V. Satyanarayana, S. Srividya, U. Yedukondalu / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 3, Issue 4, Jul-Aug 2013, pp.1946-1949

High Throughput Two- Dimensional Median Filters On FPGA for Image Processing Applications

V. Satyanarayana[1], S. Srividya[2], U. Yedukondalu[3]

[1]Senior Assistant Professor, (Department of Electronics and Communication Engineering, Aditya Engineering College, Surampalem, Andhra Pradesh

[2]Assistant Professor, (Department of Electronics and Communication Engineering, Aditya Engineering College, Surampalem, Andhra Pradesh

[3]Associate Professor, (Department of Electronics and Communication Engineering, Aditya Engineering College, Surampalem, Andhra Pradesh

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 realizable 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 Median filter is an effective method for the removal of impulse-based noise from the images. This paper suggests an optimized architecture for filter implementation on FPGA. A 3x3 sliding window algorithm is used as the base for filter operation. Partial implementation is done via soft core processor. The designs are synthesized for a Xilinx Spartan-3 EDK.

Keywords**—***FPGA, Median Filter, Soft Processor Core, Parallelism, Pipelining*

I. Introduction

Digital image processing is an ever expanding and dynamic area with applications reaching out into our everyday life such as medicine, space exploration, surveillance, authentication, automated industry inspection and many more areas. In most of the cases captured images from image sensors are affected by noise. The impulse noise is the most frequently referred type of noise. This noise, commonly also known as salt & pepper noise, is caused by malfunctioning pixels in camera sensors, faulty memory locations in hardware, or errors in the data transmission.

Median filtering is considered a popular method to remove impulse noise from images. This non-linear technique is a good alternative to linear filtering as it can effectively suppress impulse noise while preserving edge information. The median filter operates for each pixel of the image and assures it fits with the pixels around it. It filters out samples that are not representative of their surroundings; in other words the impulses. Therefore, it is very useful in filtering out missing or damaged pixels of the image.

The complexity in implementation of median filter is due to the large amount of data involved in representing image information in digital format. General purpose processor as an implementation option is easier to implement on but not time-efficient due to additional constraints on memory, I/O bandwidth and other peripheral devices. Full custom hardware designs like Application Specific Integrated Circuits (ASICs) provide the highest speed to application but at the same time they have very less scope for flexibility.

Digital Signal Processors (DSPs) and Field Programmable Gate arrays (FPGAs) are two choices under the category of semi custom hardware devices. These devices give a balanced solution for performance, flexibility and design complexity. DSPs are best suited to computationally intensive applications. FPGA has been chosen for our application because of its various properties. FPGAs are reconfigurable devices, which enables rapid prototyping, simplifies debugging and verification. Its parallel processing characteristic increases the speed of implementation.

In section 2 details of median filter algorithm are presented. In section 3 FPGA implementation of the algorithm is discussed. In section 5 we provide the simulation results. The discussion of the implementation is concluded in the last section

II. Filter Algorithm

An image is stored in the form of 2D matrix of pixel values. We have referred sliding window algorithm described by R. Maheshwari and S.P.Rao [2]. The median is defined as the middle of a group of numbers when the numbers are sorted. The group should contain odd number of elements. For the 2D image, a standard median operation is implemented by sliding a window of odd size (e.g. 3x3 windows) over an image. A 3x3 window size is chosen which is considered effective for most commonly used image sizes. At each position of the window, the nine pixels values inside that window are copied and sorted. The value of the central pixel of the window is replaced

with the median value of the nine pixels in the window. When applied on Grayscale images, pixels are ranked for their brightness. When applied on Color scale images, the pixel whose red, green, and blue components have the smallest sum of squared differences from the color coordinates of its neighbors is then chosen to replace the central pixel of the neighborhood. The window shifts right by one column after every clock cycle. Window-I is read in three clock cycles. To read window-II, only the fourth column has to be accessed and column two and three can be retained from the previous window.

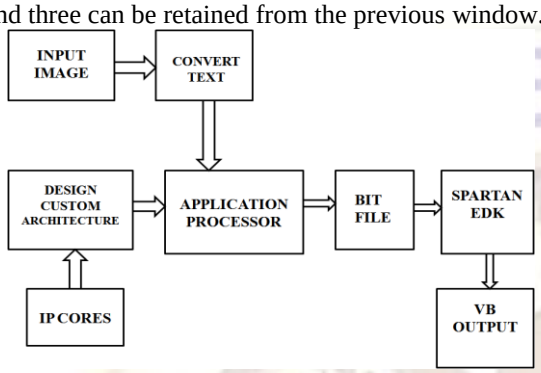

Fig.1 block diagram

III. Architecture

To build an embedded system on Xilinx FPGAs, the embedded development kit (EDK) is used to complete the reconfigurable design. Figure 1 shows the design flow.

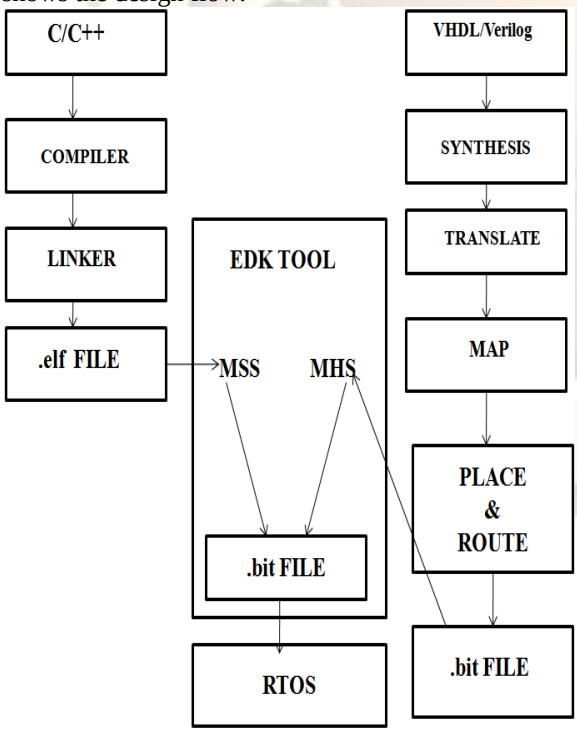

Fig.2 design flow of EDK

Unlike the design flow in the traditional software design using C/C++ language or hardware design using hardware description languages, the EDK enables the integration of both hardware and software components of an embedded system. For the hardware side, the design entry from VHDL/Verilog is first synthesized into a gate-level netlist, and then translated into the primitives, mapped on the specific device resources such as Look-up tables, flip-flops, and block memories. The location and interconnections of these device resources are then placed and routed to meet with the timing Constraints. A downloadable .bit file is created for the whole hardware platform. The software side follows the standard embedded software flow to compile the source codes into an executable and linkable file (ELF) format. Meanwhile, a microprocessor software specification (MSS) file and a microprocessor hardware specification (MHS) file are used to define software structure and hardware connection of the system. The EDK uses these files to control the design flow and eventually merge the system into a single downloadable file. The whole design runs on a real-time operating system (RTOS).

IV. Filtering Co−Processor

There are different ways to include processors inside Xilinx FPGA for System-on-a-Chip (SoC): PowerPC hard processor core, or Xilinx MicroBlaze soft processor core, or user-defined soft processor core in VHDL/Verilog. In this work, The 32-bit MicroBlaze processor is chosen because of the flexibility. The user can tailor the processor with or without advance features, based on the budget of hardware. The advance features include memory management unit, floating processing unit, hardware multiplier, hardware divider, instruction and data cache links etc. The architecture overview of the system is shown in Figure 2. It can be seen that there are two different buses (i.e., processor local bus (PLB) and fast simplex link (FSLbus) used in the system [5-6]. PLB follows IBM Core connect bus architecture, which supports high bandwidth master and slave devices, provides up to 128- bit data bus, up to 64-bit address bus and centralized busArbitration. It is a type of shared bus. Besides the access overhead, PLB potentially has the risk of hardware/software incoherent due to bus arbitration. On the other hand, FSL supports point-to-point unidirectional communication. A pair of FSL buses (from processor to peripheral and from peripheral to processor) can form a dedicated high speed bus without arbitration mechanism. Xilinx provides C and assembly language support for easy access. Therefore, most of peripherals are connected to the processor through PLB; the DWT coprocessor is connected through FSL instead.

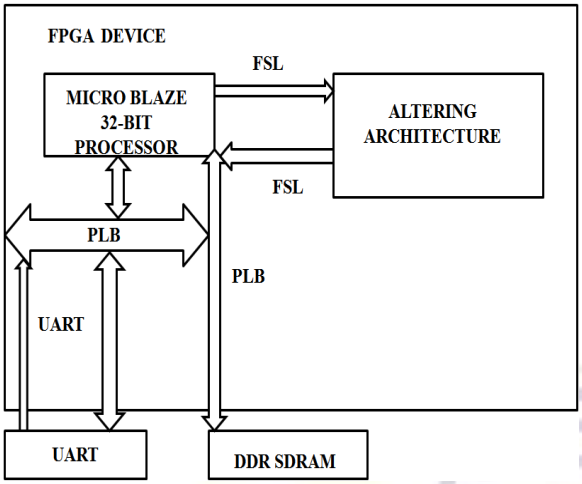

Fig.3 system overview

The current system offers several methods for distributing the data. These methods are a UART, and VGA, and Ethernet controllers. The UART is used for providing an interface to a host computer, allowing user interaction with the system and facilitating data transfer. The VGA core produces a standalone real-time display. The Ethernet connection allows a convenient way to export the data for use and analysis on other systems. In our work, to validate the DWT coprocessor, an image data stream is formed using VISUAL BASIC, then transmitted from the host computer to FPGA board through UART port.

V. Experimental Results

Experiments are performed on gray level images to verify the proposed method. These images are represented by 8 bits/pixel and size is 128 x 128. Image used for experiments are shown in below figure.

Fig.4 input image

The measurands used for proposed method are as follows:

The entropy (E) is defined as Where s is the set of processed coefficients and p (e) is the probability of processed coefficients. By using entropy, number of bits required for compressed image is calculated. An often used global objective quality measure is the mean square error (MSE) defined as

Where, nxm is the number of total pixels. f (i,j) and f(i,j)' are the pixel values in the original and reconstructed image. The peak to peak signal to noise ratio (PSNR in dB) [11-13] is calculated as

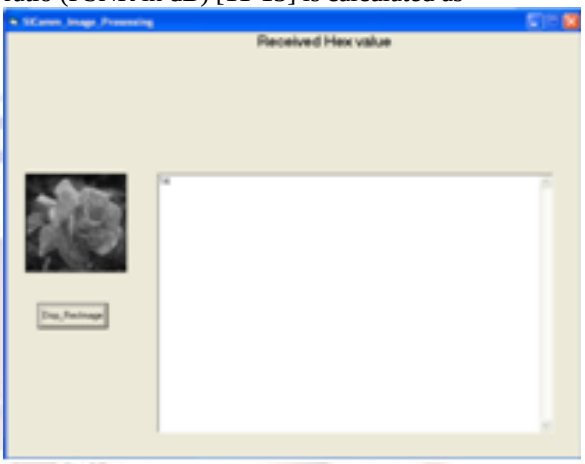

Fig.5 output image

Other Example:

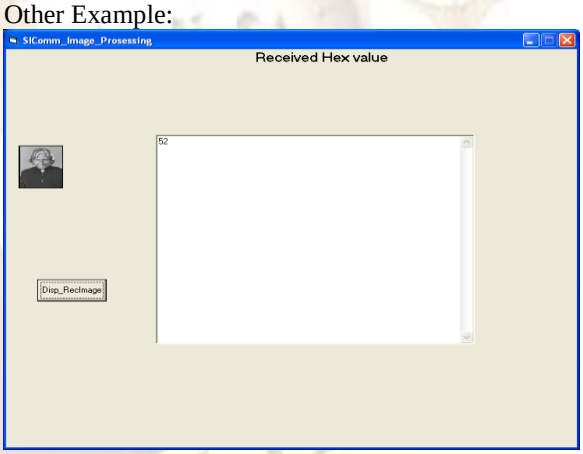

Fig.6 input image

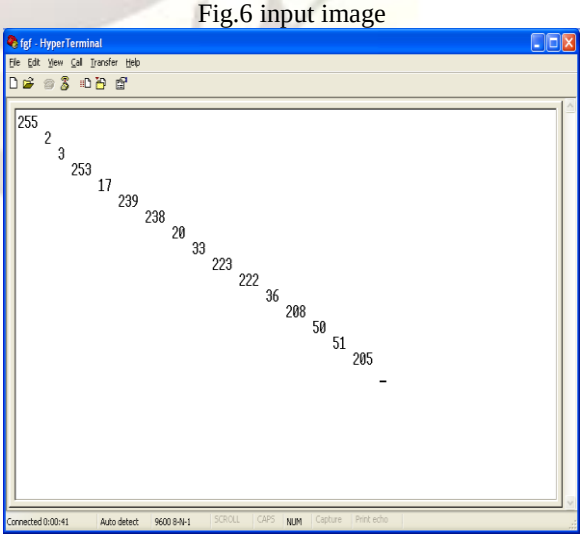

Fig.7: output image

And the synthesis report is below

```
Selected Device : 3s500efg320-4

 Number of Slices:                  2649  out of   4656    56%
 Number of Slice Flip Flops:        3343  out of   9312    35%
 Number of 4 input LUTs:            3794  out of   9312    40%
    Number used as logic:           3118
    Number used as Shift registers:  356
    Number used as RAMs:             320
 Number of IOs:                       83
 Number of bonded IOBs:               40  out of    232    17%
    IOB Flip Flops:                   55
 Number of BRAMs:                      7  out of     20    35%
 Number of MULT18X18SIOs:              3  out of     20    15%
 Number of GCLKs:                      7  out of     24    29%
 Number of DCMs:                       2  out of      4    50%

Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 12.384ns (Maximum Frequency: 80.749MHz)
   Minimum input arrival time before clock: 41.553ns
   Maximum output required time after clock: 13.840ns
   Maximum combinational path delay: 3.344ns
```

Fig.8 synthesis report

VI. Conclusion

In this paper, 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 another and we finally presented the results which are implemented on the Spartan-3 EDK evolution board.

References

[1] 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.*

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

[3] G. Angelopoulos and I. Pitas, A fast implementation of twodimensional 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.*

[4] 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.*

[5] 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.*

[6] 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.*

[7] 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.

[8] 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.*

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

[10] 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.*

[11] 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.*