# An Investigation of FPGA Implementation for Image Processing

JianXiong   Q. M. Jonathan Wu

Department of Electrical and Computer Engineering University of Windsor, Windsor, ON N9B 3P4
xiongj@uwindsor.ca, jwu@uwindsor.ca

***Abstract*-** Field Programmable Gate Arrays (FPGAs) were not specifically invented for image processing application but its intrinsic parallel computation capability offers high-performance image processing solutions, which are often tenfold and even hundredfold faster than traditional processors. However, not every image processing algorithm is suitable for FPGAs. It is significant to understand the strengths and limitations of FPGA, which we will address in this paper. Further we will discuss the following two concerns 1) what kinds of algorithms can be easily implemented in FPGAs? 2) System architecture that meets the needs of video-rate processing. Finally, this paper introduces the system-level tools required for transforming the algorithm into an FPGA implementation and describes an Intellectual Property (IP) design example.

## I. INTRODUCTION

Image processing has been used in a wide range of industrial, commercial, civilian and military applications for over two decades and some notable applications include medical image analysis, public video surveillance, and automatic vehicle guidance and human machine interface. One of the inherent limitations encountered when dealing with images is large data size that impedes development of systems for real-time implementation.

In order to enhance real-time implementation, two aspects of implementation are endeavored; one is to optimize algorithm and the other is to adopt a novel hardware platform. The most popular hardware platform used is the general purpose central processing unit due to its matured operating system and user-friendly interface. However, due to increase in image size, data width, interruption of operating system by user instructions and other regular management real-time application becomes less realizable.

Two possible approaches to enhance the performance of a hardware processor are to increase the operation frequency and to use parallel operation. For the latter case research community has already witnessed the adoption of a variety of processors, such as multi-core CPU, Digital Signal Processors (DSPs), Single Instruction Multiple Data (SIMD) processor, and Graphics Processing Unit (GPU). These processors are designed to enhance parallel processing. Take DSP as an example, its Harvard architecture separates storage and signal pathways for instructions and data. This is done to ensure that both data and instruction can be fetched simultaneously to realize temporal parallelism. In addition, there are also other components inside DSP to realize parallel operation such as Multiply and Accumulate (MAC).

Field Programmable Gate Arrays (FPGAs) have emerged decades ago but were tradionally used for glue logic. With an increase in its logic density, FPGA began to become a useful parallel platform for image processing. It is made up of a large number of logic arrays and abundant I/O pads, and forms a general and unspecified logic circuit ready for custom configuration. FPGA's advantage relies in its parallel processing capability, which offers temporal parallelism at the expense of spatial parallelism. Another significant feature of FPGA is its software like reconfiguration flexibility [2].

Some FPGA novice may think that FPGA's advantage rests with its super high operation frequency. Unfortunately, owing to the limitation of its architecture and manufacture process, contemporary FPGA can only run at a maximum of several hundred MHz, while a common CPU in our PC can easily reach several GHz clock frequencies. In terms of operation frequency, FPGA is absolutely a loser.

The reason why FPGA can outperform other processors is that FPGA is a real parallel processor, for example it will take 5 operation cycles for CPU to finish an addition: (1) fetch instruction from memory, (2) decode the meaning of fetched instruction, (3) fetch data from memory, (4) execute addition operation, (5) write the result back to memory; while for FPGA, it will only take one operation cycle to finish this operation. If there are 20 addition operations, a CPU will take 100 cycles to finish while FPGA still can finish it in one cycle because FPGA can use 20 adders simultaneously. Now assuming that the CPU can run at 2GHz clock frequency and each operation will take only one clock cycle, then to finish 20 additions will cost $0.05\mu s$. Assuming that the operation clock of FPGA is 100MHz, then it will only take $0.01\mu s$ to finish the same 20 operations. Hence it is easy to realize the benefits offered by FPGA's parallelism. In particular for operator intensive processing, more significant improvements can be achieved by FPGA compared to serial processors like CPU.

However, not every image processing algorithm is suitable for FPGAs. It is significant to understand FPGAs' strength and limitation. The positive side of FPGA has been addressed above, and the remaining paper will present its negative side. In addition, some issues of FPGA implementation for image processing will be discussed.

The following content is organized as follows. Section 2 points out FPGA's weakness, and analyzes the features of algorithms that can be implemented with FPGA with relative ease. Section 3 describes general system architecture of FPGA platform for video rate image processing. In section 4, the system-level tools for mapping algorithm into FPGA Implementation are introduced and an IP design case is presented. Finally conclusion is given in section 5.

## II. ALGORITHM FEATURE AND FPGA LIMITATION

Image processing algorithms vary depending on different application, hence numerous algorithms exist. But they can be classified into two groups: memory-independent algorithm and memory-dependent algorithm.

Memory-independent algorithms have the following two features enabling it to perform in a stream-like mode and thus are easy to implement in FPGA.

1) Neighboring operation.

The so-called Neighboring operation is popular among many image processing algorithms, for example, Median Filter, Edge detector (including Sobel, Prewitt, Laplacian and Gaussian, Canny), Harris Corner detector, and stereo vision algorithms. They all can be processed with a sliding window in a raster scan order (similar to the incoming pixel stream from a digital camera). In literatures, these algorithms are called neighbor operation or kernel operation or point operation.

2) One-pass operation.

It means that there are no iterations in the algorithm and image processing can be done in just one pass. This feature eliminates the demands for storing one whole frame of image data into memory.

Many neighboring operators are also one-pass operators but some are not, for example one basic image processing algorithm called Connected Components Labeling, also belongs to a neighboring operator class but it cannot be completed in one pass. In [3], a single pass connected components algorithm is presented but it is not practical in real applications since it occupies too many on-chip memories. When it comes to High Definition (HD) image, off-chip memory is necessary to store image data.

Memory-independent algorithms are suitable for FPGA implementation because it eliminates the requirement of off-chip memory. Usually, only a small amount of image data is stored temporarily in the on-chip memory of FPGA for window processing. This reduces the cost of board components and increases the speed of system so as to realize real video-rate processing.

On the contrary, memory-dependent algorithms usually require at least two iterations in operation, and cannot finish operations in one pass. So it has to store incoming image data into external memory for practical application.

When it comes to memory-dependent algorithms, the limitation of FPGA becomes evident.

a) The introduction of external memory will hinder the speed of the whole system since the bottleneck lies in the interface between FPGA and the external memory. The memory bandwidth will determine the overall system performance.

b) Memory-dependent algorithms often concern complicated operation. It may be relatively easier to develop from the view point of pure software like Matlab or C language, but it is not always easy to change software into pure hardware [1].

In addition, FPGA has some other issues that should be paid attention to, for example the difference between fixed point number and floating point number, as well as the available operations in FPGA.

## III. SYSTEM ARCHITECTURE FOR VIDEO RATE IMAGE PROCESSING

Real-time image processing is closely related with hardware structure. Assuming that one digital camera sensor is used for capturing image data, if the system can handle every frame at the speed of camera's video output, then we say it is a video rate image processing system.

According to the discussion in section 2, it is apparent that it is relatively easier to implement memory-independent algorithms in FPGA for video rate processing. Hence, the following general system architecture is provided for solving this problem.

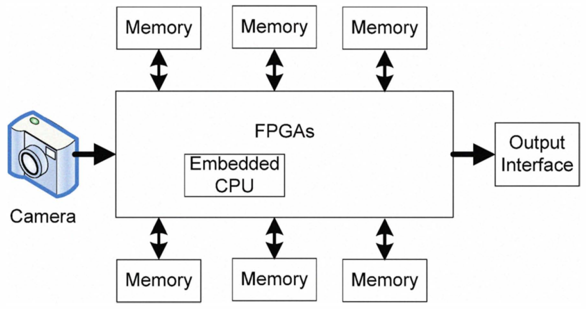

Figure 1. System Architecture

Figure 1 displays an image acquisition and processing system. Image video is captured via camera, and then transferred to FPGA for processing and the processed result is output via output interface. Here, one FPGA and six external memories are displayed for concept illustration. In real hardware platform, multiple FPGAs can be used, and the number of external memories can be any number larger than or equal to 2. The key point of this general architecture is the adoption of multiple independent external memories, which could work as buffer and assist pipe-line processing or ping-pang operation. Addition of external Memories will only increase output latency while the system throughput will remain the same.

Furthermore, no matter how complex the algorithm is, it is still possible to reach video-rate operation by simply expanding the number of external memory units. For instance if the image processing operation will take a period of 1 video frame, we can use only 2 external memories for video-rate processing. If the processing is too complicated and will take a period of 10 video frames, then we merely increase the number of memory units to 11. In short, it is always possible to reach video-rate operation at the expense of extra memories.

Figure 1 also shows an embedded CPU inside an FPGA. Its purpose is to take care of the tradeoff between development time and system performance. With the introduction of embedded CPU, the time spent on algorithm implementation will be greatly reduced. But of course, the overall performance will be impacted. Therefore it is necessary to understand every part of the algorithm and make an informed division between software and hardware requirements.

The following table lists part of the contemporary FPGA-based processors [4]. Some of them exist in silicon as a hard IP, and some can be incorporated within the FPGA as a soft IP.

TABLE I. FPGA-BASED PROCESSOR

| **Processor name** | **Type/Bits** | **Interface bus** | **FPGA vendor** |
|---|---|---|---|
| MicroBlaze | Soft/32 | IBM Coreconnect | Xilinx |
| NIOS | Soft/32 | Avalon | Altera |
| LatticeMico32 | Soft/32 | Wishbone | Lattice |
| CoreMP7 | Soft/32 | APB | Actel |
| ARM Cortex-M1 | Soft/32 | AHB | Vendor independent |
| LatticeMico8 | Soft/8 | Input/Output ports | Lattice |
| Core8051 | Soft/8 | Nil | Actel |
| Core8051s | Soft/8 | APB | Actel |
| PicoBlaze | Soft/8 | Input/Output ports | Xilinx |
| PowerPC | Hard/32 | IBM Coreconnect | Xilinx |
| AVR | Hard/8 | Input/Output ports | Atmel |

## IV. IP DESIGNED WITH SYSTEM-LEVEL TOOLS

One of the goals of FPGA design is to ease the transformation from algorithm to real hardware circuit. In recent years, a number of system-level tools began to emerge. Table 2 lists some of the system-level tools that have emerged in the past.

TABLE II. SYSTEM-LEVEL ALGORITHM MAPPING TOOLS

| | **Tool Name** | **Tools Developer** |
|---|---|---|
| Matlab based | System Generator for DSP | Xilinx |
| | AccelDSP | Xilinx |
| | Simulink HDL Coder | Mathworks |
| | Synplify DSP | Synplicity |
| C/C++ based | System-C | OSCI |
| | Catapult-C | Mentor Graphics |
| | Impulse-C | Impulse Accelerated Technologies |
| | Mitrion-C | Mitrionics |
| | DIME-C | Nallatech |
| | Handel-C | Celoxica |
| | Carte | SRC Computers |
| | Streams-C | Los Alamos National Laboratory |

Compared to C/C++ based tools, Matlab based tools have a much shorter learning curve. While among Matlab based tools, AccelDSP is more flexible than others, and therefore is recommended here.

AccelDSP is a high-level DSP synthesis tool facilitating the mapping from a Matlab floating-point design to Xilinx FPGA fixed-point design. It reads and analyzes Matlab code and then automatically generates a fixed-point version Matlab design. Next, this fixed-point design will be verified, simulated and finally a synthesizable RTL HDL code will be generated.

Every AccelDSP project must have two ".m" files i.e. a script m-file and a function m-file. The script m-file is used to apply stimulus and plot results whereas the function m-file is used for realizing the design functions. Certain style of Matlab codes are mandatory and synthesizable Matlab codes must use loops to process every pixel, and at the same time complicated functions and operations like convolution cannot be adopted.

The users can decide if the auto-referred data precision is enough; and if not, users are allowed to manually adjust the model to reduce quantization error. In the flow of AccelDSP design, this conversion from floating-point to fixed-point is the most critical and time consuming process.

In the following example, the design of a Canny Edge Detector [5, 7] is described to point out some tricks for designing IP in AccelDSP.

Canny is a popular edge detector and has a complex operation but it is still a one-pass algorithm that is suitable for hardware implementation. It consists of four stages: image smoothing, gradient calculation, non maximum suppression, and hysteresis threshold. The following figure shows the flow of a canny edge detector algorithm. Incoming pixels walk through these four blocks and get processed sequentially in a raster scan order.

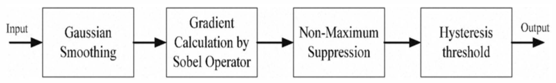

Figure 2. Flow of Canny Detector

The first block adopts a Gaussian 2D filter shown below to filter out unwanted noise.

$$coeff=\begin{bmatrix} 1 & 4 & 7 & 4 & 1\\ 4 & 16 & 26 & 16 & 4\\ 7 & 26 & 41 & 26 & 7\\ 4 & 16 & 26 & 16 & 4\\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}/273;$$

Figure 3. Gaussian 2D filter

Here, multiplication instead of division is recommended for handling the parameter "273" shown in Figure 3, since dedicated hardware multiplier is available in contemporary Xilinx FPGAs.

The second block is to find the edge strength by calculating the gradient of image. Sobel filter is used for simplifying hardware since only shift operation is needed. The following gives the coefficient of Sobel.

$$Sx=\begin{bmatrix} -1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1 \end{bmatrix} \qquad Sy=\begin{bmatrix} 1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1 \end{bmatrix}$$

Figure 4. Sobel filter

Horizontal and vertical edge strength Gx, Gy can be obtained. The magnitude of the gradient is then approximated using the following formula:

$$G=|Gx| + |Gy| \tag{1}$$

In block 3, non-maximum suppression is done in the following steps. At first, edge direction is evaluated using the value and sign of the gradient components. For each pixel, by just simply comparing Gx and Gy, the direction can be approximated by one of the sectors shown in Figure 5. For example, at one pixel, if ($Gy <= 0$ & $|Gx|/2 >$

-Gy), the direction of this point can be approximated by section of 0 or 180 degree. The benefit of this approximation is that no complicated division operation is incurred but simple shift operation is used.

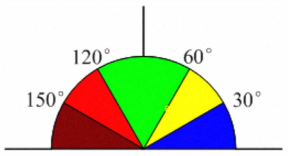

Figure 5. Direction Approximation

Once the approximation of the gradient direction is known, the gradient magnitude of the neighborhood pixels along the calculated direction will be compared and the pixel that has no local maximum gradient magnitude is eliminated.

Finally, in block 4, hysteresis threshold is used as a means of eliminating streaking. This helps to reduce the breakup of an edge contour in the final output image. Two thresholds are required and their choice will exert a strong influence on the final appearance of the output image. Here the thresholds are fixed and chosen through experimental evaluations.

The Matlab code is written according to the aforementioned processing procedure, and is synthesized with AccelDSP by using Xilinx xc5vlx110 FPGA. The generated IP can support up to 134.2MHz clock speed, and costs 25 multipliers, 49 adders as well as 35 subtracters. 33 startup clock cycles exists, while one effective output data can be generated every clock cycle. Generally speaking this is an acceptable IP at the cost of improved design time.

Compared to hand-coded RTL module, the results of AccelDSP are less efficient in terms of area and timing. The generated RTL code also lacks readability, making it difficult to maintain, however, it is still worthwhile to use AccelDSP since it can dramatically reduce design time.

## V. CONCLUSION AND FUTURE WORK

This paper addresses some basic concerns about FPGA implementation for image processing, presents feature algorithm as well as FPGA implementation issues, and also provides a general FPGA platform for video-rate application. Finally system-level algorithm mapping tools are introduced and one case study is presented to point out some tricks of IP design algorithm with AccelDSP. Based on our experiments it is safe to claim that AccelDSP is a useful tool for designing small and less complex IP blocks.

Our future work is to design a FPGA based platform with flexible memory interface to realize video-rate applications and to adopt the system-level design tool for accelerating IP development.

## REFERENCES

[1] K. T. Gribbon, D. G. Bailey, A. Bainbridge-Smith, ‘Development Issues in Using FPGAs for Image Processing’, Proceedings of Image and Vision Computing New Zealand 2007, pp. 217–222, Hamilton, New Zealand, December 2007.

[2] Katherine Compton, Scott Hauck, "Reconfigurable computing: A survey of System and Software," ACM Computing Surveys, Vol. 34, No.2, June 2002, pp171-210

[3] C.T. Johnston and D.G. Bailey, "FPGA implementation of a Single Pass Connected Components Algorithm", in IEEE International Symposium on Electronic Design, Test and Applications (DELTA 2008), Hong Kong, 228-231 (23-25 January, 2008).

[4] Rahul Dubey, “Introduction to Embedded System Design Using Field Programmable Gate Arrays”, 2009 Springer-Verlag London Limited, ISBN 978-1-84882-015-9.

[5] M Venkatesan and D Venkateshwar Rao, “Image Processing Algorithms on Reconfigurable Architecture Using Handel-C”, Journal of Engineering and Applied Sciences 1 (2): 103-111, 2006, Medwell Online, 2006.

[6] T.Saegusa, T.Maruyama, Y.Yamaguchi, “How fast is an FPGA in image processing?”, FPL 2008, pp.77-82.

[7] J. F. Canny. A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6):769–798, November 1986.