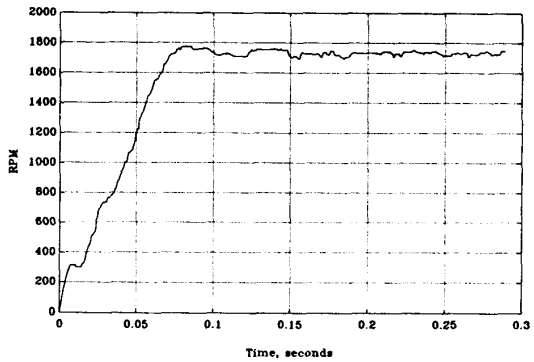

Figure 7: Median Filtered Free-Acceleration Curve

ad hoc off-line scheme for removing such spikes before further processing of the waveform could proceed. Figure 7 shows the results of filtering this waveform with a median filter of size $N = 4$.

The success of the median filter in the off-line processing of a variety of waveforms from power electronic circuits and motor drive systems has led us to explore potential filter architectures for use in real-time monitoring and control applications.

III. Design Overview

Three broad classes of digital median filter architectures are described in [10]:

- *Original order* designs, which maintain the window of samples in chronological order in a buffer. The median is produced by some "median finding" [11] or sorting hardware. See [4], for example.
- *Sorted order* designs, which maintain the window of samples in sorted order. A linked-list arrangement is used to tag each point with an "age" index to indicate its time of arrival in the window. See [5], for example.
- *Bitwise* designs, which implement a variety of sophisticated algorithms to determine the median entry in the data window based on the assumption that the number of bits representing individual samples is relatively small. See the "stack filter" proposed in [6] for one approach in this class.

An excellent and detailed summary of the different tradeoffs between speed and complexity for these approaches, including several important variations and hybrids, may be found in [11].

Minimizing delay or *latency* is an important design constraint in the development of a hardware median filter. Referring to Fig. 1 we see that, in mathematical terms, filter operation on the data window at time n for a filter of size $2N+1$ may be described as

$$Y[n] = \text{Median of } (X[n-N], ..., X[n], ..., X[n+N]),$$

where $X[n]$ represents a discrete input data sample and $Y[n]$ is the filter output, both at time n. In principle, the median filter is non-causal; that is, it requires future samples of the input data stream X to compute the output value at time n. In practice, there must be a pure delay associated with an implementation of a median filter. For a filter of size N, the delay cannot be less than N sample intervals since N "future" points are required to predict an output point. In the off-line results presented in Section II, the output waveforms have been shifted by N samples to correct for this pure delay. In a real-time setting, the delay is inescapable, and the decision to use a median filter with its nonlinear smoothing properties must be carefully balanced against other filter types, which may have frequency-dependent or pure delays. To avoid further complicating a potential control or monitoring problem, a hardware median filter of size N should, ideally, introduce no more than the inevitable N sample interval delay. Generally, we suspect that designs with excessive latency or, worse, with latencies which increase with window size or content, should be avoided.

The sorted order architecture can be constructed to provide the best overall performance (i.e., minimum latency) [11]. In short, this design implements the equivalent of a hardware insertion sort algorithm [12]. The goal of this design is to take advantage of the fact that as the window "slides" across the data point by point, only one data value is being discarded and one added at any time. The rest of the list is already in sorted order. By using a collection of parallel comparators to simultaneously compare the incoming value to each point in the sorted list, it is possible to discard the old data point and insert the new point in a fixed number of clock cycles regardless of the length of the list. In our sorted order implementation, presented in the next section, this insertion is accomplished in two clock cycles.

We discovered after the completion of our design and construction efforts that our filter is similar in many respects to the sorted order architecture reported in [5]. In our design we have replaced the bit-slice architecture in [5] with a word parallel design in which all of the W bits of each data point are written, compared, or read simultaneously. This word parallel approach divides by W the number of clock cycles needed to perform an insertion cycle compared to a "bit-serial" design [11]. In [5], the authors report only simulated performance estimates; in Section V, we present results from an actual prototype. Readers interested in further construction details are encouraged to examine [5] in addition to the next section. In particular, [5] provides guidelines for determining and comparing silicon area for VLSI implementations which are beyond our current scope. The description of our sorted order filter architecture presented in the next section is adapted in part from the thesis [13]. See [13] for more information, including detailed schematics.

IV. Hardware Implementation

A block diagram of our $2N+1$ point hardware filter window is shown in Fig. 8. The actual values from the input stream X are stored in the registers labeled data. The "age" of each sampled value, modulo $2N+1$ counting from zero, is stored in the auxiliary register labeled index associated with each data register. A window element is a combination of the data and index registers and supporting circuitry. The top element in our design contains the largest data value in the window and the bottom element contains the smallest, with all other data in rank order in the middle elements. After an initialization in which each window element is loaded with a data value of zero and an index value ranging from zero to $2N$, the steady state operation of this filter window is controlled by a finite state machine (FSM) which has two states, *shift* and *load*.

The *shift* operation prepares the window elements to receive a new input value from the analog-to-digital converter. During the *shift* state two processes occur: the value from the *data* register of the middle element is read into an output register as the median value, and each element compares the value of its *index* register with the constant $2N$. The element which matches contains the oldest point; an index of zero indicates the newest point. The result of the index age comparison in each element is or-gated with the result of the index comparison from the element above. This composite result propagates to the element below. All elements below the oldest point, determined by the propagated index age comparison, shift the values of their sampled data and index registers up one element, eliminating the oldest point by writing over it, and freeing the bottom point.

The *load* operation performs an insertion sort to place a fresh input value from the analog-to-digital converter into its correct position in the window. The fresh input value is broadcast to all of the window elements. Each window element compares the input value with the value in its *data* register. At some "break point" in the window, all of the elements below the break point will be less than the input value and all of the elements above the break point will be greater than the input value. The

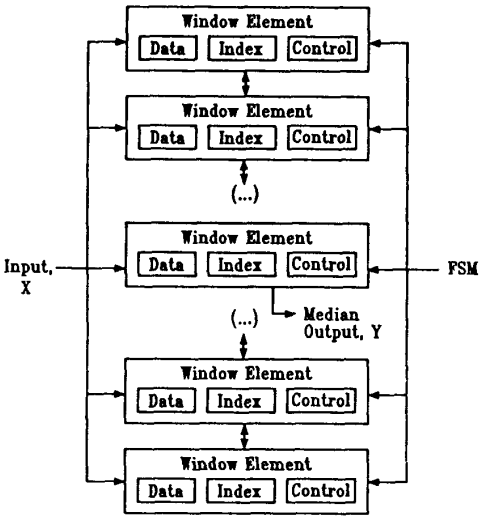

Figure 8: Median Filter Block Diagram

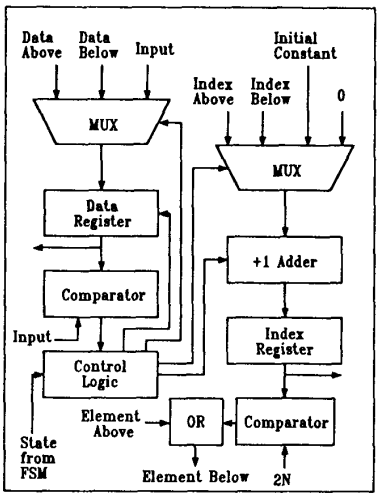

Figure 9: Typical window element

break point is unique in that the element immediately below it compares low to the input value and the element above it compares high; this condition is easily tested in parallel for all pairs of window elements. All elements below the break point shift down, opening an element in which the new data point from the input bus is inserted. The *index* register for this new point is set to zero. All other index registers are incremented by one. When the *load* operation is complete, the machine returns to the *shift* state and the process begins again. A new output point, the median data value in the current window, is made available after every two finite state machine cycles (i.e., clock cycles) regardless of the size of the window or the type of waveform being filtered.

Figure 9 shows a schematic of a typical window element which operates under the two state scheme described above. Slight modifications must be made for the top and bottom elements since these elements do not have elements above or below, respectively. An action table summarizing the behavior of the top, bottom, and middle elements during the *shift* and *load* states is shown in Table 1. The *propagated* signal, checked during the *shift* state for each middle element refers to the value of the propagation chain that indicates the presence of an index match with the constant $2N$ above an element.

Only the top and bottom elements, therefore, require special control coding. A filter of arbitrary size may be constructed by stacking sufficient "generic" middle elements between a top and bottom pair. For simplicity, analog signals are level shifted prior to analog-to-digital conversion in our prototype, so the lowest-valued digital sample stored in the window is zero. Hence, our prototype does not handle negative sample values digitally; all register values are positive binary numbers. The median output of the filter can be inverse shifted to restore the original bias.

Element	State	
	Shift	*Load*

Top	If (index = 2N+1)
	data = data below
	index = index below
		If (data <= input)
		data = input
		index = 0
		else
		index += 1

Middle	If (index = 2N+1 or *propagated*=TRUE)
	data = data below
	index = index below
		If (data <= input and data above>input)
		data = input
		index = 0
		else if (data>input and data above>input)
		index += 1
		else
		data=data above
		index = index above+1

Bottom	If (index = 2N+1)
	do nothing
		If (data above > input)
		data = input
		index = 0
		else
		data=data above
		index = index above+1

Table 1: Element Action Table

In the next section we present results from an actual prototype of our filter design.

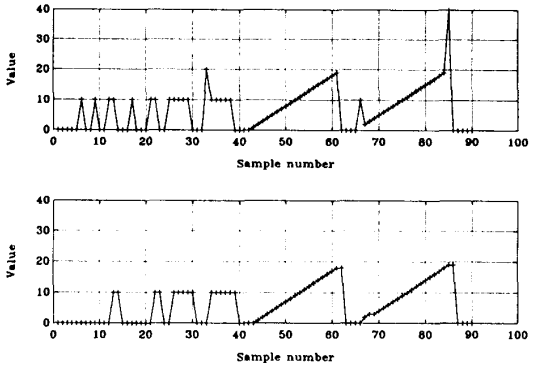

Figure 10: Test Input and Output

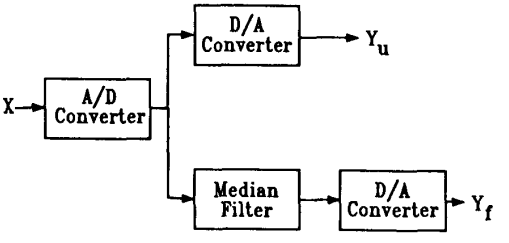

Figure 11: Median Filter Test Configuration

V. Experimental Results

To study the validity and potential performance of our proposed design, we constructed a hardware prototype of a median filter of size one (a window size of three points). First, to test the hardware sorter, the prototype was used to filter digital data stored in an EPROM. A section of the digitized input waveform is shown in the top trace in Fig. 10. The precise sample locations are indicated with + symbols. The edges and constant neighborhoods in the signal are corrupted with spike noise. The bottom trace in Fig. 10, the output of the filter, shows that the noise is removed by the prototype filter with its three point window.

Next, the filter was used in the test configuration shown in Fig. 11. The signal Y_u was used to monitor the waveform of digitized samples fed as input to the median filter. The signal Y_f was a real-time, median filtered version of the digitized input waveform, Y_u.

Figure 12 shows the results of presenting a $35kHz$ triangle wave as the input to the filter test hardware. The top trace in Fig. 12 shows Y_u. For this $N = 1$ filter, the crest and trough samples of this triangle wave are part of signal oscillations where the wave is changing directions. We expect, therefore, that these oscillations will be reduced in the filtered output. The bottom trace in Fig. 12 shows Y_f, the output of the median filter. As expected, the crest and trough samples have been eliminated. The filter sampling rate for this experiment was $575kHz$.

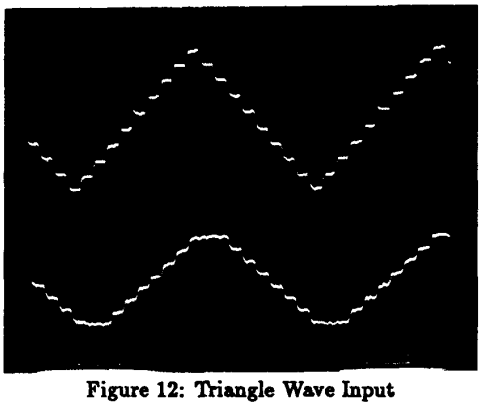

Figure 12: Triangle Wave Input

We have had similar successes operating our hardware filter at sample rates as high as $1.23MHz$, near the limit of our digital-to-analog conversion hardware. Even at this reasonably fast sample rate, we saw no signs in our breadboarded prototype of performance degradation which would prevent the filter from running at even higher sample rates. A robust, tightly-packed hardware layout on a printed circuit board or custom integrated circuit would presumably perform well at much higher frequencies.

VI. Conclusions

This paper has presented a hardware median filter design which meets the stringent requirements necessary for control and monitoring applications. The results from the hardware prototype substantiate the expected performance of the filter. Several important problems remain in the development of specific applications.

A point mentioned previously that cannot be overemphasized is that the median filter is nonlinear. Standard linear analysis techniques for the design of feedback control systems, for example, cannot be directly applied if a median filter is incorporated in the closed loop. Developing satisfactory techniques for determining stability and other properties of systems which contain nonlinear building blocks like the median filter is an open and active area of research.

Of course, any field use of the median filter would necessitate a more rugged construction technique than the solderless breadboarding used here. The highly modular structure of the filter naturally points towards a custom VLSI implementation, an approach taken by many re-

searchers studying nonlinear filtering in the signal processing fields. We hope to report on our experiences with a custom integrated circuit application at a later date. The commercial success of fast floating point units for modern digital signal processing applications is well known. We suspect that, in the long run, the power of nonlinear filtering may generate a demand for the development of commercially available chip packages for nonlinear operations such as sorting.

Acknowledgements

The authors gratefully acknowledge the support of the Electric Power Research Institute. SBL thanks Professor George C. Verghese, W. Clem Karl, Kwaku O. Prakah-Asante, and Lawrence A. Jones for valuable assistance and discussions.

References

[1] G.R. Arce and N.C. Gallagher, Jr.,"BTC Image Coding Using Median Filter Roots," *IEEE Transactions on Communications*, June 1983, pp. 784–793.

[2] P.A. Maragos and R.W. Schafer, "Morphological Skeleton Representation and Coding of Binary Images," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, October 1986, pp. 1228–1244.

[3] W.C. Karl, S.B. Leeb, L.A. Jones, J.L. Kirtley, and G.C. Verghese, "Applications of a Class of Nonlinear Filters to Problems in Power Electronics," *IEEE Power Electronics Specialists Conference*, June 1990, pp. 35–42.

[4] K. Oflazer, "Design and Implementation of a Single-Chip 1–D Median Filter," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, October 1983, pp. 1164–1168.

[5] G.R. Arce and P.J. Warter, "A Median Filter Architecture Suitable for VLSI Implementation," *23rd. Allerton Conf. on Commun., Cont., and Comput.*, October 1984, pp. 172–181.

[6] E.J. Coyle, "The Theory and VLSI Implementation of Stack Filters," *VLSI Signal Processing*, IEEE Press, New York, NY, 1986.

[7] G.R. Arce, N.C. Gallagher, and T.A. Nodes, "Median Filters: Theory for One– and Two–dimensional filters," *Advances in Computer Vision and Image Processing*, JAI Press, Greenwich, CN, 1986.

[8] P. Maragos and R.W. Schafer, "Morphological Filters – Part II: Their Relations to Median, Order-Statistic, and Stack Filters," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, August 1987, pp. 1170–1184.

[9] B. Szabados, J.H. Dableh, G.M. Obermeyer, R.E. Draper, and R.D. Findlay, "Measurement of the Torque-Speed Characteristics of Induction Motors Using an Improved New Digital Approach," *IEEE Transaction on Energy Conversion*, September 1990, pp. 565–571.

[10] G.R. Arce, P.J. Warter, R.E. Foster, "Theory and VLSI Implementation of Multilevel Median Filters," *International Symposium on Circuits and Systems*, June 1988.

[11] D.S. Richards, "VLSI Median Filters," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, January 1990, pp. 145–153.

[12] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes in C*, Cambridge University Press, Cambridge, 1988.

[13] A. Ortiz, "Flash Filter: Median Filtering by Insertion Sort," S.B. Thesis, MIT Department of Electrical Engineering and Computer Science, May 1990.