ISSN (online): 2250-141X
Vol. 1 Issue 1, November 2011

Noval approach image processing algorithms on hardware implementation for surveillance systems

K.SivaNagi Reddy
Associate Professor of **Swathi Institute of technology and science**,
Hyderabad,
E-Mail:sivanag1979@gmail.com

Dr.Bhanu .M. Bhaskara

Professor &principal of **Nexus College of technoly and science**,

Hyderabad.

Abstract

FPGAs are often used as implementation platforms for real-time image processing applications because their structure allows them to exploit spatial and temporal parallelism. Such parallelization is subject to the processing mode and hardware constraints including limited processing time, limited access to data and limited resources of the system. These constraints often force the designer to reformulate the software algorithm in the process of mapping it to hardware. To aid in the process this paper proposes the approaching of design process to implement hardware patterns which embody experience and through reuse provide tools for solving particular mapping problems.

Keywords: image processing, real-time, hardware, pipeline, parallelism Field programmable Gate Arrays.

1. Introduction

Imaging and video applications are one of the fastest growing sectors of the market today. Typical application areas include e.g. medical imaging, CCTV, digital cameras, set-top boxes, and Machine vision and security surveillance. As the evolution in these applications progresses, the demands for technology innovations tend to grow rapidly over the years. Driven by the consumer electronics market, new emerging standards along with increasing requirements on system performance imposes great challenges on today's imaging and video product development. To meet with the constantly improved system performance measured in, e.g., resolution, throughput, robustness, power consumption and digital convergence (where a wide range of terminal devices must process multimedia data streams including video, audio, GPS, cellular, etc.), new design methodologies and hardware accelerator architectures are constantly called for in the hardware implementation of such systems with real-time processing power. This thesis tries to deal with several design issues normally encountered in hardware implementations of such image processing systems.

2. Pipelined image processing

ISSN (online): 2250-141X
Vol. 1 Issue 1, November 2011

Consider a memory based image processing system which implements local operators, defined for a given neighborhood of the currently processed image pixel Assuming the region of interest $W \times H$ inside the frame buffer of line width L, a standard way of accessing pixel $f(j, j)$ is to calculate its address as shown in the figure. Many processors (especially DSPs) provide memory address generator blocks, facilitating this task. Nevertheless, a more effective way is to set up a pointer to the memory and provide consecutive accesses with auto post incrimination (available in most advanced processors). This resembles the situation when we receive the image as a sequential data stream (e.g. from CCD or CMOS image sensors or USB/Fire wire/Ethernet devices).In a general case, the local operator calculates the resulting value of the pixel $g(i, j)$ on the basis of the values of all the pixels from the given window, accessible with constant offsets from the current (central pixel) pointer. Actually, there is no need for multiple pixel access

As summing an 8-connected neighborhood of radius r (a square window of the size $(2r + 1) \times (2r + 1)$, one can create a pipeline consisting of $2r$ delay lines (SISO registers) of the image line length L. The currently accessed (received, in the case of a serial input data stream) pixel, together with the outputs of all $2r$ delay lines, forms one column of the requested window (the data are accessible in parallel). In a general case, a $(2r+1)\times(2r+1)$ array of additional pixel-size registers (forming $2r$ +1) SIPO row buffers) provides simultaneous access to the surrounding pixels. The delay TD introduced by such a pipeline depends on pixel sampling period TS, image line period TL and the neighborhood radius r:

$$TD = rTL + (r + 1)TS.$$

In the case of a line containing L pixels with no blanking period, we have

$$TD = TS(r(L + 1) + 1).$$

Any local operator can thus be implemented as a static function Φ of multiple inputs and one output:

$$g(i, j) = \Phi(f(i + m, j + n) | -r \le m \le r, -r \le n \le r).$$

The output $g(i, j)$, delayed from the original data stream by TD, can be used as input data for the next processing stage of the same form. The delays of the cascaded procedures accumulate, but the overall latency remains strictly defined and constant. Operators that use only one pixel value to perform the transformation can be considered as a special case of the local ones, with the neighborhood radius $r = 0$. The implementation is much simpler, as the delay lines are not needed and we use only one input. Typically, such transformations are realized via programmable LUTs (*Look Up Tables*), memory arrays addressed by the input value and Containing the output values for all possible input values. The presented implementation concept is suitable for a great variety of early image processing (linear and non-linear): filtering (hi- and low-pass, gradients, edge enhancement, background subtraction, etc.), segmentation (Thresholding, clipping, double thresholding, template matching, etc.), morphology (hit-or-miss, dilation, erosion, opening, closing, etc.), parameterization (labeling, moments, moment invariants, etc.). The implemented procedures can be cascaded and combined parallel, forming Fast image preprocessing systems, well suited to a given task. Note that a brute force implementation of Φ is not always efficient, or even possible. Even in the case of the smallest non-trivial 3×3($r = 1$) neighborhood and 8-bit gray-scale image, Φ requires a 72-bit input word.

Good results can be obtained via the decomposition of the operator, which will be shown next.

3. Filters operators

A special class of local operators (both linear and nonlinear) is separable ones. The problem size decreases significantly if the operator Φ can be decomposed in such a manner that every column is processed independently and the

ISSN (online): 2250-141X
Vol. 1 Issue 1, November 2011

partial results for the columns are composed to form the result.
Consider the so-called Gaussian filter defined by the convolution kernel:

$$\begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix}$$

It is widely used for image smoothing (low-pass filtering), as it is easily normalized by 16 (bit shift instead of division). Moreover, to obtain a Gaussian filter of a greater radius, we can compose (cascade) two Gaussian filters: $Gr1 \_ Gr2 = Gr1{+}r2$,

which implies that it is sufficient to implement $G1$. Introducing a three-input operator Γ:

$$\Gamma(a, b, c) = a + 2b + c$$

we can calculate partial results $\gamma(i, j)$

$$\gamma(i, j) = \Gamma(f(i, j - 1), f(i, j), f(i, j + 1)).$$

The final result $g(i, j)$ is calculated in another Γ block, using $\gamma(\cdot, j)$ as inputs:

$$g(i, j) = \Gamma(\gamma(i - 1, j), \gamma(i, j), \gamma(i + 1, j)).$$

Instead of a function with a 72-bit input, we need two copies of a function with a 24-bit input.
Local minimum and local maximum operators on large windows are used for finding lower/upper image envelopes: $l\min(l\max)$, $l\max(l\min)$,
which are very useful in background subtraction methods. Both the operators are separable. Every neighborhood column can be minimized in the pipeline:

$$\mu(i, j) = MIN2r{+}1 - f(i, j{-}r), \ldots,$$
$$f(i, j), \ldots, f(i, j{+}r),$$

and the final result $g(i, j)$ is the minimum of $2r{+}1$ partial results (Fig. 3):

$$g(i, j) = MIN2r{+}1\_ \mu(i, j{-}r), \ldots,$$
$$\mu(i, j), \ldots, \mu(i, j{+}r).$$

Thus the decomposition results in reducing the problem size from $(2r + 1)2$ to $2(2r + 1)$.

4. Weighted Order Statistics filter

The Weighted Order Statistics (WOS) filter applies weights to each of the inputs in the window. The center block is assigned a weight of 3, while the four corners are assigned a weight of 1, and the remaining blocks a weight of 2. The WOS algorithm is given by the following equation:

$$Y(n_1,n_2)=\text{MED}\begin{matrix} W_1*(n_1\text{-}1,n_2\text{-}1) \\ W_2*(n_1,n_2\text{-}1) \\ W_3*(n_1+1,n_2\text{-}1) \\ W_4*(n_1\text{-}1,n_2) \\ W_5*(n_1,n_2) \\ W_6*(n_1+1,n_2) \\ W_7*(n_1\text{-}1,n_2+1) \\ W_8*(n_1+1,n_2\text{-}1) \end{matrix}$$

where 'w x' means x repeated w times. For example,4 3 means 3 repeated 4 times. There are three parts in the WOS filter that the information goes through: *weight adding, sorting, and weighted ordering*. The values, from the universal filter registers, first get a weight value assigned to it in relation to its window position. This is comprised of a set of nine adders that places a 2-bit weight value in front of the 8-bit window value, creating a
10-bit number. These nine 10-bit values then move to the registers in the sorting section of the WOS filter. The newly weighted values are now placed into a second set of registers. These registers are used to store the data as it is being sorted. The sorting section uses eight comparators to sort the values, using only the last eight bits. These comparators compare two numbers at a time, resulting in a high value and a low value. Since there are nine values that need to be compared, the eight comparators are broken into two sets of 4 comparators.
The first set compare register 1 through register 8, and the second set compare register 2 through register 9. This allows
an odd number of values to be sorted. Multiplexers determine which value is allowed back into the register, depending on which set of comparators is being used.

ISSN (online): 2250-141X
Vol. 1 Issue 1, November 2011

Basically, what this hardware does is compare the values stored in two adjacent registers, finds the high value and the low value, and swaps the values when necessary. This process continues until the values are sorted from highest to lowest. The weighted order section is the last section of the WOS filter. The first two bits of the values are checked for their weight values. Recall that the weight of a number represents how many times a number appears in a sequence of numbers. A weight of two means the number will appear twice. For the WOS filter, the weight represents how many times a value will be entered into the shift register. The shift register is made of fifteen 8-bit registers that are wired together so that values can only enter through register 1. When another value goes to register 1, the value in register 1 gets shifted down to
register 2. This continues until all fifteen registers have values. Once all the values have been placed in the shift register, the WOS filter is finished, and the final weighted rank can be output, so it can be a selection for the universal filter. A clock is used for the WOS filter so that the sorting section and the weighted ordering section changes properly. To completely obtain an output from the WOS filter takes 20 clock cycles, which makes this the critical path for the universal filter. The next slowest filter (the rational filter) takes less than half the number of clock cycles. This worst case scenario takes 22.039ns to complete.

Figure : 1 Block diagram of the Weighted Order Statistics (WOS) filter

5. Convolution filters

Linear local operators are in general performed by convolution with a given kernel of radius *r*. Consider the one dimensional example:

$$g(i)=\sum a_k f(i+k) \quad k \text{ from } -r \text{ to } r$$

Where *ak* are kernel elements (weights).
implementation of the convolution kernels for many linear operators requires taking the above constraints into account. For example, a simple 3 × 3 averaging kernel:

$$\frac{1}{9}\begin{matrix}1&1&1\\1&1&1\\1&1&1\end{matrix}$$

Can be approximated by

$$\frac{1}{2^9}\begin{matrix}57&57&57\\57&57&57\\57&57&57\end{matrix}$$

The normalization of the operator (division by 9) was replaced by shifting the result to the right by 9 bits (division by 512). Appropriate weights ($-128 \le 57 \le 127$) were applied. Another example can be a rotation-invariant 3 × 3 Laplacian:

$$\frac{1}{6}\begin{matrix}1&4&1\\4&-20&4\\1&4&1\end{matrix} \approx \frac{1}{2^5}\begin{matrix}5&22&5\\22&-108&22\\5&22&5\end{matrix}$$

The post-processor contains a programmable LUT, which offers a possibility to implement linear and nonlinear point-based operations (e.g., negation, gamma correction).

6. HARDWARE IMPLEMENTATION (FPGA)

6.1. Introduction:

This report takes a set of macro cells suitable for designing self-timed systems and a library already created for the Actel field programmable gate arrays, and presents a library for the Xilinx FPGA. The reasons for using a two-phase transition signaling protocol for control and a bundled protocol for data paths, and an introduction to the self-timed elements have already been detailed in papers .The methods of implementation of the self-timed elements on the XC4003 FPGA, and the limitations of its basic structure that were encountered when designing the cell set will be our

ISSN (online): 2250-141X
Vol. 1 Issue 1, November 2011

main concern. Some results including the area covered by and the delay through each self-timed element will be provided.

6.2. Xilinx XC4000 FPGA

Xilinx FPGA performs the function of a custom LSI circuit using look-up tables, and has a row of logic cells (Configurable Logic Blocks, CLBs) interspersed with routing channels. It is user-programmable and re-programmable in a system. The CLB has thirteen inputs and four outputs which provide access to the function generators and flip-flops .Nine of the thirteen inputs are used by the function generators, while the other four are used only by the flip flops. Two of the outputs are clocked, while the other two are not. The structure of the CLB, shown in Figure 1, is complex compared to the Actel basic module. The CLB structure controls the properties of our cell set, both area and time, as it is the basis for all the macros in the library. The XC4003, of the 4000 series, has 3000 gates, a 10x10 array of these CLBs, 80 I/O Blocks (IOBs) and a hierarchy of interconnect lines to route all of them together.

6.3. Implementation

When designing the elements, it had to be guaranteed that on programming them onto the FPGA they were partitioned, placed and routed appropriately, so that their operation is not altered in any way. Therefore, the self-timed elements that occupied more than one CLB had to be made into Hard Macros and the ones that occupied only one CLB had to be constrained to specific function generators. Hard macros improve rout ability and performance. They can only be made up of CLBs, no routing will be included and they should be rectangular for the PPR program, i.e. the CLBs that the self-timed element occupies should be arranged so that they fit into a rectangle area to minimize wastage of resources. A hard macro is created by running the HMGEN program on the *.lca file. This file can be created by the XACT Design Editor (XDE) or by running the XMAKE program on a schematic from Work view (schematic capture program used in designing the cell set). The major drawback of using Xilinx for this cell set or using it in self timed design was the lack of control over the routing, which can make the bundling constraint harder to guarantee.

Consequently, the correct operation of the self-timed elements cannot be assured. The Xilinx CLB structure is large and complex compared to, for example, the Actel basic module, and was thus harder to use to full capacity when implementing the self-timed elements. For example, the flip-flops and their clocked outputs were never used in designing any of the elements. Most of the self-timed elements contain some kind of feedback for memory. This was difficult to implement using the Xilinx CLB as it did not allow local feedback and all feedback had to be outside the CLB.

The necessity for more outputs per CLB arose, as the flip-flop outputs could not be used without an additional clock input to the element. A third function generator, H, inside a CLB could have been more useful, if it had more inputs independent of the other function generators and had two unclocked outputs. Another limitation was the maximum number of inputs to a CLB, although increasing this would make the CLB bigger

The CCL circuit has been implemented for Xilinx XC4000 series FPGAs. The Threshold block is bit parallel, and uses dedicated fast carry logic; it is highly scalable. A 16 bit threshold block occupies 9 CLBs. The Initial Labelling block occupies 3 CLBs per bit. The non-zero maximum unit occupies 14 CLBs whereas the other maximum units occupy 4 CLBs. The line buffer is implemented using XC4000 Select-RAM^{TM}. An N pixel line buffer occupies N/2 +2 CLBs. The whole CCL circuit has been implemented in an XC4010E-1 FPGA chip (20 by 20 CLBs). For a 256 by 256 image, it fits easily on to the XC4010E.

7. Results Obtained

ISSN (online): 2250-141X
Vol. 1 Issue 1, November 2011

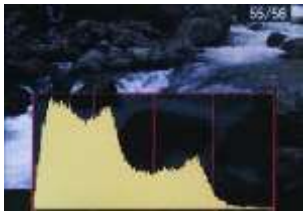
We currently support the color to grayscale conversion algorithm, convolution algorithm and histogram. The color to grayscale conversion and the histogram runs without any constraints on the size of image. Convolution algorithm can have limited width.

7.1 Colors to Grayscale

Each pixel of the image contains three bytes representing each color. To obtain a grayscale image, we have to average the color value for each pixel. The program buffers 64 consecutive pixels, calculates the average and sends the new color values of each pixel to the computer through the USB cable. The program does not impose any size limits on the image, but larger images take more time to produce results. Shown below is an example input image and the result of gray scale conversion.

7.2 Histogram

The histogram program takes a color image as input. It calculates the number of pixels with a particular color value of the three colors in R-G-B order. For each color, an array of 256 locations is created and the number of pixels, in the entire image, of a particular color level is recorded in the respective array. This information is pushed to the USB cable starting from number of pixels with color value equal to zero in the R-G-B order.



7.3 Sobel Edge Detection

We have applied two Sobel filters one after the other to perform edge detection in both directions. For this, we modified our convolution algorithm to do the same computation twice. However, since we did not have a square root function, we had to approximate the merging of two filters using simple addition operator. We have some errors in the results we obtained.

7.4 Convolution

The convolution program reads the convolution mask from the Flash and reads the amount of shift required to approximate the division operation. Originally, we had put independent statements inside par blocks, but we saw drastic improvements in the compile time when the par was removed. The mask is applied over the image starting from the first row. Remember that since BMP images are stored starting from the bottom, the first row is the bottommost row.

The program buffers three consecutive rows and then applied the convolution mask on new columns from left to right. The result of the convolution is stored in the center row left pixel. Due to this, the output image is shifted to the left by one pixel. The mask does not convolute with the last one or two columns depending upon whether the width of the image is a multiple of three (the mask size). After the result of one row is available, it is returned to the computer through the USB. An inherent inefficiency in the program is present because we are buffering three rows at a time and then next time, we again buffer two of these rows. This improvement is left for future work.

The results for two different masks are shown below. The first one is the original image, the second one is Gaussian smoothened and the third is smoothened by an all ones mask. The third figure is lighter because we

ISSN (online): 2250-141X
Vol. 1 Issue 1, November 2011

have approximated the division-by-nine with a shift-by-three.

8. Conclusions

The pipelined architecture implemented in hardware, especially in programmable logic devices, provides a constant latency of the image processing path, which fulfills the main condition of real-time systems. Implementing the procedures is well supported by widespread design tools (VHDL compilers, libraries, etc.). Low power consumption and small size of the devices encourage constructors to put the preprocessor into the image sensing unit. The cost of the image processing hardware is relatively low, and will decrease with FPGA chips enhancement. In the case of remote vision systems, this can lead to reducing the bandwidth between the vision-based sensor and the host (e.g., a robot controller). Moreover, the possibility to implement selected microprocessor and FPGA provides means of implementing the required low level post-processing and additional, high level procedures (image analysis, pattern recognition, etc.).

References

(1) Dudani S., Breeding K. and McGhee R. (1977). Aircraft identification by moment invariants, *IEEE Transactions on. Computers*, **26**(1):39-46
(2) http://www.datasheetcatalog.com.
Xilinx, Inc. (2007). *Spartan-3A FPGA family: Complete Data Sheet*, DS610,
(3) http://www.datasheetcatalog.com.
(4) Drzazga A., Hajdul J., Malec J. and Wnuk M. (1983). Hardware image preprocessor, *Technical Report*, Wrocław University of Technology (in Polish).
(5) SGS-THOMSON Microelectronics (1994). *IMSA110 Image and Signal Processing Sub-system*,
(6) S. K. Mitra and Giovanni L. Sicuranza, eds. *Nonlinear Image Processing*, Academic Press, San Diego, 2001.
(7) A. M. Grigoryan, "Mixed Median Filters and Their Properties," *Proceedings of SPIE Nonlinear Image ProcessingVIII*, vol. 3026, February 1997, pp. 8-20.
(7) L. Breveglieri and V. Piuri, "Digital Median Filters, "*Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology,* vol. 31, no. 3, July 2002, pp. 191-206.
(8)K. Egiazarian, O. Vainio, and J. T. Astola, "Implementation of Cascaded and Recursive Stack Filters," *Circuit Systems and Signal Processing*, vol. 15, no. 1, 1996, pp. 93-111.
(9) S.S.Erdongan, Abdul Wahab, T. H. Hong. "VHDL Modeling and Simulation of the Back-Propagation Algorithm and its Mapping to the PM". IEEE 1993 Custom
Integrated Circuits conference.
(10) Aldec, "Aldec-HDLTM Series User Guide Version 4.1", August 2000.