Dynamic Image Filter Selection using Partially Reconfigurable FPGAs for Imaging Operations

V.TADIGOTLA, S.COMMURI
Stephenson Research and Technology Center
101 David L. Boren Blvd, Room 1050
University of Oklahoma, Norman, 73019
UNITED STATES OF AMERICA

Abstract: - Real time image processing requires the selection of appropriate image filters. Static implementations of filter circuitry can be optimized if the image type is known at the time of reception. However, if the type of the incoming image is unknown, then improper selection of the filters could severely impact the quality of the received image. In this paper, an effective technique to select an imaging filter at run-time using dynamically reconfigurable Field Programmable Gate Arrays (FPGAs) is presented. Hardware – software co-design and hardware reconfiguration is utilized to design flexible and reliable image filtering applications. This approach is validated through a case-study where an appropriate filter is dynamically implemented in Virtex-II Pro FPGA and the resulting improvement in the image quality is demonstrated.

Key-Words: - Non-linear Imaging Filters, Implementation, FPGA, Dynamic Partial Reconfiguration

1 Introduction

Image Processing is an ever expanding and dynamic area with applications impacting our everyday life in such diverse areas as medicine, space exploration, surveillance, and authentication [1]. In real-time image transmission, images are mostly affected by additive noise (like Gaussian noise) or impulse noise (e.g. salt and pepper noise). The received images should be enhanced in such a way that the Mean Square Error (MSE) of the received image with reference to the original image is minimized. The image enhancement requires a series of filtering operations on the received image. These filters are serially connected since image processing design is divided into a number of processing stages. Implementing such filters on a general purpose computer is straightforward, but not very time efficient due to constraints on processor speed and available memory [2], [3].

The run-time efficiency of filters can be improved through the use of specially designed Application Specific Integrated Circuits (ASICs) and Digital Signal Processors (DSPs). Fast execution times for complex computations can be achieved by optimizing the circuits for the specific application [4].

System On Chip (SoC) platforms based on ASICs and DSPs can exploit the parallelism and pipelining algorithms to allow for greatly reduced execution times [4]. However, these designs result in a fixed hardware architecture and circuitry, and therefore cannot implement complex systems in an efficient and flexible manner [2], [4]. These limitations can be overcome by leveraging recent advances in Reconfigurable SoC’s based on Field Programmable Gate Arrays (FPGAs) [5-7]. The latest version of these reconfigurable systems introduces the concept of ‘Dynamic Run-time Partial Reconfiguration,’ where only a small portion of the circuitry is modified at run-time while the system remains functioning [8], [9]. Implementing image processing algorithms on reconfigurable hardware allows for the dynamic selection of the filter depending on the characteristics of the received image. This improves the image quality for a wide range of images, while simplifying the debugging and verification process [10-12].

In this paper, implementation of non-linear imaging filters on partially reconfigurable FPGAs is discussed. Prototyping of non-linear filters is especially important because the efficiency and properties of non-linear filters are difficult to quantify using formal methods. Therefore determining the effects of these filters on overall system performance often requires simulations or prototyping [13]. The design flow for the image processing system discussed in this paper is shown in Fig 1. Here the input to the design is the raw image corrupted with noise (salt and pepper noise, Gaussian noise, etc.). The main intent of this design is to use a suitable filter to eliminate the noise from the raw image and to display the processed image...
The challenge addressed in this design is the selection of an appropriate filter to filter out the noise. Control logic (shown in Fig 1) determines the selection of the appropriate filter by measuring design parameters (noise statistics) of the input image. Then the control logic block reconfigures the FPGA accordingly, filters the image and writes it to the memory (static function block in Fig 1).

The rest of the paper is organized as follows: The non-linear filters used in image processing are discussed in Section 2. Partial reconfiguration in Xilinx FPGAs is discussed in Section 3. Finally the implementation of the design and various performance indices that depict the advantage of partial reconfiguration over full/without reconfiguration is discussed in Section 4. Finally conclusions are presented in Section 5.

2 Selection of Imaging Filters

There are number of filters available for imaging operations. Based on the operation performed, these filters can be classified either as linear filters or nonlinear filters. Linear filters provide an optimal solution when the input image is corrupted by a Gaussian noise and the mean-square criterion is used to improve the image [14]. While there are a number of image processing applications that benefit from the design of linear filters, there are classes of images that require the use of nonlinear filters. Especially important amongst these classes are those of images corrupted by additive impulse noise common in the real-time transmission of images [15]. This paper addresses the design of such nonlinear filters using dynamically reconfigurable FPGAs for image processing applications.

Nonlinear filters are easy to design when compared to linear filters but their efficiency and properties are difficult to quantify using formal methods [13]. One technique of determining the effects of a nonlinear imaging filter on overall system performance is through simulation and prototyping. This can be easily achieved using FPGAs.

Some of the non-linear filters are discussed below:

2.1 Median Filter

One of the most commonly used nonlinear filters in Image Processing is the Median Filter. These filters are commonly used for smoothing of images and the removal of the noise from a corrupted image. These filters preserve sharp signal changes and are very effective in removing impulse noise (salt and pepper noise) [1]. In median filters, a window whose size is determined by the order of the filter, slides along the image and filters the content of the pixels covered by the window (see Fig 2). An impulse noise, for example salt and pepper noise, causes isolated pixels in an image to have exceptionally low or high pixel intensity [1]. These types of variations can be minimized through the averaging operations performed by the median filter.

<table>
<thead>
<tr>
<th>14</th>
<th>32</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>121</td>
<td>34</td>
</tr>
<tr>
<td>60</td>
<td>41</td>
<td>22</td>
</tr>
</tbody>
</table>

Sorted Numbers: 6, 14, 21, 22, 32, 34, 41, 60, 121

Fig 2. Median Filter

Linear filters cannot filter this type of noise without affecting the vital characteristics of the image. The main problem with the median filter is its high computation costs.

2.2 Morphological Filters

Morphological filters are another class of non-linear filters. These filters can be used on binary and gray scale images, and are useful in many areas of image processing, such as skeletonization, edge detection, restoration and texture analysis. These filters use a similar structuring element as the median filter. The basic building blocks used for many morphological operations are the erosion and dilation operations [1].

Other morphological filters such as OPEN, CLOSE, OPEN-CLOSE and CLOSE-OPEN are formed by the different combinations of Erode and Dilate filters given by [1], [17].

The implementation of these filters is discussed in the Section 4.

2.3 Ordered Statistics (OS) Filters

Ordered Statistics filters are a powerful set of non-linear filters. Ordered Statistics of a set X can be specified as

\[ X_{OS} = \{X_1, \ldots, X_{2M+1}\} \]  

Such that \( X_1 \leq \ldots \leq X_{2M+1} \)
If a linear combination of order statistics is defined as
\[ \sum_{i=1}^{2M+1} A_i X_i = A^T X_{OS} \] (2)
then \( A = [A_1, \ldots, A_{2M+1}]^T \) defines the basis for a new set of non-linear filters called order statistics filters or OS filters [17]. The filter weights are always defined such that they sum to 1. The OS coefficients are usually symmetric.

3 Partial Reconfiguration in Xilinx FPGAs

FPGAs provide an array of logic cells that can be configured to perform a function. This is accomplished by loading a configuration file into the FPGA. Today, some FPGAs support partial reconfiguration, where a portion of the device is reconfigured while the rest of the device remains operational and is unaffected by reconfiguration.

Currently, the most widely used Xilinx FPGA chips with partial reconfiguration capability are the Virtex II and Virtex-II Pro. Virtex-II Pro FPGAs are used for demonstrating the results in this paper. These devices are composed of an array of Configuration Logic Blocks (CLBs) comprising of four slices, with each slice providing two 4-input function generators, carry logic, arithmetic logic gates, multiplexers and two storage elements [18].

Reconfiguration in FPGAs can be accomplished using two different methods: Difference-based flow and Module based flow [19], [20]. Difference-based flow is suited for those applications requiring a minor change in the hardware configuration. In Module-based flow on the other hand, entire modules can be reconfigured.

Difference-based reconfiguration is useful if the modification desired in a particular FPGA implementation is small. Module-based reconfiguration is useful for applications requiring large modifications in the implementation. In the Module-based reconfiguration method, the reconfigurable elements are organized into modules at design time, and the CLBs containing these modules are rewritten at run-time. Module-based design flow has the additional advantage of allowing the use of background scripts to perform operations like placement and routing of the design elements, generation of partial reconfiguration files etc. This significantly improves the efficiency of the reconfiguration process. Since module-based design flow is more dynamic and flexible than the difference-based flow, it is preferred in partial reconfiguration process [20] – [21].

In a typical design, the filtering requirements determine the static components and the components requiring modification during the implementation. The steps involved in determining the partitioning of the static and reconfigurable modules is shown in Fig 3. On power-up, the initial configuration is loaded onto the FPGA. The initial configuration consists of the Static Modules and the initial Partially Reconfigurable Modules. The internal BlockRAMs are initialized with the program code for the PPC405, and the program code starts automatically after the configuration is loaded. Partial dynamic reconfiguration is then accomplished under the control of the software running on the PowerPC.

The partitioning of the imaging filters into static and dynamic components and their implementation is described in the following section.

4 Implementation and Results

This section discusses the implementation of the nonlinear filters discussed in Section 2 on a Xilinx Virtex-II Pro FF672 FPGA. The device is a user programmable gate array with embedded PowerPC processor and embedded high speed serial transceivers. The architecture is coarse grained and consists of a number of basic cells called CLBs.

The implementation of each of the filters, i.e. Median, Dilate/Erode, Averaging, and OS filters, is first presented. The design is studied and portions of the filter circuit common to the different filter implementations are used to design the static
portions, while the reconfigurable aspects of the filters are modified dynamically at run-time.

4.1 Implementation Steps:
The implementation consists of the following basic steps:
- Design of each filter module (using VHDL or Verilog).
- Synthesis of each module.
- Performing the initial budgeting of the top level, i.e., assigning constraints to the design, namely, area constraints for each filter module on the FPGA.
- Implementation of each individual module.
- Assembly of the entire design.

<table>
<thead>
<tr>
<th>Filter</th>
<th>Occupied Slices (total: 4928)</th>
<th>Equivalent Gate Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median Filter</td>
<td>721</td>
<td>14.7 %</td>
</tr>
<tr>
<td>Median Filter + Dilate/Erode Filter</td>
<td>739</td>
<td>14.99 %</td>
</tr>
<tr>
<td>Median Filter + Dilate/Erode Filter + Averaging Filter</td>
<td>764</td>
<td>15.5 %</td>
</tr>
<tr>
<td>Median Filter + Dilate/Erode Filter + Averaging Filter + OS Filter</td>
<td>801</td>
<td>15.73%</td>
</tr>
</tbody>
</table>

Table 1. Design space occupied by group of filters using ASICs/DSPs or FPGAs without reconfiguration

If these four filters were to be implemented in conventional ASICs or DSPs, then the total design space required is the sum of the space occupied by each filter. From Table 1, it can be seen that the implementation of all the filters in an ASIC would require a total of 1,399,702 gates. This places a serious constraint on design space.

Often an image processing design requires the use of different filters as the noise is unpredictable. This is especially true in real-time imaging applications like Defense target detection. The filters given in Table 1 are implemented in the FPGA using partial reconfiguration and the design space occupied by different groups of filters is given in Table 2.

In partial reconfiguration, the reconfigurable block should be fixed in the design space. The size of the reconfigurable block is fixed based on the space required in each of the designs.

The bit-streams of each of the reconfigurable filters are stored in an external memory and the reconfiguration is accomplished through the SelectMAP interface at 50 MHz clock speed. The average resulting bit-stream for each of the configurations is 513 kBytes resulting in a reconfiguration time of 10.26 milli-seconds and a worst net delay of about 6.381 nano-seconds. It can be seen that the run-time reconfiguration and the processing of the received image can be achieved within standard image transmission cycles.

4.2 Case Study
In this case study, the FPGA implementation of various filters identified in section II and dynamic image filter selection using FPGAs is discussed. A 256 X 256 grayscale image corrupted by 50% impulse noise (salt and pepper noise) is given as the input to the FPGA and is filtered by each of the filters. The four filters implemented in this design are the Median, Dilate/Erode, Averaging and Order-Statistics filters. The interface modules for these filters are the reconfigurable modules in this design.
For different cases of Gaussian Noise affected 256 X 256 Lena Image:

<table>
<thead>
<tr>
<th>Variance of the Input Image ($\sigma^2$)</th>
<th>When ROW (3) is used as structuring element</th>
<th>When CROSS(3) is used as structuring element</th>
<th>When SQUARE (3) is used as a structuring element</th>
<th>Selected Structuring element by the FPGA</th>
<th>MSE of the Filtered Image</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MSE of the Filtered image</td>
<td>Computation time (ms)</td>
<td>MSE of the Filtered image</td>
<td>Computation time (ms)</td>
<td>MSE of the Filtered image</td>
</tr>
<tr>
<td>25</td>
<td>307</td>
<td>0.96</td>
<td>336</td>
<td>1.01</td>
<td>318</td>
</tr>
<tr>
<td>100</td>
<td>321</td>
<td>0.964</td>
<td>330</td>
<td>0.98</td>
<td>345</td>
</tr>
<tr>
<td>225</td>
<td>342</td>
<td>0.956</td>
<td>347</td>
<td>1.05</td>
<td>358</td>
</tr>
<tr>
<td>400</td>
<td>371</td>
<td>0.9</td>
<td>369</td>
<td>1.0043</td>
<td>378</td>
</tr>
<tr>
<td>625</td>
<td>406</td>
<td>0.9266</td>
<td>397</td>
<td>1.1</td>
<td>402</td>
</tr>
<tr>
<td>900</td>
<td>448</td>
<td>0.971</td>
<td>431</td>
<td>1</td>
<td>432</td>
</tr>
<tr>
<td>1225</td>
<td>496</td>
<td>0.973</td>
<td>471</td>
<td>1</td>
<td>467</td>
</tr>
<tr>
<td>2025</td>
<td>611</td>
<td>0.9634</td>
<td>568</td>
<td>0.993</td>
<td>555</td>
</tr>
<tr>
<td>3025</td>
<td>753</td>
<td>0.961</td>
<td>665</td>
<td>1.001</td>
<td>686</td>
</tr>
</tbody>
</table>

Table 3. Dynamic Structuring element selection for filtering Gaussian noise using Averaging filter

The FPGA utilization and efficiency becomes significant as the number of required filters in the design increase.

The filters implemented above, share a number of common processing elements which are recreated for each implementation. Thus while the process leads to a flexible design, the implementation itself is inefficient. The implementation process can be streamlined by identifying the common elements and implementing these in the static area of the FPGA. The identification of common elements and the dynamic reconfiguration of the filters from one form to another is the topic of this case study. Dynamic selection of the filter window based on the noise statistics will be used to design the reconfiguration process.

For dynamic image filter selection using FPGAs, the filtering of images corrupted by Gaussian noise of zero mean and variance $\sigma^2 (N(0, \sigma^2))$, is considered. The filtering mechanism used is Average filter i.e., Order Statistics filter with equal weights on each coefficient. The input image is first compared with the reference image and the Mean Squared Error is calculated. The MSE obtained when using different windows (ROW(3), CROSS(3), and SQUARE(3) shown in Fig 4) for images corrupted with noise of different variances is shown in Table 3. The performance of these three structuring elements for images corrupted with different noise characteristics is then used to determine the control logic for the run-time selection of the filter. The control mechanism that is followed is discussed below.

Fig 4. Window Structures
The summary of the results for different input images corrupted by Gaussian noise of different standard deviations was shown in Table 3. It can be seen from the table that for less noisy images ROW (3) filters the image better (i.e. preserves edges better than CROSS (3) and SQUARE (3)) with less computation time. Similarly CROSS (3) performs better for medium noisy images and SQUARE (3) performs better for more noisy images. The selection of appropriate window becomes crucial in some applications like real-time image sequence streaming at a faster rate where computation time is critical. The selection of appropriate window by the FPGA is shown in Table 3.

For the designs which require more number of filters and structuring elements, hardware implementation based on the partially reconfigurable FPGAs is the better solution because using these FPGAs, more circuitry can be accommodated on the single FPGA without increasing significantly the design space.

5 Conclusions

In recent years, reconfigurable computing has emerged as a tool for rapid prototyping and for reducing the time to market for numerous products. While comparatively expensive, these platforms are well suited for image processing applications requiring run-time flexibility in the choice of filters. In this paper, the dynamic implementation of various image processing filters using reconfigurable FPGAs was examined. The implementation of Discrete Cosine Transforms (DCT), the Inverse Discrete Cosine Transforms (IDCT), and their application in real-time video processing is being investigated and is the scope for our future work.

References