

# *Pipelined median architecture*

Article

Accepted Version

Cadenas Medina, J. (2015) Pipelined median architecture. *Electronics Letters*, 51 (24). pp. 1999-2001. ISSN 0013-5194  
doi: <https://doi.org/10.1049/el.2015.1898> Available at  
<https://centaur.reading.ac.uk/39965/>

It is advisable to refer to the publisher's version if you intend to cite from the work. See [Guidance on citing](#).

To link to this article DOI: <http://dx.doi.org/10.1049/el.2015.1898>

Publisher: Institution of Engineering and Technology (IET)

All outputs in CentAUR are protected by Intellectual Property Rights law, including copyright law. Copyright and IPR is retained by the creators or other copyright holders. Terms and conditions for use of this material are defined in the [End User Agreement](#).

[www.reading.ac.uk/centaur](http://www.reading.ac.uk/centaur)

## **CentAUR**

Central Archive at the University of Reading

Reading's research outputs online

# Pipelined median architecture

J. Cadenas

The core processing step of the noise reduction median filter technique is to find the median within a window of integers. A four-step procedure method to compute the running median of the last  $N$   $W$ -bit stream of integers showing area and time benefits is proposed. The method slices integers into groups of  $B$ -bit using a pipeline of  $W/B$  blocks. From the method, an architecture is developed giving a designer the flexibility to exchange area gains for faster frequency of operation, or vice versa, by adjusting  $N$ ,  $W$  and  $B$  parameter values. Gains in area of around 40%, or in frequency of operation of around 20%, are clearly observed by FPGA circuit implementations compared to latest methods in the literature.

*Introduction:* The median filter is a well-established technique for noise reduction in image processing and yet the concept of the median finds new applications in image forensics [1], electrocardiography [2] and in fast processing of real-time systems [3]. For  $N = 2k + 1$  sorted integers the median is the integer at the middle position. Hardware architectures for computing the median are broadly classified in sorting-based methods [4] and non-sorting-based methods [5][6]. As sort is theoretically bound by  $O(N \log N)$  time, non-sorting methods have emerged driven by the idea of completing the median in  $O(W)$  time, where  $W$  is the bit length of the integers of the unsorted set from where the median is sought. Typically, in hardware scenarios,  $W$  is restricted to high resolution analogue-to-digital converters with  $W \leq 24$ , or to bytes for image pixels, thus even for modest values of  $N$ , non-sorting median calculation methods gain an advantage.

This Letter develops a non-sorting method to calculate the median as a four step procedure; this follows from a reformulation of parallelism at the bit level of a previous method [6]. Each step is easily implemented for fast computation with the overall result that the median is computed three times faster than before, as confirmed here by a timing analysis. The main features of the previous method are preserved; it computes the median on a set of  $N$   $W$ -bit integers by  $W/B$  processing blocks, where  $B$  is a parameter of how many bits of the integers are sliced for processing. Each block contributes  $B$ -bit towards finding the median in a pipeline stage. Two key ideas are put forward. The first is a parallel addition at the bit level within each block of computation, whereas previously, this addition was computed serially. The second is a parallel decision and selection to carry forward the computation to subsequent blocks. These key ideas are facilitated by encoding slices of bits using a binary-to-thermometer code, referred to as  $B$ -to- $T$  encoding.

*B-to-T encoding:* A code of  $r-1$  ones followed by a zero is referred to as a thermometer code; this is common in fast ADCs [7]. For instance, the binary code for decimal value  $1_{10} = 01_2$  can be written in thermometer code either as  $0001_2$  or as  $1110_2$ ; this Letter uses the latter. For a binary pattern of  $B$ -bit we will express the thermometer code as an output string of  $r = 2^B$  bits; the bit vector is denoted as  $q_i$  for  $i = 0, \dots, 2^B - 1$ . In short, for an input  $i_{10}$  the  $B$ -to- $T$  encoder sets bits in vector  $q$  with indices  $i, \dots, 2^B - 1$ .

*Small example:* Consider a data set of  $N = 9$  integers,  $x_j = \{6, 0, 12, 13, 10, 3, 15, 5, 9\}$ , each of  $W = 4$  bits (labelled as  $[3:0]$ ). If  $x_j$  is sorted the median is at position  $P = 5$ ; integer 9 for this set. Partitioning each  $x_j$  with  $B = 2$  bits forms  $W/B = 2$  blocks. The two MSBs of  $x$  ( $x_j[3:2]$ ) are processed first in Block 2. Block 1 processes the two LSBs ( $x_j[1:0]$ ) as shown in Table 1. Integers are processed sequentially, and a  $B$ -to- $T$  encoding on the integer slice is performed on the fly on a 4-bit vector ( $2^B$ ) as previously stated. Each one of the bits of this encoding vector is added vertically, also on the fly, as  $A_i$ ,  $i = 0, \dots, 3$ , starting from a count of 0. Note this addition is parallel. For instance, bit slice “ $01_2$ ” for integer 6 is encoded as “1110” (and added to an initial “0000”); then bit slice “ $00_2$ ” for integer 0 is encoded as “1111” and added to the running count of “1110” gives a count of “2221” (second row under Block 2 in Table 1). After all nine integers are processed by Block 2  $A_3, A_2, A_1, A_0$  have counts 9, 6, 4, 2 respectively. A block finds  $B$ -bit of the median as the first occurrence of the index  $i$  where  $A_i \geq P$ , (for Block 2 this occurs at  $i = 2$  under column  $A_2$ ).

**Table 1:** Window with integers  $x_j = \{6, 0, 12, 13, 10, 3, 15, 5, 9\}$  and on-the-fly addition of  $B$ -to- $T$  encoding on 2-bit slices.

$(x_j)_{10}$	$(x_j)_2$	Block 2	$A_3$	$A_2$	$A_1$	$A_0$	Block 1	$A_3$	$A_2$	$A_1$	$A_0$
6	0110	01	1	1	1	0	10	4	4	4	4
0	0000	00	2	2	2	1	00	4	4	4	4
12	1100	11	3	2	2	1	00	4	4	4	4
13	1101	11	4	2	2	1	01	4	4	4	4
10	1010	10✓	5	3	2	1	10	5	5	4	4
3	0011	00	6	4	3	2	11	5	5	4	4
15	1111	11	7	4	3	2	11	5	5	4	4
5	0101	01	8	5	4	2	01	5	5	4	4
9	1001	10✓	9	6	4	2	01✓	6	6	5	4
		$A_i \geq 5$	1	1	0	0	$A_i \geq 5$	1	1	1	0

The two MSBs of the median, are then found as  $M[3:2] = "10_2"$ ; integers 10 and 9 remain median candidates (ticks for Block 2). Next, Block 1 is processed. First, we copy the final sum value to the right of  $A_2$  (this is  $A_1$  with value 4, underlined in Block 2) as the initial value for the addition in Block 1. Secondly, the slices for integers 6, 0, 12, 13, 3, 15, and 5 (slices in grey in Block 1) get nullified so they cannot update any  $A_i$  for Block 1. Computing  $A_i$  is as before, on the remaining integer 2-bit slices. The condition  $A \geq P$  is now first satisfied under  $A_1$  ( $i = 1$ ). The two LSBs bits of the median are thus  $M[1:0] = "01_2"$ . Concatenating the results from blocks 2 and 1 give the median as  $M = "1001_2" = 9_{10}$  (tick in Block 1).

*Median calculation method:* From Table 1 a method to calculate the median is presented as a four-step procedure;

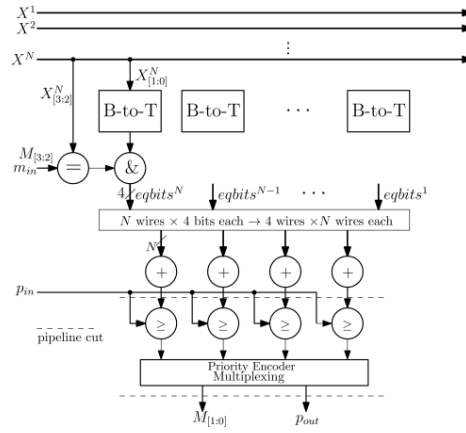
1. Binary-to-thermometer slice encoding
2. Parallel addition of encoding bits
3. Selection of median slice and setting of sum initial values
4. Nullification of non-median integers

Step 1 is a code conversion; for slices of  $B = 2, 3$  and 4 bits, fast hardware implementations are easily achieved using look-up tables. Step 2 computes  $A_i$  in parallel, and as selecting the median slice value for a block are made on sums  $A_i$ , the method must be faster than previous method [6]. Step 3 requires  $2^B$  parallel comparisons; the selection of the median is also fast for  $B = 2, 3$  and 4 through the use of look-up tables or logic. Note median position  $P$  remains a constant for all blocks; setting adders' initial values for subsequent blocks requires a selection operation and is made fast with a suitable multiplexer. Setting the initial values to adders is implemented through a simple truth table. Step 4, the logic for the nullification of integers, that cannot be the median, is also achieved with simple logic blocks.

*Parallel operation in a sliding window:* Maintaining a parallel update on sums  $A_i$  is required for pipelined operation on a streaming sliding window. Observe  $B$ -to- $T$  (step 1) is paramount here to allow for additions in step 2 to proceed in parallel and to keep them coherent. Fig. 1 shows a block diagram to compute the median slice  $M[1:0]$  of  $N$  integers,  $B = 2$ . This requires an array  $r = 2^2$  adders working in parallel on  $B$ -to- $T$  encoded bits. Each sum is of  $\log_2 N$  bits wide to hold a value of up to  $N$ . The block of computation has two outputs; the median slice for the block,  $M[1:0]$ , and the initial set value for the sum of the next block ( $p_{out}$  in the figure). These two outputs are shown at the bottom of the figure, all based on an array of parallel comparators. Notice each  $B$ -to- $T$  encoding is inhibited by a single enable bit (circle with '&') to account for nullification of integers.

*Sliding window median architecture:* In general, for  $W$ -bit integers, physical blocks of  $B$ -bits each account for  $W/B$  processing blocks. Input elements  $x_j$  are pipelined with  $N$  stages to arrange for  $x^1, x^2, \dots, x^N$  as

shown in Fig. 1. An array of  $2^B$  adders, each of  $\log_2 N$  bits, is maintained per block. The current integer gets nullified by comparing, for equality, the median bits found thus far, by previous blocks, with the corresponding slice bits of  $x_j$ . Note the required equality comparison to nullify integers is of only  $B$ -bit for all blocks in a complete architecture and can be omitted for the first block since all integer slices must be processed by the first block.



**Fig. 1** A median slice block of computation, encompassing the four steps, to calculate the median on a sliding window of  $N$  integers

All previous outputs from a block, namely  $M$  and  $P$ , are pipelined from one block to the next; after  $W/B$  processing blocks, the median  $M$  is found, with each block contributing  $B$ -bit to  $M$ . The median for each window emerges every clock cycle. Given the median computation requires  $W/B$  blocks with two pipeline registers each, the latency for the architecture in Fig. 1 is of  $N+2(W/B)-1$  clock cycles.

*Timing analysis:* Fig. 1 layout shows pipeline cuts that can be conveniently made anywhere to modify the critical path delay  $T$ . Therefore, a pipeline cut is made such that the critical path of Fig. 1 is essentially due to the parallel comparator array plus the slower of a priority encoder or multiplexer. As, at most, a two-level logic is required for a multiplexing operation, then  $T = \log_2 N + 2$ . Previous work had a critical path of  $T_{[6]} = 3\log_2 N + 6$  for  $B = 2$  [6], so this proposal makes a processing step, at least, three times faster regardless of any  $B$  value. For comparison, the median architecture in [5],  $T_{[5]}$  is basically the delay cost of the Carry-Save Adder tree (CSA) and at least of  $\log_{1.5}(N/2) + \log_2 N$  to account for the final adder [8]. For convenience, Fig. 1 can also be cut such that the critical path remains essentially the delay cost of the adder (to sum  $N$  bits); this is also the delay cost of a Carry-Save Adder tree in practical terms. Thus, in this work the claim is that the presented architecture can be conveniently made as fast as previous methods for all practical values of  $N$ . Note this is independent of parameter  $B$ ; however it seems convenient to maintain  $B$  as small as 2, 3 or 4 bits, in order to keep the on-the-fly encoding as a small look-up table. The sorting-based method in [4], presented as hardware area-efficient, has a critical path of  $N-1$  logic levels and therefore is slower than the time complexity of this work.

*Circuit results in FPGA technology:* The results for a median architecture in [5], referred to as LCBP, and the one presented here for the same FPGA device is shown in Table 2. These are for the case of  $N = 9$  and 25 integers each of  $W = 8$  bits. The proposed processing block has two pipeline registers delay and so latency is of 7 clock cycles ( $2(8/2) - 1 = 7$ ). Note that this latency (latency per block \* number of blocks - 1, caused by waiting for a window of  $N$  integers to fill) is common to both the LCBP and the proposal here. Table 2 shows that for  $N = 9$  and  $B = 2$ , the proposed method here runs roughly at the same frequency as LCBP architecture but with much less FPGA resources; this saving can be as high as 40%. For  $N = 9$ , and  $B = 3$ , the advantage of the reduced latency (5 clock cycles vs 8 for LCBP) can be exchanged for a faster frequency (from 286 MHz to 335 MHz) while settling for the same latency as LCBP. This is so, since each

of three processing cells ( $W/B = 8/3$ ) is cut-pipelined with three internal registers, so latency is of  $3*3-1 = 8$  clock cycles. Area saving is also obvious from the table for the case  $B = 3$ . Once more, the smaller latency can be traded off (from 7 to 11 clock cycles,  $3*4-1 = 11$ ) for a frequency increase (from 259 MHz to 330 MHz); this is a gain of over 20%. The proposal gives a designer the flexibility to easily trade off an increase in frequency for a small penalty in latency to any specific architecture of  $N$  integers by choosing a suitable parameter value for  $B$ .

*Conclusion:* The four-step median calculation method proposed here makes either faster or as fast computations, than previous hardware algorithms in the literature. The median on  $N$  integers completes after  $W/B$  processing blocks for a serial stream of  $W$ -bit integers when slicing the integers by  $B$  bits. As processing more bits per block may result in shorter latency, this can be traded off for faster operation. Smaller area is also observed for some  $N$  design parameters. One further improvement to this design requires finding a way to fuse the adder and the comparison of Fig. 1 into a single optimised block to make it smaller and faster. The four-step median method given here is also easily implemented as a fast programming solution using arrays or trees.

**Table 2:** FPGA resources and frequency of operation for the median architecture LCBP in [5] and the one presented here.

	$N = 9$			$N = 25$	
	LCBP	Here		LCBP	Here
		$B = 2$	$B = 3$		$B = 2$
CLB (Slices)	459	254	284	668	652
DFF	516	507	478	964	1303
LUT	632	336	567	766	975
$f_{\max}$ (MHz)	327	332	286/335	318	259/330
Latency, clocks	8	7	5/8	8	7/11

J. Cadenas (University of Reading, Reading, RG6 6AX, UK)

E-mail: o.cadenas@reading.ac.uk

## References

- 1 Kang X., Stamm M.C., Peng A. and Liu K.J.R.: ‘Robust median filtering forensics using an autoregressive model’, *IEEE Trans. Inf. Forensics and Security*, 2013, **8**, 9, pp. 1456-1468.
- 2 Niederhauser T., Wyss-Balmer T., Haeberlin A., Marisa T., Wildhaber R.A, Goette J. and Vogel R.: ‘Baseline wander filtering algorithms for long term electrocardiography’, *IEEE Trans. on Bio. Eng.*, 2015, **62**, (6), pp. 1576-1584
- 3 Atia M.M, Georgy J., Korenberg M.J., and Noureldin, A: ‘Real-time implementation of mixture particle filter for 3D RISS/GPS integrated navigation solution’, *Electron. Lett.*, 2010, **46**, (15), pp. 1083-1084
- 4 Chen, R. D., Chen, P. Y., and Yeh, C. H.: ‘Design of an area-efficient one-dimensional median filter’, *IEEE Trans Circ. and Syst. II*, 2013, **60**, (10), pp. 662-666
- 5 Prokin, D., and Prokin, M.: ‘Low hardware complexity pipelined rank filter’, *IEEE Trans. Circ. and Syst. II*, 2010, **57**, (6), pp. 446-450

6 Cadenas, J., Megson, G. M., Sherratt, R. S. and Huerta, P.: 'Fast median calculation method', *Electron. Lett.*, 2012, **48**, (10), pp. 558-560

7 Hieu B. V., Beak S. Choi S, Seon J. and Jeong T. T.: 'Thermometer-to-binary encoder with bubble error correction (BEC) for flash analog-to-digital converters (FADC)', 3<sup>rd</sup> Int. Conf. on Communications and Electronics, 2010, pp. 102-106.

8 Parhami, B.: 'Computer arithmetic, algorithms and hardware designs' (Oxford, 2000)

---