UNIVERSITY OF CALIFORNIA
RIVERSIDE

The FPGA Implementation of Computer Vision Algorithm

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Electrical Engineering

by

Zhonghua Zhou

June 2014

Thesis Committee:

Dr. Qi Zhu , Chairperson
Dr. Amit Roy-Chowdhury
Dr. Ming Liu

The Thesis of Zhonghua Zhou is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I am grateful to my advisor, without whose help, I would not have been here.

To my parents and my love for all the support.

ABSTRACT OF THE THESIS

The FPGA Implementation of Computer Vision Algorithm

by

Zhonghua Zhou

Master of Science, Graduate Program in Electrical Engineering
University of California, Riverside, June 2014
Dr. Qi Zhu , Chairperson

Computer vision algorithms play an significant role in vision processing, and it is widely applied in many aspects such as geology survey, traffic management and medical care, etc.. Most of the situations require the process to be real-timed, in other words, as fast as possible. Field Programmable Gate Arrays (FPGAs) have a advantage of parallelism fabric in programming, comparing to the serial communications of CPUs, which makes FPGA a perfect platform for implementing vision algorithms. These algorithms usually have a very high computation power because the objects, a large amount of pixels of a single picture, have to be proceeded not once, but many times. This project reconfigured onto the FPGA board, a partial of a algorithm that has multiplexing computations. The algorithm of Harris corner detection is chosen, which contains an important step in many vision processing and is of good performance. The reconfiguration of a portion that is the most time-consuming part has been synthesized onto the board and a result is presented to demonstrate the performance and the capacity of the FPGA. It shows that FPGAs can achieve a faster speed than that of some CPUs due to the parallelism, which is the specialization of FPGAs and is unfeasible to apply to software.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Prior and related works

For applying image processing algorithms onto hardware, the most popular used algorithm that is implemented on the FPGA is the convolution. Many researchers have used FPGA to be the platform to accelerate the image processing system to approach their needs. Sridharan and Priya [9] took the advantage of parallelism to implement visibility graph on FPGA. Perri et al. [13] introduced a single-instruction multiple-data method to achieve the fast convolution on FPGA. Further more, Benedetti et al. [1] designed a multi-FPGA first-in first-out (FIFO) structure to implement the convolver. Zhang et al. [8] used a multiwindow partial buffering scheme to achieve a faster data flow.

Although these researchers did many works on dealing with the timing of the convolution to optimize the hardware image processing, they did not pay many efforts on fully using the FPGA board that they were implementing on, in specific, combining other resources to further accelerate the speed of a certain algorithm. In this paper, we have designed a co-op system that uses both CPU and FPGA chip, and have also

purposed a new approach to speed up the convolution by using a single-input multi-output FIFO. Claus et al. [5] optimized a SUSAN corner detection onto FPGA, and we have decided to implement the most widely used Harris corner detector so that this implementation may have a better compatibility.

1.2 Purpose of the Thesis

This thesis demonstrates the advantage of hardware by implementing a specific software algorithm onto the DE2i-150 board, differ from many attempts that have been designed for serial processing [12], although it is possible to actualize such serial operations on a architecture of FPGA, its potential of speed-up has been wasted because the availability of parallelism may not be fully used. Moreover, most of the hardware devices like FPGAs have only very low clock frequency, usually in range of megahertz, so sometimes those implementations were but a simple transplant and could even be much slower compare to CPUs, which are always operate in a frequency range of gigahertz in nowadays industries.

The reason that FPGA is chosen is because which has a very flexible structure. For instance, data can be stored in both memory or logic, which gives the programmer more than one method to achieve one certain goal. Unlike GPUs, they are more like a software implementation though parallelism is also available.

Implementing a algorithm onto FPGA is a rough task, even if there are many approaches, but you can hardly tell which is the easy way or the best way to achieve it. It's also a challenge to meet the requirements of clocks for different modules, which you will never have to worry about on a CPU platform. In addition, due to FPGA's specialism, everything has to be fixed, otherwise the synthesis will not go through successfully

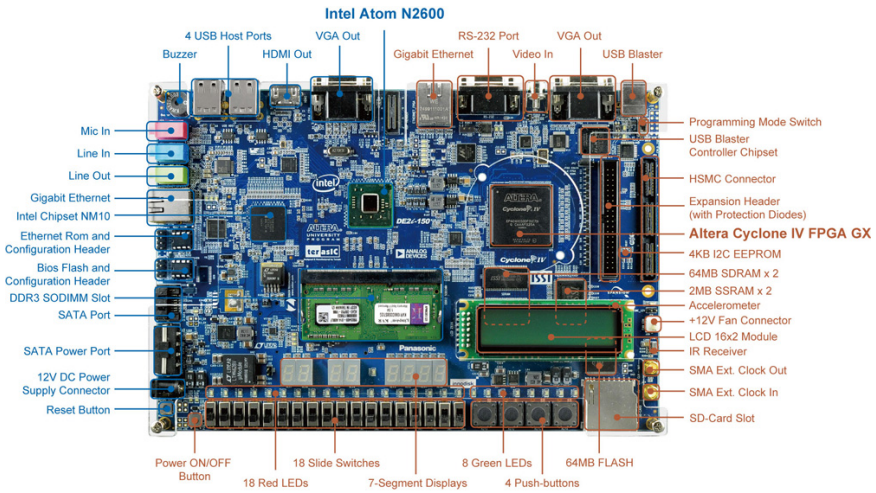

Figure 1.1: DE2i-150, used in this thesis

, which largely limits the design of functioning. Besides, a single block would consist of many low-level logic resources, which means for programmers it is more time-consuming and laborious comparing to the software developing.

1.3 Development of image processing and its hardware platforms

It is quite relatively related between the development of digital image processing and that of computer and integrated circuit. The platforms that has been used to process digital images are: PCs, professional integrated circuits, DSPs and FPGAs.

In 1946 to 1964, PCs had entered the era of "transistor", until then, we consider computers modern. Because transistors have better reliability, faster processing speed and lower cost of energy, the size and price of computers had both decreased a lot, which made it more and more popular for people to use. So the actual development of image processing started with the popularize of computers. After then, along with

the development of integrated circuit, computers had entered an era of "large scale circuit". Computers in this period are often used as a work space for non-real-time image processing and management. In order to improve the capacity of image processing, CPU was introduced to work with the CPU, but GPU was but a accelerator of CPU commands, which was not parallelism and result in the ability of image processing remains poor.

A solution of handling real-time image processing was eagerly needed, equipment manufacturers started producing professional image processing chips using integrated circuit to lower the cost. Such chips have the following advantages comparing to the old image processing equipments: these chips have small size and weight, fewer output ports and bonding points, longer life of use, lower cost and feasible for large-scale production. Nowadays, a huge amount of professional image processing chips like ASIC and ASSP are still holding the majority of the market share.

It is too much cost to apply professional chips for the system of image processing in small quantities, and it would also spend a longer cycle of research and developing. Digital Signal Processor (DSP) is a microprocessor that has a special architecture and it well meets the above needs. It has its own DSP commands, hardware multiplications, the Harvard structure that separates the procedures and data, which makes DSP the perfect processor of image processing in small quantities for recreating 2D and 3D image, image express and robot visions, etc.. DSPs have a very high speed of proceeding commands but are still limited in processing image data. So it is more suitable for image processing systems of small quantities and lower data handling capacities.

In the early days. FPGAs were usually used as a contemporary platform for image processing in many research facilities and companies, or sometimes a substitution replaced by ASIC or ASSP. FPGAs might have been equipped in some image processing

systems but are considered to be companion chips. In general, FPGAs are added for functions like porting, driving, and flexibility, while the main task of image processing is being proceeded by ASIC or ASSP.

The cost of ASIC and ASSP is increasing sharply with the capacity development of which. By applying FPGA, one can freely edit the structure of circuit, and the EDA tools provided by FPGA manufacturers can easily help equipment manufacturers doing the work of layout, which greatly shortened the research cycle and cost comparing with ASIC or ASSP.

Semiconductor manufacturers produce FPGAs as general productions when most of the equipment manufacturers starts to use FPGA as the main processor for image processing. If FPGAs were to achieve the same capacity and circuit scale as ASIC or ASSP, it would replace most or even all of the functions of ASIC or ASSP in many fields, become the main processor instead of "companion" chip.

FPGA manufacturers are advancing the development of FPGAs rapidly to accelerate the process of replacing ASIC or ASSP. Take Altera for example, which is also the supplier of the FPGA that is being used in this thesis project. Altera introduced Stratix IV in 2008 that has 40nm fabrication and a 350HMz operating frequency, it is very competitive to standard cell ASIC in 90nm.

Still, limitations like price exist if FPGAs were to replace ASIC or ASSP. Although FGPAs could achieve a high performance but which has a price over thousands dollars, way higher than competitive ASICs. In addition, because of the universality of FPGAs, some functions might be wasted if it were aimed to a specific image processing equipment.

1.4 Objectives

- Implementation of partial algorithm onto FPGA
  - The complexity of computation is obtained from GProf, the portion is selected that has been called most and has the most computation power. In the algorithm of Harris corner detection, the Gaussian smoothing has been called 6 times and consumed more than 75% of the total run time.
- Implementation of PCI express (PCIe) system
  - Because the algorithm was based on the software so the input must be sent from the CPU as well as the result is to be sent back to it. The PCI express is the bridge that connects the FPGA and the Intel chipset, where the CPU is linked through DMI bus.
- Implementation of protocol functions
  - The integrated PCIe in the DE2i-150 board is the only interface can be used for the communication. But it only handles slow PIO transaction of double-word size data per time and cannot allow the host PC nor the FPGA to access the same memory. It means most hard IPs are unavailable so that a directly communication will be present for the FPGA and the CPU.

1.5 Thesis outline

The rest of this thesis is organized as:

Ch. 2 Framework - The basic knowledge and structure of the algorithm will be described here.

Ch. 3 Algorithm - In this chapter we go through the details of the algorithm, background of the development, usage in image processing, and compare with other similar algorithms.

Ch. 4 Hardware - Here we introduce the hardware that we used and developed in the project, implementation of the portion of the algorithm is to be demonstrate in specific.

Ch. 5 Result - This chapter shows the result we have, and compares the speed of FPGA with the original algorithm.

Ch. 6 Conclusion - The last chapter contains the conclusion about the performance of the implementation together with a briefly discussion that concerning the possible future work.

Chapter 2

Framework

This chapter describes the structure of the algorithm framework as well as the reconfiguration of the portion on the FPGA.

2.1 Overview of algorithm

The overview procedure of the chosen algorithm is shown in fig. 2.1. Once the image is sent from a camera or devices of any kind, derivatives will be derived from two directions, horizon and vertical, respectively. Then, products of derivatives

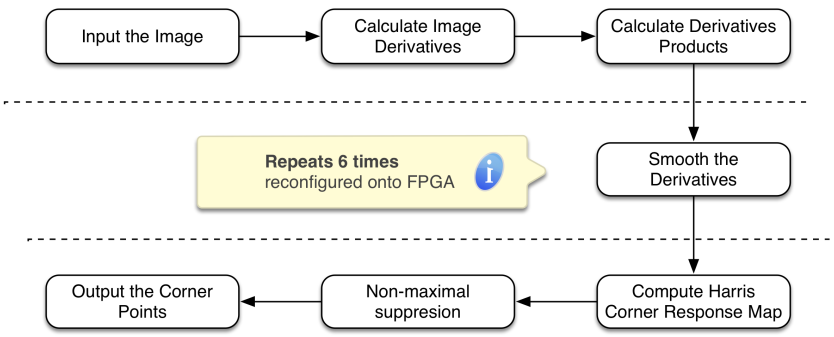

Figure 2.1: The Block diagram of the Harris corner detection

will be calculated in order to meet the need of Gaussian smoothing. The Gaussian smoothing portion is the non-trivial task not only for corner detection, but for other image processings as well. Once the smoothing has done its work, scores of pixels, or values of corners will go through the computation known as non-maximal suppression and finally the corners will be drawn differently on a copy of the original image. The details of the algorithm including mathematics and implementations will be discussed later in chapter 3 and chapter 4, respectively. Note that the entire procedure is proceeded on a copied image, the original one remains untouched.

2.2 Overview of the implementation

The implementation of the partial algorithm is the main task of this project. Motivated by the real-time demand of image processing, I decided to take the advantage of parallelism that FPGA has to offer. The given algorithm Harris corner detection has been designed and optimized for CPU serial processing, the basic idea of this implementation is to design a block logic that can be simply duplicated to take the whole serial process apart. Because each pixel of the image is treated exactly the same, so that we can transform the serial processing into multiple parallel processing by splitting pixels into different blocks. Fig 2.2 shows a diagram of the implementation. The Gaussian kernel is saved in a logic onto FPGA, consecutive input is to be stored in a buffer, together proceed to the next stage, convolution. Convolution is basically a combination of addition and multiplication, and these operators have different operating clock cycles, in order to secure the accuracy of the computation, a state machine is introduced to control the timing of these operators. Results after the convolution will be saved in a certain buffer, which with the input buffer (pixels buffer) decides what the current

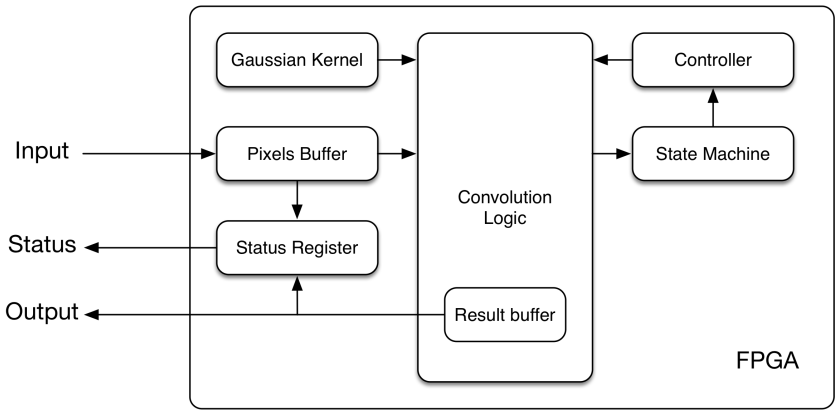

Figure 2.2: The Block diagram of the FPGA implementation

status the FPGA is. Finally the CPU can determine whether or not the current output is valid to retrieve.

2.3 Communications

Because the implementation is the partial of the whole, the internal signals exchange protocol must also be developed. Fig. 2.3 shows the CPU activity of data exchanging with the FPGA. Whenever the CPU wants to send data, which is pixel in this case, it has to confirm that the FPGA is in ready stats by sending a "request" signal to the FPGA, error might occur if former pixels had not finish the calculation. If the FPGA is stood by, then a "ready" signal will be sent back to CPU. As soon as CPU has received that signal, a new pixel is to be input into the FPGA, then CPU sets itself idle, meanwhile, it keeps trying to obtain a signal that tells the CPU the newest result has been calculated. Once "retrieve" signal is obtained, the final output data will be achieved and be stored, at the same time, the CPU has to tell the FPGA that the transmission is done, after the entire cycle has completed, CPU will go back to the very first step and start over again untill the whole image has been proceeded. Please note

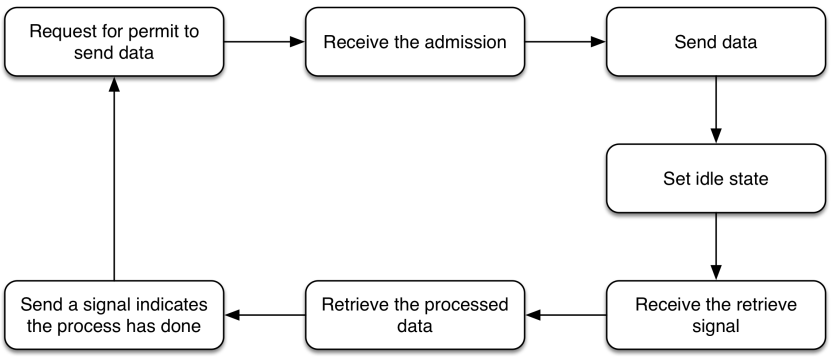

Figure 2.3: Data exchanging cycle of CPU

that in case of security and robustness, none of the above steps we talked about can be eliminated, which, at least in this project, are the only weaknesses that vastly slowed down the speed if we take CPU and FPGA as a integral system. The details of the signals and how they are sent through two platforms will be described in chapter 4.

Chapter 3

Algorithm

The algorithm of corner detection is a widely used pre-processing for many advanced image processing algorithms as well as an important algorithm itself. Normally, corner detection algorithms are realized in software domain, but due to its complexity in computation we mentioned in previous chapter, it is necessary to propose a implementation on hardware that can significantly improve the time in processing.

3.1 Background of corner detection

The corner detection, or interest point detection, is a useful method that used of an image to extract the feature or infer the context. It is predominantly applied in many aspects such as image mosaicing, tracking and recognizing, etc..

What is a corner, it is extremely easy for human beings to recognize a corner, but how can computer distinguish whether a point is corner or not? Fig. 3.1 shows three different kinds of situations. In computer vision, a point can be considered as a "corner" if there is a intersection of two edges. Also, corner can be well recognized if there are two different and obvious edge directions of a point. edge is a line that has a

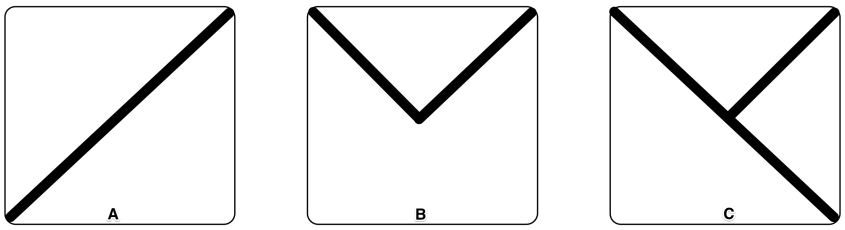

Figure 3.1: Definition of Corner

strong contrast comparing to the background. Back to the figure, now we can say that object A is a edge and object B and C are both corners. Notice that there is a difference between the computer world and the human vision, we don't think object C is a corner but for computers, it is a well-defined corner.

The reason that corner detection is called "interest point" detection is because in general the algorithm not only detects corners, but also interest points. corners are all interest points but sometimes interest points are not always corners. If a point can be well detected and has a certain defined position, then it is considered an interest point. Which means the definition of corner is included. If one wants to eliminate non-corner interest points from corners, then a necessarily analyze is needed. For example, a post-processing with edge detection can well detects real corners by eliminating line endings and isolated points.

Nowadays, most corner detections are still not very robust, they can not handle themselves to prevent some errors and it is still a topic for people to discuss. An good indicator that determines the quality of a corner detection algorithm is to see if it can detect the same corner under multiple circumstances, in other words, different similar pictures that had done some other image processings such as rotation, darken, etc..

The most frequently used algorithm for corner detection is proposed by Harris and Stephens, which is a further work on a method developed by Moravec and was

first published in 1988. In this thesis, the Harris corner detector is chosen and partially implemented.

3.2 The Harris and Stephens algorithm

3.2.1 Concept

In some situations, we wish to find corners, corners have an advantage over edges, edges are not localize-able, they could be anywhere on an "edge". But a corner is just a corner, could be easily localized, which makes it very useful in computer vision.

The Harris corner detector [14] is really a simple algorithm. Suppose we are looking for a corner like fig. 3.2. In the small region of the red window where the corner resides, exists a lot of gradients for both horizontal and vertical. Now, how do we find them? Or how do we figure gradients out exactly? Once if we know about the horizontal gradients I_x and the vertical gradients I_y of the image I, we can have the summation over the red window, $\sum(I_x)^2$ and $\sum(I_y)^2$. If these two summations are large, we can say we have a corner. If one of them is large but the other is small, then we likely have a corner. If it is none of the above, both of them are small, then it is not a corner at all. This whole process is called Harris corner detector

In practical, we might have a corner like fig. 3.3, which is rotated from the original corner. The horizontal gradients of an image like this is not quite as pronounced as the vertical gradients of that image. But if we rotate it back to the normal orientation, we can reduce the case to the one above. By applying this trick, a de-rotate is used. We introduce a matrix that slightly generalizes $\sum(I_x)^2$ and $\sum(I_y)^2$, then we apply eigenvalue decomposition to this matrix, we will get two eigenvalues, the evaluation that we used for summations is also used for these two eigenvalues. So we apply this

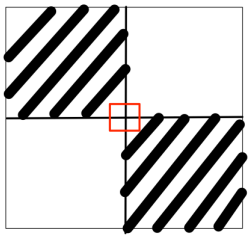

Figure 3.2: Basic Corner

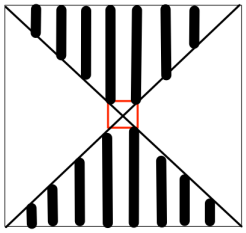

Figure 3.3: Rotated Corner

eigenvalue decomposition to every positive pixel of the image, and then take the local maximum of the result where both eigenvalues are large, we find the corners in a very robust way of exactly the Harris corner detector.

3.2.2 Mathematical Calculation

First, let us assume there is a 2-dimensional image, say it is I, then we select a path over the area (u, v) and shift it by (x, y), The weighted sum of squared differences

(SSD) between these two could be obtained by:

$$S(x,y) = \sum_u \sum_v w(u,v)(I(u+x, v+y) - I(u,v))^2 \tag{3.1}$$

where $S(x,y)$ represents the SSD, $I(u+x, v+y)$ could be approached by a Taylor expansion, .so now, let I_x and $I_y$be the partial derivatives of I, so that we can have:

$$I(u+x, v+y) \approx I(u,v) + I_x(u,v)x + I_y(u,v)y \tag{3.2}$$

By using the above approximation, we have then rewrite $S(x,y)$ as:

$$S(x,y) \approx \sum_u \sum_v w(u,v)(I_x(u,v)x + I_y(u,v)y)^2 \tag{3.3}$$

this can be reorganized in a matrix form:

$$S(x,y) \approx \begin{pmatrix} x & y \end{pmatrix} A \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.4}$$

where A is:

$$A = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{3.5}$$

this matrix A is a Harris matrix, $w(u,v)$ is a weighted window, usually a Gaussian is used here.

In order to evaluate the corners, we have to derive the Harris matrix, from equation 3.5 we know that it can be easily obtained by simply calculating the derivatives of two directions.

As we have discussed in the previous subsection, this matrix A is to be used to apply de-rotate. I_x actually consists of the horizontal derivative of every single pixel of the image, so does I_y. So after the eigenvalue decomposition of A, we can analyze all of the pixels and tell which is a corner by judging the two eigenvalues of A at each certain point. we can have three inferences based on the theory of Harris corner detector:

- If both eigenvalues are small, usually approximately equals to zero, then this specific point has nothing to do with interest point.
- If one eigenvalues is small and the other one is positively large, then it is a interest point, but it is more likely an edge instead of a corner.
- If both eigenvalues have large positive values, which indicates the change in intensity of all directions of the point is large, then a corner is found.

The way to compute the exact value of eigenvalues is very computationally expensive and complex, in which the calculation of square root is needed, so Harris and Stephens developed another approach to "measure" the eigenvalues:

$$M_c = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2 = det(A) - \alpha trace^2(A) \tag{3.6}$$

where the variable α is a constant, and the value of which is between 0.04-0.15, and it is empirically chosen. Therefore, we do not have to actually do the eigenvalue decomposition of the Harris matrix, we only evaluate the determinant and the trace of that to find corners instead, or we should say interest point in general.

If you find it difficult determining the parameter α, there is a way of avoiding it by measuring the harmonic mean of the eigenvalues, this method is proposed by Julia Alison Noble [11]

$$M_c^{'} = 2\frac{det(A)}{trace(A) + \epsilon} \tag{3.7}$$

where ϵ is a small positive constant.

Shi and Tomasi [15] then proposed a corner detector that directly calculates the $min(\lambda_1, \lambda_2)$, the smaller eigenvalues, by which it is more stable to track corners under certain assumptions. This method is sometimes known as the Kanade-Tomasi

corner detector, another approach of finding corners besides Harris and Stephens corner detector.

Chapter 4

Hardware

4.1 Architecture of FPGAs

FPGAs are the members of the programmable logic device family, it can be treated as a array consists of configurable logic block (CLB). These CLBs are connected by a programmable network, together they are controlled by memory cell, which makes FPGAs satisfy different specific needs conveniently. Until now, the structure of FPGAs are based on the following techniques: Flash, EPROM, SRAM and anti-fuse.

The most widely used FPGAs are based on SRAM and the major suppliers of which are ALTERA and XILINX. The mainstream of FPGAs have a fabrication of 65nm, the highest clock frequency has reached 500MHz, meanwhile, such variables of FPGAs like frequency and logical gate are still developing rapidly.

A FPGA is a array consists of CLB, the biggest CLB so far has 192 rows and 116 columns, programmable I/Os form the outta frame, sometimes a high-level FPGA could have more than thousands of I/Os. Together with the interconnections they are the inner structure of FPGAs, Fig. 4.1 showns the CLB and a example of user logic. We

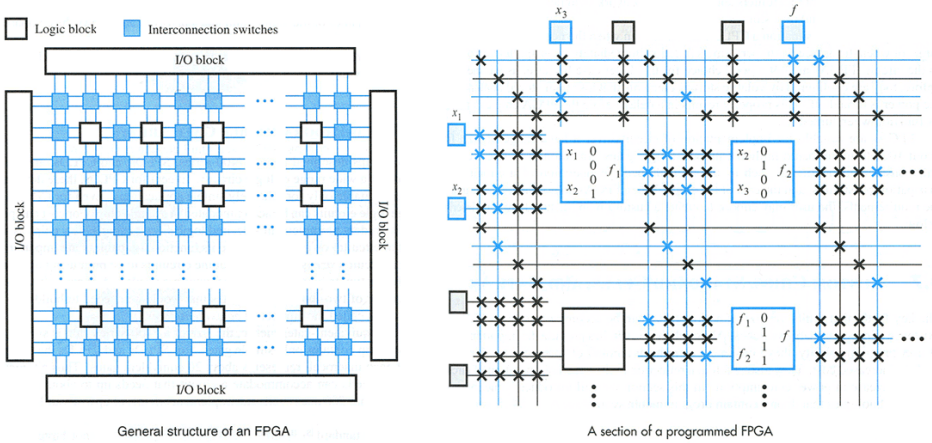

Figure 4.1: Internal structure of FPGA

have four I/Os activated, which are x_1, x_2, x_3 and f, three logic blocks where inputs and outputs are clearly indicated. The blue lines and crosses mean connected and black ones mean disconnected. Take the first block for example, x_1 and x_2 are connected to the block by using different lines and output f_1 is ported to another block as an input.

In order to improve the capacity of FPGAs, some other specific modules are integrated onto FPGA besides the traditional resources. Such as RAM, DSP, embedded hard core processor (PowerPC or ARM) and soft core processor (Nios or Microblaze). It does not mean to replace the structure of traditional FGPAs but to complement the capacity of FPGAs and to give more functions and options that can be applied to programs that are becoming more and more complex.

Each CLB can have 2 or 4 or even more logic elements (LE), which is the basic unit of FPGAs, Fig. 4.2 demonstrates the structure of LE. A LE includes a 4-bit look up table (LUT), it can be allocated as a 16-bit RAM or ROM, or as a combined logic which outputs combinatorial output. There is a carry path involved so that the LE can do relative calculations. The last part of IE is a D flip-flop with its enable, reset and clock signals, this register can store the output of LE. Because the output of LE could

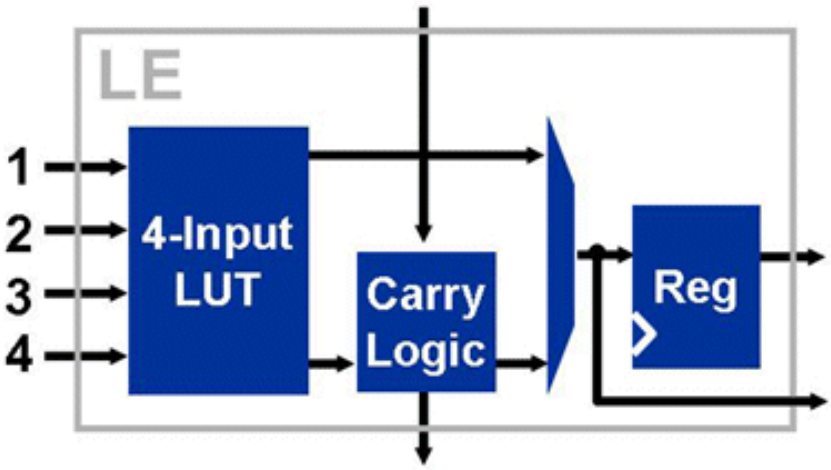

Figure 4.2: Block diagram of LE

be set as the input of another LE, so that the LE can also be considered to be a micro state-machine.

The standard development of FPGA programming is: logic design, simulation, layout, synthesis, setups and debug. three out of the six procedures above are proceeded by the tools provided by manufacturers, which are simulation, synthesis and debug. the capacity of synthesis tool directly decides that of the whole design. So in order to obtain a better result of synthesis, one must fully consider the specialism of the tool and design the project according to the structure of the chip is used.

At the very beginning of the history of FPGAs, developments are quite simple, so the logic designs are usually done by using the develop tool of schematics. In the late 80s, the complexity and scale of design are both increasing so that the designs which are based on schematics started to be eliminated. because of the development of EDA technology, HDL designs become more and more popular and now is also applied to FPGAs. IEEE standard makes the HDL based CAD tools widely used in the field of microelectronics, programmers can design their own circuits in layering or modularity

freely. Such programming methods are "from top to bottom" and can be categorized as follow:

(1) system layer description, describing the design specifications of the entire system.

(2) Behavior layer description, describing the mathematical model of the system.

(3) RTL layer description, using the basic units of the system to characterize the mathematical model.

(4) physical layer, using the simple gate circuits to represent the system.

In general, the design of the system is usually worked on the behavior and RTL layers, one can choose a certain layer to work on according to the complexity of the algorithm or the familiarity to the synthesis tool. The significant point of using HDL to describe the circuit is to fully understand the relationship between the hardware design language and the hardware circuit. One should have a basic concept of the circuit that is to be synthesized when writing the code.

4.2 Design of logic blocks

Designing hardware is somehow pretty much like programming software in assembly language, in which a simple calculation consists of lots of machine code instructions and on the other side, many logic blocks form a single hardware module.

As mentioned before, Fig. 2.2 shows the basic concept of how the blocks of FPGA work, The rest of this chapter will go through all of the logic blocks that are introduced to reconfigure the partial algorithm.

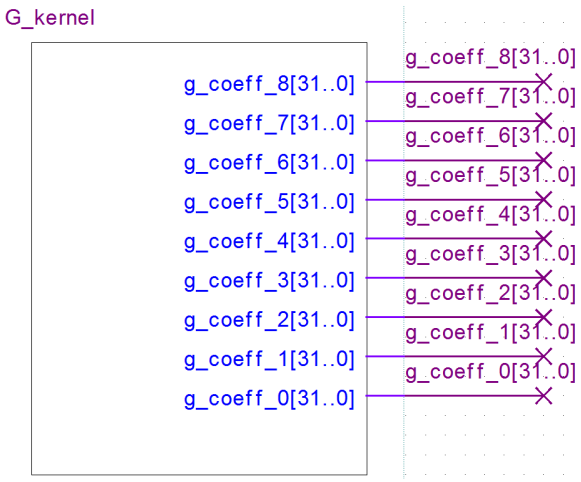

Figure 4.3: Gaussian Kernel

4.2.1 Gaussian Kernel

Fig. 4.3 shows the Gaussian kernel, which is used to obtain the Harris matrix A we discussed in ch. 3.2.2. This block is the simplest one in the entire project, it requires no inputs and output the kernel constant variables. Each output corresponds one element of the matrix, coefficient 8 represents very first element 0.0001, coefficient 5 is the center element 0.3989, and so on. Note that every parameter is a float number and it acquires 32 bits memory to be saved, so outputs are all designed to be 32 bits wide.

0.0001	0.0044	0.0540
0.2420	0.3989	0.2420
0.0540	0.0044	0.0001

Table 4.1: The Harris matrix

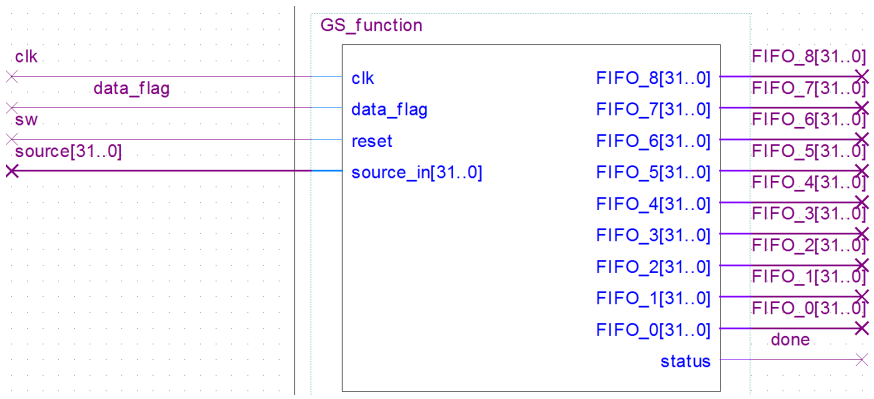

Figure 4.4: input FIFO

Signal name	Description
clk	The clock signal
data_flag	Flips when new data is input
sw	A switch that controls the reset function
source[31..0]	Input pixels, which is a 32-bit float type number
FIFO_ x[31..0]	The No.xth output pixel, paired with the corresponding element of Harris matrix in the further stages
done	Set to high when FIFO is full, otherwise, remains low

Table 4.2: Signals of Input FIFO

4.2.2 Input FIFO

In order to maximize the speed, we have to make the hardware as parallel as possible. The FIFO IPs that Altera provided are all "single input and single output" mechanism, which is not quite what we want, so a "single input nine outputs" FIFO is implemented to meet the need that we can run in one pass a whole convolution with the 3*3 Gaussian matrix.

Fig. 4.5 shows how this logic block proceeds. Initially, all of the 9 outputs are set to zero, and data are sent in through one to nine. When the 10th data is sent in, all of the outputs are given variables in a particular order, oldest is on the bottom and newest is on to top, for this FIFO, No.8 is the top and No.0 is the bottom. When the

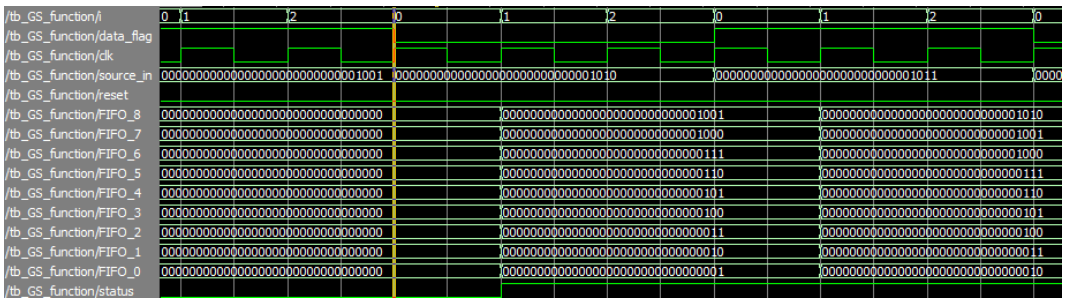

Figure 4.5: Simulation of the GS_function block

11th data(1010) is received after three clock cycles, then in the next clock cycle, the oldest data(0001) is dropped and the 10th data is added to the FIFO and stored in the No.8 register, and so on. Note that whenever a new data is received, the data_flag signal is flipped, this mechanism secured the system would not mess up two consecutive data that have the exact the same value. Also, by saving data in the logic, the problem that the wasting of resources is solved. each single pixel is used many times in convolutions of pixels by it's side, so in software the variable has to be called multiple times but in hardware it will be called and sent into the FIFO only once, it lasts until it finishes all of its related works.

4.2.3 Addition and Multiplication

The convolution involves both the FIFO and the Gaussian kernel, each output of these two blocks will be paired up into nine different groups. Eventually they will all be added up into a final result. Because the algorithm in this project uses 32-bit float type number to represent pixels, normal calculation blocks can not be applied here due to the complexity of float value. Altera provides the mega-function that fully supports the float value calculations makes it the perfect choice in this project.

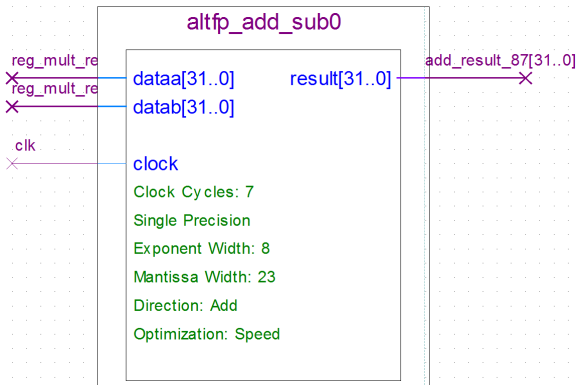

Figure 4.6: The adder

Fig 4.6 is the addition block after setting up the mega-function wizard. The optimization of speed is selected because which is the main gold of the thesis, any function and ability except pure adding are disabled in order to save resources(Fig 4.7), such block is to be duplicated many times to form a functional algorithm so one must consider carefully about what is needed and what is not. 32-bit float type number means it is single precision and has 8-bit exponent and 23-bit mantissa. The purpose of the ports are pretty clear and a global clk signal will be connected to the adder.

Fig 4.8 is the simulation of the mega-function of the adder, which obtains the result right after 6 clock cycles since the sources are inputted in clock "0", the total proceeding period is exactly the same as the text indicated on the block, which is 7 clock cycles.

The multiplication module has the same procedure of setting up and ports, except the function of it is multiply instead and takes only 5 clock cycles to calculate the result of two inputs. Some specific library has to be imported when using the simulation tool to simulate such given logic blocks, for the above two blocks, the "lpm_ver" library must be included in the executing command, otherwise, unexpected errors will occur.

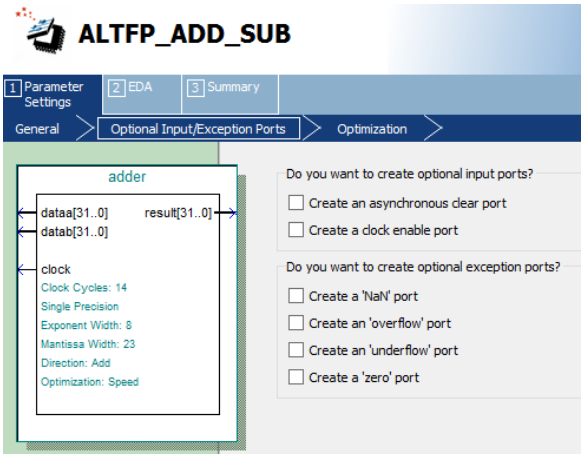

Figure 4.7: Optional ports of the adder

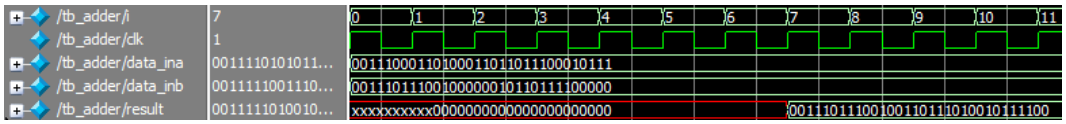

Figure 4.8: The simulation of the adder

4.2.4 State machine

For a single convolution of two 3*3 matrices, nine multiplications and eight adders are implemented. Each one of the block has a proceeding period and the latter calculation can only executes when the former has done its work. For the robustness of this entire process, a state machine is introduced. Fig. 4.9 is the scheme of the state machine. If the done signal is "high", then the arbiter is triggered, it starts counting inside and output a value to spliter, the spliter uses this value to decide what the state now is and to set the corresponding output to "high" while setting others to "low".

Fig. 4.10 is the simulation of the arbiter, when done signal is set to "high", the arbiter starts to count. The output of the arbiter is just numbers in sequence, each number represents a state in the spliter, and the spliter tells which register is in active.

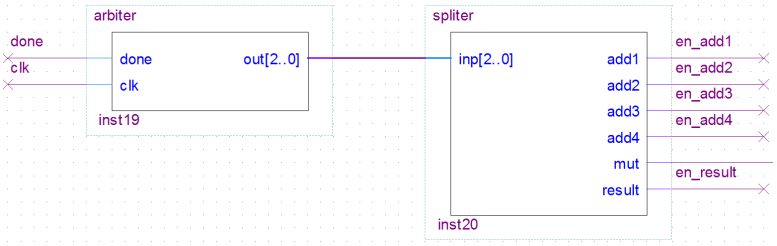

Figure 4.9: The State machine

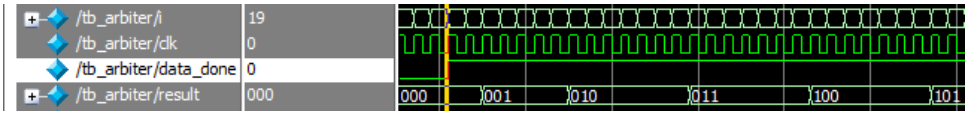

Figure 4.10: The simulation of the arbiter

4.2.4.1 Adder register

There are four addition registers and a result register that play an intermediary role between different stages of the convolution. Take Fig. 4.11, the add1_register for example, inputs are connected to the former stage of the calculation, no matter what the values are, the register will always store what it receives, once the enable signal is obtained "positive edge", output ports will immediately output the data that it currently stored and remain the output until the next "positive edge". The latter stage of the calculation, which would be add2, uses these outputs as the input sources, and gives the result to add2_register, and the process keeps doing the same work to add3 and add4.

4.2.4.2 Result register

Fig. 4.12 is the result register, not only does the result register store and send data, but also it has other functions to keep the system robust. Table. 4.3 describes the functions of the additional ports of result register comparing to the adder registers.

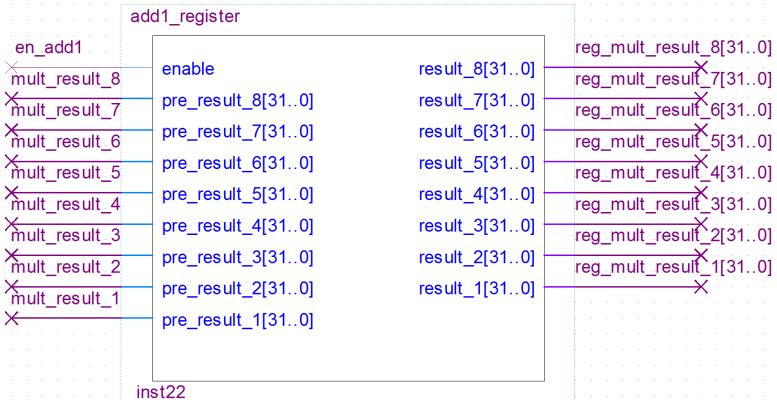

Figure 4.11: The adder register

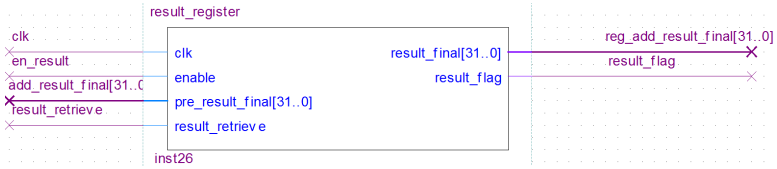

Figure 4.12: The result register

Signal name	Description
result_retrieve	A signal comes from the CPU that is set to "high" when result is being retrieved and reset to "low" when the retrieval has done.
result_flag	Set to "high" when the enable signal is "positive edge", indicates that the new result is eligible to retrieve. Set to "low" when the result_retrieve signal is "negative edge".

Table 4.3: The signal of result register

The reason that the above two signals are implemented is because, unlike the calculation portion, there exists no interruptions or duplicates between stages, former result would never change if latter had not taken it. But to the result register, it is a different situation. The result register communicates with the CPU, which is a completely separated system. The CPU could have requested for result multiple times even though there is only one result has been calculated. because the difference of the

frequency between CPUs and FPGAs, CPUs are always in GHz level and FPGAs are in MHz, which is quite slow comparing to that of CPUs. So there must be a way to secure each request applies to a new data.

The flag signal tells the CPU that whether or not the current data is eligible and safe to retrieve and it is always the first signal that is going to be interacted in a communication process between the CPU and the FPGA, once CPU received the "high" flag signal, it would start to retrieve the data, meanwhile, a result_retrieve would be sent to FPGA indicates that the process has begun. After the result is obtained, CPU sets back to "low" the result_retrieve signal, tells the FPGA that the process has done. Then the FPGA also sets the result_flag signal to "low" so that by letting CPU receives the flag signal first, it can keep CPU to wait until a new result is derived. Such mechanism can well improve the robustness of the system.

4.2.5 PCIe system

Besides the partial reconfiguration, the FPGA has to communicate with the "outside world", which in this case is the CPU.

Fig. 4.13 shows all the interconnections between different modules. The only achievable communication is the PCIe bus that between the Atom processor and the FPGA chip. By using the development tool Qsys [7], the PCIe system can be easily generated. Fig. 4.14 shows the Qsys designing framework, hardware controllers are connected with each other through the Avalon Memory-Mapped interface, so master controllers can easily access slave controller in a memory-mapped manner. The host can directly control user logic through the PCI Express Avalon MM master port bar1_0. PCI Express bar2 Avalon MM master is provided for host to configure the SG-DMA

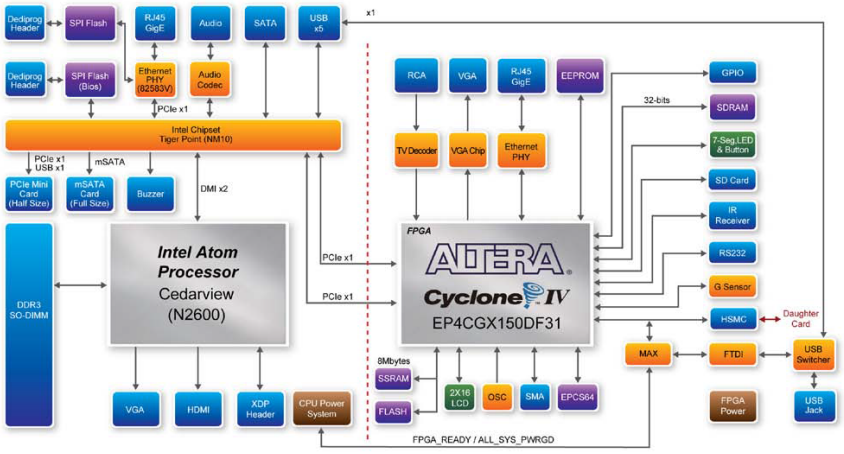

Figure 4.13: The Block diagram of DE2i-150

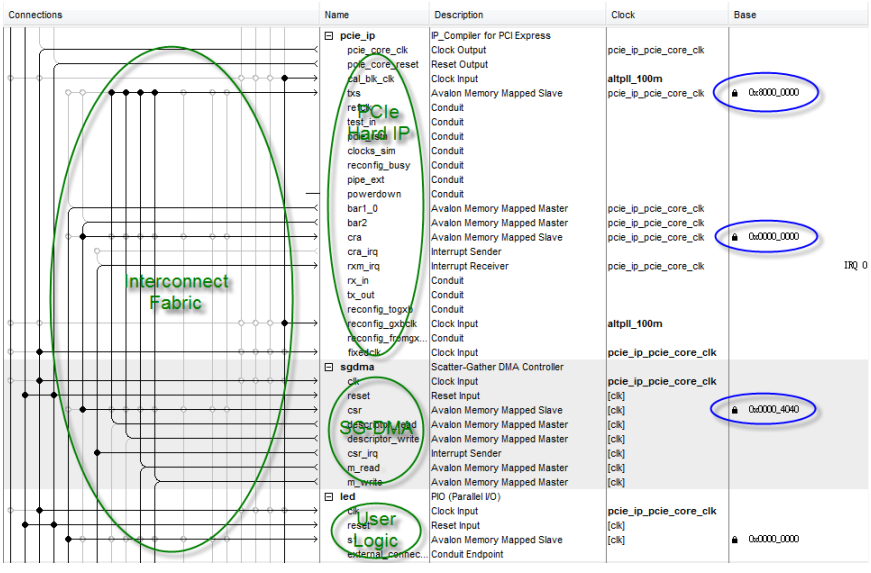

Figure 4.14: PCIe Framework based on Altera Qsys

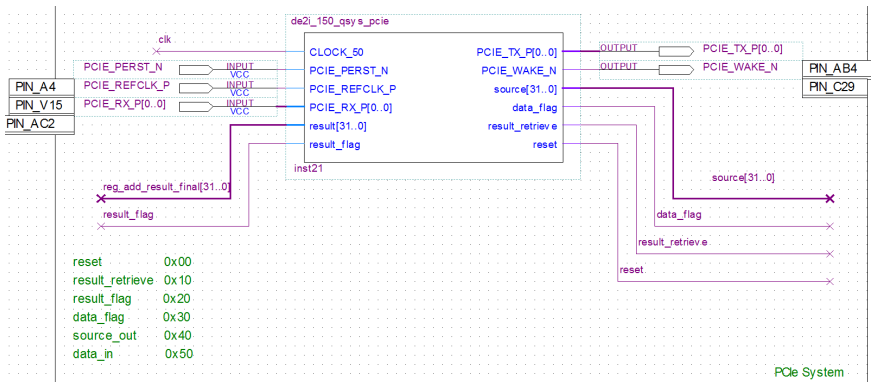

Figure 4.15: The PCIe system

engine. The SGDMA can perform data transmission between any two Avalon MM slave controllers.

Signal name	Description
reg_add_result_final[31..0]	The data that is obtained from the result register.
result_flag	The flag signal of result register
source[31..0]	Float type pxiel data
data_flag	Flips when new source comes in
result_retrieve	"high" is retrieving data and "low" means retrieval has done
reset	Reset the FIFO

Table 4.4: The signal of PCIe system

Fig. 4.15 shows the final generated PCIe system including all of the necessary user logic, the PCIe can be considered as the CPU, data comes in through input ports go directly to the CPU and data send by CPU goes into FPGA through the output ports of the PCIe system. The major problem of the entire thesis project is encountered here, the PCIe bus is only capable of sending 32-bit data at a time, which means only one pixel can be sent to FPGA per command, because the pixels in the algorithm are float type. In addition, to execute a send-to-FPGA or a retrieve-from-FPGA command

also spends a lot more extra time, with that being said, the interaction process vastly slowed the speed of the project and which so far is unfeasible to improve or avoid.

Chapter 5

Result

Because of the weakness of the PCIe system, it is impossible to make a bi-system such as the one in this project run faster than GHz level CPU, so the main goal of the testing is the find out the partial algorithm run-time on both CPU and FPGA and then compare with each other.

Fig. 5.1 shows the original picture with a size of 1330*1110 and the edited one. The result is pretty acceptable.

The table 5.1 shows the result of the run-time for both platform, please note that this timing is the partial algorithm run-time only, the communication between CPU and FPGA is excluded, it will take up to minutes for the signal-pixel in-and-out interconnection. The CPU result is obtained by using GProf, and the result of FPGA is observed from a clock timer through the simulation. The design of the partial reconfiguration is actually an IP, so although the speed is somehow slightly slower than the original algorithm, it is just the result by applying one block, the speed can be accelerated by simply copy and paste more IPs into the design, just like how the convolution is done by using duplicated adders and multiplications. Also, in order to have a direct

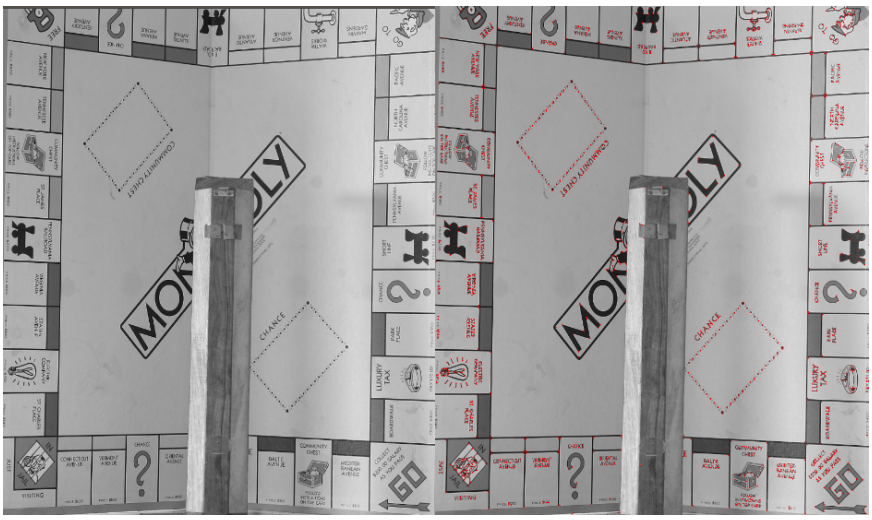

Figure 5.1: The picture before and after corner detection

comprehension about how the parallelism and hardware take advantage in algorithms, there is a assumption in the table, we can see that the run-time is significantly shortened more than 20 times without any optimization.

Platform	Run-time	Result
Atom N2600 CPU 1.6Ghz	171.82 ms per call, 6 calls in total	1.030s
DE2i-150 FPGA 50MHz	28 clock cycles per output, 826.7 ms per call, 2 calls in total	1.653s
FPGA optimal implementation (theoretically)	826.7/x ms per call, after dividing the input picture into x parts, 2 calls in total	1.653/x s
FPGA implementation in 1.6GHz	25.84 ms per call, 2 calls in total	0.052s

Table 5.1: The run-time comparation

Chapter 6

Conclusions

This thesis discussed the implementation of the partial algorithm of corner detector, a complete result is presented, which shows the capability of supporting corner detection for real-time image processing. we reconfigured a functional IP that can be used not only for corner detection, but also for other processes that involves convolution.

The computation increases as the complexity of algorithm does, FPGAs are good alternatives, we have shown that the computation-intensive and repetitive functions can be off-loaded onto FPGA and it can be used as a co-processor.

The work in this thesis also points out the need for a development of the architecture of FPGAs, if in the future manufacturers should improve even a little bit of the frequency of FPGAs could make a huge acceleration and decrease the price of which will make it more popular to be accepted for customers. Also the way of interconnection between the CPU and the FPGA significantly decides the fate of practical use. I believe that in the future a high-speed bus line will be introduced so that the FPGAs can be a very useful plug-in or co-processor to the CPUs, just like how we use modules in designing hardware, the FPGA could as well be designed to be a specific module that is

to expend the performance and the flexibility of a PC in many fields. This project can be continued in two directions:

- Implementing more portions of the algorithm, as shown in Fig. 2.1, the non-suppression and the derivative can also be reconfigured onto FPGA. Although there is no repetition of these two portions that can be optimized into one or two passes, it can still benefit from parallelism by calculating horizon and vertical derivative at the time.

- Change to another FPGA board so that there might have some other way to achieve a better interconnection between CPU and FPGA, for example, if both chips are capable of accessing a same RAM, then the pixels of a picture can be sent and retrieved in a very high speed level, additionally, a vast amount of data transmitting per command could also be feasible.

Bibliography

[1] A. Prati A. Benedetti and N. Scarabottolo. Image convolution on fpgas: The implementation of a multi-fpga fifo structure. In *Proc. Euromicro Conf.*, pages 123–130, 1998.

[2] Altera. de2i-150. `http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=529`.

[3] Altera. Fpgas. `http://www.altera.com/devices/fpga/fpga-index.html`.

[4] D. Bailey. Adapting algorithms for hardware implementation. In *7th IEEE Workshop on Embedded Computer Vision*, pages 177–184, 2011.

[5] J. Rausch C. Claus, R. Huitl and W. Stechele. Optimizing the susan corner detection algorithm for a high speed fpga implementation. In *Field Programmable Logic and Applications, FPL 2009, Int. Conf.*, pages 138–145, 2009.

[6] T. Kanade C. Tomasi. Detection and tracking of point features. *Pattern Recognition 37*, pages 165–168, 2004.

[7] Altera Corporation. Sopc builder user guide. `http://www.altera.com/literature/ug/ug_sopc_builder.pdf`, 2010.

[8] M. Xia H. Zhang and G. Hu. A multiwindow partial buffering scheme for fpga-based 2-d convolvers. In *IEEE Trans. Circuits Syst.*, volume 54, pages 200–204, 2007.

[9] T. K. Priya K. Sridharan. The design of a hardware accelerator for real-time complete visibility graph construction and efficient fpga implementation. *IEEE Trans. Ind. Electron.*, 52(4):1185–1187, 2005.

[10] W. J. MacLean. An evaluation of the suitability of fpgas for embedded vision systems. In *1st IEEE Workshop on Embedded Computer Vision*, 2005.

[11] A. Noble. *Descriptions of Image Surfaces*. PhD thesis, Department of Engineering Science, Oxford University, 1989.

[12] M. J. Jones P. Viola. Robust real-time face detection. *Int. J. Computer Vision 57*, pages 137–154, 2004.

[13] P. Corsonello S. Perri, M. Lanuzza and G. Cocorullo. Simd 2-d convolver for fast fpga-based image and video processors. In *Proc. MAPLD*, page D2, 2003.

[14] C. Harris; M. Stephens. A combined corner and edge detection. In *Proceedings of The Fourth Alvey Vision Conference*, pages 147–151, 1988.

[15] J. Shi; C. Tomasi. Good features to track. In *9th IEEE Conference on Computer Vision and Pattern Recognition*, 1994.