2011

FPGA Implementation of Blob Recognition

Jian Xiong
University of Windsor

FPGA Implementation of Blob Recognition

by

Jian Xiong

A Thesis
Submitted to the Faculty of Graduate Studies
through Electrical and Computer Engineering
in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada

2010

FPGA Implementation of Blob Recognition

by

Jian Xiong

APPROVED BY:

Dr. Xueyuan Nie
Department of Mechanical, Automotive and Materials Engineering

Dr. Narayan Kar
Department of Electrical and Computer Engineering

Dr. Jonathan Wu, Advisor
Department of Electrical and Computer Engineering

Dr. R. Rashidzadeh, Chair of Defense
Department of Electrical and Computer Engineering

November 26, 2010

DECLARATION OF CO-AUTHORSHIP/PREVIOUS PUBLICATIONS

0.1 CO- Authorship Declaration

I hereby declare that this thesis incorporates material that is result of joint research as follows:

This thesis also incorporates the outcome of a joint research undertaken in collaboration with Mr. Thanh Nguyen under the supervision of Professor Jonathan Wu. The collaboration is covered in Chapter 3 and 4 of the thesis as the development of the PC based software algorithm for this project has been done by Mr. Thanh Nguyen. In all rest of the cases, the key ideas, primary contributions, experimental designs, data analysis and interpretation, were performed by the author, and the contribution of co-authors was primarily through helping in obtaining experimental results presented in section 4.3.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-author(s) to include the above material(s) in my thesis.

I certify that, with the above qualification, this thesis, and the research to which it refers, is the product of my own work.

0.2 Declaration or Previous Publication

This thesis borrows some of its material from one original paper that has been previously published as follows:

1. Jian Xiong, Q.M. Jonathan Wu, "An Investigation of FPGA Implementation for Image Processing", ICCCAS.2010

I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as graduate student at the University of Windsor.

I declare that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyright material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis. I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University of Institution.

ABSTRACT

Real-time embedded vision systems can be used in a wide range of applications and therefore the demand has been increasing for them.

In this thesis, an FPGA-based embedded vision system capable of recognizing objects in real time is presented. The proposed system architecture consists of multiple Intellectual Properties (IPs), which are used as a set of complex instructions by an integrated 32-bit CPU Microblaze. Each IP is tailored specifically to meet the needs of the application and at the same time to consume the minimum FPGA logic resources. Integrating both hardware and software on a single FPGA chip, this system can achieve the real-time performance of full VGA video processing at 32 frames per second (fps). In addition, this work comes up with a new method called Dual Connected Component Labelling (DCCL) suitable for FPGA implementation.

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my supervisor Dr. Q.M. Jonathan Wu, who not only provided me with a good experimental environment and sophisticated experimental setup, but also encouraged me with his great kindness and patience. All of this support enabled me to finish this thesis successfully. In addition, I would like to thank my co-worker Mr. Thanh for discussing and sharing his ideas for algorithms. Without his support, it would have been hard for me to realize this project on time.

This work is funded in part by The Auto21 Network of Centers of Excellence of Canada Research Chair program, and the NSERC Discovery grant.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

1.1 Problem Statement

In computer vision, object recognition is to identify a given object in an image or a series of images (video), for the purpose of extracting some explicit information that is to be used in subsequent analysis or further operations.

Image recognition is still an active and challenging research area in general, especially for handling unconstrained environments, and usually incorporates a variety of steps. According to [50], there are no shortcuts and the following six steps must be noticed: image formation, conditioning, labeling, grouping, extracting and matching. Image formation is a function of multiple variables including the camera sensor, lens, illumination and the surface reflectivity condition, etc. The remaining five steps constitute a canonical decomposition of the recognition problem, each step preparing and transforming the data in the right way for the next step.

Simply focusing on the image analysis problem and viewing it from another angle, we can divide it into two steps: location and identification [15]. Location is pinpointing the position of the expected object that is usually unknown in an image. Based on some certain properties such as intensity, color, texture, etc, the image under detection is segmented into separate regions, and some of which are selected and named candidates for further analysis. Following the candidates selection, an identification function analyzes what the object is inside the candidate region.

With the advance of computer industry, one of the tremendous changes of computer vision system is the shift of processing platform from conventional desktop computers or powerful workstations to embedded processors [56]. This is a shift representing the trend of computer technology, and is also a natural shift catering to the market needs. As we already witnessed over the past decades, embedded vision technologies have emerged in a wide variety of important applications from industry to commerce and from civilian to military.

Typically an embedded CPU (16-bit or 32-bit processor) or a Digital Signal Processor (DSP) is used as the system controller and algorithm processor in embedded vision system. The Field Programmable Gate Array (FPGA) has also been adopted in the last decade, but is often only limited to glue logic for interfacing various electronic devices on board, while the complicated image processing algorithms are implemented on a DSP or an embedded CPU. This is because in part that FPGA development is a trial and error process, presents many challenges, and requires designers to simultaneously cope with both high level (algorithm and system architecture, etc,) and low level (logic circuit, memory management, time domain, etc.) design [1]. For many designers, especially those under the heavy pressure of time-to-market, it is unaffordable to develop a pure FPGA-based system.

However, FPGAs have some unique features [2, 4], which make itself stand out from other processors: real hardware parallel processing capabilities enables FPGA technology to have higher data throughput than MCU or DSP; Reconfigurability makes it possible to update internal logic, which is far more flexible than custom designed ASIC;

Abundant logic and I/O resources make an FPGA the perfect platform for developing System on Chip (SOC).

SOC means that all components of an electronic system including software and hardware are integrated into a single chip. It is a hot topic in embedded system researches because it is compact in size and highly integrated, and thereby can not only reduce the board BOM (Bill of Material) but also can enhance system performance within small size scale. Thanks to the tremendous improvement in FPGA technology over the last decade, for example higher chip density, smaller package, more special features and better development suit, it certainly became one of the most promising platform for SOC development.

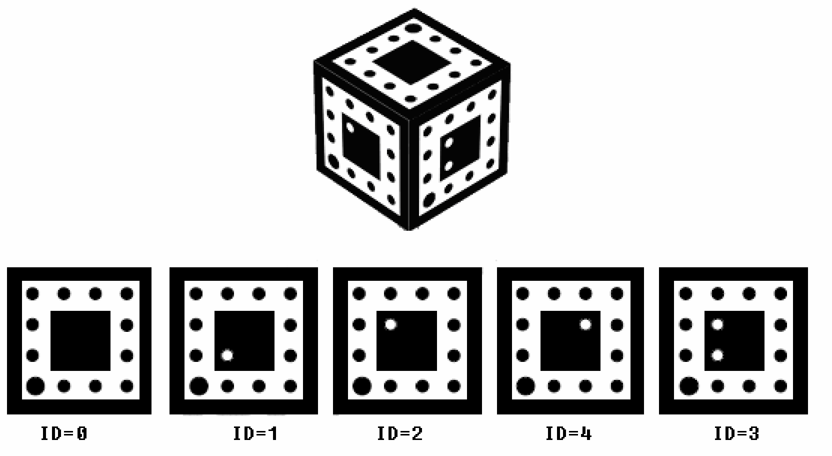

Figure 1: Target under Detection

The primary goal of this thesis is to develop an FPGA based embedded vision system for recognizing in real time the certain blob pattern shown in figure 1. As an example, this vision system can be used to track and monitor the process of assembling nuts on an engine automatically. As we can see from the application scenario shown in figure 2, the target is assembled on a torque gun and is rotating when the torque gun is fastening or unfastening the nuts. By identifying the location and orientation of the target, it is possible to make sure that every nut is mounted correctly. Hence this vision system can assist by increasing nut assembly accuracy, reducing rework, and thereby increase the productivity and profitability of the automotive manufacturing process.

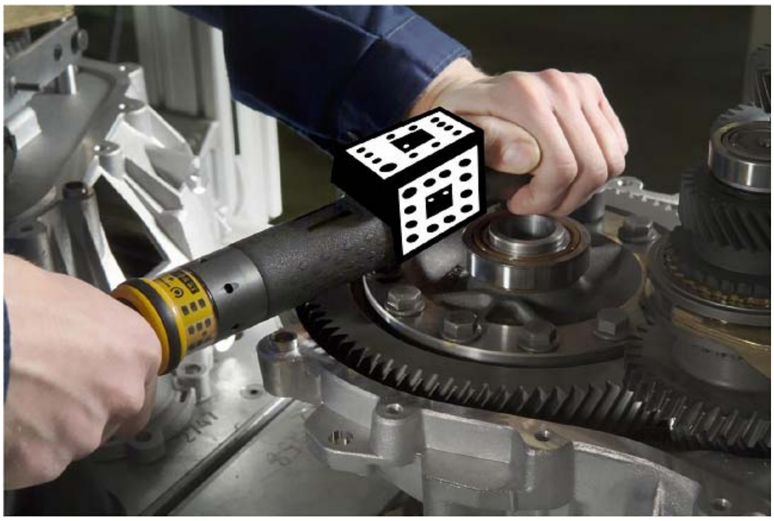

Figure 2: Application Scenario

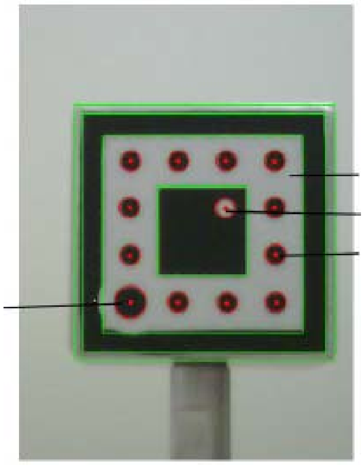

Figure 3: Blob Pattern

As we can see, there are five faces on a target. Each of them has a black square in the center named the heart block. The heart block contains up to 4 white dots, and is also surrounded by 12 smaller black dots. The largest black dot is defined as the origin of the face. The ID of each face is determined by the relative position of the origin and the white dots.

The main focus of this project has been designing hardware architecture and porting the PC based algorithm to the FPGA system, as well as developing the reusable Intellectual Property (IP).

1.2 Thesis Organization

The following content is organized as follows. The second chapter provides a review of the FPGA implementation for image processing, analyzes the features of image processing algorithms, and explains the advantages as well as the challenges of FPGA implementation for image recognition. In addition, the general architecture of FPGA implementation for image identification is developed. Furthermore, the recent advances and state-of-art FPGA technologies, including embedded CPU integrated with FPGA and the advanced tools for developing the FPGA system, are introduced. The final part of the second chapter also describes some existing FPGA-based embedded vision system.

In chapter three, the FPGA based embedded vision system developed in this thesis is explained. Its hardware architectures, including board level architecture, IP level architecture as well as circuit level architecture are described first. Here, the architecture design is focusing on reducing the silicon area and occupying the least resources. Several reusable IPs are designed and reused in the whole architecture. The usage of each reusable IP will be explained in depth. Next the proposed recognition algorithm is explained in detail from the view point of software flow. In the subsequent sections, the related software design is described, including a Windows GUI application and the firmware running on an embedded CPU (called Microblaze) integrated in FPGA

In the fourth chapter, the experiment results are presented and the related analysis is explained. During the whole design process, multiple design options are chosen and then implemented. Their advantages and disadvantages will be addressed and the experiment results will be explained in detail.

Finally, the fifth chapter presents the conclusion of this thesis as well as the suggestions for the future work of this FPGA-based embedded vision system.

2 REVIEW OF LITERATURE

2.1 Overview of FPGA Implementation for Image Recognition

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

In order to enhance real-time implementation, two aspects of implementation are endeavoured; 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 traditionally 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 Giga Hz 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, and 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 improve-ments 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.

2.2 Algorithm Feature and FPGA Implementation Challenges

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) Neighbouring operation.

The so-called Neighbouring 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 neighbour 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 neighbouring 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 neighbouring 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. During algorithm development, floating-point numbers are often used because they represent infinite precision. If the algorithm is to be implemented in hardware, floating-point numbers are not always feasible. The solution is to convert very precise floating-point numbers to less precise fixed-point numbers. This process in Matlab is called quantization. Quantization is an iterative process, and requires comparing the results of floating-point and fixed-point process, so becomes the most difficult step for designers. As for available operators, during algorithm development, there is no limitation on the type of operations. But, it is not true for FPGA development. Only a limited set of operations can be synthesized, such as addition and deduction. Arbitrary division and multiplication should be avoided as long as it is possible for the purpose of saving resources and reducing circuit size.

2.3 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.

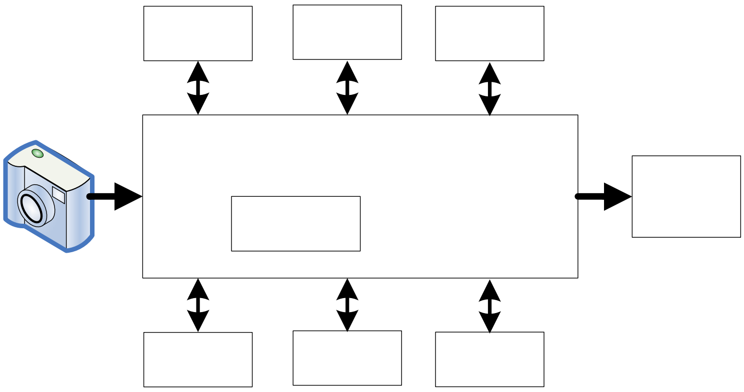

Figure 4: General Architecture of FPGA Implementation for Image Processing

This figure 4 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 compli-cated 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.

Here it should be pointed out that it is possible for multiple circuit blocks to access one single external memory in a Time Division Multiplexing (TDM) way, which could get an effect of near-parallel processing. But there are two problems: 1) a memory controller that can handle multi-port access is needed. This demand will increase the complexity of FPGA design. Fortunately, current FPGA vender already began to provide this kind of controller for free, such as the MPMC (Multi-Port Memory Controller) provided by Xlinx. 2) But another problem cannot be avoided: memory throughput. With the increase of the problem scale, the throughput of memory will become a bottleneck for the whole system. Unlike average CPU or DSP, FPGA is a parallel processor by nature. It is better to use multiple memories so that the potential parallel processing capability of FPGA could be fully exploited.

2.4 FPGA Based Embedded Processor

In recent years, embedded processors are provided by many FPGA venders in the form of hard IP existing in silicon fabric or in the form of soft IP which can be incorporated within FPGA. Figure 4 also shows an embedded CPU inside an FPGA. The coexistence of an embedded processor and traditional digital logic fabric is to grant the flexibility of incorporating both software and hardware in one chip. In this way, the contradiction between the challenge of FPGA implementation and the system performance can be compromised since different algorithm fit different platform. The algorithms suit parallel application can be realized in FPGA logic fabric, while the algorithm suit serial processor can be realized in embedded CPU inside the FPGA.

The partition of hardware and software can be determined by two factors. One is the required update time of processing. For example, if one part of a process needs microsecond or millisecond update time, then it can be processed by software. But if one part of a process requires 10-100 μs update time, then hardware logic must be exploited instead of software. The other factor influencing our choice of choosing hardware or software is the feature of algorithm itself. Only when the algorithm has a large portion suitable for parallelization, the potential speedup can be achieved by employing FPGA. Therefore it is necessary to understand every part of the algorithm and make an informed division between software and hardware.

Table 1 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 1: 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

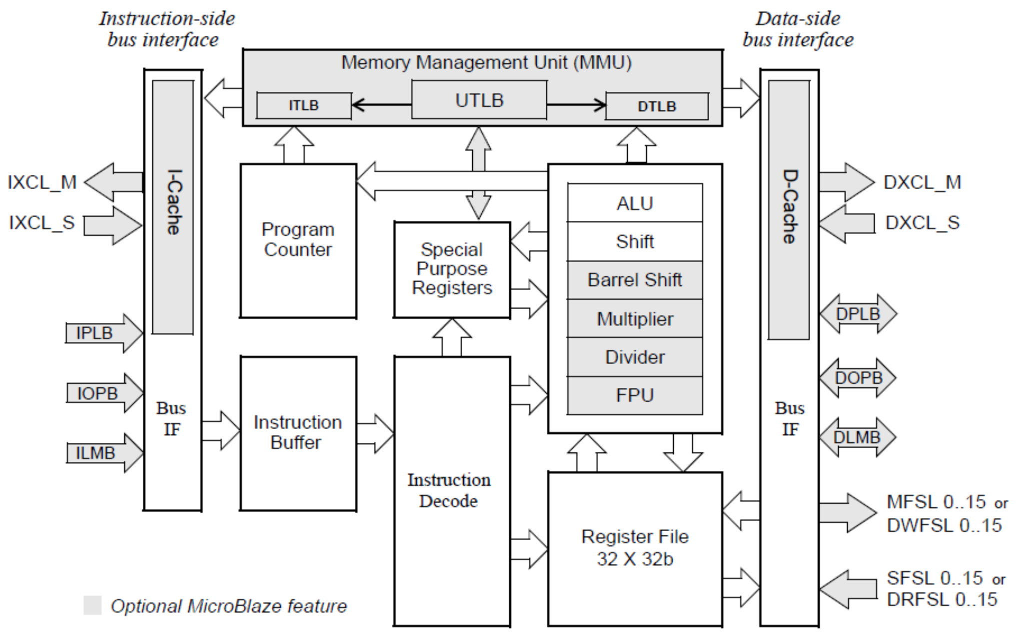

Figure 5: Functional Block Diagram of MicroBlaze Core

In this project, a MicroBlaze embedded processor soft core is adopted. It is a 32-bit Reduced Instruction Set Computer (RISC) optimized in Xilinx FPGA. As a kind of soft IP, it can be synthesized and incorporated into some particular Xilinx FPGA by using general logic fabric resources. Figure 5 shows a functional block diagram of the MicroBlaze core.

The Microblaze is a reconfigurable soft core, containing both fixed and configurable features. Its fixed features include:

- ✓ Thirty-two 32-bit general purpose registers
- ✓ 32-bit instruction word with three operands and two addressing modes
- ✓ 32-bit address bus
- ✓ Single issue pipeline

There are dozens of configurable features including optional on-chip bus, optional data and instruction cache to name a few. This flexible reconfiguration enables the users to choose their desired set of features according to their design requirements.

2.5 IP Designed with System-Level Tools

One of the goals of the 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 so far.

Table 2: System-Level Algorithm Mapping Tools

<table>
<tr><th></th><th>Tool Name</th><th>Tools Developer</th></tr>
<tr><td rowspan="4">Matlab based</td><td>System Generator for DSP</td><td>Xilinx</td></tr>
<tr><td>AccelDSP</td><td>Xilinx</td></tr>
<tr><td>Simulink HDL Coder</td><td>Mathworks</td></tr>
<tr><td>Synplify DSP</td><td>Synplicity</td></tr>
<tr><td rowspan="4">C/C++ based</td><td>System-C</td><td>OSCI</td></tr>
<tr><td>Catapult-C</td><td>Mentor Graphics</td></tr>
<tr><td>Impulse-C</td><td>Impulse Accelerated Technologies</td></tr>
<tr><td>Mitrion-C</td><td>Mitrionics</td></tr>
</table>

	DIME-C	Nallatech
	Handel-C	Celoxica
	Carte	SRC Computers
	Streams-C	Los Alamos National Laboratory

C/C++ based tools are developed for designing the hardware and the software simultaneously, and for easing the creation of test benches. Based on this tool a software programmer are not supposed to understand the hardware in depth, and thereby expatiating the design process, increasing productivity. To take Impulse-C as an example, it uses the communicating process programming model to develop highly parallel, mixed hardware/software algorithms and applications. But as stated in [57], C/C++ language programming is not a replacement for the existing hardware description languages such as Verilog HDL and VHDL. It can be used to describe a wide variety of functions that fit the FPGA hardware, but we cannot expect that it is used to describe a low-level hardware structures.

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

The AccelDSP is a high-level DSP synthesis tool facilitating the mapping from a Matlab floating-point design to a Xilinx FPGA fixed-point design. It reads and analyzes the Matlab code and then automatically generates a fixed-point version of 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 the Matlab codes are mandatory and the 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.

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.

2.6 Existing FPGA-Based Embedded Vision System

In the past literatures, lots of works have been done to implement the image recognition algorithms on a FPGA. However, many of them only focus on some simple ones which can be finished in one pass and do not require the aid of external memories. For instance, some neighbourhood operations, which typically include median filter, Sobel, Prewitt, Laplacian, Gaussian, Canny [5, 7] and Harris corner detector as well as stereo vision algorithms [51], etc. In addition some researchers also exert themselves to tailor their algorithms to eliminate the introduction of off-chip memory, for example [48].

However, many image recognition algorithms are by nature iterative operations and cannot work without the aid of off-chip memory. For example, one basic image processing algorithm named the Connected Components Labeling, which also belongs to neighbourhood operator, cannot be completed in one pass. In [3], a single pass connected components algorithm is presented, but it is difficult to be used in real applications since it consumes too many on-chip memories. Optical flow is another example that needs the introduction of external memory, and a corresponding solution is described in [52], which unfortunately can only deal with a QVGA-size (320 240) video image.

As a matter of fact, it is better to introduce off-chip memory when implementing complicated algorithms on FPGA for the purpose of reducing costs and making the system flexible. One solution described in [55] is a negative example. It requires a bigger FPGA when dealing with higher resolution images, making it too expensive to affordable and loosing the expandability.

In recent years, CPU integrated in FPGA enhances the processing capability of FPGA, and making FPGA a promising platform to develop SOC. [53] describes a FPGA-based people detection system, which adopts a 32-bit soft processor named the Microblaze. However, in [53], not only the hardware logic circuit on the FPGA is used, but also the embedded soft CPU is involved in computation, which jeopardizes the system performance deeply: only a low system speed of 2.5 frames per second (fps) is reached. It is hard to be used in real-time application. In [54] another FPGA-based vision system adopting integrated CPU is described. In this work, all the algorithm operations are performed on hardware logic of FPGA, while the soft CPU is only used to sequence the general operation. However, the system only achieves a low speed performance of processing VGA (640 480) at 10fps.

The following two figures display the FPGA board used in [51] and [54] respectively. They both adopt Xilinx Virtex-4 series FPGA.

Figure 6: FPGA Board Used in [51]

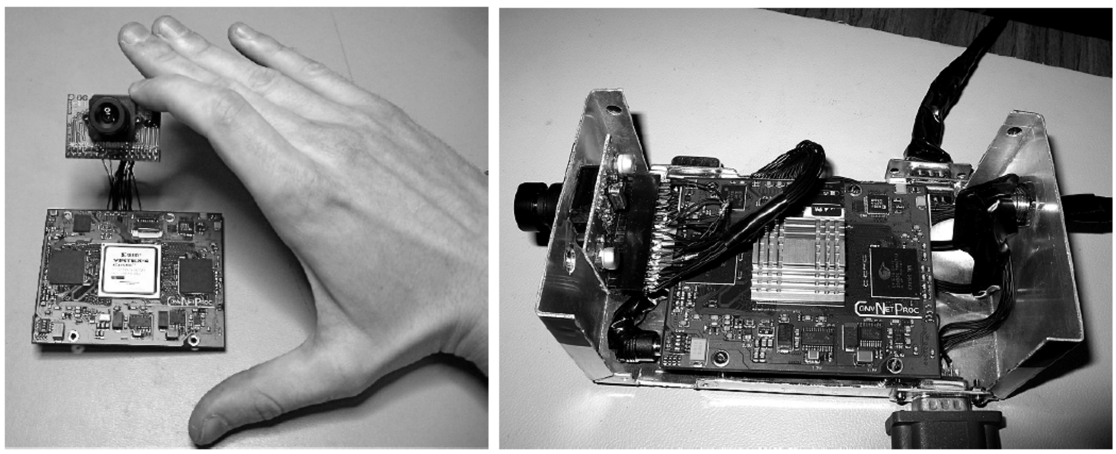

Figure 7: FPGA Board Used in [54]

The work in this thesis presents an expandable FPGA-based vision system integrating both hardware and software on a single FPGA chip. Multiple IPs are called as a set of complex instructions by the embedded Microblaze CPU to perform the whole algorithm. By virtue of the hardware parallel architecture, a real-time performance of full VGA process at 32 fps rate is achieved.

3 DESIGN AND METHODOLOGY

3.1 Board-Level System Architecture

The proposed embedded vision system is capable of acquiring and processing VGA video to extract the pre-defined object information in real time. Its board-level diagram is shown in the following figure.

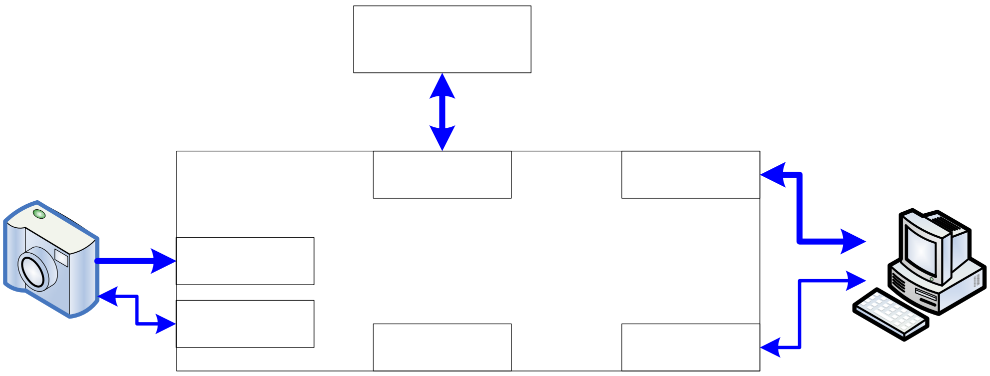

Figure 8: System Architecture Diagram in Board Level

The camera on the left hand side provides FPGA with a raw video stream through the Image Acquisition port under the control of I2C bus. An external DDR2 memory is introduced for the algorithm operation, because the algorithm is an iterative process and requires handling large amount of image data. Through the Multi-Port Memory Controller (MPMC), and embedded Microblaze CPU and internal digital logic are connected to external DDR2 memory for fetching and storing data. The peripheral GPIO is used to control several LEDs on the board so that the internal logic status can be observed. Finally, considering the bandwidth usage and the system flexibility, we use USB and RS232 to respectively transmit the video stream and the analysis to a computer to display on a Windows Graphic User Interface (GUI) application. A RS232 port is used to transfer analysis because it consists of only the object ID along with its coordinates, and occupies a small bandwidth. However, the video stream is another story, consuming a large amount of bandwidth, so a USB 2.0 is adopted for video transmission.

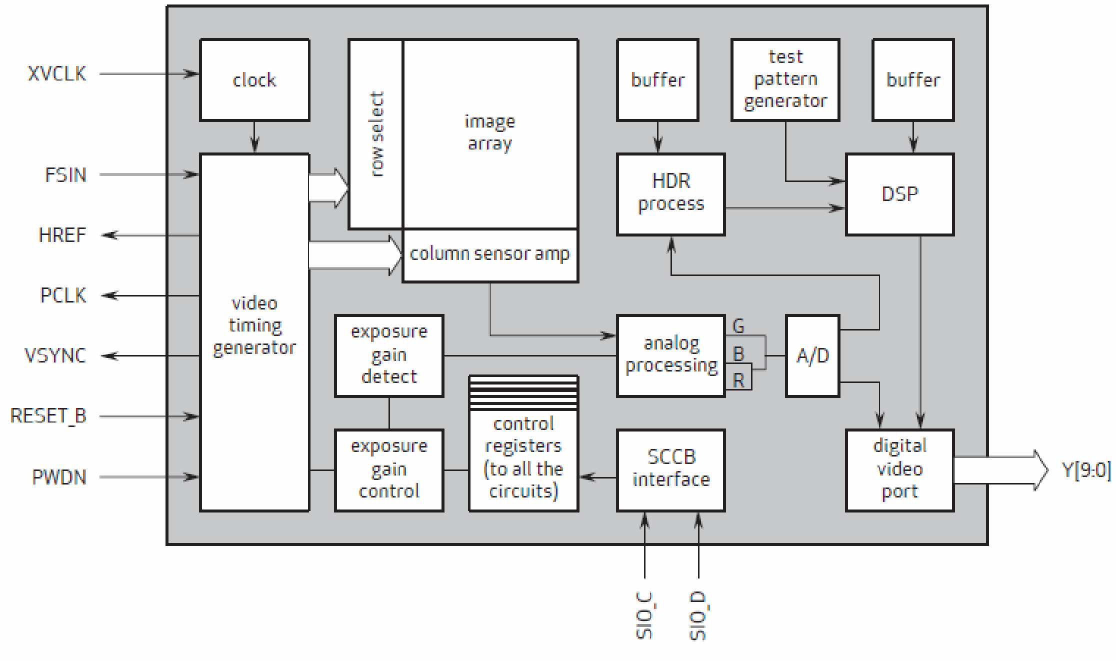

Figure 9: Functional Block Diagram of OV10121

1) Camera: an omnivision OV10121 Black/White CMOS WVGA High Dynamic Range (HDR) Camera Sensor is selected. This device incorporates a 768-by-506 image array capable of operating at up to 30 frames per second. Through the Serial Camera Control Bus (SCCB) interface, all the camera functions including exposure control, gain, white balance and windowing, etc., can be programmed. The SCCB is an updated I2C bus and conforms to the conventional I2C protocol. The most important feature of this device is its proprietary HDR technology that enables the OV10121 to handle extreme variations of bright and dark conditions within the same scene,

making it perform like the human eye under quickly changing lighting conditions. Figure 9 displays the functional block diagram of OV10121.

2) FPGA: Xilinx Virtex-5 XC5VLX110-FF676 FPGA is adopted. Built on a 65-nm copper process technology, Virtex-5 FPGAs are a programmable alternative to custom ASIC technology. It offers a good optional solution for addressing the needs of high-performance logic designers, high-performance DSP designers, and high-performance embedded systems designers, with unprecedented logic, DSP, hard/soft microprocessor, and connectivity capabilities. In addition to its huge amount of logic resources (110,592 logic cells), XC5VLX110 contains many hard-IP system level blocks, including powerful 36-Kbit block RAM/FIFOs, 550 MHz second-generation 25×18 DSP slices, enhanced clock management tiles with integrated DCM (Digital Clock Managers), phase-locked-loop (PLL) clock generators, and advanced configuration options. All of these features make XC5VLX110 a good platform to develop sophisticated image processing algorithms.

Figure 10: Virtex 5 FPGA

3.2 IP-Level System Architecture

Board-level architecture shown in the above subsection only focuses on the interconnection of components on a board, but does not display the circuit architecture inside the FPGA. The following figure illustrates the system architecture inside the FPGA in IP level.

Each square block in Figure 11 represents a circuit unit, and is called an IP. Each IP can perform a certain function, and is equipped with a uniform interface called PLB so that each IP can be connected together with Microblaze (the soft-CPU core provided by Xilinx), to form an integrated system.

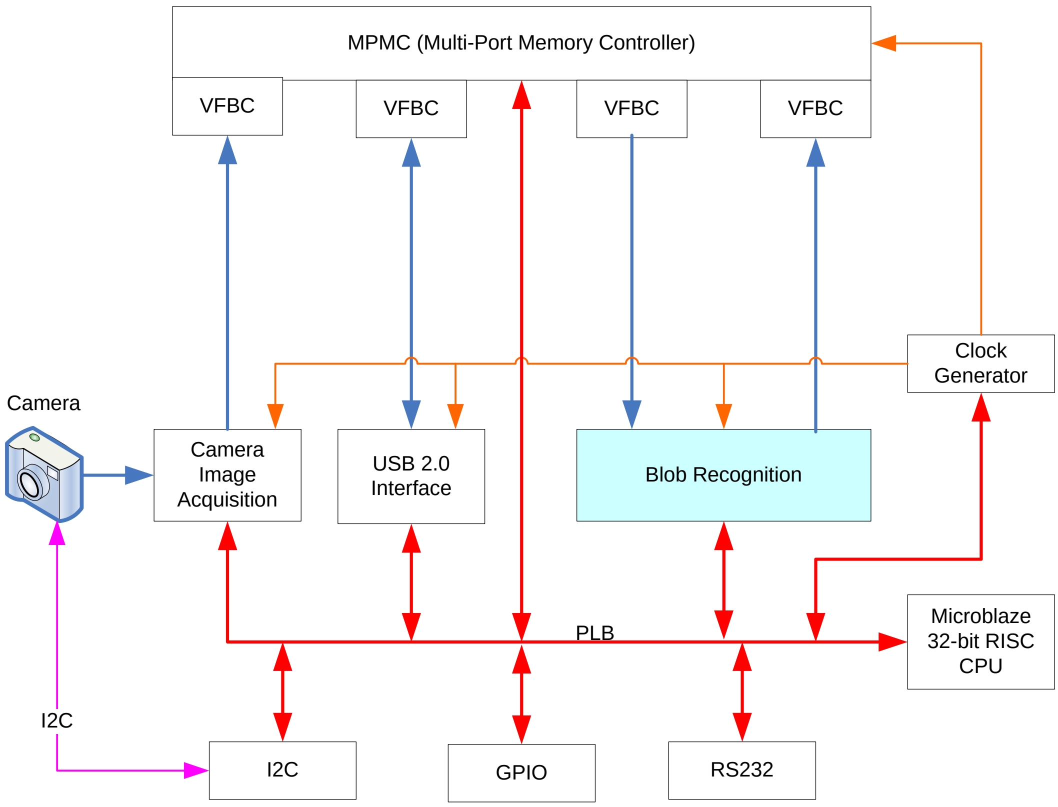

Figure 11: System Architecture Diagram In IP Level

- Microblaze is a 32-bit RISC CPU soft core, and is integrated here for controlling through the PLB bus the operation of every other IP in the system. A whole suite of development tools including compiler and assemblers are provided by Xilinx to help the designer code in C or C++.

- The PLB is one element of the IBM CoreConnect architecture, and is a high-performance synchronous bus designed for connection of processors to high-performance peripheral devices. The PLB includes multiple advanced features concerning its data transfer, bandwidth usage, pipeline architecture, bus arbitration, and memory guard methodology to name a few. Most of these features map well to the FPGA architecture, however, some can result in the inefficient use of FPGA resources or can lower system clock rates. As a result, Xilinx uses an efficient subset of the PLB for Xilinx-developed PLB devices.
- The Clock Generator is an IP provided by Xilinx, and is responsible for providing clocks for every other IP according to system wide clock requirements.
- MPMC stands for Multi-Port Memory Controller, and is used to access external DDR2 memory. It is a fully parameterizable memory controller that supports SDRAM/DDR/DDR2/DDR3/LPDDR memory. In addition, MPMC provides access to memory for one to eight ports.
- VFBC stands for Video Frame Buffer Controller, and is a connection layer between each IP and the MPMC. It provides each IP with access to external DDR2 memory, and allows IPs to read and write data in two dimensional (2D) sets regardless of the size or the organization of external memory transactions. The VFBC includes separate Asynchronous FIFO interfaces for command input, write data input, and read data output. This is useful to decouple the video IP from the memory clock domain.

- GPIO stands for general purpose I/O, and is a port to control some external LEDs for indicating the system status.
- RS232 is a serial communication port to communicate with a computer for receiving instructions and outputting results.
- Image Acquisition is a video timing generator to match the video output port of an Omnivision camera, and then capture its generated video stream.
- I2C is a 2-wire bus to write/read the registers of an Omnivision camera OV10121, thereby controlling its operation mode. Omnivision uses an updated I2C bus interface called Serial Camera Control Bus (SCCB) interface, which conforms to the protocol of conventional I2C bus interface.
- USB stands for Universal Serial Bus, and is used in this project to communicate with a computer for receiving instructions from user and outputting video stream to a computer for display. This USB IP is designed to interface with an on-board Cypress EZ-USB FX2™ USB Microcontroller. The EZ-USB FX2 device is a single-chip integrated USB 2.0 transceiver, Serial Interface Engine (SIE) and 8051 microcontroller. This device supports full-speed (12 Mbps) and high-speed (480 Mbps) modes. The FX2 interface to the Virtex-5 FPGA is a programmable state machine that supports 8- or 16-bit parallel data transfers. The USB FX2 device is used in a slave mode where the FPGA accesses the FX2 like a FIFO.
- Blob Recognition is the IP handling the blob recognition algorithm. The embedded algorithm is iterative by nature and operates a large amount of data. So two VFBCs are adopted here to ease the operation of the frequent fetching of data from an external memory and storing the temporary results back to it.

Among all of these IPs, four of them, including Image Acquisition, I2C, USB and Blob Recognition, are designed by the author. In the following subsections, the design of these four IPs will be addressed.

3.3 Peripheral IP Design

3.3.1 Image Acquisition IP

Image Acquisition IP is a video timing generator to match the video output port of Omnivision OV10121 camera, so that the generated video stream can be captured and stored in an external DDR2 memory through the FPGA. The related input and output pins of the OV10121 are listed in Table 3.

Table 3: Pin of OV10121 Digital Video Interface

Item	Signal	Pin Type	Function/Description
1	PWDN	Input	Power down mode selection 0: Normal Mode 1: Power down mode PWDN has an internal pull-down resistor.
2	HREF	Output	Horizontal valid pixel reference signal output
3	VSYNC	Output	Vertical sync output
4	PCLK	Output	Pixel clock output
5	XCLK	Input	System clock input; 6~30MHz
6	Y[9:0]	Output	10-bit digital video output data port

The video timing specification of OV10121 is shown below.

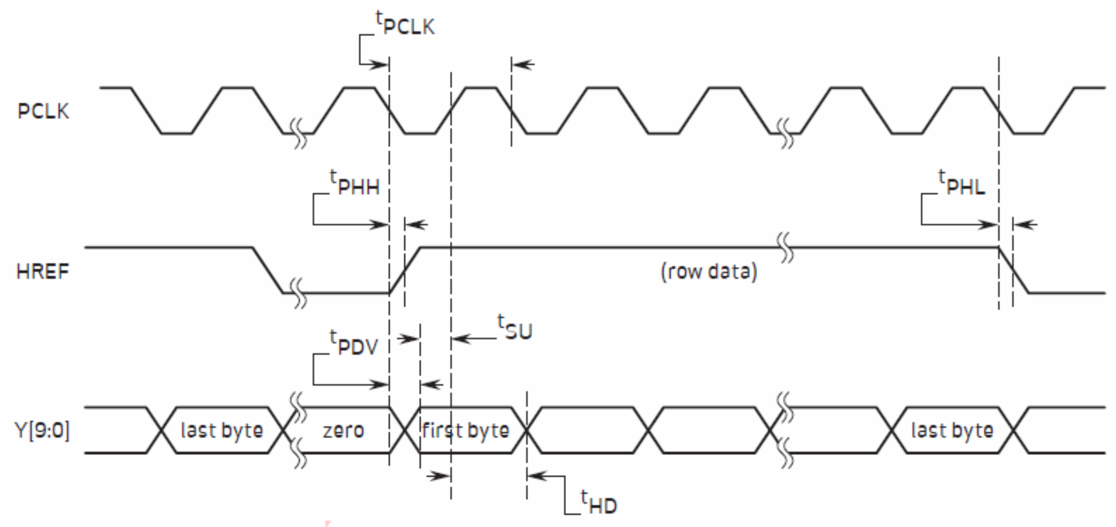

Figure 12: Horizontal Timing

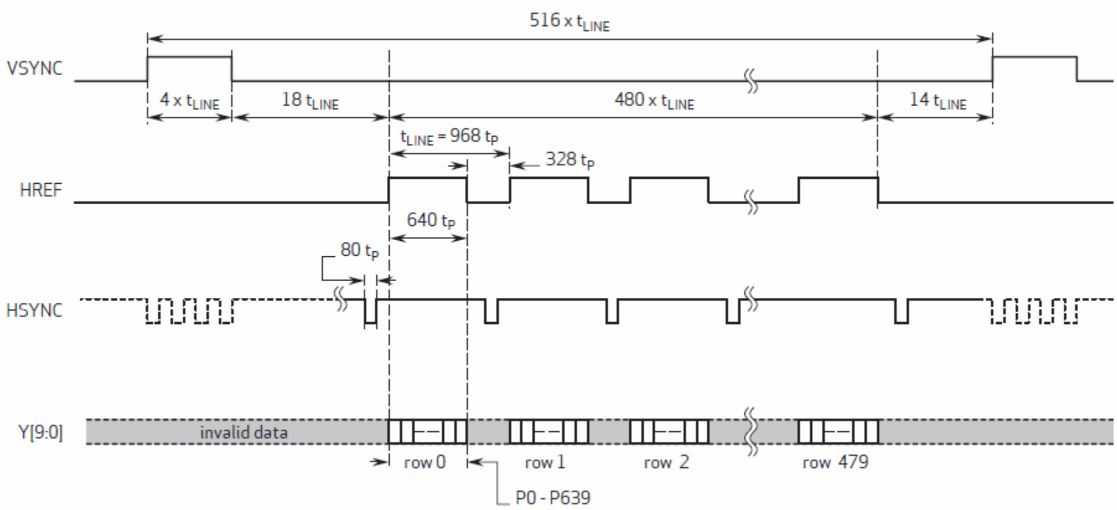

Figure 13: VGA Frame Timing

The Image Acquisition IP can be divided into three parts. The first is a timing generator called an Image Capture Interface. It is used to generate the two inputs of the OV10121, XCLK and PWDN, and also receives the incoming 10-bit digital video stream Y[9:0] according to the timing of PCLK, HREF and VSYNC. The second part is called Video to VFBC, and is used for transferring the captured video stream to an external DDR2 memory via the VFBC port. The third one is a PLB interface for connecting this

IP to a PLB bus so that it can be under the control of the Microblaze. The following is the functional diagram of Image Acquisition IP.

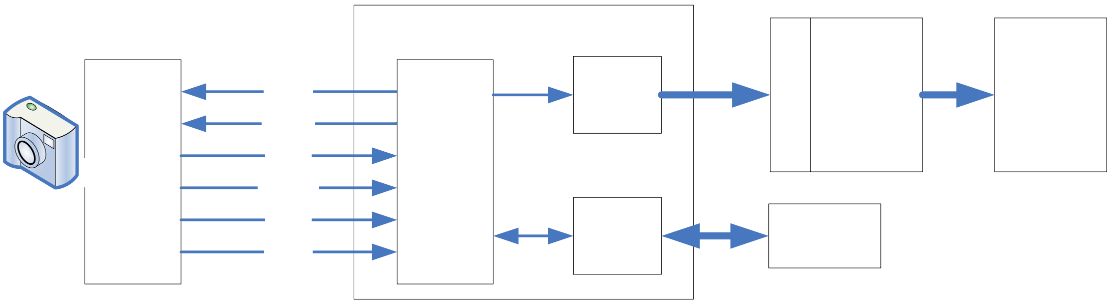

Figure 14: Image Acquisition IP Functional Diagram

3.3.2 I2C IP

The Omnivision OV10121 camera uses an updated version of the I2C bus interface called Serial Camera Control Bus (SCCB) interface, which conforms to the protocol of the conventional I2C bus interface.

The I2C bus is a popular serial, two-wire interface used in many systems because of its low overhead. The two-wire interface minimizes interconnections so ICs have fewer pins, and the number of traces required on printed circuit boards is reduced. Capable of 100 KHz or 400 KHz operations, each device connected to the bus is software addressable by a unique address with a simple Master/Slave protocol.

The I2C bus consists of two wires, a serial data (SDA) and serial clock (SCL), which carry information between the devices connected to the bus. The number of devices connected to the same bus is limited only by a maximum bus capacitance of 400 pF. Both the SDA and SCL lines are bidirectional lines, connected to a positive supply voltage via a pull-up resistor. When the bus is free, both lines are High. The output stages of devices connected to the bus must have an open-drain or open-collector in order to perform the wired-AND function.

Each device on the bus has a unique address and can operate as either a transmitter or receiver. In addition, devices can also be configured as Masters or Slaves. A Master is the device which initiates a data transfer on the bus and generates the clock signals to permit that transfer. Any other device that is being addressed is considered a Slave. The I2C protocol defines an arbitration procedure that insures that if more than one Master simultaneously tries to control the bus, only one is allowed to do so and the message is not corrupted. The arbitration and clock synchronization procedures defined in the I2C specification are supported in this project.

Data transfers on the I2C bus are initiated with a START condition and are terminated with a STOP condition. Normal data on the SDA line must be stable during the High period of the clock. The High or Low state of the data line can only change when SCL is Low. The START condition is a unique case and is defined by a High-to-Low transition on the SDA line, while the SCL is High. Likewise, the STOP condition is a unique case and is defined by a Low-to-High transition on the SDA line, while the SCL is High. The definitions of data, START and STOP, insure that the START and STOP conditions will never be confused as data. This is shown in the following figure.

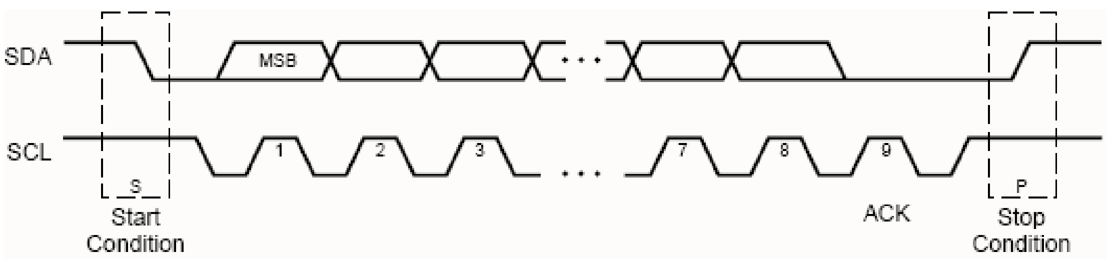

Figure 15: I2C Bus Protocal

In this thesis, a custom I2C IP is designed and has the following specifications:

- Only supports master operation. Does not support slave mode.
- Supports multiple-master operation
- Supports I2C WAIT state (I2C clock line SCL is held LOW by external devices)
- Only supports byte write/read, and does not support page write and sequential read
- Data transfer format is seven-bit address format
- The PLB controller interface
- 400KHz Operation since the Omnivision OV10121 camera sensor requires 400Khz operation.

The following figure displays the diagram of the designed I2C IP. It can be divided into two major blocks, the PLB interface and the I2C controller. The PLB interface is for connecting the I2C controller to Microblaze through a PLB bus, and the I2C controller is for generating the I2C signal timing conforming to the I2C protocol.

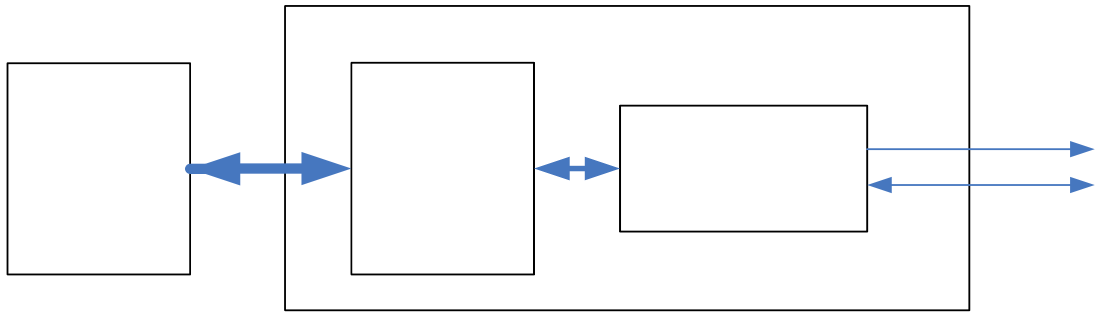

Figure 16: Block Diagram of I2C Master Controller

The I2C bus interface logic consists of several different processes.

- Arbitration is to insure that if more than one Master simultaneously tries to control the bus, only one is allowed to do so and the message is not corrupted.

- Generation of SCL, SDA, START and STOP Conditions is to generate the SCL and SDA signal outputs on the I2C bus when in Master mode.
- Start/Stop Detection is to monitor Start/Stop conditions on the I2C bus.
- I2C State Machine contains two state machines: Main State Machine and One-Bit State Machine. The first one is responsible for the whole flow of the I2C controller, while the latter one is responsible for generating one bit.

3.3.3 USB IP

USB stands for Universal Serial Bus, and is used in this thesis to communicate with a computer for receiving instructions and outputting video stream to display. This USB IP is designed to interface with an on-board Cypress EZ-USB FX2 USB Microcontroller CY7C68013A. The EZ-USB FX2 device is a single-chip integrated USB 2.0 transceiver, Serial Interface Engine (SIE) and 8051 microcontroller. This device supports full-speed (12 Mbps) and high-speed (480 Mbps) modes. The FX2 interface to the Virtex-5 FPGA is a programmable state machine that supports 8 or 16-bit parallel data transfers. The USB FX2 device is used in a slave mode where the FPGA accesses the FX2 like a FIFO.

USB development requires the knowledge of both developing a Windows Operation System based application and designing embedded systems, and requires mastering high-level protocol and low-level circuit design, which is a very complicated process. In this thesis, the FPGA based glue logic to interface the CY7C68013A is designed. In addition, with the aid of the development suit provided by Cypress, a Windows GUI application is designed to display the video stream and to control the operation of the FPGA system. In this subsection, only the logic design of the USB interface will be addressed.

Figure 17 displays the diagram of the designed USB interface IP. It can be divided into three major blocks:

- ✓ FIFO Controller generates the Cypress defined slave FIFO signal timing to interface with the on-board Cypress USB controller.
- ✓ PLB interface connects the USB IP to the Microblaze through a PLB bus.
- ✓ USB To VFBC transfers the video data stored in an external memory to a computer via a VFBC port.

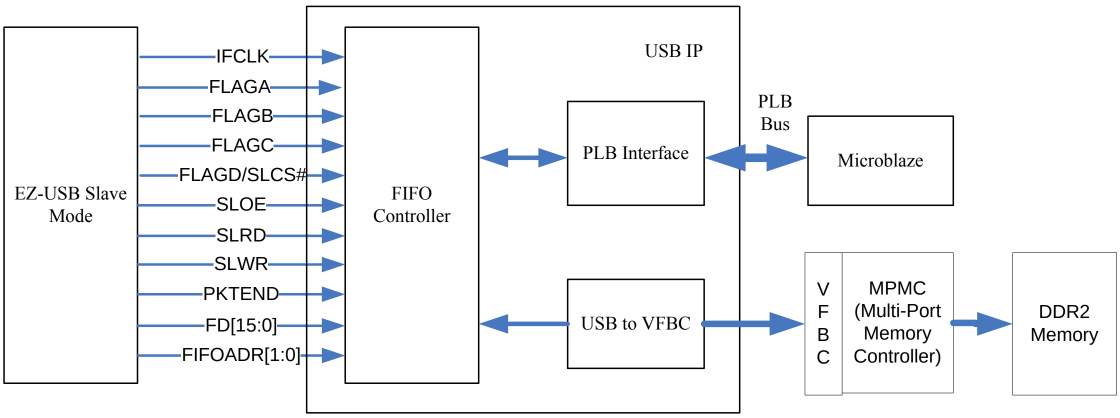

Figure 17: Interface between Slave mode EZ-USB and FPGA USB IP

The on-board Cypress EZ-USB FX2 USB Microcontroller is configured to work in slave mode, and is controlled by an external master: the FPGA. FPGA accesses it like a 16-bit FIFO.

- The USB Interface logic inside the FPGA accesses the FIFOs through an 8 or 16-bit wide data bus, FD[15:0]. The data bus is bidirectional, with its output drivers controlled by the SLOE pin.

- Slave-mode EZ-USB FX2 contains four slave FIFIOs. The FIFOADR[1:0] pins select which of the four FIFOs are connected to the FD bus and which are controlled by the FPGA.
- In asynchronous mode (IFCONFIG.3 = 1), SLRD and SLWR are read and write strobes; in synchronous mode (IFCONFIG.3 =0), SLRD and SLWR work as enable signals. In this project, synchronous mode is adopted.
- The slave FIFO interface can be clocked from either an internal or an external source. In this thesis, the internal clock source is adopted and is also configured to output on the interface clock (IFCLK) pin to clock the USB Interface logic inside the FPGA. In this way, the FPGA logic design is simplified.

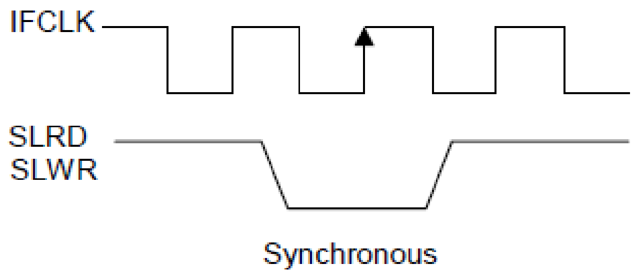

Figure 18: USB Slave FIFO Synchronous Timing Models

- Four pins — FLAGA, FLAGB, FLAGC, and FLAGD— are adopted to indicate the status of the EZ-USB's FIFOs: 'FIFO full' or 'FIFO empty'.
- PKTEND is asserted by the FPGA to commit an IN packet to the USB regardless of the packet's length. Usually it used when the master wishes to send a 'short' packet.

In this project, the USB IP will not only transfer the video stream to a computer, but also will receive some instructions from it. So the logic of this USB IP can be divided into two state machines: one is for writing the slave FIFO, and the other is for reading the slave FIFO. These two state machines will have a similar operation process, so only the FIFO write state machine is shown below as an example.

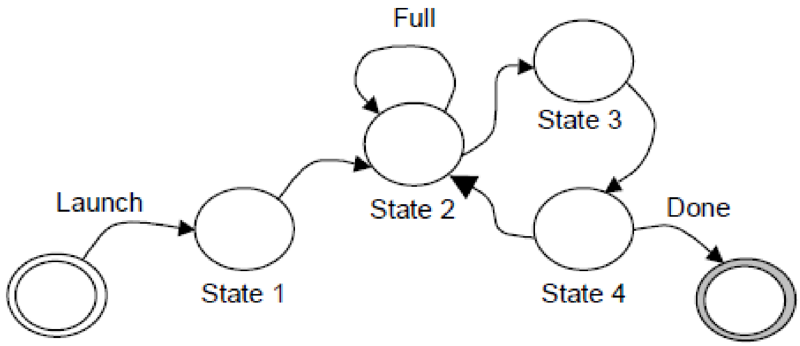

Figure 19: State Machine of Synchronous FIFO Writes

3.4 Core Algorithm IP Design

Blob Recognition IP is just the circuit block that contains the core algorithm to search and identify the target. In the following subsections, this algorithm will be explained from the viewpoint of software at first, and then the FPGA implementation will be addressed.

3.4.1 Blob Recognition Algorithm

The proposed blob recognition algorithm is aiming at processing a gray-scale image in VGA size, and can be divided into three major steps as below:

- Image scale down: the incoming video stream is in VGA (640x480) size. In order to decrease the processing time and to reduce the resources for computation, this incoming VGA video image is downscaled by factors of four to QQVGA (160x120).

- Candidate location: given a downscaled QQVGA image in which the position of the target is unknown, this step is supposed to search for every possible candidate. Multiple candidates may be chosen according to a certain criterion, which exerts strong influence upon the performance of the whole system.

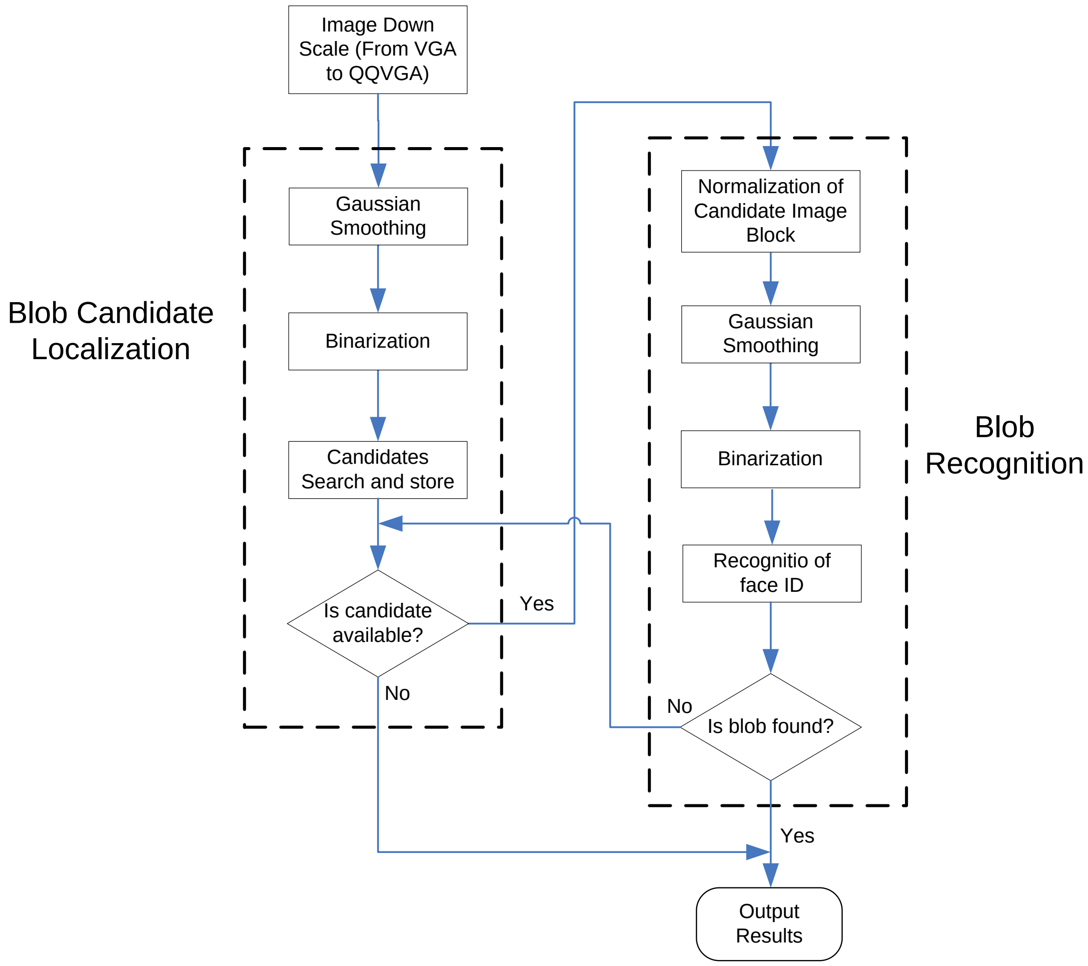

Figure 20: Flow Chart of Proposed Algorithm

- Object identification: given each candidate, the related image data will be captured from the main memory to process and analyze. An identification algorithm inspired by the unique feature of the blob face is adopted to extract the explicit information of each face.

The latter two steps can be divided into finer processes, and are illustrated in Figure 20.

3.4.1.1 Gaussian Smoothing

The Gaussian Smoothing is used to blur the image under detection for further processing. Each pixel is replaced with a weighted average of its neighbourhood when the mask is sliding over the image. It belongs to the neighborhood operation (or is called a pixel operation or a kernel operation), and is a kind of widly used image processing filter defined by the following equation.

$$g(x, y) = Ke^{\frac{-(x+y)^2}{2\sigma^2}}, \quad (1)$$

where, (0, 0) is the center of the mask, and σ determines how fast the weight decay. Usually, the sum of the mask coefficients is 1. The selected Gaussian mask is shown in Figure 21.

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

Figure 21: 5-by-5 Gaussian Kernel

It is worthwhile to notice the parameters inside the mask since they are all selected carefully for the purpose of simplifying computation and reducing logic resources. In this Gaussian kernel, only three parameters, 7, 26 and 41, need the multiplication operation, while the other parameters only require a simple shift operation. This is because that performing multiplication or division of unsigned integers by powers of two only involves a logic shift. Taking the parameter 4 as an example, to shift left by two bits on a number has the same effect of multiplying it by 4.

3.4.1.2 Binarization

1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1		1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

Figure 22: 11-by-11 Average Kernels

Binarization is to binarize the gray-scale image under detection. In the literature, there are many algorithms about binarization, such as the Otsu method. The Otsu method is a kind of global threshold method, and is sensitive to abrupt change of lighting conditions. So researchers turn to adaptive binarization algorithms focusing on local threshold. When it comes to different applications, different binarization methods are used. In this project, based on the size of the image under detection, the specific blob pattern and the experience accumulated through many experiments, an average filter based adaptive threshold method is adopted. The kernel of this average filter is 11-by-11 in size, and is shown in Figure 22.

Each pixel is replaced with ‘1’ or ‘0’ according to the following equation when the mask is sliding over the image. Where, ‘1’ represents white pixel, and ‘0’ represents a black pixel,

$$BW(0,0) = \begin{cases} 1 & if \quad Int(0,0) > Int\_avg + Delta \\ 0 & if \quad Int(0,0) \le Int\_avg + Delta \end{cases} , \qquad (2)$$

where, (0, 0) is the center of the mask, and BW(0,0) is the value that is going to be assigned to the center pixel of the mask; Int(0,0) represents the current intensity value of the center pixel, and the Int_avg is the average intensity value of the neighbourhood; Delta is a parameter determining the threshold.

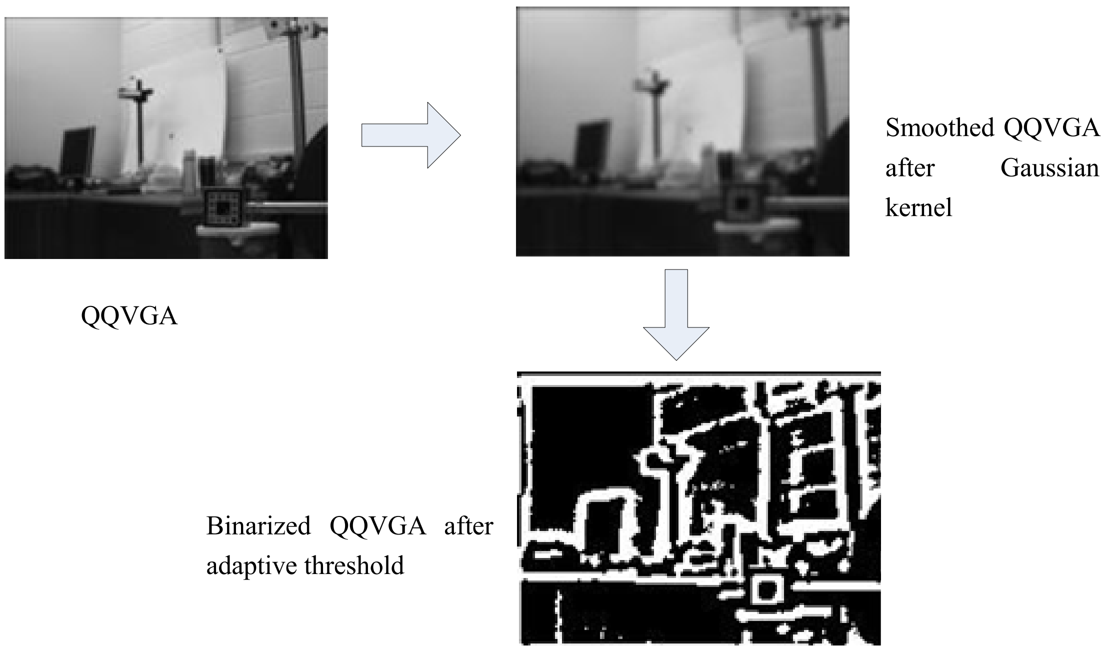

Figure 23: Example of Image Smooth and Binarization

The Gaussian Smoothing and Binarization methods are tailored in such a way that the black heart block in a face can be correctly segmented from its background, which is just the key to successfully selecting a target candidate. Figure 23 shows an example of the results of these two steps.

3.4.1.3 Candidates Search and Store

This step tries to find all the sub-images that contain an object and then stores into memory the coordinates and the size of the selected sub-images (candidates). In order to search for the possible candidates, the following method is proposed: given a binarized image, each component is grouped and labelled by using a Connected Component Labelling (CCL) method, and then the dimension of each component is measured, including maximum x axis, maximum y axis, minimum x axis, minimum y axis and center of the component. It is expected to choose all the possible candidates according to this simple measurement and some certain criterion, for example the ratio of width and height and the distance between the center of the component to the border of the image.

CCL scans an image and groups its pixels into components based on pixel connectivity. But the traditional CCL can only handle one single type of pixel at a time, either black or white. In this project both black and white components are taken into consideration, hence the following Dual Connected Component Labeling (DCCL) method is proposed.

Similar to the conventional CCL method described in [50], DCCL also adopts a 2×3 mask and an equivalent table.

- 2×3 mask: the connectivity of each pixel is checked when a 2×3 window is sliding across an image, where the 2×3 window is shown below. “A” is the pixel under detection.

- Equivalent table: it is used to store the label values that actually belong to one single component.

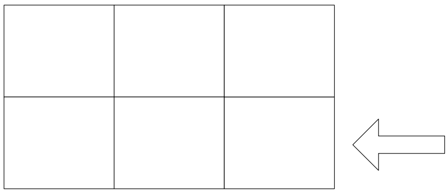

Figure 24: 2x3 Mask of DCCL

However, unlike the conventional CCL method, DCCL changes the data structure and adopts another equivalent table called a BW_EQ table for storing the information of connectivity between different types of pixels.

- Data structure: In the 2×3 mask, each pixel is represented by an *N*-bit data.

E D

  - ✓ (*N*-1 downto 1) bits represent label value.
  - ✓ *N*th bit indicates the type of pixel: 1 ->black, 0 ->white.

B A

For example:

B(*N*) = 1 indicates that pixel B is a pixel in black color.

B(*N*-1 downto 1) = 3 indicates that B is labelled number 3.

- BW_EQ table: an equivalent table that stores the information of connectivity between black and white components.

Each DCCL processing will be done according to the following order: E -> D -> C -> B. A is the pixel under detection.

- Label Assignment: starting from pixel E, it searches the first pixel which has the same Nth bit as that of pixel A, and also has a non-zero label value. If found, then its label is assigned to pixel A. Or if not found, a new label is assigned to A.

- Equivalent Table: after the label assignment, label comparison continues between A and other pixels with the same Nth bit. If different labels are found, the labels are stored into equivalent table.
- BW_EQ Table: it finds all the pixels with a different Nth bit from that of A, and then store the labels into the BW_EQ table.

After black and white components are grouped and labelled by the DCCL method, the dimension of each labelled component will be measured for the purpose of selecting target candidates. In this thesis, based on the measured size of the segmented heart block, it is possible to estimate the size of the candidate image block, since there is a certain geometric relationship between the blobs inside a face: the heart block is a square shape and the length of its side is almost one third of the length of the side of the square bound. This relationship is shown in Figure 25.

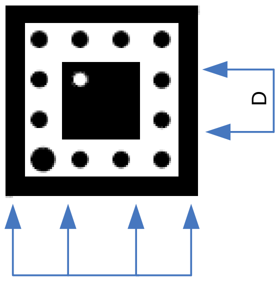

Figure 25: Geometric Relationship of Blob Face

Finally, the coordinates and the dimension information of all the chosen candidates are stored in a buffer for further processing one by one.

3.4.1.4 Normalization of Candidate Image Block

Every candidate image may have different image sizes. It is helpful for the next ID identification process to have the candidate image block normalized. In this project, the normalization size is 96-by-96. However, owing to the limitation of the FPGA memory interface, only the transformation listed in Table 4 is adopted.

Image normalization scales an image to a certain size. It is a popular image processing operation. In this project, a traditional bilinear interpolation method is adopted: the incoming image data comes through a filter, and the output pixel value of the filter is a weighted average of pixels in the nearest 2-by-2 neighbourhood. Figure 26 is an example of normalization from 128-by-128 to 96-by-96.

Table 4: Normalization List

96x96 -> 96x96	256x256 -> 96x96	384x384 -> 96x96
128x128 -> 96x96	288x288 -> 96x96	416x416 -> 96x96
160x160 -> 96x96	320x320 -> 96x96	448x448 -> 96x96
192x192 -> 96x96	352x352 -> 96x96	480x480 -> 96x96
224x224 -> 96x96		

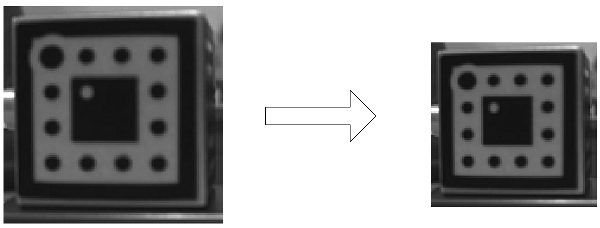

Figure 26: Normalization from 128-by-128 to 96-by-96

3.4.1.5 Recognition of Face ID

The target is a cube, and has 5 faces with different blob patterns in black and white. These blob patterns look like nested squares. From Figure 27, it can be seen that there is a black square named a heart block in the center of each face. This heart block is surrounded by 12 small black dots, while the 12 black dots are bounded by another black square border again.

Among the 12 black dots, the largest one is defined as the origin of the face. The relative position between the origin and the white dots inside the heart block determines the ID of the face. In this project, up to four white dots may be located inside the heart block, and the face ID is calculated by the following equation,

$$ID = W1 * 2^0 + W2 * 2^1 + W3 * 2^2 + W4 * 2^3 \quad , \qquad (3)$$

where, $Wx = 1$ only if a white dot appears on the position of Px (x=1 or 2 or 3 or 4). Position Px is determined based on the relative position between the corner of the heart block and the origin in a clock-wise direction.

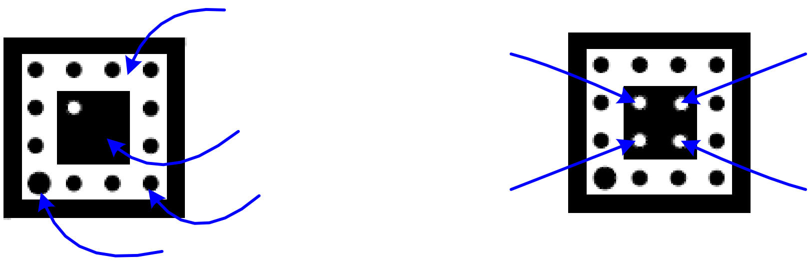

Figure 27: Definition of Blob Face

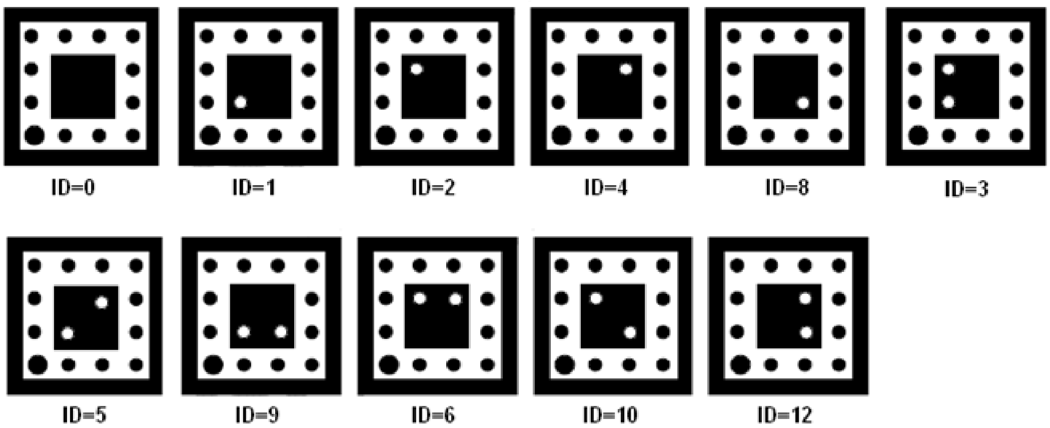

Figure 28: Example of Blob Faces

White Frame

Figure 28 gives examples of face ID. Where, up to 2 white dots exist inside the heart block.

An intuitive ID recognition algorithm inspired by the feature of the blob face is proposed. The whole flow chart is illustrated in Figure 29.

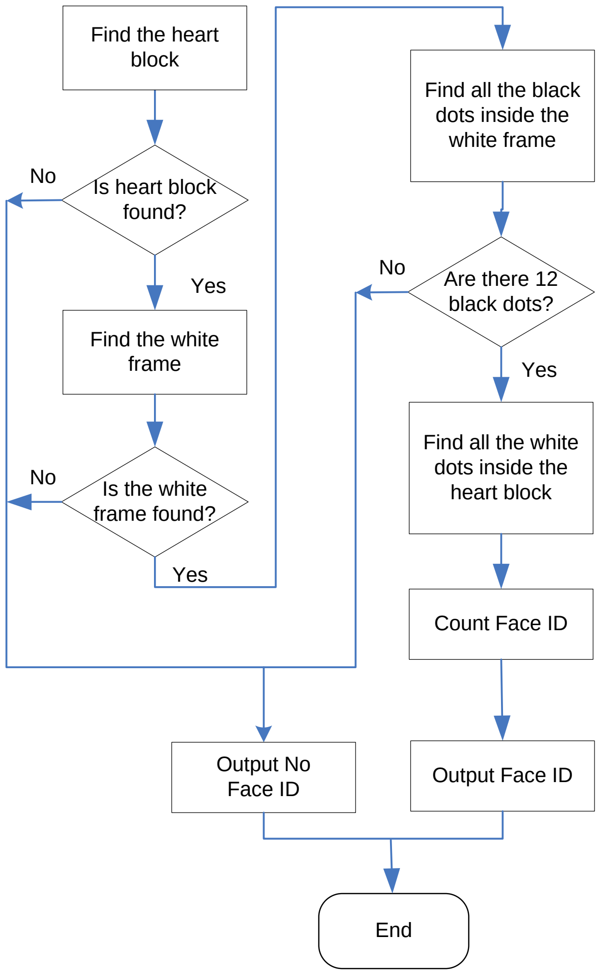

Figure 29: Recognition of Face ID

1) Find out the heart block: every face has a heart block in square shape. Since the candidate image block is located based on the segmented heart block, the center of the cropped image block is by nature the center of the heart block.

In addition, we introduce a bias parameter for the tolerance of the possible mismatch caused by image distortion or previous image processing.

2) Find out the white frame: two features are helpful to find the white frame: 1) The central coordinates of the white frame should be very close to the center of the heart block; 2) The white frame is surrounding the heart block. The purpose of obtaining the white frame is to search for the 12 black dots surrounding the heart block.

3) Find out 12 black dots surrounding the heart block: as mentioned before, the connectivity information between black and white components can be achieved through DCCL method, and is stored in BW_EQ table. Combining the information of white frame and the BW_EQ table, the black dots can be easily screened out.

4) Find all the white dots inside the heart block: with the aid of BW_EQ table and the measured geometric dimension of each labelled component, it is easy to get all the white dots. These white dots are used to calculate the ID of the face.

5) Count face ID: given the information of all the black dots and the white dots, it is possible to calculate the ID of a face. However, in order to make sure the position of the origin of the face as well as the position of the white dots inside the heart block, a fine division of the blob face is illustrated in Figure 30.

- Locate origin: the origin of a face is the largest black dot among the 12 black dots inside the white frame. Its location is defined by 8 zones based on its relative position with the heart block:

TOP_LEFT, TOP_MID, TOP_RIGHT,

MID_LEFT, MID_RIGHT,

BOTTOM_LEFT, BOTTOM_MID, BOTTOM_RIGHT

- Locate white dots inside the heart block: the location of white dots inside the heart block is defined by the following 8 zones.

UP_LEFT, UP_RIGHT, DOWN_LEFT, DOWN_RIGHT,

LEFT_EDGE, RIGHT_EDGE, TOP_EDGE, BOTTOM_EDGE

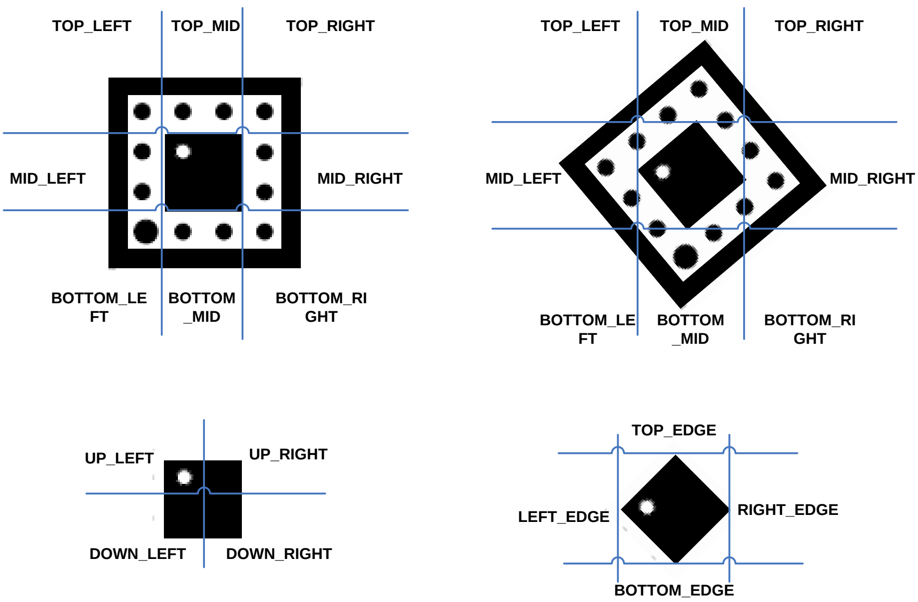

Figure 30: Division of Blob Face

- If the origin is located in the following zones:

TOP_LEFT, TOP_RIGHT, BOTTOM_LEFT, BOTTOM_RIGHT

Then the following zones are checked to see where the white dot lies in,

UP_LEFT, UP_RIGHT, DOWN_LEFT, DOWN_RIGHT

- If the origin is located in the following zones:

  TOP_MID, MID_LEFT, MID_RIGHT, BOTTOM_MID

  Then the following zones are checked to see where the white dot lies in,

  LEFT_EDGE, RIGHT_EDGE, TOP_EDGE, BOTTOM_EDGE

In short, according to the geometric relationship of the origin, white dots and the heart block, it is possible to calculate the ID of a face.

3.4.2 FPGA Implementation of Algorithm

The algorithm described in the section 3.4.1 is a specifically tailored version of algorithm for FPGA implantation, since there are some challenges when it comes to FPGA, such as limited computation resources and contentions caused by parallel processing. In this section, the practical FPGA implementation of each function in the algorithm is addressed.

3.4.2.1 Gaussian Smoothing Circuit

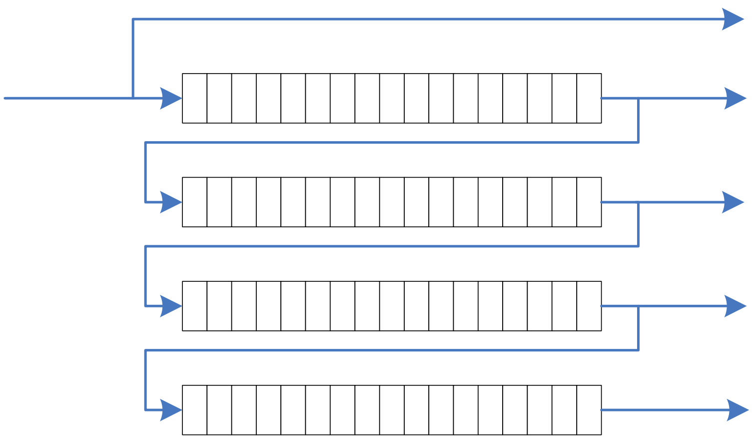

Figure 31: Gaussian Smoothing Circuit

Gaussian smoothing belongs to neighborhood operation (or called pixel operation or kernel operation), and is suitable for FPGA implementation. Observing the flow of the whole algorithm, it can be noticed that the Gaussian Smoothing method is used twice: one is to blur the incoming QQVGA image, and the other is to blur the normalized candidate image block. In order to reduce FPGA resources, only one Gaussian smooth IP is built inside and reused.

In order to handle both QQVGA and normalized candidate image block, this Gaussian Smooth IP adopts Dual Port RAM (DPRAM) because the output tap can be changed accordingly. In total, four DPRAM are used to compose a 5-by-5 kernel.

This Gaussian smoothing IP is designed by using Xilinx AccelDSP. The synthesized results are shown below.

- Performance Summary

Startup Clock Cycles	Hardware Clock Cycles Per Design Function Call
19	1

- Utilization Summary

Mult	Adder	Subtractor
10	24	0

- Interface Configuration

Name	Value	Information
Register Inputs	1	Inputs were registered (this will add one cycle of latency to the design)
Interface Protocol	push	Push-mode/feed-forward mode was implemented

- Performance Information

Clock Name	Requested Frequency	Estimated Frequency	Estimated Period	Max Throughput	Input Sampling
Clock	200.0 MHz	262.7 MHz	3.8070 ns	1	262.674 MSPS

Figure 32: Synthesized results of Gaussian Smoothing IP by Xilinx AccelDSP

3.4.2.2 Binarization Circuit

Just similar to the Gaussian Smooth function, this Binarization function is also used twice in the algorithm flow: One is to binarize the QQVGA image, and the other is to binarize the normalized image. For the same reason of reducing resources, Binarization IP adopts the architecture similar to that of Gaussian Smoothing IP, and uses DPRAM to handle both QQVGA and normalized image. However, 11 DPRAM instead of 5 DPRAM are used for composing a 11-by-11 average kernel.

This Binarization IP is also designed by using Xilinx AccelDSP. The synthesized results are shown below.

– Performance Summary

Startup Clock Cycles	Hardware Clock Cycles Per Design Function Call
13	1

– Utilization Summary

Mult	Adder	Subtractor
2	120	2

– Interface Configuration

Name	Value	Information
Register Inputs	1	Inputs were registered (this will add one cycle of latency to the design)
Interface Protocol	push	Push-mode/feed-forward mode was implemented

– Performance Information

Clock Name	Requested Frequency	Estimated Frequency	Estimated Period	Max Throughput	Input Sampling
Clock	200.0 MHz	126.1 MHz	7.9320 ns	1	126.072 MSPS

Figure 33: Synthesized results of Average Fitler IP by Xilinx AccelDSP

3.4.2.3 DCCL and Label Group Circuit

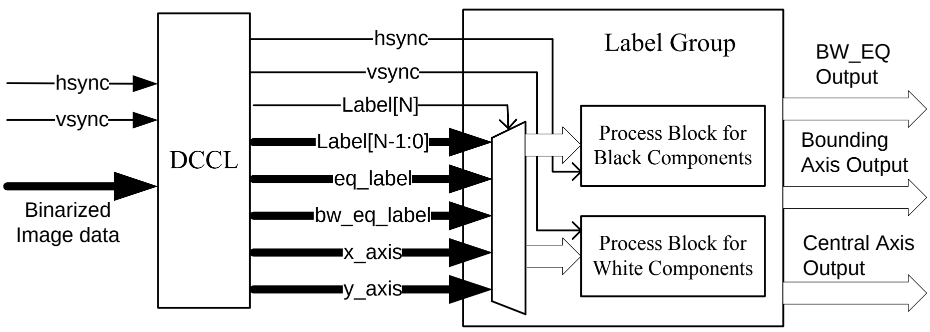

Figure 34: Block Diagram of Candidates Search and Store

This IP finds out all the possible candidates and then stores them into the memory for further processing. The diagram of this IP is divided into two main blocks and is shown in Figure 34: One is DCCL and the other is Label Group.

Figure 35 illustrates the architecture of DCCL. Given a binarized image, DCCL groups and labels pixels based on their connectivity information. The results of DCCL will be stored in the memory of the Label Group block. The Label Group block organizes all the collected labels so that each labelled component's dimension can be measured.

In DCCL, two Dual Port RAM (DPRAM) are used for forming the 2-by-3 mask operation. The reason why DPRAM is used instead of FIFO is that DPRAM is more flexible than FIFO to handle images in different sizes. Here each pixel in DPRAM has N-bit data, and the Nth bit indicates the type of pixel: 1 means black, and 0 means white. The remaining N-1 bits represent the label value. By using a 2-by-3 mask operation, each pixel is assigned a triplet including a label, an equivalent label indicating the connectivity with the same type of component, as well as a list of bw_eq_label indicating the connectivity with different type of components.

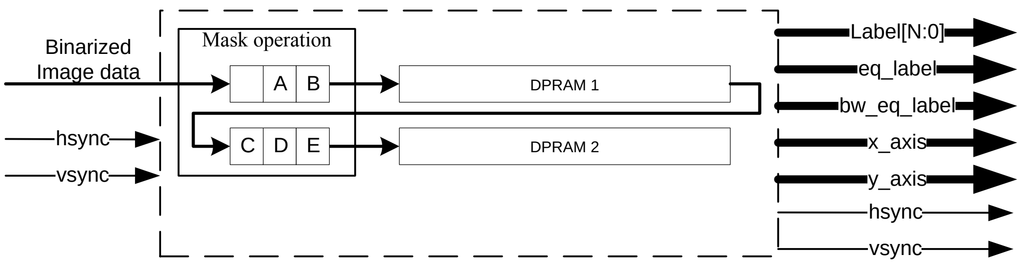

Figure 35: Block Diagram of DCCL

DCCL is a kind of one pass method and processes one pixel at a time. The output label information should be collected and reorganized to get each component's dimension. This is done by the Label Group circuit block shown in Figure 36.

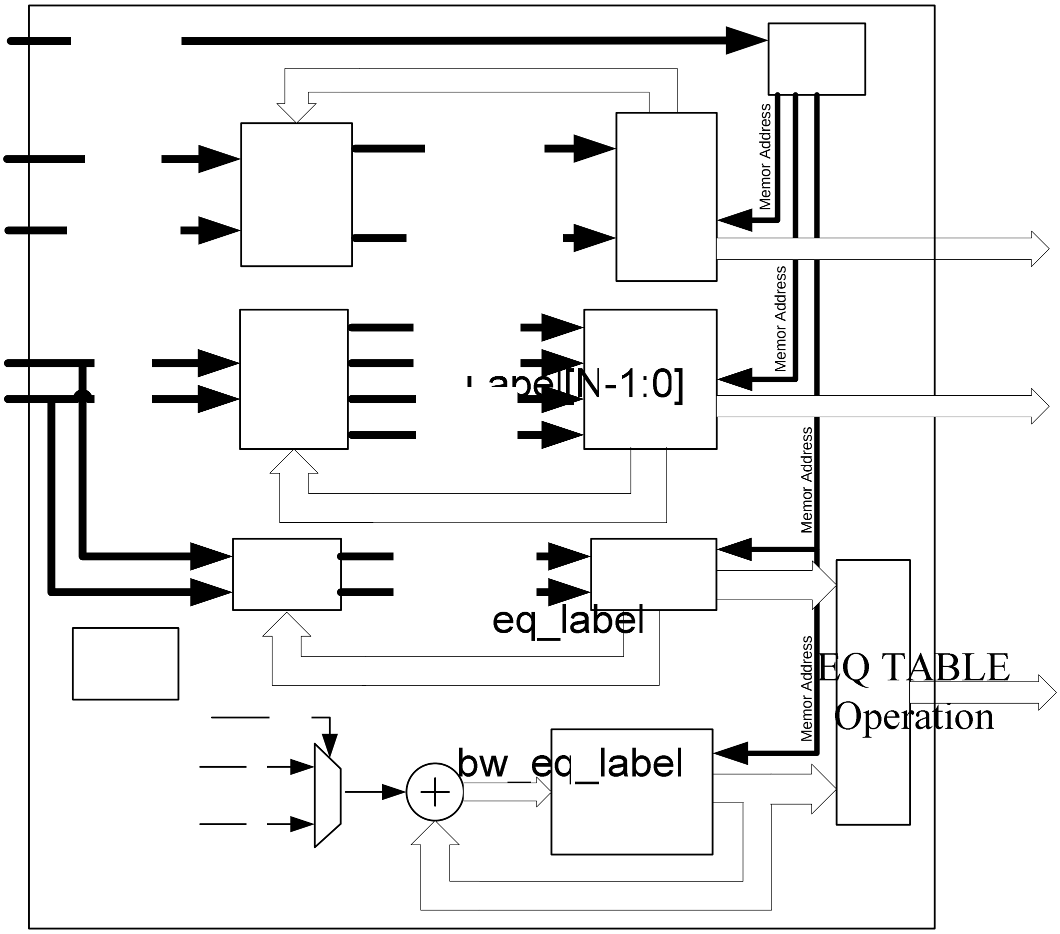

Figure 36: Block Diagram of Label Group

In stead of storing every labeled pixel, Label Group circuit block only records the maximum, minimum and central coordinates of each labeled components and their corresponding equivalent label, so that memory recourses can be saved. The six measured coordinates includes Maximum *x* axis, Maximum *y* axis, Minimum *x* axis, Minimum *y* axis, Central *x* axis and Central *y* axis. The maximum and minimum axis on x and y direction roughly indicates the border of a component, and are only updated by comparing the coordinate of the incoming labeled pixel with the labeled component border. So this operation only uses a comparator. The calculation of central coordinate involves addition and division according to the following equation,

$$x_c = \frac{\sum_{i=1}^{N} x_i}{N}; \quad y_c = \frac{\sum_{i=1}^{N} y_i}{N}, \qquad (4)$$

where (x_c , y_c) refers to the coordinates of the center point of a labeled component, N is the pixel number of a component, and (x_i , y_i) is the ith pixel coordinates of a component.

The final output of the Label Group block consists of the maximum and minimum coordinates on border, one central coordinates, and the BW_EQ label information.

3.4.2.4 Normalization Circuit

Normalization IP is responsible for resizing each candidate image to a predefined size, which is 96-by-96 in this thesis. This is because that every candidate image may have different image size, and it is easier for ID identification function to handle some images with fixed size than to process a series of images with random size. In this thesis, only image downscale is used since the minimum candidate image size is 96×96.

To downscale an image, a conventional bilinear interpolation method is adopted: the incoming image data get through a filter, and the output pixel value of the filter is a weighted average of pixels in the nearest 2-by-2 neighbourhood. The parameters of the bilinear interpolation are selected carefully to reduce FPGA resources. Only the following 4 sets of parameters are used, (1,0), (0.5, 0.5), (0.75, 0.25), and (0.25, 0.75). It is easy to understand that (1, 0) does not require any computation, and the (0.5, 0.5) only needs logic shift right by 1 bit. The parameter 0.25 is done by shit left by 2 bits. In terms of 0.75, the calculation is operated according to the following equation:

$$result = indata \times 0.75 = indata \times 3/4 = (indata + indata \times 2)/4 \qquad (5)$$

As a result, the whole normalization operation is simplified to only involve addition and shift operations, and multiplier is not used.

Figure 37 displays the block diagram of the architecture of normalization IP. Only one FIFO is used to store one row of normalized image data, which will be used to process for interpolation in the column wise direction.

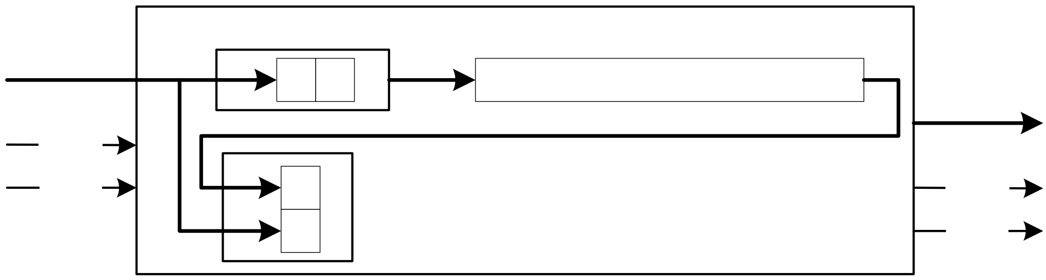

Figure 37: Block Diagram of Image Normalization

Here, since normalization IP only resizes an incoming image to an image in 96-by-96 size, only one FIFO in 96 bytes length is adopted. There are two mask operations in Figure 37: row interpolation mask and column interpolation mask. Each mask contains only two registers. The remaining circuits in this IP are all combination logic circuit such as adders, shifters and multiplexers. In short, this IP can perform a simplified version of image resize function with limited number of scale factors within small area of FPGA logic gates. In addition, this IP is a one-pass circuit and does not require any iterative operation.

<u>*3.4.2.5 Face ID Recognition Circuit*</u>

Face ID Recognition IP consists of majorly 6 circuit blocks:

- ✓ Heart Block Search: this circuit block is to find out the black heart block of a face. This operation requires checking each component and selecting heart block according to the fact that the heart block is a black square nested by another black square. So the ratio of width to height is examined to find a component in square shape. In addition, the geometric size is checked to ignore the background noise which is too big or too small. Furthermore, the center coordinates of a component is checked to make sure that the full blob face can be cropped from original VGA image.

- ✓ White Frame Search: this circuit block finds out the white frame surrounding the black heart block. Just similar to the operation of searching the black heart block, this circuit only checks the size and the center coordinates of each white component. Once a white component is surrounding the black heart block, and its central point is within a certain distance of the center of the black heart block, and then we can make sure to choose a white frame.
- ✓ Black Dots inside White Frame Search: this circuit block finds out the 12 black dots inside the white frame. Combining the information of BW_EQ table with that of white frame, it is easy to find all the black dots inside the white frame.
- ✓ White Dots inside Heart Block Search: this circuit block finds out all the white dots inside the black heart block. Just like the process of Black Dots inside White Frame Search, this circuit just checks the information stored in BW_EQ table to find out all the white dots.
- ✓ ID Calculation: this circuit block calculates the ID of a face. Given all the information provided by the previous four circuit blocks, this circuit makes sure the location of the origin of the face at first, and then calculates the face ID according to the relative position between white dot and the origin. Here a

look up table storing the information of every possible position combination of the origin and white dots are used. So that the operation of the logic can be simplified and the speed of the circuit can be faster.

- ✓ State Machine: this circuit controls the operation of the other logic circuit blocks. Face ID Recognition processes the labelled component in a iterative way. It fetches the stored information of labelled component frequently to compare and sort. So a state machine is used to manage the operation of the whole IP.

The following functional block diagram illustrates the architecture of the IP.

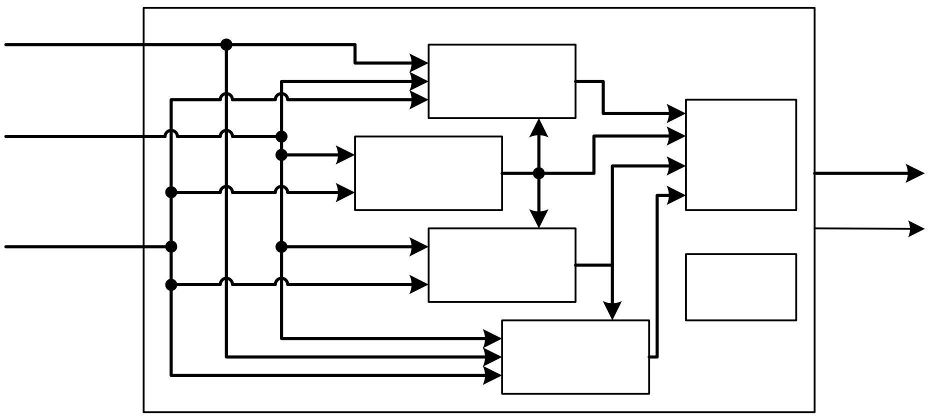

Figure 38: Block Diagram of ID Recognition IP

BW_EQ
Output

Bounding
Axis Output

Hear

4 ANALYSIS OF RESULTS

4.1 IP Design Verification

Verification is a key step in the FPGA design flow. Without complete verification, there is no guarantee that the designed source code is safe to use. As for verification, there are several stages as described below:

- Function simulation: to make sure the correctness of RTL code.
- After synthesis simulation: to make sure the correctness of synthesized net list.
- After PAR (Place and Route) simulation: to verify net list and timing.

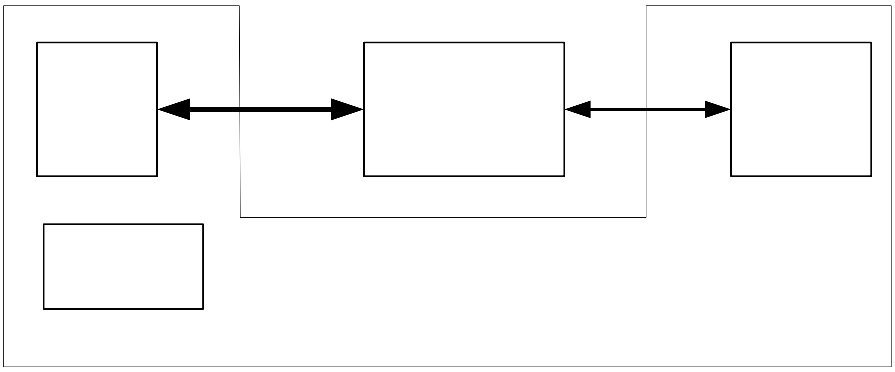

Figure 39: Testbench Architecture

Testbench is an environment built for providing the Device under Test (DUT) with stimulus, and for receiving and analyzing the output of the DUT. Figure 39 illustrates the general architecture of a testbench. DUT communicates with and receives input signals from the Stimulus module. DUT is also connected to a Sink module to output its generated results. The final simulation results are verified in Result

Comparison module. Building testbench is the necessary step for verifying an IP. Every different IP requires a different testbench. The verification method in this work is described as follows,

- ✓ Image Acquisition IP: a dummy camera which can generate a set of predefined dummy data in VGA size is designed for the stimulus of the IP. The output of the IP is automatically checked with the predefined VGA-size dummy data.
- ✓ Gaussian Smoothing IP: the stimulus is a set of images in QQVGA size or normalization size. The output of the IP is compared with the results of the corresponding Matlab code.
- ✓ Binarization IP: the stimulus is a set of images in QQVGA size or normalization size. The output of the IP is compared with the results of the corresponding Matlab code.
- ✓ DCCL IP: the stimulus is a set of images in QQVGA size or normalization size. The output of the IP is compared with the results of the corresponding Matlab code.
- ✓ Normalization IP: the stimulus is a set of images in different size defined in subsection 3.4.1.4. The output of the IP is compared with the results of the corresponding Matlab code.
- ✓ Blob Identification IP: the stimulus is a set of images in normalization size. The output is checked by manual comparison since the output is just the face ID.

- ✓ I2C IP: a custom defined microcontroller interface is used for stimulus and an EEPROM is introduced for verifying the write/read of I2C bus.

4.2 Hardware Environment Description

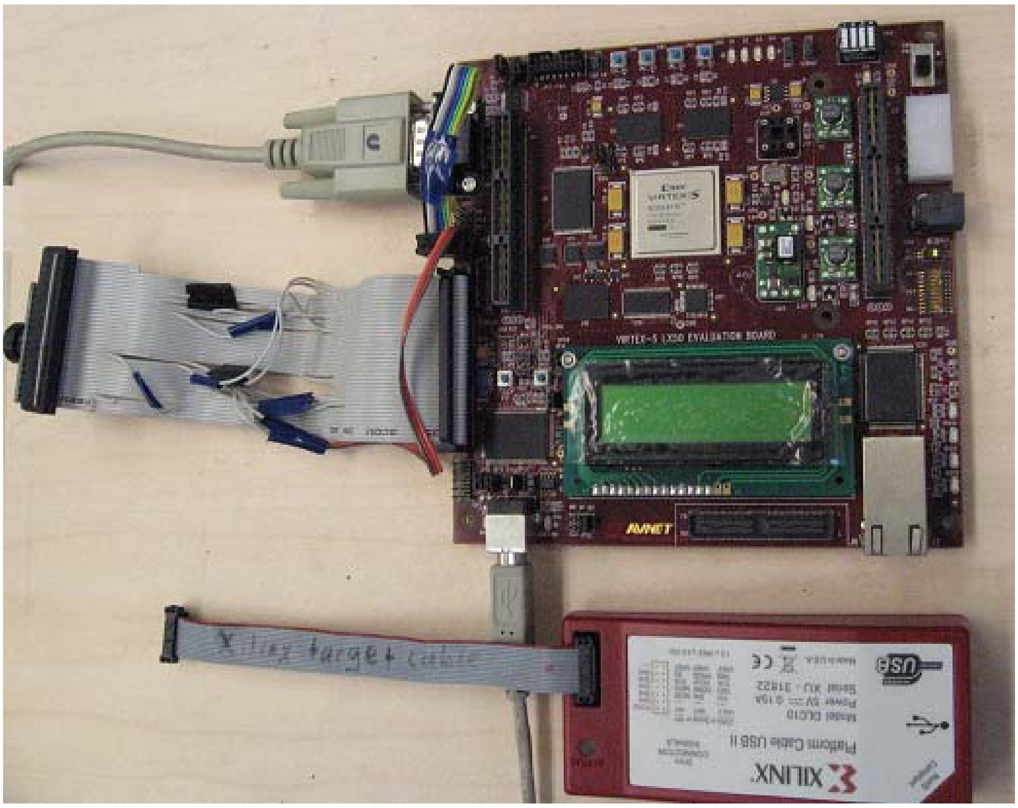

Figure 40: Experiment FPGA Board

The proposed solution is entirely integrated into an AVNET Xilinx XC5VLX110 Evaluation Kit shown in Figure 40. An Omnivision OV10121 camera is manually wired to the FPGA board for capturing video image.

This FPGA board has the following features:

- ➢ FPGA

— Xilinx Virtex-5, XC5VLX110-FF676 FPGA

- ➢ I/O Connectors

— Two EXP™ general-purpose I/O expansion connectors

— One 0.1” USB Debug Header

— One 50-pin 0.1” Header supports Avnet System ACE Module (SAM)

— 80 pin LVDS connector supports 10-bit plus Frame and Clock TX and RX data.

- Memory

— 64 MB DDR2 SDRAM

— 16 MB FLASH

- Communication

— RS-232 serial port

— USB 2.0

— 10/100/1000 Ethernet

- Power

— Regulated 3.3V, 2.5V, 1.8V, and 0.9V supply voltages derived from an external 5V supply

— SSTL2 Termination Regulator

- Configuration

— XCF32P 32Mbit configuration PROM

— Xilinx Parallel Cable IV or Platform USB Cable support for JTAG Programming/Configuration

- Display

— 2x16 character LCD display

4.3 Experiment Results

The Microblaze and the Blob Recognition IP are both running at 100MHz clock frequency, and the DDR2 Memory is running at 200MHz clock frequency.

The consuming time of the Blob Recognition IP can be estimated by adding up the processing time of the three processing steps together.

The first step, image downscale, is just a data decimation operation. It only increases one clock delay, which can be ignored here. The second step, candidate location, consumes about 4.1 milliseconds. However, the consuming time of the third step is unpredictable since it depends on the number of the candidates and the size of the candidates. The more numbers of candidate or the larger the candidate's image size, the more consumed time. However, the candidate's number and the candidate's size will affect each other. If the candidate's size is very big, there must be small number of candidates. Otherwise, a lot of candidates may exist there. In addition, the processing time of normalization circuit varies from 98 microseconds to 2.3 milliseconds when the candidate size changes from 96×96 to 480×480. The remaining processing time of the third step is 450 microseconds if the candidate is a true blob face; otherwise it will decrease accordingly.

To take a common case that there are 7 candidates with medium size as an example, the third step may finish within 10 milliseconds. Hence, the total consuming time for a common case will be at maximum 15 milliseconds, which means that Blob Recognition IP can process a VGA video stream at 66.7 fps. But owing to the band-width of USB2.0 and the performance of the camera, as well as the variety of environment, this system is designed to process VGA image at 32 fps.

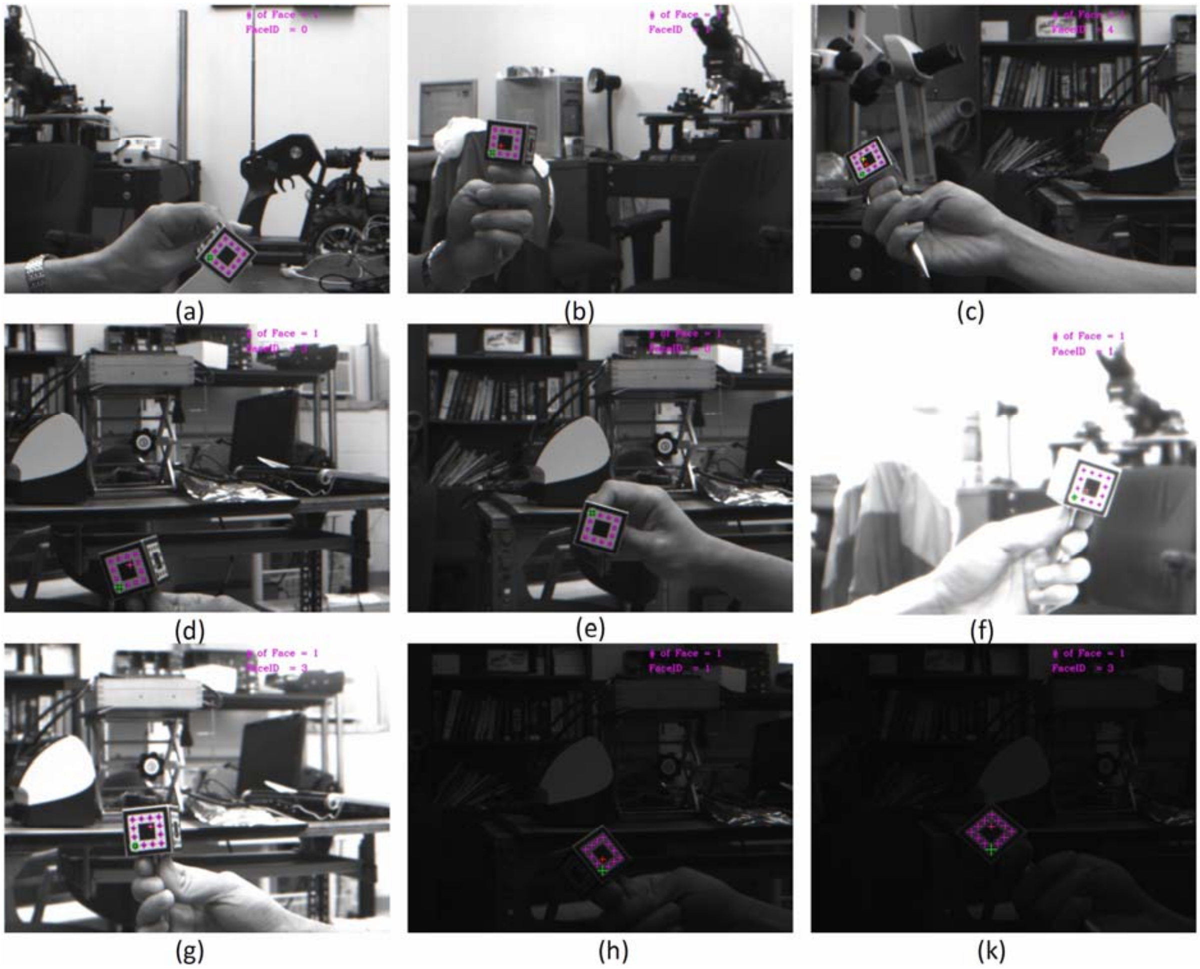

Figure 41: Experiment Results in Different Background

Experiments demonstrated that, at 32 fps speed, the system can reliably process VGA video stream and recognize, within the distance from 0.2m to 0.7m, each blob face over a large range of lighting conditions as long as the target is clearly visible in the video image. Some of the experiment cases are given in Figure 41, in which every detected blob is marked to highlight the process results. Please note that the 9 sub-images in Figure 41 represent 9 different test cases, which are arranged aiming at four variables: the angle of the target, the position of the target in a image: boundary or center, the background objects and the illumination conditions. In these cases the huge difference of the variables cannot fail the work of the system, and thereby demonstrates its robustness.

The consumed FPGA resources are listed in Table 5. We can see that only half of the FPGA resources are occupied, which makes room for integrating more complicated algorithms or adding additional peripherals into the FPGA such as Ethernet.

Table 5: Occupied FPGA Resources Summery

Slice Logic Utilization	**Used**	**Available**	**Utilization**
Slice Registers	21,878	69,120	31%
Slice LUTs	18,212	69,120	26%
Occupied Slices	9,237	17,280	53%
BlockRAM/FIFO	66	128	51%
Total Memory used	2,250	4,608	48%
DSP48Es	15	64	23%
PLL_ADVs	2	6	33%
BUFG/BUFGCTRLs	14	32	43%

There are two interfaces between the FPGA system and the PC: one is a RS232 port and the other is a USB 2.0 port. These two ports can be used for both input and output.

- RS232:
  - Output: Display the results of the face ID recognition.
  - Input: Receive the instructions from user.
- USB:
  - Output: Display the VGA video stream on the monitor of a PC.
  - Input: Receive the instructions from user.

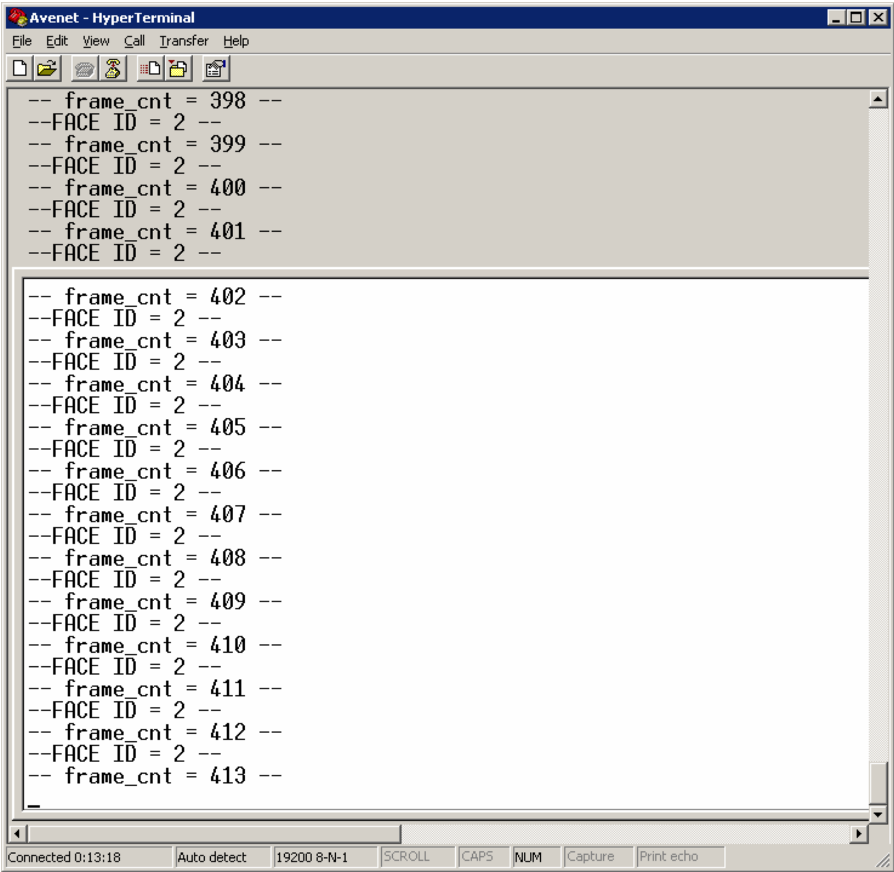

Figure 42: Hyper Terminal GUI

Users can use either RS232 port or USB port to control the operation of the FPGA board. Only the Hyper Terminal application provided by Windows Operation System is required if RS232 interface is used for instruction input. From Figure 42, we can also see that the result of ID recognition is displayed on the GUI of Hyper Terminal.

In terms of the USB port, based on the USB 2.0 Windows driver provided by the Cypress, a Windows GUI application is designed for displaying video stream and inputting instructions, which is shown in Figure 43.

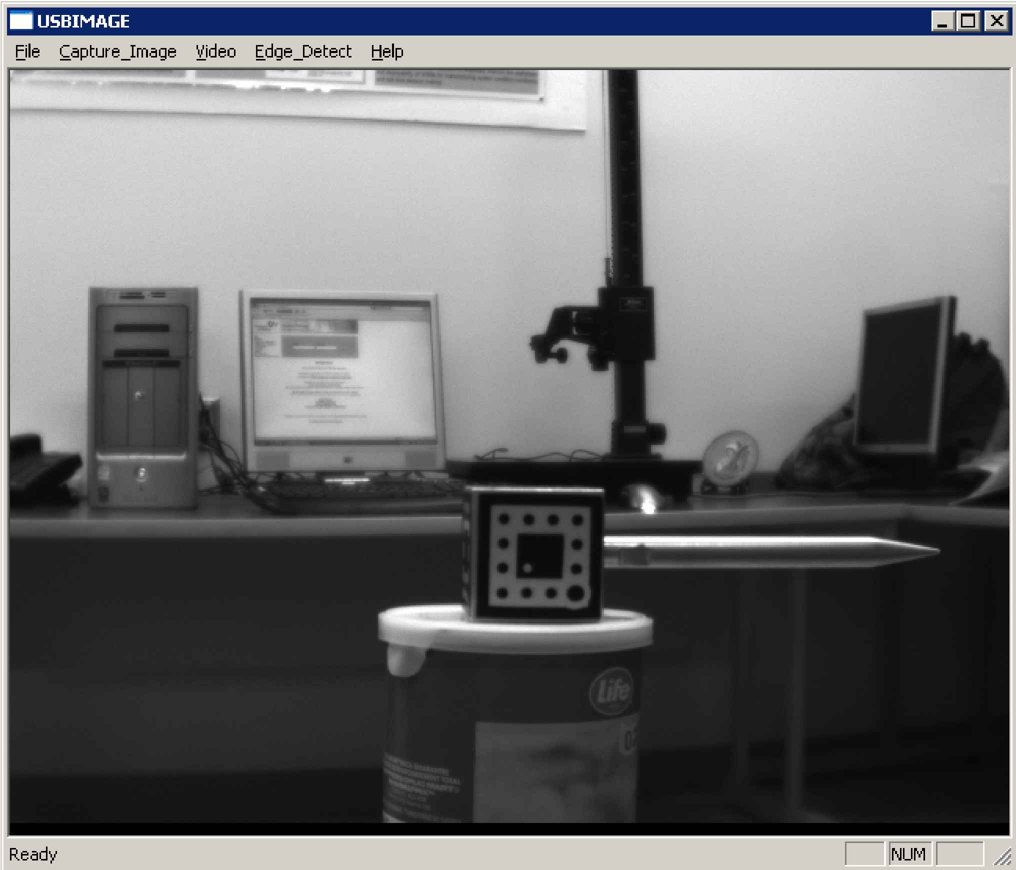

Figure 43: Windows GUI Application

Two menus are supposed to be used frequently. The first menu, Capture_Image, is used to capture just one still image. The second menu, Video, is used for capturing dynamic video image. When different menu is selected, the corresponding instructions are transmitted to FPGA via the USB port, and then the related mode registers of the FPGA circuit are updated.

4.4 Experiment Analysis

FPGA design is a trial and error process, concerning both high level algorithm development and low level logic circuit design. In this thesis, the final architecture and the parameters described previously are the results of iterative experiments and improvement. This thesis also developed some other options and these are described bellow.

4.4.1 The Development of System Architecture

Architecture: Many FPGA implementations in the literature majorly focus on one-pass algorithm. Once the algorithms become complicated, they need iterative operations and require accessing external memory for example, many designers will resort to DSP or CPU. This is because it is inefficient for FPGA to access external memory. In addition, designing memory controller takes a lot of time.

So the initial solution of this thesis is to find a one-pass algorithm that can take advantage of on-chip memory. But three facts make this kind of thought impossible. 1) Image processing needs to process large amount of data; 2) Iteration is a must. The problem in this thesis is in nature cannot be solved with a one-pass algorithm. Hence it is necessary to introduce a memory for temporarily storing the intermediate data. 3) The on-chip memory integrated on the FPGA cannot handle one frame of VGA image data. So it is impossible to fulfill the whole algorithm without the introduction of an external memory.

As a result, MPMC and VFBC are studied and adopted for accessing external DDR2 memory. VFBC is specifically developed by Xilinx for image processing application. Unfortunately, there is something wrong with the VFBC in the version 10.1 ISE Design Suite, which is solved by Xilinx in the version 11.4 version ISE Design Suite. However, this bug in the VFBC costs the author lots of time and energy, and delays the project a lot.

When the MPMC and VFBC are verified to be usable, the remaining IPs are organized in such a way that each IP can access external DDR2 memory respectively. It is believed that only in this way the true parallel processing can be realized, and the parallel processing capability of an FPGA can be fully exploited.

Besides parallel processing, IP reuse is an important aspect in FPGA development. It is not only for reuse in the next project, but also for the reuse in the current project. In this way, the logic resources will be less occupied. For example, Gaussian Smoothing IP and Binarization IP are both reused in different positions of the algorithm flow. But at the beginning of the development, different IPs are used because different mask size is necessary for different image size. Fortunately, after the optimization of the architecture and the parameters, the same IP can be used for images with different size. This will also be addressed in the next subsection again.

4.4.2 The Development of Mask Size

In the proposed algorithm, mask operation or kernel operation is used. A 5-by-5 mask is chosen for Gaussian Smoothing IP, and 11-by-11 is selected for Binarization IP. In addition, the normalization IP uses a 96-by-96 normalization image size.

Please note that the mask size and the normalization size are not decided randomly. On the contrary, they are selected very carefully after many experiments.

It is easy to understand that the bigger the mask size is, the more neighbourhoods will be involved into computation. As a result, in many cases, for example adaptive threshold, bigger mask can behave better. But big mask size will cost too many on-chip memories on FPGA, which is often unaffordable. Usually, 3-by-3, 5-by-5 and 7-by-7 mask are used for FPGA-based image processing. But there are more concerns about mask size in this thesis.

- As for Gaussian Smoothing IP, the bigger the mask size is, the more blurred the image will be. The benefit of a big mask size is to reduce the number of components that are grouped and labelled in DCCL step. As a result, the on-chip memories are reduced and the computation time can be decreased. But the side effect is that it may connect the black heart block of a face to its peripheral black pixels, and then it is difficult to segment the correct candidate image.
- As for Binarization IP, two factors affect its performance. One is the size of the mask; the other is the threshold bias for binarizing each pixel. The performance will be very sensitive to the threshold bias if the mask is too small compared with the normalization image. Figure 44 shows the results segmented by different mask size as well as different threshold bias. In figure 44, A is a normalized candidate image. B is the binarized image by using a 5-by-5 mask with a big threshold bias. We can see that any unnecessary black curves appeare in image B, which is not expected. C is the binarized image by using the same 5-by-5 mask with a small threshold bias. It has much less unexpected black curves than that of B, but introduces some additional white dots inside the black heart block, which are also not expected. D is the binarized image by using an 11-by-11 mask. It is clear that the segmented image is very clean inside the face area and does not contain any unexpected

black or white dots. In addition, it is found that 11-by-11 mask is not sensitive to the threshold bias when the normalization size is 96-by-96.

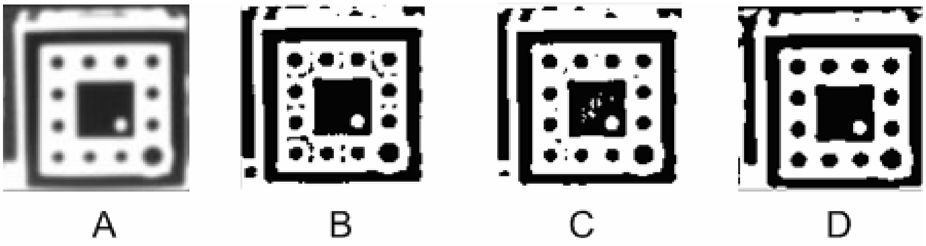

Figure 44: Results of Binerization IP with Different Parameters

5 CONCLUSION AND RECOMMENDATIONS

5.1 Summary of Contributions

Embedded vision systems capable of recognizing objects in real time can be used in a wide range of applications and therefore their demand is increasing day by day. Traditional embedded vision system often adopts embedded CPU or DSP for algorithm implementation. When it comes to an FPGA, most time only glue logic instead of core algorithm is realized on it. This is because that FPGA development is a complicated process requiring both high level (algorithm development and computer architecture, etc,) and low level (logic circuit design, memory management, and clock distribution, etc.) development. It often costs so much time that the implementation of algorithm on FPGA becomes unaffordable.

But FPGA based embedded vision system has the following unique features that lend itself great advantages over other processors, and also make itself a perfect SOC development platform.

- Real hardware parallel signal processing capability
- Flexible reconfigurability
- Abundant interface resources for connecting external devices

The main contribution of this thesis is to develop a FPGA based embedded vision system, which can identify each blob face ID in real time. Differing from many other FPGA platforms that introduce another processor for core algorithm implementation, this system realizes all the algorithms on a single FPGA chip. The architecture of this FPGA system is designed in such a way that each IP block has the capability to access external DDR2 memory so that the parallel processing capability of FPGA can be fully exploited. Besides, each IP is tailored carefully in order to reuse them in the same system and then to reduce occupied logic resources. Hence, fully system-level pipeline operation and compact size scale of FPGA circuit are two important features of this FPGA system.

In addition, it should be pointed out that the algorithm implemented on the FPGA is an updated version of PC based algorithm designed by C language. However, lots of operations used in this PC based software are not suitable for FPGA implementation. Therefore, another contribution of this thesis is to change the original C language software for FPGA implementation. Besides, during this algorithm update, a new Dual Connected Component Labelling method is also proposed.

5.2 Future Work

Under certain lighting conditions and certain view angles, the developed FPGA system may fail to identify a visible blob face. Therefore, there is still room for improving the ID recognition algorithm.

Besides, it is worthwhile to add in more external memories for enhancing parallel processing capabilities. Actually, in this project, the existence of only one external DDR2 memory does impact the performance of the whole system. When FPGA is configured to process VGA video stream at 32 fps rate, and simultaneously to transfer VGA video to a PC via USB2.0 port, the system may become unstable. This is because that the bandwidth of the external DDR2 memory is not big enough to handle such a huge data processing throughput. Hence, the bottleneck of the system lies in the external memory bandwidth. In the future, it is better to add in one more external DDR2 memory to ease this bottleneck.

Furthermore, based on the designed embedded vision system, more complicated algorithms can be integrated to realize more advanced application. For example, it is possible to extend the current algorithm to detect the orientation and the 3-D location of the blob target within a given space by using one or more cameras. This will concern 3-D object identification and tracking, as well as the resultant camera calibration. Based on the current designed FPGA system, the period of further development can be shorten a lot since only the Blob Recognition IP is required to be updated for the new algorithm.

REFERENCES

[1] K. T. Gribbon, D. G. Bailey, A. Bainbridge-Smith, “Development Issues in Using FPGAs for Image Processing,” Image and Vision Computing New Zealand, pp. 217–222, 2007.

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

[3] C.T. Johnston, 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, D Venkateshwar Rao, “Image Processing Algorithms on Reconfigurable Architecture Using Handel-C,” Journal of Engineering and Applied Sciences, vol. 1, no. 2, pp. 103-111, 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.

[8] A. Yilmaz, O. Javed, M. Shah, "Object Tracking: A Survey," ACM Journal of Computing Surveys, Vol. 38, No. 4, 2006

[9] M. Venkatesan, 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.

[10] Trucco, Alessandro Verri, “Introductory Techniques for 3D Computer Vision”, Prentice Hall, March 16, 1998

[11] Bruce G. Batchelor, Paul F. Whelan, “intelligent vision systems for industry,” Springer; 1st Edition, June 13, 1997

[12] Junichi Nakamura, “Image Sensors and Signal Processing for Digital Still Cameras,” CRC Press, August 5, 2005

[13] Rafael C. Gonzalez, Richard E. Woods, “Digital Image Processing”, Prentice Hall; 3 edition, August 31, 2007

[14] Ramakant Nevatia, “Machine Perception”, Prentice-Hall, Inc., 1982

[15] David Vernon, “Machine Vision, Automated Visual Inspection and Robot Vision,” Prentice Hall International (UK), 1991

[16] Robert B. Fisher, “From Surface To Objects: Computer Vision and Three Dimensional Scene Analysis”, John Wiley & Sons Inc., June 1989.

[17] Roger Woods, John McAllister, Gaye Lightbody, Ying Yi, “FPGA based Implementation of Signal Processing Systems”, John Wiley & Sons, Ltd, 2008

[18] Axel Jantsch, Hannu Tenhunen, “NETWORKS ON CHIP”, Springer; 1 edition, January 2003

[19] Scott Hauck, Andre DeHon, “Reconfigurable Computing, The theory and Practice of FPGA Based Computation”, Elsevier Inc, 2008

[20] AMOS R. OMONDI, JAGATH C. RAJAPAKSE, “FPGA Implementations of Neural Networks”, Springer, 2006

[21] Steve Kilts, “Advanced FPGA Design, Architecture, Implementation, and Optimization”, John Wiley & Sons, Inc., 2007

[22] E. R. DAVIES, “MACHINE VISION Theory Algorithms Practicalities”, Elsevier Third Edition, 2005

[23] “Virtex-5 User Guide”, UG190 (v3.2) December 11, 2007

[24] “Virtex-5 FPGA Configuration User Guide”, UG191 (v3.1) April 25, 2008

[25] “Virtex-5 FPGA Data Sheet DC and Switching Characteristics”, DS202 (v3.7) December 11, 2007

[26] “Device Driver Programmer Guide”, Xilinx, v1.4 - June 22, 2007

[27] “BFM Simulation in Platform Studio”, Xilinx UG644 EDK 11.2

[28] “MicroBlaze Processor Reference Guide, Embedded Development Kit EDK 11.4”, UG081 (v10.3)

[29] “Multi-Port Memory Controller (MPMC) v5.03.a”, DS643 September 16, 2009

[30] “PLB Usage in Xilinx FPGAs”, Xilinx, September 2005

[31] “Embedded Processor Block in Virtex-5 FPGAs, Reference Guide”, UG200 (v1.7) October 6, 2009

[32] “PLBV46 Interface Simplifications”, SP026 (v1.2) December 5, 2008

[33] “Platform Specification Format Reference Manual, Embedded Development Kit (EDK)”, UG642, EDK 11.2

[34] “PLB v3.4 and OPB to PLB v4.6 System and Core Migration User Guide”, UG443 (v1.4) August 11, 2008

[35] "EDK Concepts, Tools, and Techniques, A Hands-On Guide to Effective Embedded System Design", UG683 EDK 11

[36] "ChipScope Pro 11.4 Software and Cores User Guide", UG029 (v11.4) December 2, 2009

[37] "AccelDSP Synthesis Tool User Guide", UG634 (v11.4) December 2, 2009

[38] "MATLAB for Synthesis Style Guide", UG637 (v11.4) December 2, 2009

[39] Jianbo Shi, Carlo Tomasi, "Good Features to Track," IEEE Conference on Computer Vision and Pattern Recognition (CVPR94) Seattle, June 1994

[40] Eli Brookner, "Tracking and Kalman Filtering Made Easy", John Wiley & Sons, Inc., 1998

[41] "EZ-USB Technical Reference Manual", Document # 001-13670 Rev. *A, Cypress

[42] "Migrating From EZ-USB FX2(TM) to EZ-USB FX2LP(TM)", AN4078_C, Cypress, June 8, 2005

[43] "Endpoint FIFO Architecture of EZ-USB FX1-FX2(TM)", AN4067, Cypress, October 19, 2004

[44] "Datasheet of OV10620/OV10121 CMOS WVGA High Dynamic Range (HDR) Camera Chip Sensor", Version 2.0, August 27, 2008, Omnivision

[45] Javier Díaz, Eduardo Ros, Francisco Pelayo, "FPGA-Based Real Time Optical Flow System," IEEE transactions on circuits and systems for video technology, vol. 16, no. 2, February 2006

[46] Hussein M. Alnuweiri, "Parallel Architectures and Algorithms for Image Component Labeling," IEEE transactions on pattern analysis and machine intelligence, vol. 14, no. 10, October 1992

[47] D.Crookes, K.Benkrid, A.Bouridane, "Design and implementation of a high level programming environment for FPGA-based image processing," IEE Proc.-Vis. Image Signal Process., Vol. 147, No. 4, August 2000

[48] Ping Zhou, Yue Liu, Yongtian Wang, "Adaptive Real-time Labeling and Recognition of Multiple Infrared Markers Using FPGA," 2009 Fifth International Conference on Image and Graphics

[49] Philip H.W. Leong, "Recent Trends in FPGA Architectures and Applications," 4th IEEE International Symposium on Electronic Design, Test & Applications

[50] Robert M. Haralick, Linda G. Shapiro, "Computer and Robot Vision, Volume 1", Addison-Wesley Publishing Company, Inc., 1992.

[51] S. Jin, J. Cho, X. D. Pham, K. M. Lee, "FPGA Design and Implementation of a Real-Time Stereo Vision System," IEEE transactions on circuits and systems for video technology, vol. 20, no. 1, January 2010

[52] Javier Díaz, Eduardo Ros, Francisco Pelayo, "FPGA Based Real Time Optical Flow System," IEEE transactions on circuits and systems for video technology, vol. 16, no. 2, February 2006

[53] Vinod Nair, Pierre-Olivier Laprise, "An FPGA-Based People Detection System," EURASIP Journal on Applied Signal Processing 2005:7, 1047–1061

[54] Clement Farabet, Cyril Poulet, Yann LeCun "An FPGA-Based Stream Processor for Embedded Real-Time Vision with Convolutional Networks," 2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops

[55] Nicolas Farrugia, Franck Mamalet, “Fast and Robust Face Detection on a Parallel Optimized Architecture Implemented on FPGA,” IEEE transactions on circuits and systems for video technology, vol. 19, no. 4, April 2009

[56] Branislav Kisacanin, Shuvra S. Bhattacharyya, Sek Chai, “Embedded Computer Vision (Advances in Pattern Recognition),” Springer; 1st Edition. October 9, 2008

[57] David Pellerin, Scott Thibault, “Practical FPGA Programming in C”, Prentice Hall PTR, April 22, 2005

VITA AUCTORIS

Jian Xiong was born in 1978 in Sichuan Province, P.R. China. In the year of 2000, he received his Bachelor degree in Electrical Engineering from Chongqing University, China. In 2005 he received his Master of Science degree in Electronic Engineering from the University of Electronic Science and Technology of China. He has been working as a hardware and FPGA designer in several IC design houses for several years. In the year of 2009, he went to Canada to pursue another graduate studies. He has been working as a research assistant under the supervision of Professor Jonathan Wu (CRC in automotive sensor systems) in the last two years.