

University of Windsor

Scholarship at UWindor

Electronic Theses and Dissertations

Theses, Dissertations, and Major Papers

2008

A Compact Camera with a Reconfigurable Real-time Embedded Image Processor for Pharmaceutical Capsule Inspections

Anthony Karloff
University of Windsor

Follow this and additional works at: <https://scholar.uwindsor.ca/etd>

Recommended Citation

Karloff, Anthony, "A Compact Camera with a Reconfigurable Real-time Embedded Image Processor for Pharmaceutical Capsule Inspections" (2008). *Electronic Theses and Dissertations*. 8030.
<https://scholar.uwindsor.ca/etd/8030>

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

A Compact Camera with a Reconfigurable Real-time Embedded Image Processor for Pharmaceutical Capsule Inspections

by

Anthony Karloff

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
2008



Library and
Archives Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*
ISBN: 978-0-494-47052-7
Our file *Notre référence*
ISBN: 978-0-494-47052-7

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.


Canada

© 2008 Anthony Karloff

All Rights Reserved. No Part of this document may be reproduced, stored or otherwise retained in a retrieval system or transmitted in any form, on any medium by any means without prior written permission of the author.

Declaration of Co-Authorship/Previous Publication

I. Co-Authorship Declaration

This thesis also incorporates the outcome of a joint research undertaken in collaboration with Neil Scott and Mohammad Islam under the supervision of Dr. Roberto Muscedere and Dr. Majid Ahmadi. The collaboration is covered in Chapter 2 of the thesis. In all cases, the key ideas, primary contributions, experimental designs, data analysis and interpretation, were performed by the author.

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.

II. Declaration of Previous Publication

This thesis includes one original paper that has been previously published in a peer reviewed journal, as follows:

Thesis Chapter	Full Citation	Status
Chapter 2	A. Karloff, N. Scott, and R. Muscedere. A flexible design for a cost effective, high-throughput inspection system for pharmaceutical capsules. In Proc. IEEE International Conference on Industrial Technology, April 2008	Published

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 anyones 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 copyrighted 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

The following thesis presents the system requirements, design methodology, final hardware design and system integration of a custom digital camera for high-speed pharmaceutical capsule inspections.

The primary goals of the camera design were to minimize the cost of the device and to have a flexible design that could be easily upgraded in the future. For this application, a 3.1 mega pixel CMOS image sensor was used with a USB 2.0 interface. In addition, the custom camera can pre-process image data in an embedded, reconfigurable real-time image processor implemented in a FPGA. All data processing in the camera occurs with only buffering four rows of an image, eliminating the need for RAM on the device and lowering the overall cost.

The final design was manufactured and implemented into a complete inspection system which used 16 of these cameras to inspect up to 60 000 capsules per second.

To my family and friends for their endless support and encouragement. This work would not have been possible without you.

Acknowledgments

There are several people who deserve to be acknowledged for their generous contributions to this project. I would first like to express my sincere gratitude and appreciation to Dr. Roberto Muscedere, my supervisor, for his invaluable guidance and involvement throughout the course of this thesis. I would also like to extend a very special thanks to Dr. Majid Ahmadi and Dr. Maher Sid-Ahmed for their expert guidance, encouragement and constant support throughout my studies.

Contents

Declaration of Co-Authorship/Previous Publication	iv
Abstract	vi
Dedication	vii
Acknowledgments	viii
List of Figures	xiii
List of Tables	xv
List of Abbreviations	xvii
1 Introduction	1
1.1 Project Overview	2
1.2 Quality Control of Two Part Gelatin Capsules	2
1.3 Methods of Defect Detection	3
1.3.1 Current Inspection Methods	3
1.3.2 Introduction to Machine Vision	5
1.3.3 Commercial Systems	6
1.4 Proposed Solution	7

1.5	Thesis Organization	7
2	System Overview	9
2.1	Introduction to the Optisorter	9
2.1.1	Background Information	9
2.1.2	Summary of Operation	10
2.2	System Timing	11
2.3	Imaging Environment	13
2.4	Objectives for Modification	13
2.4.1	Selecting a Camera	14
2.4.2	Modifying the Imaging Environment	16
2.5	Proposed Design	18
3	Camera Design Methodology	21
3.1	Camera Design Flow	21
3.2	Design Specifications	23
3.3	Major Design Methods	23
3.3.1	Reconfigurable Hardware Programming	24
3.3.2	Simulation Tools	24
3.3.3	Prototyping Hardware	25
3.4	Testing	25
4	Hardware Design	26
4.1	Component Selection	26
4.1.1	Imaging Sensors	27
4.1.2	Reconfigurable Devices	28
4.1.3	Communication Interface	31
4.1.4	Component Summary	32

4.2	Circuit Schematics	32
4.2.1	Micron MT9T001 CMOS Imaging Sensor	32
4.2.2	Xilinx Spartan-3E500 FPGA	33
4.2.2.1	I/O Connections	33
4.2.2.2	Clock Connections	34
4.2.2.3	Power Connections	34
4.2.3	TPS Triple Supply	35
4.2.3.1	Limiting Buck Converter Current	36
4.2.3.2	Setting VCCO	37
4.2.3.3	Sizing Soft Start Capacitors	37
4.2.4	Cypress FX2 USB Microcontroller	38
4.2.5	I ² C Communication Bus	39
4.3	PCB Layout	40
4.3.1	PCB specifications	41
4.3.2	Component Placement	43
4.3.3	Power and Grounding	46
4.3.4	Bypass capacitors	47
4.3.5	Routing	51
4.3.6	Manufacturing Files	52
5	HDL Blocks and Programming	54
5.1	FPGA Programming Overview	54
5.2	Frame Timing and Data Synchronization	55
5.3	I ² C Write Slave	57
5.3.1	I ² C Bus Overview	58
5.3.2	I ² C Slave VHDL implementation	59
5.4	Asynchronous FIFO	61

5.5	Output Controller	62
5.6	Image Processing Block	64
5.7	Trigger Delay	65
5.8	Synthesis Constraints and Results	65
6	Image Processing	68
6.1	Demosaicking	68
6.2	Hardware Implementations	71
6.3	Edge-Enhanced Real-Time Hardware Demosaicking	73
6.4	Implementation and Results	77
7	Conclusion	79
7.1	System Integration	79
7.2	Summary	82
7.3	Future Work	84
	References	86
A	System Requirements	88
B	Camera Board Schematics	92
C	VHDL Code	99
D	MATLAB Code	124
	VITA AUCTORIS	129

List of Figures

1.1	Typical Defects in Gelatin Capsules	4
2.1	Optisorter External View	10
2.2	Quadrant 1 Detailed	12
2.3	Camera Configuration	14
2.4	System Block Diagram	18
2.5	Camera Block Diagram	19
3.1	Hardware Design Flow	22
4.1	DCM with Off-Chip Delay Feedback	35
4.2	PCB Design Flow	41
4.3	PCB Partitions and Final Layout	44
4.4	PCB Power Plane Layout	47
4.5	Bypass Capacitor Values	48
4.6	Bypass Capacitor Impedance Profile [1]	48
4.7	Capacitor Placement Inductive Loop[1]	50
4.8	Capacitor Impedance and Resonant Frequency[1]	50
4.9	Bypass Capacitor Critical Current Path	51
5.1	FPGA VHDL Modules and Data Flow	55

5.2	MT9T001-3100 Timing [10]	56
5.3	MT9T001-3100 Synchronization Flow	57
5.4	I ² C Bus Communication	59
5.5	I ² C FSM	60
5.6	I ² C write in FPGA.	61
5.7	Timing for Cypress FX2 Slave FIFO	63
6.1	Bayer Pattern CFA on a CMOS Image Sensor	69
6.2	3x3 Data Window for Bilinear Interpolation	71
6.3	5x5 Data Window for Edge Weight Function	73
7.1	Camera PCB in an Enclosure	80
7.2	Final Capsule Images	81
B.1	TPS Triple Supply Schematic	94
B.2	Spartan-3E FPGA	95
B.3	Cypress FX2 USB Microcontroller	96
B.4	Micron Sensor Schematic	97
B.5	I2C Component Schematic	98

List of Tables

1.1	Current MV Capsule Inspection Systems and Proposed Model	6
1.2	Proposed MV Components	7
3.1	Design Specification Summary	23
4.1	Spartan FPGA Summary[19]	30
4.2	Component Summary	32
4.3	Spartan-3E Supply Voltage Ramp Rate[21]	36
4.4	I ² C Devices: Loading Capacitance	39
5.1	Write Controller Cases	64
5.2	FPGA Utilization with Edge Enhanced Demosaicking	66
5.3	FPGA Utilization without Edge Enhanced Demosaicking	67
6.1	Bilinear Output	72
6.2	Demosaicking Results	78
A.1	Defect List and Tolerances[5]	88
A.2	High Level Business Requirements[5]	90
A.3	High Level Performance Requirements[5]	90
A.4	Business Scenarios[5]	91

B.1 Bill of Materials 92

List of Abbreviations

CFA	Color Filter Array
DCM	Digital Clock Manager
DSP	Digital Signal Processing (Processor)
EEPROM	Electrically Erasable PROM
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
HMI	Human Machine Interface
I/O	Input/Output
I ² C	Inter Integrated-circuit Communication
IC	Integrated Circuit
LED	Light-emiting Diode
LUT	Look Up Table
MV	Machine Vision
PAL	Phase Alternating Line
PCB	Printed Circuit Board
PLC	Programmable Logic Controller
PROM	Programmable ROM
RAM	Random Access Memory
ROM	Read Only Memory
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

Chapter 1

Introduction

As technology advances, there is an ever increasing demand for faster, smaller and more affordable technology in all corners of industry. The field of machine vision is no exception. Advances in imaging devices and computing power has allowed machine vision based inspection systems to increasingly appear in a variety of industrial and manufacturing settings. The manufacturing of pharmaceutical gelatin capsules is an extremely high volume and high throughput manufacturing environment which has a strong need for quality control inspections to take place. However the intricacies of performing such inspections make both an effective and affordable solution a challenge to develop.

1.1 Project Overview

The objective of this project was to work in collaboration with a local pharmaceutical capsule manufacturer, Pharmaphil Inc., with funding from the Ontario Centres of Excellence (OCE), to develop a cost effective prototype inspection system for two-part gelatin capsules.

The development of this system entailed upgrading and retrofitting an existing capsule sorting device, the Optisorter, to perform detailed, high-throughput visual inspection of these capsules at a low cost. The overall project was divided into three major parts to be developed by individual students. These included: development of control hardware and a PC interface, development of image processing software, and finally, the development of custom digital cameras for image acquisition, construction and processing. The final contribution listed above is the focus of this thesis along with contributions to co-ordinating system timing and modifying the imaging environment.

1.2 Quality Control of Two Part Gelatin Capsules

The manufacturing of pharmaceutical two part gelatin capsules is a highly sensitive process to both environmental and process variations which lead to undesired flaws in some of the product. Currently there are limited methods of quality control that provide a flexible, accurate and cost-effective solution. As a single capsule is essentially valueless, neither time nor expense can be afforded to the quality assurance of the product [8], yet each capsule must be fully inspected for potential defects so that the manufacturer can provide a marketable quality guarantee for their product. The ability to ensure that the capsules are within certain manufacturing specifications and free of defects, without adding substantial cost the process, can give the manufacture

an edge in sales and increase profit in this highly competitive market.

The two-part telescoping gelatin capsule was patented in London in 1847 by James Murdock [14]. They are made in two parts by dipping metal rods in a liquid gelatin. The two ends are trimmed, and supplied as partially closed units to various pharmaceutical companies who then separate, fill and close the two halves.

Typical defects found in these two part gelatin capsules include, but are not limited to: holes, dents, bubbles, missing halves, incorrect dimensions, and foreign product (such as a different colour or sized capsule). A complete list of defects required for detection is provided in Table A.1 and some examples of these defects can be seen in Fig. 1.1. The cost of discarding a defective capsule is negligible especially when compared to the potential cost that could be incurred by its accidental distribution. Defective capsules can disrupt the filling process performed by drug companies that purchase the product, leak contents into packaging, or in the case of a foreign capsule (such as a red pill appearing in a batch of blue pills) promote a lack of confidence in the drug distributor or even pose legal issues. Hence, there is a strong desire for quality assurance in the manufacturing of these capsules. The following sections detail current methods of defect detection, state of the art solutions on the market and the advantages of a custom Machine Vision (MV) solution.

1.3 Methods of Defect Detection

1.3.1 Current Inspection Methods

There are currently two primary methods being used to inspect two part gelatin capsules for quality assurance. The simplest method is manual inspection, whereby an individual attempts to identify defective capsules as they pass through an inspection station. This usually involves a large quantity of capsules moving over a conveyor belt

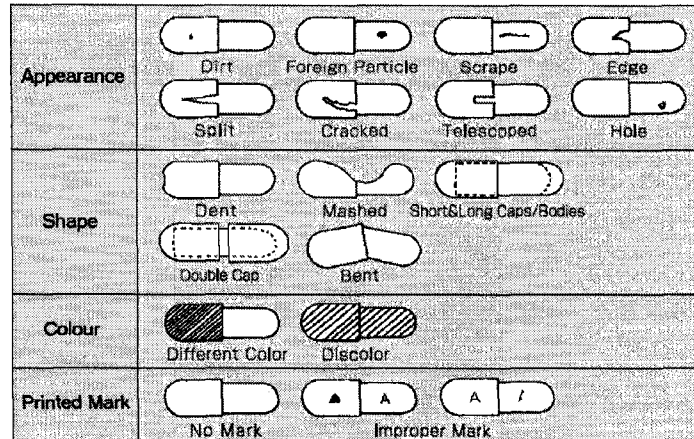


Figure 1.1: Typical Defects in Gelatin Capsules

that is illuminated from the bottom. Although this is effective for processing very large quantities at a very fast rate, the accuracy in which the capsules are inspected is greatly compromised. This is because not every capsule can be fully inspected by the individual and the integrity of the inspection fluctuates greatly due to human error, fatigue, and focus. Also, the capsules are susceptible to areas of occlusion where they may overlap or touch one another, making it impossible for every capsule to be fully inspected. Generally, if an excessive number of defective capsules are identified, the entire manufactured batch is discarded, adding greatly to the overall manufacturing cost. The benefits of such a manual system are the reduced equipment and maintenance cost, especially where labour is relatively inexpensive.

The second predominate method of capsule inspection involves the use of machine vision systems to attempt to identify defective capsules. While the benefits of such systems include improved accuracy and consistency in the inspection of each capsule, they are typically very expensive to setup, may suffer reduced inspection rates depending on the hardware and demand on the system and generally lack flexibility and the ability to upgrade.

1.3.2 Introduction to Machine Vision

A Machine Vision (MV) System is the application of computer vision to industry and manufacturing. A typical machine vision system consists of several of the following nine components[16]:

1. One or more digital or analog cameras (black-and-white or colour) with suitable optics for acquiring images.
2. Camera interface for digitizing images (widely known as a “Frame grabber”).
3. A processor, often a PC or embedded processor, such as a DSP. In some cases, all of the above are combined within a single device, called a “Smart Camera.”
4. Input/Output hardware (e.g. digital I/O) or communication links (e.g., network connection or RS-232) to report results.
5. Lenses to focus the desired field of view onto the image sensor.
6. Suitable, often very specialized, light sources (LED illuminators, fluorescent or halogen lamps, etc.).
7. A program to process images and detect relevant features.
8. A synchronizing sensor for part detection (often an optical or magnetic sensor) to trigger image acquisition and processing.
9. Some form of actuators used to sort or reject defective parts. Each of the following must be carefully considered and customized for a specific application.

In this thesis, the complete specifications of the MV system used for this application will not be discussed in extensive detail, as the focus of this document is on the imaging sensor.

1.3.3 Commercial Systems

There are currently a variety of MV systems on the market targeted at defect detection of pharmaceutical gelatin capsules and tablets. These range in price from \$4,200 USD to over \$600,000 USD. Table 1.1 outlines a few current systems as well as the specifications for the proposed system. The approach these systems use to acquire and process images varies greatly. Almost all the systems listed rely on line scan cameras that process visual information as it is acquired. This benefits the system by working extremely fast, but it limits the variety and accuracy of inspections that can take place because only a single line of the image is available to process. In addition, storing images of defective capsules is difficult unless image buffering occurs in the system. Table 1.1 shows that regardless of the type of camera used, generally an inspection rate of at least 60,000 capsules per hour is attained by these systems.

Some current systems such as the CVIS-SXX-E consist of a very elaborate mechanical system which adds significantly to the cost of the overall system. This seems typical for most of the systems, as fixturing the capsules proves to be a difficult task. Finally, while an average of 100 micron resolution is maintained, each system does vary to some degree in detail and resolution.

System Model	Cost	Caps/h	Camera Type	Processing	Resolution
CTI-1	\$4,200	50,000	Unknown	Unknown	0.1mm
InspeCaps 150	Unknown	120,000	3x Linescan CCD	Visicard 4	0.1mm
CVIS-SXX-E	\$600,000	100,000	8x Linescan CCD	Analog Sig.	0.1mm
MVT	\$350,000	60,000	Unknown	Unknown	0.2mm
Proposed	<\$35,000	60,000	12x CMOS	Digital PC	0.01mm

Table 1.1: Current MV Capsule Inspection Systems and Proposed Model

1.4 Proposed Solution

The proposed system maintains the competitive throughput requirements of competing systems with an inspection rate of 60,000 capsules per hour, while showing greatly reduced cost and an increase in inspection accuracy. This is achieved by taking a completely customized approach to the hardware of the system and combining it with an affordable existing mechanical design. The additional benefits of such an approach lay in both the flexibility and the ability to upgrade the proposed system. Table 1.2 outlines the general MV components used in the proposed system and the advantages of using these components over existing MV systems.

Component	Typical	Proposed	Advantage
Camera	CCD Linescan	CMOS	Full digital image
Interface	Frame Grabber	USB 2.0	Cost effective
Processor	DSP	PC	Easy to upgrade
Lens	Standard	Standard	None
Lights	Red LED	White LED	Full colour images
IP Program	Custom	Custom	Flexible to change
Sync. Sensors	Unknown	Inductive proximity.	Easy to interface
Reject Mechanism	Mechanical	Air actuator.	Touchless

Table 1.2: Proposed MV Components

1.5 Thesis Organization

This thesis discusses the design, build and testing of a custom digital camera used as part of a MV system for quality control in the manufacturing of two part gelatin capsules. Chapter 2 begins by giving an overview of the proposed MV system including the current mechanical setup, imaging environment and system timing constraints. Following this, Chapter 3 discusses the design methodology for the development of the camera component of the MV system. This includes the design specifications for

the camera, the design flow methodology used for high level device design and finally introduces the development tools and equipment used for programming, simulating and testing the design. Chapter 4 discusses the actual hardware design of the camera including detailed schematic designs as well as physical component layout and PCB design consideration for the final camera. Chapter 5 will detail the VHDL code developed for the FPGA on the camera and will discuss the various blocks and their role on the camera. Chapter 6 introduces the image processing elements of the camera, specifically discussing Colour Filter Array (CFA) imaging sensor data and the use of “demosaicking” techniques to perform image reconstruction. This chapter will also cover the software simulation and hardware implementation of several demosaicking methods including a novel real-time edge enhancement method proposed in this thesis. Finally, Chapter 7 will discuss the integration of the camera with the MV system as well as conclude the work and provide a discussion for future development of the system.

Chapter 2

System Overview

2.1 Introduction to the Optisorter

2.1.1 Background Information

The Optisorter, seen in Fig. 2.1, was a German engineered MV system built in the early 1990s. A number of these systems were acquired by Pharmaphil Inc. to be implemented as an affordable quality control method for the manufacturing of their size #00-#5 two part gelatin capsules. Although the Optisorter has a solid mechanical foundation for an MV system, the hardware was essentially obsolete. The exact functionality of the Optisorter is still unknown, however the hardware contained therein gives a good indication of what functions this system may have performed.

With only analog PAL cameras and analog processors, the existing machine was most likely only able to identify foreign capsules and measure basic geometric tolerances such as the length and width of the capsule.

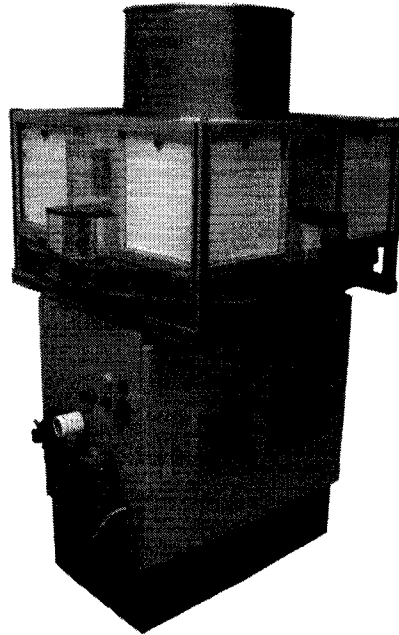


Figure 2.1: Optisorter External View

2.1.2 Summary of Operation

The system original system is comprised of four identical inspection stations designed to operate in parallel to provide the desired system throughput. Capsules are initially loaded into a large hopper seated on top of the machine. This hopper feeds a series of 24 radial arms that rotate counter clockwise within the system. As an arm enters a quadrant, the holder is first cleared of any stray contents such as a double loaded capsule from the previous station. Next, a loading mechanism allows a single capsule to descend down the arm into a holder that seats the capsule. The arm continues to spin until the capsule passes beneath a series of four cameras which are triggered by proximity sensors that track the arm positions. The four photographs are inspected using simple analog circuits and an accept or reject decision is made as the capsule passes over an air valve actuator responsible for ejecting the capsule into the appropriate bin.

2.2 System Timing

To understand the most important factors in system timing, let an *event* represent an operation that will be occurring in the system when an arm arrives at a certain location and a *cycle* represent the time it takes for a new arm to arrive at an event, in other words, the time between arms. Major events will include:

1. Clearing the holder.
2. Loading the holder.
3. Imaging the top of the capsule.
4. Imaging the bottom of the capsule.
5. Accepting the capsule.
6. Rejecting the capsule.

Fig. 2.2 shows the details of quadrant one with six of the 24 arms that will appear in the quadrant at a single cycle. The general location of the enumerated events above are show as circled numbers.

It is important to note that because each event is performed for a new arm on every cycle, the amount of time allocated for an event is dependent on the cycle time and not on the amount of time between events. This means greater time allocation for critical events can only be gained by increasing the time allocated to a cycle, not by increasing the physical spacing between events. Since cycle time is the product of the physical spacing between arms and the rotational speed of the system, the system timing is directly proportional to the rotational speed. This ultimately defines the total system throughput. To achieve the desired inspection rate of 60,000 capsules per hour, the 24 radial arms must be spinning at 10.411 revolutions per minute.

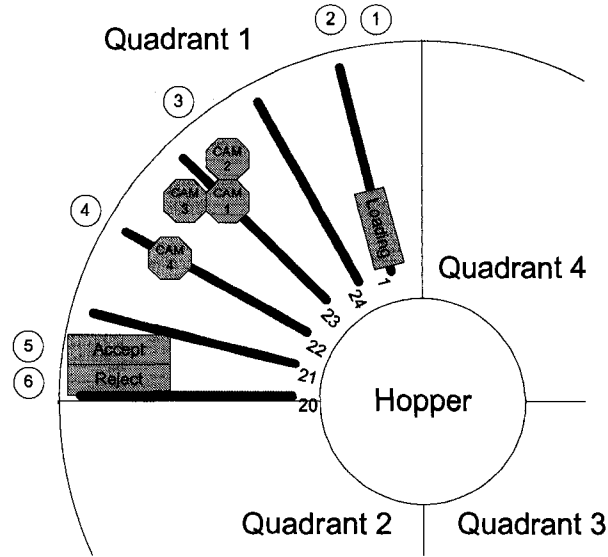


Figure 2.2: Quadrant 1 Detailed

This was found by determining the number of capsule a single arm would have to inspect in this time and then computing how fast this arm would have to move in order to accomplish this task. Equation 2.1 shows this computation.

$$\frac{1000 \text{ caps}/\text{min}}{(24 \text{ arms}/\text{rev} \times 4 \text{ caps}/\text{arm})} = 10.4167 \text{ rev}/\text{min} \quad (2.1)$$

Since a cycle represents $1/24$ of a revolution, the maximum time allocated to any event can be computed as shown equation 2.2.

$$\frac{60 \text{ sec}/\text{min}}{10.4167 \text{ rev}/\text{min}} \times (1/24) \text{ rev} = 0.24 \text{ sec} \quad (2.2)$$

Therefore, the most critical event of the system (arguably the transfer and processing of image data) must occur in < 240 ms. Both equations 2.1 and 2.2 are simply theoretical calculations based on the operation of the the system as described above.

2.3 Imaging Environment

The current Optisorter system uses four cameras to capture the full 360 degree surface of the capsule. The capsule is seated in an opaque holder with a slot down the middle which allows both the top and back surfaces of the capsule to be visible. The capsules are illuminated with back lighting to cast the outline of the capsule and any defects in shadow while illuminating the background and flawless portions of the body for the capsule. Three cameras are positioned to examine the top surface of the capsule while a single camera is positioned from the reverse angle to examine the bottom of the capsule. The cameras are angled to image as much of the surface as possible, however, the solid metal holder occludes a small portion where the capsule makes contact with the holder. In an ideal imaging environment, the back lighting would remain perpendicular to the camera in order to reduce any uneven illumination that results from reflected light. It would appear that due to space constraints, a single panel of LEDs was used for back lighting the three top images while a separate panel of LEDs was used to back light the bottom of the image. Fig. 2.3 shows the details of the camera and lighting setup.

2.4 Objectives for Modification

As the project objectives state, the ultimate goal is to retrofit the existing Optisorter system with upgraded hardware to perform faster, more comprehensive inspections of the capsules. The following modifications were proposed as part of the final system upgrade. These modifications were to be developed as three separate parts by individual students.

1. New hardware to perform motor control, monitor proximity sensors, control accept and reject air actuators, trigger cameras, and interface with the HMI .
-

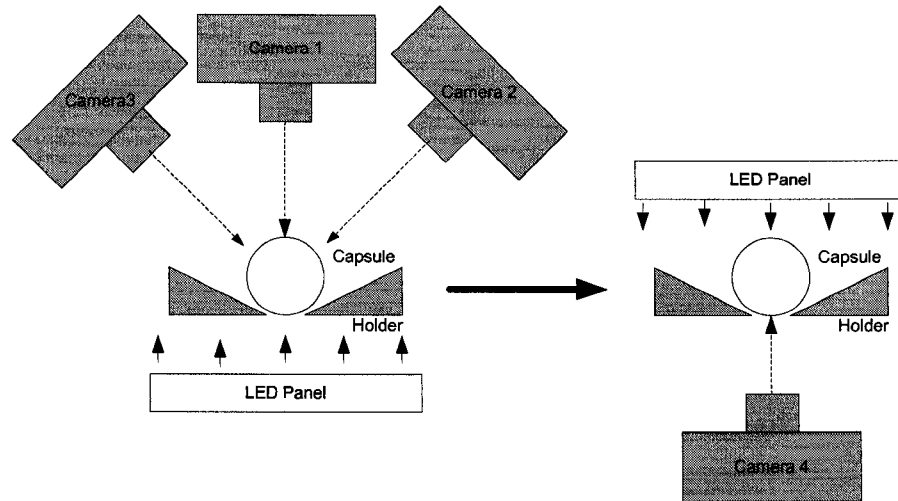


Figure 2.3: Camera Configuration

2. Develop a PC based computer vision system to perform image processing and analysis.
3. Develop new low cost digital cameras to acquire and transfer images.

The third modification listed above is the focus of this thesis where the first two items were developed by Neil Scott and Mohammad Islam respectively. More specific technical details regarding the requirements of these cameras will be discussed in subsequent section. For more information regarding the other project objectives, refer to thesis of Neil Scott and Mohammad Islam.

2.4.1 Selecting a Camera

After determining the overall system requirements, it was necessary to select the hardware for capturing images and transferring them to PCs. The first and foremost consideration was the potential cost of a camera, as 16 cameras were required for a complete system (four cameras for each of the four quadrants). There were three possible approaches, each with inherent advantages and disadvantages.

1. Frame Grabber

The first possible approach would be to purchase very inexpensive image sensors that could be connected via a Frame Grabber card to a PC. This would result in extremely fast implementation, but the high cost of a frame grabber made this solution very undesirable, especially if the final system were to be mass produced.

2. Firewire PC Interface

The second consideration was to create custom cameras with a high speed Firewire interface. Producing custom cameras would cost effective compared to the cost of a frame grabber. It would also allow the hardware to be reasonably upgraded. However the design of such cameras would require longer development time. This was still advantageous over the high cost of the frame grabber interface, however developing hardware and PC drivers for the Firewire interface would have been quite difficult due to the interface complexity and lack of commercial hardware and software support.

3. USB PC Interface

The final option was to continue with the idea of custom cameras, only with a slightly slower speed USB interface. The advantages of USB lay not only in the cost effectiveness, but in ease of development since many USB ready micro controllers and open sourced software were readily available. This was determined to be the optimal approach, as long as the USB transfer rates could keep up with required data throughput.

Since the most desirable data interface was USB 2.0, a quick calculation was made to verify that the USB transfer speeds could keep up with the system timing demands. Assuming a simple 1024x768 image window at 8 bits per pixel, a single image would

consist of 786432 bytes of data. USB 2.0 has a theoretical maximum transfer rate of 60 MB/s but this is unattainable due to packet communication overhead. Bulk transfer rates of up to 40 MB/s are possible with a realistic observable transfer rate of around 30 MB/s [3]. Preliminary USB 2.0 transfer test conducted by Neil Scott (a partner in this collaborative project) found a sustained bulk transfer rate of 31145280 bytes per sec or equivalently 30 MB/s. At this rate, it would take 0.1 seconds to transfer 4 full images to the PC. As 240 ms is the maximum cycle time available in order to achieve the desired system throughput, the USB interface would be sufficient and still leave up to 140 ms to be allocated for image processing to occur within the PC.

The preliminary system timing consideration outlined above showed that a USB 2.0 interface would be sufficient, and two demo boards were obtained to perform initial hardware feasibility tests.

2.4.2 Modifying the Imaging Environment

Although the current imaging setup may be sufficient, a number of improvements were proposed to provide improved images for processing. These included:

1. Improved lighting
2. Transparent holders
3. Three camera setup
4. Reflected front lighting
5. Reduced stray reflections

Each item listed above was identified as a potential area of improvement not only for the quality of images, but in the case of a three camera setup, reduced system load and cost. Of these items, only the first was implemented in the final system. Instead

of using red LEDs, the diffused back lighting of the current system was upgraded to high intensity white LEDs. The higher intensity light allowed for a shorter exposure time while taking an image of the capsule, resulting in less blur introduced due to the movement of the capsule. The white light allowed for full color spectrum exposure of the color CMOS image sensors used in the final custom cameras.

The use of transparent holders for fixturing the capsules during imaging was tested, however, due to difficulties manufacturing perfectly clear holders free of scratches, the existing opaque holders were used. The use of clear holders should be considered as an area of future development.

A three camera setup was also tested, however without perfect clear holders, there are areas of the capsule that are occluded using three cameras and opaque holders. Again, the use of a three camera setup should be further investigate along with the use of clear holders.

The idea of reflected front lighting was proposed as a way of illuminating the front of an opaque capsule without needing an additional front light. The idea was to place a reflective surface over the front of the camera (around the lens) so that stray light from the back lighting of the capsule would be reflected and illuminate the surface of the capsule. This would provide the visibility required to identify color and surface defects. This item was tested and verified, however, was not implemented since the prototype was only desired to inspect clear capsules due to lack of project time.

Finally, as another point to consider, it was found that under certain circumstances, light from an inspection was reflecting off the various shiny metallic surfaces within the system, causing bright areas on the capsule surface to appear. This type of reflection could be reduced by ensuring all surfaces within the system had a dark matte finish as to reduce the reflection of light within the system. This was not implemented into the final system, but could be considered as an area for improvement in the future development of the inspection system.

2.5 Proposed Design

Using the understood operating principles of the existing mechanical system along with the desired areas of improvement, a basic proposed system overview was composed. This can be seen in Fig. 2.4.

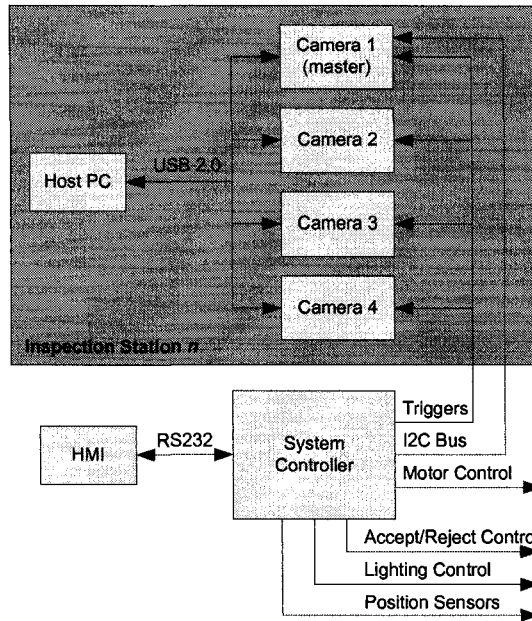


Figure 2.4: System Block Diagram

The design maintains the principal concept of having four identical stations work in parallel to accomplish the desired inspection throughput. These four inspection stations are connected via a custom control board that is responsible for co-ordinating the entire system. The control board acts as an interface between the mechanical systems and the computer vision stations, triggering the stations to capture an image of the capsules at the appropriate time and using the inspection results from the station to mechanically accept and reject the appropriate capsules.

The proposed custom digital cameras receive triggers from the control board and

send image data via a UBS 2.0 interface to a PC that is responsible for performing the necessary inspections. The results of the inspections are relayed through the master camera, back to the control board. Thus the proposed custom cameras must provide the following functionality:

1. Maintain the desired inspection rate.
2. Communicate image data to PC via USB 2.0.
3. Receive external triggers.
4. Output inspection results.
5. Remain competitive in cost compared to commercial systems.
6. Satisfy size constraints of existing imaging fixtures.

At this point, a proposed camera architecture was developed. Fig. 2.5 shows the most basic required functional blocks.

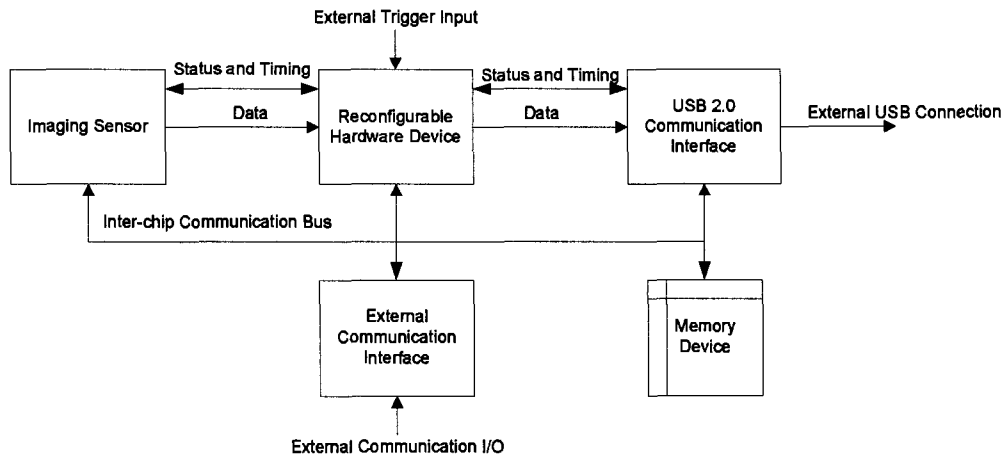


Figure 2.5: Camera Block Diagram

In this design, a suitable imaging sensor provides image data to a reconfigurable hardware device. This device acts as the camera controller, initially processing the

image data and preparing the image data output. The image data is then be sent to a USB 2.0 capable microcontroller, which is responsible for the data communication to the PC for image processing. To satisfy the external communication requirements, an I²C bus extender was used along with simple 3.3 V inputs for receiving triggers. A small EEPROM is also added to the design, in order to store basic configuration information such as the device identification for the USB enumeration as well as a master camera setting.

A unique feature to this proposed architecture is that there will be no frame buffering taking place within the camera. Data is processed and transfered in real-time. This will greatly reduce the unit cost, as well as assist in maintaining the desired inspection rates for the system.

Chapter 3

Camera Design Methodology

3.1 Camera Design Flow

After identifying the objectives for the custom digital camera and how it was to operate as part of the larger system, a design procedure was developed. The major steps involved in the camera design methodology include:

1. Deriving device specifications
2. Simulating hardware operation
3. Prototyping functional groups
4. Testing hardware functions
5. Building complete device
6. Testing device operation

7. System integration

8. System testing

Fig. 3.1 shows the general bottom-up design flow for the proposed custom camera. With our camera specifications, performance requirements, desired operation and basic overview in mind, this design procedure was developed to assist in the realization of the custom camera.

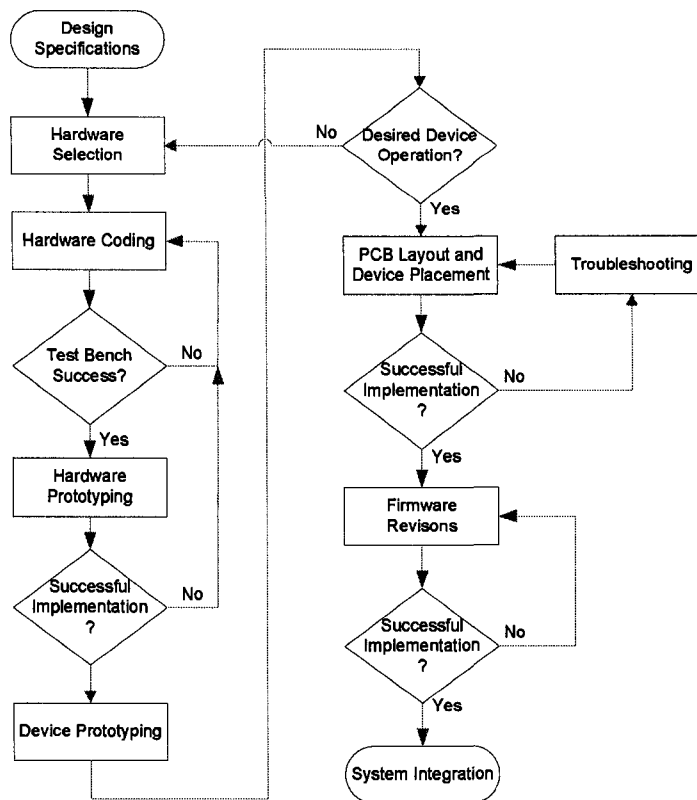


Figure 3.1: Hardware Design Flow

3.2 Design Specifications

The system design specifications were based on inspection requirements outlined by Pharmaphil Inc. The details of both the business and defect detection requirements can be found in Appendix A. Table A.1 outlines the inspection requirements used to derive the required resolution of the system. Table A.2 outlines the desired operation and is used to determine the required operating speed of the system as well as the test parameters. Finally, Table A.3 describes the desired features and functionality that must be incorporated into the modified design. Since the focus of this thesis is on the Camera Design with some insight into the imaging environment, Table 3.1 summarizes the pertinent specifications and the area of the system design to which they apply to.

Requirement	Specification	Applies to
Defect size	0.2mm or larger	Imaging sensor Imaging Environment
Capsule Colour	RGB/YUV	Imaging sensor Imaging Environment Reconfigurable Hardware
Speed of Inspection	1000 caps/min	Imaging Sensor Data Interface Memory/Buffering
Products Inspected	All	Reconfigurable Hardware

Table 3.1: Design Specification Summary

3.3 Major Design Methods

The following sections summarize the procedures used while developing the camera. These cover the actual software and hardware tools used and their function in the

design process.

3.3.1 Reconfigurable Hardware Programming

One of the largest portions of this design project involved developing hardware code for the reconfigurable device used on the camera. All coding was done in VHDL because of its modularity and easy readability (though Verilog coding would have been a viable substitute). The development environment used was Xilinx ISE 10.1 where the final VHDL code was synthesised.

The required VHDL coding was broken down into various behavioural blocks. Each block was behaviourally verified using an appropriate test bench when required, also coded in VHDL.

The top level VHDL block was assigned timing constraints with respect to the system clocks that would be provided to the system and final timing and resource utilization reports were generated. More specific information pertaining to the actual design results can be found in Chapter 6.

3.3.2 Simulation Tools

Some aspects of development for the system, such as the development of a demo-saicking (image reconstruction) algorithm required rapid logical implementation for verification before realization in hardware as part of the final design. For these purposes, Matlab was used. Algorithms were implemented as functions in “.m” files and called from either the Matlab command line or by a series of tests coded in additional “.m” file. Although Matlab is a relatively slow simulation tool, it was selected because of the wide range of library support in image processing as well as its ability to easily interface with external data sources. More detail about the specific functions implemented in Matlab, as well as the corresponding code, can be found in Chapter

6 and Appendix D.

3.3.3 Prototyping Hardware

Only limited hardware prototyping took place concerning the camera design. This was primarily due to the nature of the components, specifically their unique footprints that could not easily be mounted to a breadboard. Essentially, hardware prototyping took place by modifying demo boards. Ideally, multiple revisions of the final PCB would have been developed allowing for measurement of circuit noise and further adjustments. Details on the camera demo board used are outlined in the component selection section of the following chapter.

3.4 Testing

Testing the design occurred on two levels. The first testing occurred independently of the inspection environment, ensuring that the resulting camera was capable of generating the required hardware. The second form of testing occurred with the camera integrated into the system, increasing the operating speed and analyzing the resulting quality of image until failure in either the device or in the transferred data. The test results were used to verify that the final design met the system requirements. The details of these test are outlined in the conclusion of this thesis.

Chapter 4

Hardware Design

4.1 Component Selection

After the design specifications for the camera were determined, the next step was to select specific components capable of satisfying these requirements. Fig. 2.5 showed the major components that were to be included in the design. These include:

1. Imaging sensor
2. Reconfigurable device
3. USB 2.0 compatible microcontroller
4. External communication interface
5. Small memory device

The following sections detail the selection of these devices and justifies their use in the design with respect to the system specifications.

4.1.1 Imaging Sensors

Two of the most heavily weighted factors in selecting hardware were: cost and availability. Under these constraints, a variety of imaging sensors were compared for suitability. One critical decision was on whether to use a CMOS sensor or a CCD sensor as both have specific advantages and disadvantages.

Although they are faster and often cheaper than CMOS sensors, Charge Coupled Device (CCD) sensors only output an analog signal representing the charge resulting from a pixel being exposed to light. This analog value must be amplified and converted to a digital value by additional components and circuitry. On the other hand, a CMOS image sensor has an internal CCD with supporting digital circuitry integrated into the sensor itself. The sensor data output from a CMOS is in digital format and accompanied by associated timing signals for tracking the row and column of the current pixel being read out. In addition, CMOS sensors allow for windowing an area of the sensor's pixel array so that the entire image frame does not need to be outputted from the device. This provides greater control over both the exposure time of the sensor as well as the frame readout time. Micron, one of the primary suppliers of CMOS image sensors, provided a variety of demo boards with USB support. The sensors, though slightly more expensive than typical CCD sensors, were readily available in large quantities and would not require as much additional hardware (i.e. a controller to integrate into the design) providing overall savings in development costs.

The Micron MT9T001 3.1 mega pixel sensor was selected as it was readily available with a demo board featuring a chip memory, Virtex II FPGA and a Cypress FX2 USB microcontroller. This demo board provided a solid backbone to the camera design, even though it possessed expensive components and unnecessary features for the requirements of our design, especially in terms of memory and logic capacity of the FPGA. Nonetheless, the demo board acted as the basis for the custom camera's

hardware selection in the first iteration of the design flow.

4.1.2 Reconfigurable Devices

The reconfigurable device on the camera plays a vital part in synchronizing the various hardware components as well as managing and processing image data. The reconfigurable device acts as the camera's memory for all necessary image data buffering as well as a communication interface to the imaging sensor. Selecting a suitable device for such a dynamic role involved many considerations and making a number of trade offs. The decision parameters for selecting the appropriate device were as follows in order of priority.

1. Memory
2. Cost
3. Device Capacity
4. Development Time
5. Footprint
6. I/O Capacity

Initial firmware development was done on a Xilinx Virtex II series device (XC2V2500) as it was part of the Micron Demo board (MI3100 sensor head with DEMO2 FPGA base) which was used as a basis for the final design. Since expandability of the initial design was part of the camera design specifications, a reconfigurable device was required.

There were two predominate reconfigurable devices considered for this design. They were the FPGA and CPLD. FPGAs are "fine-grain" devices, meaning they

contain many tiny blocks of logic made up of flip-flops. CPLDs are “coarse-grain” devices with relatively few larger blocks of logic made up with flip-flops. FPGAs are RAM based and need to be configured each time they are powered up. CPLDs are EEPROM based and retain their memory after being programmed. CPLDs have faster input-to-output timing than FPGAs because FPGAs have a coarse-grain architecture where a single logic block can implement a more complex function. However, FPGAs have special routing resources to implement efficiently binary counters and arithmetic functions (adders, comparators, etc.) and RAM, where CPLDs do not. In general, FPGAs can contain very large digital designs, while CPLDs can contain small designs only [11]. In order to satisfy the more sophisticated operations required by the reconfigurable device, an FPGA was selected as the more suitable device for this role.

Next, an appropriate FPGA was selected. As Xilinx ISE development software was readily licensed by the university with full access to support and CoreGEN designs, Xilinx FPGAs were favored over competing FPGAs such as Altera. This decision to use a Xilinx device was simply due to the availability of licensed development software as well as compatibility of code being developed on the demo board with a Xilinx FPGA.

As demo board firmware was already being developed for this application using the Xilinx Virtex II device, Virtex was the first family investigated as a suitable Xilinx FPGA for the final camera design. However, the cost for the Virtex family far exceeded the cost parameters outlined, so the lower cost Spartan family was investigated. Table 4.1 shows the Spartan series descriptions used in selecting the appropriate device. Comparing the Spartan summary to our decision metrics, it was quite clear that the Spartan-3E series was most suitable for our application due to the high logic density, low I/O count and overall low cost.

Series	Domain	Description	Cost(CAD)
Spartan-3A DSP	DSP optimized	For applications where integrated DSP MACs and expanded memory are required.	\$140 to \$215
Spartan-3AN	Non-Volatile	For applications where non-volatile, system integration, security and large user flash are required.	\$13 to \$80
Spartan-3A	I/O optimized	Ideal for bridging, differential signaling and memory interfacing applications, requiring wide or multiple interfaces and modest processing.	\$63 to \$88
Spartan-3E	Logic optimized	Ideal for logic integration, DSP co-processing and embedded control, requiring significant processing and narrow or few interfaces.	\$11 to \$75
Spartan-3	I/O + Logic Opt.	Ideal for highly-integrated data-processing applications.	\$10 to \$130

Table 4.1: Spartan FPGA Summary[19]

The final step was to select a specific device from the Spartan-3E series FPGAs. As per the reconfigurable device requirements listed at the start of the section, the specific Spartan-3E device was not only selected based on its internal specifications, but also based on a common footprint that would allow for future upgrades. The Spartan-3E500 was selected as a balance of cost and logic capacity in the series of devices that shared the FT256 footprint. This footprint was selected not only for its compact nature, but also so that the more powerful Spartan-3E1200, or cheaper Spartan-3E250 could replace the selected FPGA without having to modify the physical layout and placement of these devices in the final camera design.

A final point on the FPGA selection relates to the powering of the device. Unlike the more expensive Virtex II devices, the Spartan series has some specific power requirements. In order to supply the correct voltages to the device with the proper power on start-up conditions, a separate component was chosen to perform this role. The TPS7500 Triple supply by Texas Instruments was chosen based on a recommendation in the device data sheet [21].

4.1.3 Communication Interface

There were two main components selected to act as communication interfaces for the camera. The primary device responsible for transferring image data to the PC is the Cypress CYCFX2. This specific device was selected by the developer of the USB 2.0 communication drivers so the details of this device are not covered in detail as part of this thesis.

One benefit of using this device is its ability to act as a data slave device. A 16bit data bus allows an external device to write data to a 4 KB FIFO within the Cypress controller that is automatically packeted and sent according to USB 2.0 communication specifications. The device also supports an I²C interface which is utilized on the camera as part of the external communication interface. Finally, the device is relatively cheap which is an extremely important factor in component selection.

The second major device is the selection of an I²C bus extender. This allows the local I²C bus on the camera to be connected to the control board for relaying the accept and reject singles from the PC. The NXP P82B715TD-T device was simply selected because of its low cost, availability and ability to provide the required range of communication dictated by the physical system.

4.1.4 Component Summary

Table 4.2 summarizes the major components selected for the device design noting how they will be referenced for the remainder of the thesis. For a complete list of components used in the device design, a bill of materials can be found in appendix C.

Device	Manufacturer	Part Number	Referenced As
Image Sensor	Micron	MT9T001P12STC	MT9T001
FPGA	Xilinx	XC3S500E-4FTG256C	Spartan-3E
USB MCU	Cypress	CY7C68013A-100TAXC	FX2
Bus Extender	NXP	P82B715TD-T	I ² C BE
Memory	Microchip	24LC128-I/ST	128kB EEPROM
Triple Supply	Texas Instruments	TPS75003RHLT	TPS supply

Table 4.2: Component Summary

4.2 Circuit Schematics

After having selected the specific devices to use in the camera design, the device connections must be carefully made to ensure their desired and correct operation. Some of the major considerations entail: powering the devices, terminating I/O, routing I/O and external connectors, sizing filtering capacitors, providing clocks, building reset circuits, designing power supplies and even placing test points on critical nets.

4.2.1 Micron MT9T001 CMOS Imaging Sensor

The MT9T001 CMOS imaging sensor is one of the most critical devices in the camera design. This device is responsible for acquiring image data and thus any error introduced at this point will propagate throughout the remaining components. There were

some particular considerations that had to be made when designing the schematics for this device as it required separate analog and digital power supplies. The purpose of these separate supplies is to isolate the very noisy digital circuitry from the analog portion of the IC, which is highly susceptible to noise in the power supply.

To improve the sensor's operation, separate analog and digital low drop-out linear regulators were used to power this device. However, the ground pins share the same grounding net. The appropriate placement of grounding points in the PCB layout of this device made this possible, reducing the effect any ground noise may have on the analog ground of the device by providing a direct path for the current to flow, preventing potential current leaks towards the other ground pins.

The power saving features of the chip have been grounded to simplify the PCB routing. The remaining data bus, timing signals and reset and status signals have been connected to pins on the FPGA for flexible control over this device. Fig. B.4 in Appendix B shows the final schematic drawing for the Micron MT9T001 sensor.

4.2.2 Xilinx Spartan-3E500 FPGA

4.2.2.1 I/O Connections

Since the FPGA acts as the main controller for the custom camera, almost every device in one way or another is connected to the FPGA. The I/O connections for the FPGA to the neighbouring devices were placed on pins whose functions did not change between the three different Spartan-3E devices that share the FT256 footprint. This provided greater flexibility for future upgrades. All I/O on the device were configured as 3.3 V Low Voltage CMOS (LVCMOS33) capable of syncing or sourcing up to 16 mA of current [21].

4.2.2.2 Clock Connections

Special considerations were also made when connecting clock signals to the FPGA. Although the FPGA can internally route any I/O to any cell, the Spartan-3E device has dedicated Digital Clock Managers (DCMs) with specific pins associated with them. Any part of the design where an output clock was supplied to another device, such as the clock for controlling exposure rate of the MT9T001 sensor, a neighbouring I/O pin was shorted to the DCM output in order to provide DLL feedback for the clock manager. The configuration of such a scenario can be seen in Fig. 4.1. For the specific case of the camera schematic net CLKIN (provided to the MT9T001 sensor from the FPGA), the net IFCLK enters the FPGA at IBUFG feeding *CLKIN* of the DCM (which is optionally shifted within the DCM). This is outputted from the FPGA at OBUF as the net CLKIN which is then connected to the MT9T001 sensor. The net CLKIN also re-enters the FPGA at the IBUFG feeding the *CLKFB* of the DCM to regulate the clock output. This configuration is essential to removing any clock skew that can occur through the FPGA device. This is critical, especially when the clock net is shared by multiple devices and routed through the FPGA. A similar configuration was used on the clock net (SCL) of the I²C communication bus.

4.2.2.3 Power Connections

Unlike most other components, the FPGA requires three different supply voltages: 1.2 V for the core, 2.5 V for auxiliary features and 3.3 V for the I/O. These voltages required specific power-on conditions in order to properly power the Spartan device. The exact voltage specifications are described in the following section. For the schematics of the power connections, the most critical point was that each power pin had to be assigned at least one bypass capacitor. The sizing and placement of

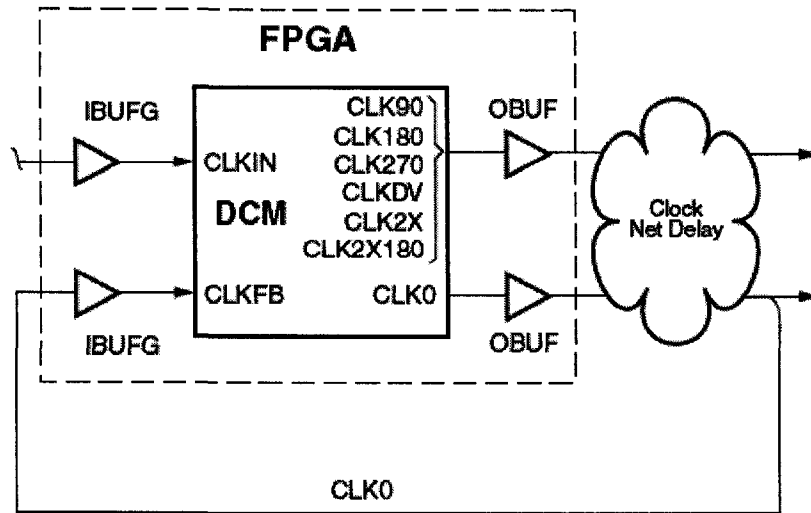


Figure 4.1: DCM with Off-Chip Delay Feedback

these capacitors is discussed in greater detail in a later section of this thesis.

4.2.3 TPS Triple Supply

The TPS7500 Triple Supply was selected as an ideal component to supply the FPGA with the required voltage levels for operation. This integrated circuit uses two non-synchronous buck converters to supply up to 3 A on the 3.3 V and 1.2 V lines. In addition, it has an integrated low-dropout linear regulator for the 2.5 V supply. Each supply has an adjustable “soft start” that allows the desired voltages to be increased at a controlled rate during power-on. This feature is used to satisfy the power on requirements of the Spartan-3E series FPGAs. The specifications for these voltages can be found in [21] and the voltage ramp rates are summarized in Table 4.3.

In order to configure the device correctly, the following design considerations needed to be made when drawing the TPS7500 schematic (seen in Fig. B.1 of Appendix B). These included: sizing soft start capacitors, selecting appropriate sized

Symbol	Description	Min	Max	Units
V_{CCINTR}	Ramp rate from GND to valid V_{CCINT} supply level.	0.2	50	ms
V_{CCAUXR}	Ramp rate from GND to valid V_{CCAUX} supply level.	0.2	50	ms
V_{CCO2R}	Ramp rate from GND to valid V_{CCO} supply level.	0.2	50	ms

Table 4.3: Spartan-3E Supply Voltage Ramp Rate[21]

components for the non-synchronous buck converters, selecting appropriate sized filtering capacitors and providing a reverse current path on the drain of the power transistors via Schottky diodes.

To suit the upgradable nature of the design, the supporting components were sized for a maximum current draw of 3 A. This is the maximum the TPS device can source. Even though the maximum current supplied from the USB controllers is limited to 500 mA, the cameras can handle external supplies capable of supplying greater currents for potential off-line application (such as additional testing and prototyping).

4.2.3.1 Limiting Buck Converter Current

The two non-synchronous buck converters that supply voltage to the VCCO and VC-CINT lines of the FPGA are both capable for sourcing up to 3 A (internally limited) unless externally limited by sizing R_1 and R_2 , as seen in Fig. B.1 of Appendix B. Unlimited current allows the converters to operate in a continuous mode, preventing “ringing” from occurring in the junction of the PMOS transistors and inductors. For this reason, R_1 and R_2 were sized as to not limit current through these devices to any less than 3 A and at the same time, be capable of dissipating the maximum power. As power is a function of I^2R with I at 3 A, R must be reduced as much as possible to minimize power dissipation. R_1 and R_2 were sized at 330 m Ω according to suggested values in [15] resulting in a maximum power of 0.297 mW. Thus 1/2 W, 330 m Ω

resistors were chosen for this application.

4.2.3.2 Setting VCCO

The non-synchronous buck converters are designed to sustain a 1.22 V output with a unity feedback. In order to set the voltage output of one of the buck converters greater than 1.22 V, the feedback to the converter must be appropriately scaled down (i.e., through a voltage divider circuit) so that the resulting output voltage is regulated at a desired level. Since the camera design operates on 3.3 V I/O, as specified in both the Cypress FX2 device as well as the MT9T001 image sensor, a single VCCO of 3.3 V was required and was generated using Buck2. The feedback output voltage for Buck2 can be written as:

$$V_{OUT} = V_{FB} \left(\frac{R_6}{R_5} + 1 \right) \quad (4.1)$$

Since $V_{FB} = 1.22$ V and $V_{OUT} = 3.3$ V then $R_6/R_5 = 1.705$. Based on recommendations from [15], R_6 and R_5 were chosen as 61.9 k Ω and 36.9 k Ω respectively.

4.2.3.3 Sizing Soft Start Capacitors

One of the most substantial benefits of using the TPS7500 supply is the soft start capability of the device. This controls the voltage ramp rate of the output supplies by appropriately sizing soft start capacitors C_4 , C_5 and C_6 in Fig. B.1 of Appendix B. The voltage ramp rate is a complex function of many variables. It is recommended that the soft start capacitors are appropriately sized by using a test bench setup and monitoring the power-on conditions outlined in Table 4.3, adjusting the soft start capacitor values until the desired power on conditions are met. This design used recommended values supplied by [15] and [20] in the sizing of these capacitors, and the resulting voltage ramps were verified on the first hardware revision of the board.

4.2.4 Cypress FX2 USB Microcontroller

The schematics seen in B.3 for the Cypress FX2 microcontroller were essentially replicated from the Cypress CY3684/3674 EZ-USB Advanced Development Board. Slight modifications were made, such as connecting the reset pin directly to power through an RC circuit designed to delay the power on transient. As in the demo board, the analog power was shared with the digital power. The data bus pins for the slave FIFO were connected to the FPGA which was used to drive data to the device.

The FX2 required an external clock. For this, a 24 MHz crystal was supplied. This clock was internally divided to produce a 48 MHz clock for IFCLK (used to synchronize the incoming FIFO data) as well as provide an external system clock CLKOUT. The CLKOUT net was supplied directly to the FPGA and was internally referenced as USBCLK, which can be scaled using a DCM of the FPGA and sent to the MT9T device to control exposure time.

The FX2 was also connected to the FPGA in a configuration that allowed it to program the FPGA in serial slave mode. With this setup, FPGA code could be loaded to the device through the USB port, eliminating the need for an additional program memory device or external programming connection. This also allowed the USB device drivers to supply the latest FPGA code whenever the device was plugged in, ensuring that the FPGA was always loaded with the most recent code. This also ensures that all cameras in a system will operate with the same code, without the need to update each FPGA of each camera separately.

The final consideration for the FX2 schematics was the connection of a bypass capacitor to each power pin to ground. The sizing and placement of these capacitors is described in more detail in a section to follow. B.3 in Appendix B shows the final schematics for this device.

4.2.5 I²C Communication Bus

The I²C components were connected so that the data net (SDA) was shared between all I²C devices, as was done with the clock net (SCL). Special considerations had to be made to size the pull-up resistors on both the local side of the I²C bus as well as the external side of the I²C bus extender.

When calculating the pull-up resistance values, the gain of the signal buffers introduces scaling factors that must be applied to the system components. In practical systems, the pull-up resistance value is usually calculated to achieve the rise time requirement of the system [7]. For the purpose of this design, the I²C bus operated at 100 kHz. Thus, the time constant of the total system (RC) is set to 1 μ s or less [12]. Equation 4.2 was used in determining the required pull-up resistance.

$$R_{pull-up} = \frac{1\mu s}{C_{device} + C_{wiring}} \quad (4.2)$$

On the local side of the I²C bus, the following components were connected with their corresponding capacitive load on the bus and factored in as part of C_{device} . Table 4.4 shows these devices.

Device	C_{device}	C_{wire}
CY7C	50 pF	10 pF
MT9T001	30 pF	10 pF
Spartan-3E500	50 pF	10 pF
EEPROM	50 pF	10 pF
Total	200 pF	40 pF

Table 4.4: I²C Devices: Loading Capacitance

The value of C_{wire} was approximated as 10 pF for any copper traces to the device as assumed in the device data sheet [12]. Thus, using Equation 4.2, the required pull-up resistance was calculated as 4.116 k Ω . To maintain a lower time constant,

this value was rounded down to the nearest common resistor value of 4 k Ω .

On the external side of the I²C bus, the load capacitance was computed by assuming 50 pF per meter as outlined by [12]. For an approximate wire length of 2 m, a total capacitive load of 100 pF was used in Equation 4.2. The required pull-up resistance on the external side of the extender was found to be 10k Ω .

Also attached to the I²C bus was the small 128 kB EEPROM used to store device settings such as master/slave settings, windowing parameters for the camera and device IDs for USB enumeration. Again, the SDA and SCL lines were connected to the bus and the device was powered appropriately. As can be seen in the schematics of B.5 of Appendix B, the I²C bus power is supplied by the same 3.3 V source as the Cypress FX2 Microcontroller.

4.3 PCB Layout

After having selected the desired components, a Printed Circuit Board was designed to create connections between the components, such as resistors, integrated circuits, and connectors [13]. In order to accomplish this task, the following bottom-up design approach was used (shown in Fig. 4.2).

A bottom-up approach was taken because specific ICs were first selected to fulfill the camera's specifications. These ICs were then combined with supporting components such as bypass capacitors and voltage regulators into larger functional groups. After prototyping the functional groups, these groups were combined to form the complete design. This section will discuss the PCB layout of these groups and the formation of the final camera hardware design.

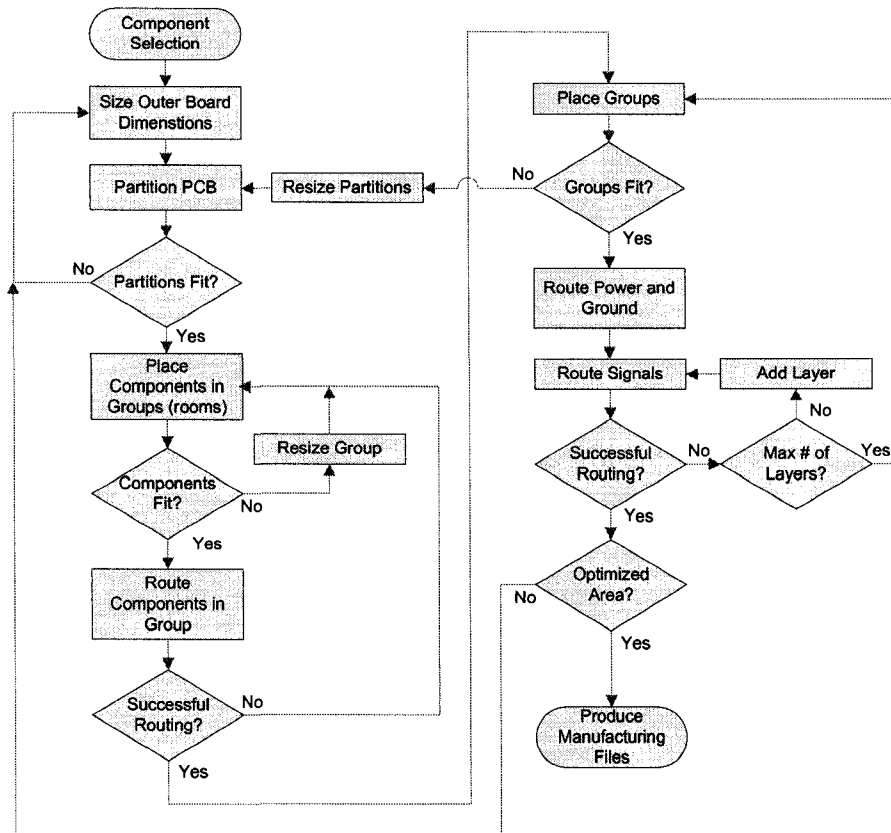


Figure 4.2: PCB Design Flow

4.3.1 PCB specifications

There were some restrictions on the physical characteristics of the PCB that had to be set before a layout could be designed. These included the following:

1. Maximum outer dimension <70 mm
2. Maximum of four layers on the PCB (two outer and two inner layers)
3. Space for four mounting through holes

In addition, the copper weight and dielectric material used in the boards needed to be specified. Copper weight defines the number of ounces of copper on one square

foot of board. From this, the thickness of copper can be computed using the known density of copper. A typical copper weight used for low voltage electronics is 1 oz copper, resulting in a layer approximately 35 μm thick [6]. To verify that this would suit the camera PCB board, the following assumptions and calculations were made. First, because a FT256 footprint was used for the FPGA, a minimum trace width of 5mil was used (1 mil = 1/1000 inch). Second, the maximum current drawn through any of these traces to I/O in the FPGA was limited to 100 mA by the input clamp diodes [20]. Finally, the maximum temperature rise permitted for a trace was limited to 10 $^{\circ}\text{C}$ as was the minimum rise value in the IPC-2221 graphs used to determine thermal conductive properties of copper traces [6]. Using these values, the following required copper weight can be calculated.

$$Area[mils^2] = (Current[Amps]/(k * Temp_{rise}[^{\circ}C]^b))^{1/c} \quad (4.3)$$

$$Width[mils] = Area[mils^2]/(Thickness[oz] * 1.378[mils/oz]) \quad (4.4)$$

where for IPC-2221 internal copper layers: $k=0.024$, $b=0.44$ and $c=0.725$

From Equation 4.3 the required trace area for our design constraints can be determined as $Area = 1.77 \text{ mil}^2$. Using the area calculation and solving for $Thickness$ in Equation 4.4, the required copper thickness is 0.25 oz.

Thus 1 oz copper would be more than suffice for the requirements of the PCB design and will maintain the desired thermal performance during operation.

As for the dielectric board material, because there was very limited structural load on the board, standard FR-4 material was selected to reduce cost. The standard board thickness of 62mil was used.

4.3.2 Component Placement

Component placement was the most critical step in the PCB layout, as this was the primary determining factor for successful routing of the design. The first step in component placement was to partition the PCB board in the major component groups. In total, there were 5 groups, one for each schematic drawing summarized above. These consisted of:

1. TPS Power Supply
2. Xilinx FPGA
3. Cypress FX2 USB
4. Micron MT9T001 image sensor
5. I²C components

Fig. 4.3 shows the final partitions and their layout on the PCB board. The top of the board is shown to the left with copper traces in red and the bottom of the board to the left with copper traces in blue. The shaded dotted rectangles outline the partition bounds and the white markings show component boundaries. However, before arriving to this final layout, a number of iterations were performed as outlined in Fig. 4.2.

When determining the placement of the major components, the largest constraint was that the MT9T001 sensor had to be placed with its optical center on the center of the PCB board with respect to the mounting holes. Fig. 4.3 shows the position of the sensor footprint on the bottom board (blue) where the optical center of the device was marked with a thick white cross-hair, offset from the footprint board center indicated by the thin white cross-hair. This offset was determined from the MT9T001 data sheet in [10]. This severely limited placement of other major components on this side

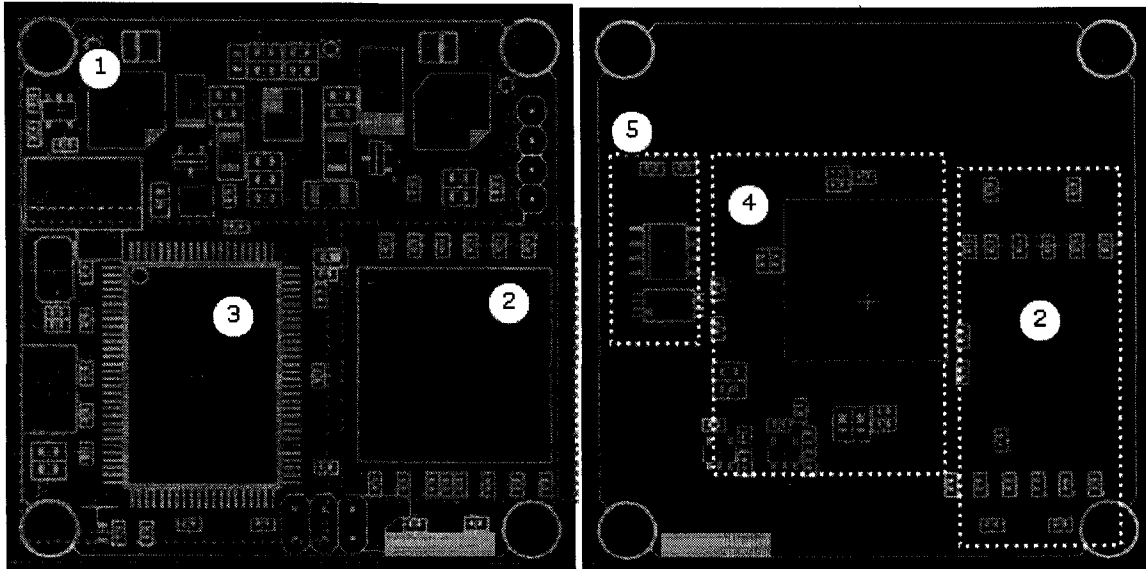


Figure 4.3: PCB Partitions and Final Layout

of the board. Thus the remaining large components were placed on the top side (red). The two linear regulators used to power this device (U6, U7) were placed adjacent to each other in the ample free space surrounding the MT9T001 with the corresponding input and output capacitors placed near their corresponding pins. These were later routed through the power plane layer to MT9T001.

The top side of the PCB board was divided into 3 major partitions as seen in Fig. 4.3. Components for the TPS (U1) power supply were placed first because they required specific locations with respect to U1. The datasheet for the TPS75003 (found in [15]) outlined critical placement for certain components and their corresponding critical paths. These components, such as the diodes (D1, D2) and especially inductors (L1, L2), required connecting traces to be less than 100 mils in length in order to minimize the equivalent series resistance (ESR) of the device and maintain the desired performance of their connected buck converters. These conditions were outlined in [15].

With the TPS75003 fully routed in place, the placement of the FPGA (U2) and Cypress FX2 (U3) were both considered simultaneously. The FX2 was oriented such that the data pins for the 16 bit FIFO bus were facing closest to the FPGA. Both devices were also placed close the edge of the partition to facilitate successful routing. The USB mini-B connector was placed within 100 mils of the device and trace lengths to the data lines were kept the same to prevent signal skew. Conveniently, the connectors were located at the edge of the PCB and appropriately spaced from any “tall” or high profile components which prevents possible obstruction from the mating connector. The 24 MHz crystal oscillator (Y1) was also placed near to its corresponding device pins to prevent signal degradation due to trace impedances. Power to the FX2 was routed through an internal power plane in the PCB, therefore, little consideration was paid to the placement of the 3.3 V linear regulator powering it. Finally, bypass, input and output capacitors were placed appropriately near their corresponding pins.

The FPGA (U2) was the simplest group to place as the only consideration necessary was the location of the 32 bypass capacitors required for the device. These were lined along the top and bottom of the device, spaced according to suggestions by [18], where very few signals were required to be routed. This left the main data buses for the MT9T001 image sensor and FX2 USB device free to be routed along the sides and center. Because there were many unused I/O on the FPGA, the device was placed at the edge of the PCB with very limited access to the pins along one side. The general signal breakout for buried pins (pins towards the center of the ball grid array) was planned according to the recommendations of [18].

The final group consisted of the I²C components. The bus extender was the most critical component in this group to place because it needed to be as near as possible to the I²C connector on the PCB. This connector (JP3) was placed in the TPS75003 partition, however, since it was a through hole component, the actual connections for

this device were made in the I²C partition on the bottom of the PCB. The I²C bus extender chip (U8) was connected directly next to the connector, closely neighbored by the I²C EEPROM (U9).

4.3.3 Power and Grounding

Typically in a four layer design such as the camera PCB, the two internal layers are reserved for power and ground. This has two purposes: The first is to provide accessibility to power and ground pins on the surface layers, without obstructing routing; The second is to create a large capacitance between the power and ground planes, which are usually separated by a dielectric pre-preg. Since the thickness of the pre-preg is easily adjustable, the capacitance created between the two inner layers can be optimised better than the FR-4 material that composes the PCB layers themselves [6].

The goal in routing the power plane layer is not only to provide surface components with accessibility to the appropriate voltages in this layer. It also to maximises the copper area, increasing the effective capacitance of the plane. In addition, minimizing bends or loops in the plane helps to reduce any parasitic inductances that could result [18].

Fig. 4.4 shows the final power plane routing with six separate planes; one for each required voltage on the device. These voltages include: 3.3 V I²C and digital/analog USB, 3.3 V FPGA I/O, 3.3 V for MT9T001 digital, 3.3 V for MT9T001 analog, 2.5 V for FPGA auxiliary and finally 1.2 V for FPGA internal power. Ideally, each voltage should be on its own plane, separated by a ground layer, and hence a 14 layer board would be required. However, due to cost considerations, the design was compacted into a single power plane to fit a four layer board.

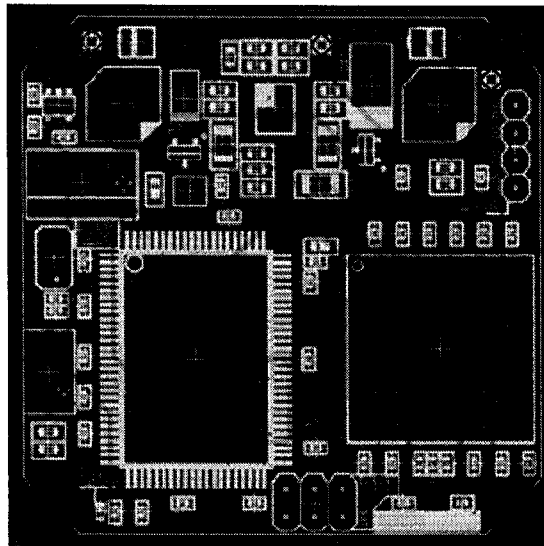


Figure 4.4: PCB Power Plane Layout

4.3.4 Bypass capacitors

The purpose of a bypass capacitor network is to filter digital switching noise, providing a smooth impedance profile over a range of frequencies, thus ensuring a static voltage level at the power pins of a device.

In order to accomplish this goal, an approximation of the desired frequency range needs to be considered first. Ideally, a prototype would be constructed and frequency spectrum analysis performed on the power lines to determine the undesired noise frequencies and to size capacitors appropriately. However, a generally safe assumption for slower speed electronics operating in the 50 MHz range is to create a smooth impedance in the range of 500 kHz to 500 MHz [1]. A capacitor only acts as a filter near its resonance frequency [1], therefore, a variety of different sized capacitors was used in order to produce the desired profile. In order for the resulting bypass network to be effective, there must be at least one capacitor for every power pin on the device. A good starting point for constructing a bypass network is to size the minimum and maximum capacitive value required for the desired frequency range and distribute

varying capacitor values throughout this range so that the total number of capacitors equals the total number of pins. However, since smaller capacitive values contribute less to the overall impedance profile [1], a greater number of these capacitors needs to be placed. In general, the number of capacitors should be doubled for every decade decrease in capacitance. Fig. 4.5 shows an example of a selection of capacitors for a 48 power pin device. Note that the quantity of capacitors increases as the capacitance decreases. Fig. 4.6 shows the impedance profile resulting from the capacitor selection of Fig. 4.5. This selection of capacitors results in a fairly even impedance over the range of 1MHz to 100 Mhz.

Quantity	Symbol	Package	Capacitive Values (μF)	Parasitic Inductance (nH)	Parasitic Resistance (ohms)
2	◇	E	680	2.8	0.57
7	▷	0805	2.2	2.0	0.02
13	◆	0603	0.22	1.8	0.06
26	■	0402	0.022	1.5	0.20

Figure 4.5: Bypass Capacitor Values

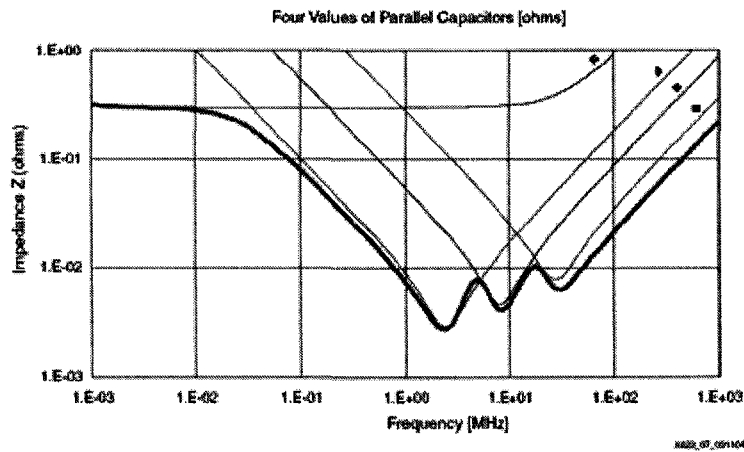


Figure 4.6: Bypass Capacitor Impedance Profile [1]

The placement of these capacitors on the PCB has a substantial effect on the per-

formance of the bypass network. This is because the routing to the bypass capacitor adds to the equivalent series inductance (ESL) of the device. Fig. 4.7 shows how an inductive loop can be formed in the cross-section view of the PCB, where current can travel through the power plane, up a via, across the capacitor, down a via and back through the ground plane. The size of this loop needs to be minimized to reduce the inductive effects contributing to the equivalent impedance of the capacitor. Due to this effect, the placement of a capacitor on the PCB can have a substantial effect on the effective resonant frequency of the capacitor. Fig. 4.8 shows the impedance profile of a capacitor and how the inductive and capacitive components influence the resonant frequency of the device. Basically, the capacitive component reduces the impedance, while the ESL of the device increases the impedance with respect to an increase in frequency. As the trace lengths between the power pin of a device and the bypass capacitor increases, the resonant frequency of the device shifts and may make the capacitor ineffective in filtering the desired noise from the system. Thus the bypass capacitors should be placed as close to the device power pins as possible, generally less than 100 mils away.

Another reason to keep to the bypass capacitors near the power pins of the device is to improve the response time of the capacitors to fluctuations in voltage. A small drop in charge at the device power pin takes time to propagate to the capacitor. The capacitor then releases charge which must then propagate back to the device. During this time, the voltage level could drop below the device's required operating range (if the distance to the capacitor is too large), indicating that the response time to these fluctuations is too slow.

For the camera board design, all bypass capacitors were kept within 100 mils of their respective power pins, and grounding vias were placed as near to the capacitor as possible to minimize the inductive loop's size, which is responsible for contributing to the ESL of the device. Fig. 4.9 shows two examples of critical paths that were

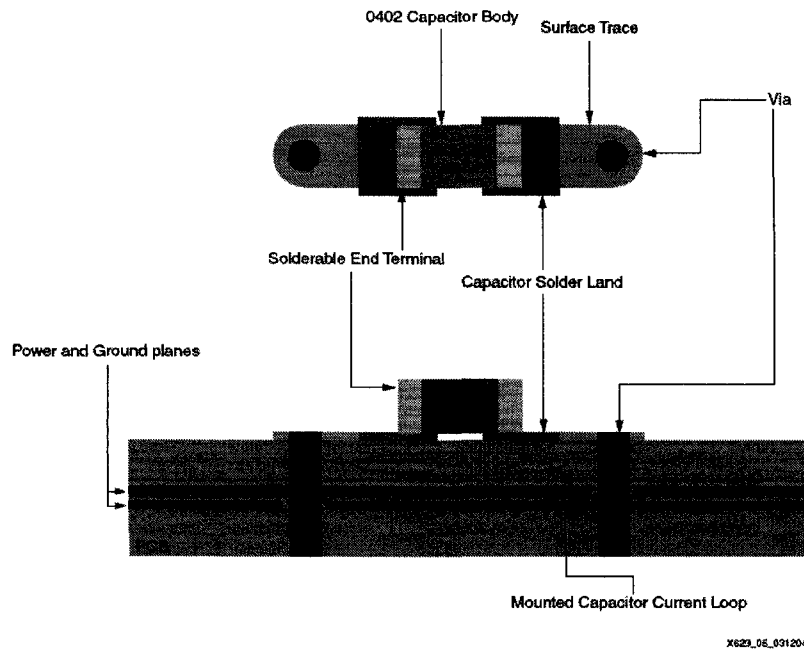


Figure 4.7: Capacitor Placement Inductive Loop[1]

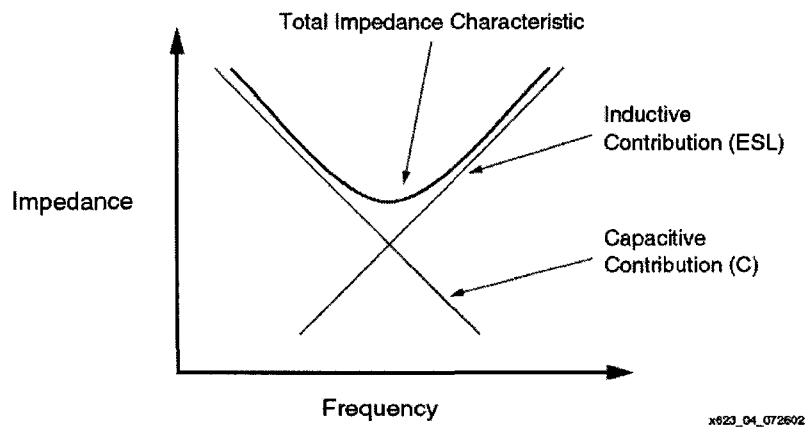


Figure 4.8: Capacitor Impedance and Resonant Frequency[1]

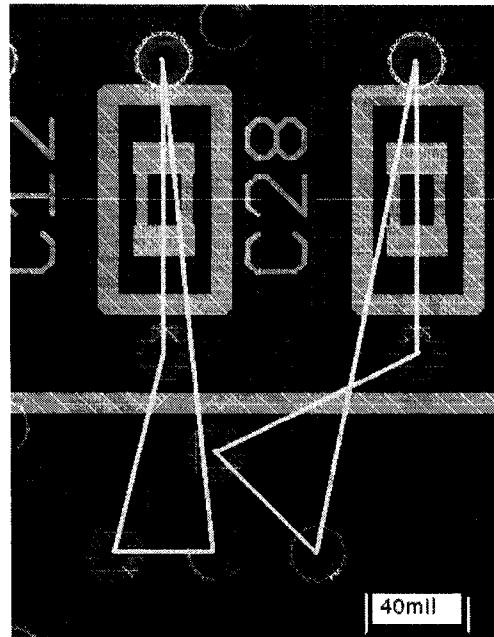


Figure 4.9: Bypass Capacitor Critical Current Path

routed in the power network where the estimated current path contributing to ESL is highlighted.

4.3.5 Routing

Routing for the design was performed in two parts. First, critical paths as well as power and ground connections were manually routed to ensure minimum bends, appropriate trace widths, and optimal connections. Next, automatic routing was performed to connect less critical nets. In the case of the FPGA, pin groups were setup to allow easy swapping of pin locations with pins that served the same function. If automatic routing of the data buses failed, pins were swapped as needed and automatic routing performed again until the entire design was successfully routed. Minimum trace widths used in the design were 5 mils with a minimum spacing of 4 mil required when breaking out signals from the FT256 footprint.

4.3.6 Manufacturing Files

The final step (after defining the PCB physical characteristics, placing components and routing connections between the components) was to generate the required files for manufacturing. These are listed below.

1. Board Stackup: defines the order of the four layers, board thickness and prepreg.
2. Drill Holes: defines location and size of holes for the PCB such as mounting points and through hole component footprints.
3. Copper Etch: defines the areas of copper on the four layers that compose the routing of the design.
4. Solder Mask: defines areas of the board protected against solder. This encompasses everything except where the component pins will be soldered to the copper board.
5. Solder Paste: defines areas where solder paste will be applied for mounting components.
6. Silk Screen: defines the colored print and artwork on the PCB used to identify components and component locations.
7. Pick and Place: defines the location of the components to be placed relative to the pick position on the components package.
8. Bill of Materials: list the components, quantities and identifier of all components mounted on the PCB.

9. Assembly Drawing: defines specification for how to assemble mechanical components on the PCB if this is required. This was not included in the camera design.

The required files listed above were provided to the PCB manufacturer, Sierra Proto Express, in Gerber RS-274-X format. The company successfully fabricated PCBs and populated the components to produce the final functioning cameras.

Chapter 5

HDL Blocks and Programming

5.1 FPGA Programming Overview

The FPGA was used in the camera design to act as the main controller to the various components within the camera. These functions included:

1. Frame timing and data synchronization
2. Digital Clock Management (DCM)
3. I²C communication
4. Data buffering FIFO
5. Output control
6. Image processing
7. Trigger delay and synchronisation

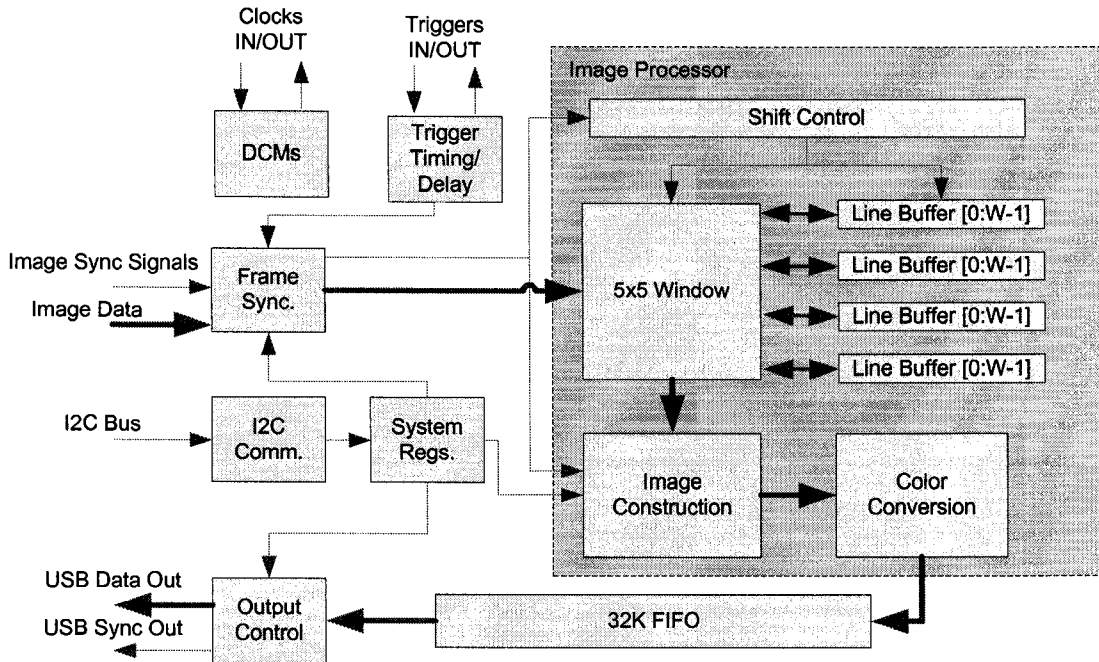


Figure 5.1: FPGA VHDL Modules and Data Flow

With these functions in mind, a block diagram was developed to show the data flow through the FPGA's various functional blocks. Fig. 5.1 shows the proposed architecture for the VHDL code developed for the FPGA in the camera design.

The following sections describe the development of these blocks and their VHDL realization.

5.2 Frame Timing and Data Synchronization

The first functional block that incoming image data from the MT9T001 sensor encounters is responsible for synchronising the image data with a data clock and using frame and line valid signals to track the row and column of current frame being outputted from the Micron sensor. According to the Micron MT9T001-3100 datasheet,

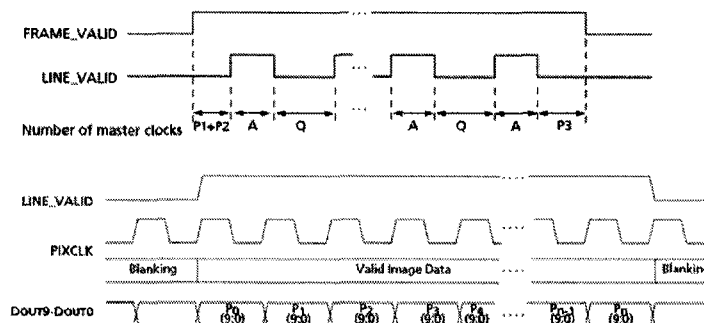


Figure 5.2: MT9T001-3100 Timing [10]

the data and timing signals appear as shown in Fig. 5.2.

Firstly, the process *SYNC_IN* in *frame_grabber_v1.vhd* latches the signal values of *LINE_VALID*, *FRAME_VALID*, as well as the current state of the data bus (*DIN*) on the rising edge of *PIXCLK*. The latched signals are also delayed by one clock cycle in order to identify subsequent rising and falling edges of these signals. Concurrently, the process *FRAME_CTLR* in *frame_grabber_v1.vhd* tracks the current row and column of data based on the latched signals above.

Essentially, the *FRAME_VALID* signal is used to indicate the start and end of a frame being read out from the sensor. While *FRAME_VALID* is high, the signal *LINE_VALID* is used to indicate the start and end of a new row of data being read out from the sensor. When both *FRAME_VALID* and *LINE_VALID* are high, either 8 bit or 10 bit data representing the intensity of the active pixel appears on the data bus and is at a stable state on the rising edge of *PIXCLK*. Fig. 5.3 shows how the *FRAME_VALID* and *LINE_VALID* signals are used to count the current row and column of data.

The register *FRAME_RESET* is used as part of the data output controller block to guarantee that a full frame of data is transmitted after the camera is triggered. This simply compensates for any loss of data in the case of a FIFO overflow. The

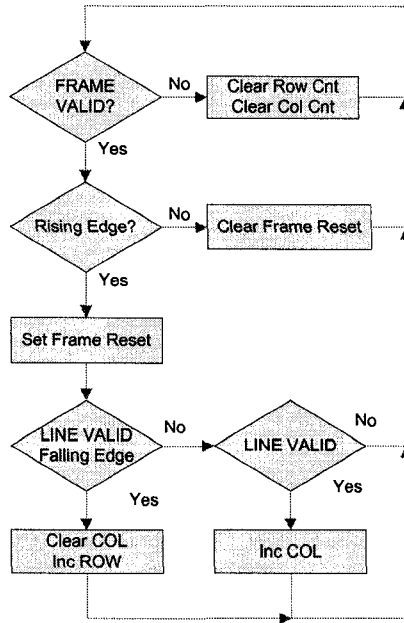


Figure 5.3: MT9T001-3100 Synchronization Flow

VHDL code for this process can be found in Appendix C.

5.3 I²C Write Slave

One of the requirements of the camera design was that all devices can share their settings and configurations via the I²C bus, as was shown in Fig. 2.5. However, there was no readily available I²C module suitable for the camera design. Thus, the most basic I²C operation, the write slave, was implemented in VHDL to allow the other devices on the bus, such as the Cypress USB Microcontroller, to send data to internal registers in the FPGA.

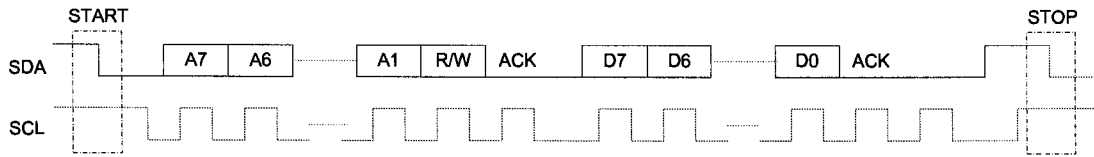
5.3.1 I²C Bus Overview

I²C is a simple two wire serial communication interface that consist of a data line (SDA) and a clock line (SCL). For this design, the I²C bus operates at 100 kHz. A unique feature of the I²C is multi-master support. Communication on the I²C bus begins when a master device transmits a *start* condition at a period when the bus is free. The bus is considered to be free again a certain time after a *stop* condition is sent. After the *start* condition, the master transmits an *address* followed by a read or write bit r/\bar{w} . The bus can operate with either 7 bit addresses or 10 bit address depending on the required number of components connected to the bus. For the camera I²C bus the 7 bit address mode was sufficient. Meanwhile, the I²C connected devices listen for this address and the appropriately addressed device sends an *acknowledge*. Data is then transfered in 8 bit words, each followed by an acknowledge from the receiving device [7].

A write slave essentially acts as an addressable memory in which a master controller, in this case the Cypress USB Microcontroller, can address and write data words to various internal registers. The slave device must be capable of the following operations:

1. Detect start/stop condition
2. Check device address and r/\bar{w}
3. Receive data words
4. Send acknowledge signal

The I²C manual in [7] outlines the following communication protocol. Fig. 5.4 shows the example timing for a master device claiming the bus with a *START* condition. The master then transmits the device address *ADDR* it wants to communicate

Figure 5.4: I²C Bus Communication

with. The master then waits to receive an acknowledge *ACK* from the slave device, which responds by pulling the SDA bus line low. The master then transfers 8 bits of data and again waits for a *ACK*. This process repeats until finally the master device terminates communication with a *STOP* condition.

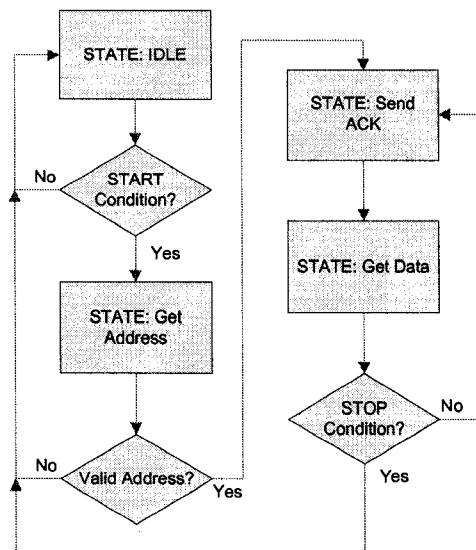
5.3.2 I²C Slave VHDL implementation

The approach taken to realize the I²C write slave device was to implement a FSM. The SDA and SCL lines of the I²C bus were sampled (using the 48 MHz clock supplied by the Cypress USB Microcontroller) at a fixed sampling interval. The sampling interval was computed based on an estimate of the rise time of the I²C bus when pulled high (as determined by the RC network resulting from the total line load capacitance and size of the pull-up resistors). The I²C lines were sampled every 64 clock cycles, which at 48 MHz translates to 750 kHz sampling rate. The sampled signals are delayed to ensure that at least two consecutive samples represent a single bus state, preventing a single sample of a transition from being considered a stable bus state.

The sampled data is then used as part of a FSM, making up the I²C slave controller. Fig. 5.5 shows the basic flow for the I²C FSM as well as the 5 states and their conditions for transition:

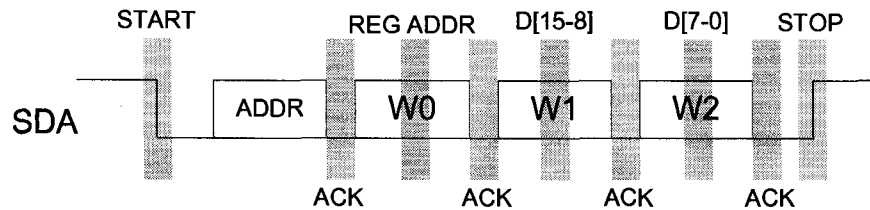
1. State 1 - IDLE: The device listens for a start condition
2. State 2 - ADDR: After a start condition, the device reads in a 7 bit address followed by a r/\bar{w} bit

3. State 3 - SND_ACK: Upon verifying itself as the addressed device, the SDA line is pulled low to indicate an acknowledge, otherwise the device returns to IDLE
4. State 4 - RCV_DATA: The device reads in an 8 bit word. Upon a successful read of 8 bit, it transitions to the SND_ACK state, otherwise it returns to IDLE.

Figure 5.5: I²C FSM

For the sake of simplicity, one state transition is not shown in Fig. 5.5. The omitted transition occurs when a *START* condition is detected while data is being received. If a *START* condition is detected, then the state machine returns to the ADDR state. According to I²C specifications, this situation would indicate that a new communication is being initialized and devices must listen for a new address. Also, although the state machine checks for a *stop* condition after receiving data, the condition on the bus itself is actually flagged by a parallel process and hence can occur anytime and not be missed.

The current state will remain in either ADDR or RCV_DATA until a full 8 bits of serial data is read from the SDA line. This can be seen in the *MainStateMach* process

Figure 5.6: I²C write in FPGA.

included in *I2c_SlaveCtrl.vhd*. The data is read off the SDA line and stored in a shift register which is enabled when the current state is ADDR or RCV_DATA. This same register is cleared when the current state is SND_ACK. A counter is used to monitor the number of incoming bits which is enabled and cleared in the same manner as the shift register.

Finally, the process *WriteI2cDataRegs* in *I2c_SlaveCtrl.vhd*. was developed to store the incoming data into an appropriate memory register in the FPGA. Registers were set up as 16 bit words to be referenced by an 8 bit address, and a communication standard was developed for the I²C master device to follow in order to write to these registers. Fig. 5.6 shows this communication format. Essentially, in the valid data transfer stage of normal I²C operation, the master device will transfer data in three word blocks. The first 8 bit word is used to address a register within the FPGA, the second 8 bit word carries the upper 8bits of the 16bit register value and the third 8 bit word carries the lower 8 bits of the 16 bit register value. The definition for each register can be found in the comments of this process.

5.4 Asynchronous FIFO

Part of the challenge of developing the FPGA code was that the system operated on two different clocks. Data read into the FPGA from the CMOS sensor operated on PIKCLK and data to be outputted to the Cypress USB Microcontroller needed

to be synchronized with IFCLK. The Asynchronous FIFO block of the FPGA was developed with two purposes:

1. To provide a small buffer to prevent data loss when transmitting data to the USB microcontroller
2. Bridge the two clock system

To realize the FIFO, Xilinx CoreGEN was used. The FIFO was simply sized as large as possible utilizing all available block rams of the FPGA. The data bus width was sized 16 bits wide to facilitate a single clock read operation as the output to the USB microcontroller is a 16 bit bus. The creation of the FIFO instances can be found in *frame_grabber_v1.vhd*. A separate read and write clock were specified and the FIFO full status simply ignored as this is later accounted for in the output controller block in the following section.

5.5 Output Controller

The output controller block was designed to act as a master device that writes image data to the slave configured FIFO of the FX2 USB microcontroller. The timing requirements of the Cypress device are outlined in the FX2 datasheet [4] and are shown in Fig. 5.7.

A write of the data lines is triggered when the SLWR line is pulled low. The data is actually written on the following rising edge of IFCLK. SLWR must then be asserted high. In this manner 16 bits are written to the FIFO every other IFCLK clock cycle. However, arbitrating data to the output bus and controlling the state of SLWR is only one function of the output controller. The controller must also ensure that a full frame of data is transmitted without loss of data to avoid any possible confusion on the PC side as to whether or not the incoming data encompasses an entire frame.

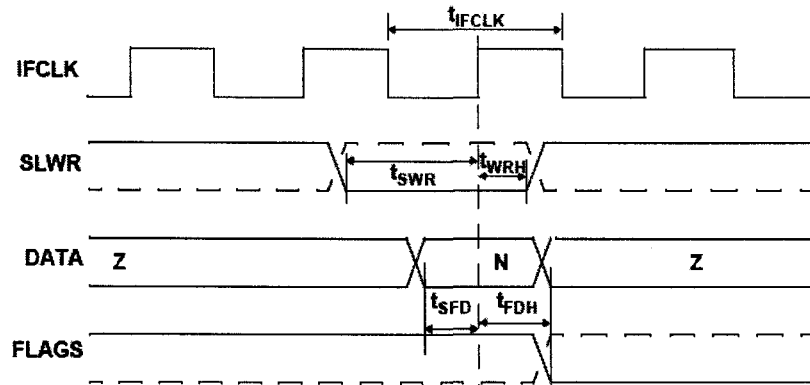


Figure 5.7: Timing for Cypress FX2 Slave FIFO

For this to occur, the output controller counts the number of bits transmitted with respect to the start of the frame and continues to output data until a complete frame worth of data has been transmitted. In the case of a FIFO overflow in the FPGA due to slow data transmission through the USB, the output data may appear erroneous and irrelevant, however the quantity of data will be consistent with a full image frame. The start of the frame condition was handled previously by the Frame Timing and Data Synchronization section above.

In addition, the controller must monitor the states of both the internal FPGA FIFO and the USB slave FIFO, ensuring that a write does not occur if either the FPGA FIFO is empty, or the USB slave FIFO is full. Table 5.1 shows the various cases that may occur and the status of the various registers and outputs under these cases.

This state table was implemented in the *USB_CLTR* process found in *frame_grabber_v1.vhd*. This generates the appropriate read and write signals for the both the internal FPGA FIFO and external USB slave FIFO, however is not responsible for arbitrating the appropriate data to the 16 bit FIFO data.

Condition	State	0		1	
	USB_FIFO_full	X	0	X	
	FPGA_FIFO_empty	X	0	1	X
	PIX_CNT < 0 and FRAME_VALID	X	X	1	X
Value	ram_ren	0	1	0	0
	slwr	0	0	0	1
	state	1	1	1	0
	PIX_CNT action	-1	-1	0	0

Table 5.1: Write Controller Cases

5.6 Image Processing Block

The image processing block in the FPGA was created for the option to pre-process image data coming from the CCD sensor. The primary use of the image processing block was to reconstruct incomplete sensor data (demosaicking) before data was transferred to the PC to allocate more cycle time to image inspection. The extra data created by reconstructing the image within the FPGA is easily handled by the USB transfer rates. Because the data was reconstructed in real-time in the camera, substantial time is saved in the PC which can be used for more complicated or thorough inspections. RGB to YUV colour conversion is also performed in this block, partially to help compress the data transferred to the PC.

The output format is 16 bit YUV with 8 bits allocated to the luminance channel Y and 4 bits to each the chrominance channels U and V. If only grayscale data is required, the chrominance channels can simply be ignored. This data is then buffered in a FIFO within the FPGA before being written to the FX2 FIFO for USB transfer.

The details behind the architecture for the image processing block as well as VHDL implementation is described in detail in Chapter 6.

5.7 Trigger Delay

The final process implemented in the FPGA is the ability to delay a trigger and control the output of various other triggers. Although this is not currently used in the system, the option for this type of control exists by routing the triggers through the FPGA. The *TRIGGER_CTLR* process in *frame_grabber_v1.vhd* was created for this purpose. The number of clock cycles to delay the trigger is controlled by I²C `data_regs(5)` with a default value of zero.

5.8 Synthesis Constraints and Results

The final VHDL code was synthesised using Xilinx ISE 10.1i for the Spartan-3E500-4 device. Constraints were specified for the two system clocks set to 48 MHz. The synthesizer option was set to minimize timing to guarantee the constraints were satisfied. The code was first synthesized to include the Edge Enhanced image processing algorithm described in the following chapter, and then synthesized with only the Nearest Neighbour and bilinear methods implemented. This was done because the current implementation of the Edge Enhanced method did not meet the required timing, which is further described in the conclusion of this thesis.

The I/O placement and timing constraints were specified in *frame_grabber_v1.ucf* and can be found in Appendix C.

Table 5.2 shows the synthesis report generated by ISE for the design that includes the Edge Enhanced demosaicking method and the Table 5.3 shows the synthesis report excluding this extra block.

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	1,016	9,312	10%
Number of 4 input LUTs	3,313	9,312	35%
Logic Distribution			
Number of occupied Slices	2,024	4,656	43%
Number of Slices containing only related logic	2,024	2,024	100%
Number of Slices containing unrelated logic	0	2,024	0%
Total Number of 4 input LUTs	3,549	9,312	38%
Number used as logic	3,305		
Number used as a route-thru	236		
Number used as Shift registers	8		
Number of bonded IOBs	45	190	23%
IOB Flip Flops	31		
Number of RAMB16s	19	20	95%
Number of BUFGMUXs	4	24	16%
Number of DCMs	2	4	50%
Number of MULT18X18SIOs	10	20	50%
Timing Results	Critical Path Delay		
PIXCLK	77.325 ns		
USBCLK	10.621 ns		
SCL	6.978 ns		

Table 5.2: FPGA Utilization with Edge Enhanced Demosaicking

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	676	9,312	7%
Number of 4 input LUTs	870	9,312	9%
Logic Distribution			
Number of occupied Slices	765	4,656	16%
Number of Slices containing only related logic	765	765	100%
Number of Slices containing unrelated logic	0	765	0%
Total Number of 4 input LUTs	1,032	9,312	11%
Number used as logic	854		
Number used as a route-thru	162		
Number used as Shift registers	16		
Number of bonded IOBs	45	190	23%
IOB Flip Flops	31		
Number of RAMB16s	19	20	95%
Number of BUFGMUXs	4	24	16%
Number of DCMs	2	4	50%
Number of MULT18X18SIOs	9	20	45%
Timing Results		Critical Path Delay	
PIXCLK	19.119 ns		
USBCLK	10.576 ns		
SCL	7.186 ns		

Table 5.3: FPGA Utilization without Edge Enhanced Demosaicking

Chapter 6

Image Processing

6.1 Demosaicking

When acquiring visual information, a Charge-Coupled Device (CCD) essentially operates by converting light intensity to charge in the Photoactive layer (an epitaxial p+ layer of silicon). This layer is made of a number of discrete points called pixels, each collecting its own charge which is proportional to the light intensity exposed on that region. Control hardware then shifts the cumulated charge to the neighbouring pixel, where the charge of the last pixel is dumped to a charge amplifier. The resulting output is a sequence of charges representing the row, or grid of pixels depending on the type of sensor: line scan or full frame. In order to obtain colour information, a Colour Filter Array (CFA) is applied over the Photoactive layer. The CFA filters light at each pixel, such that each pixel is only exposed to one spectrum of light. The most common CFA used in commercial electronics is the Bayer pattern, shown in Fig 6.1,

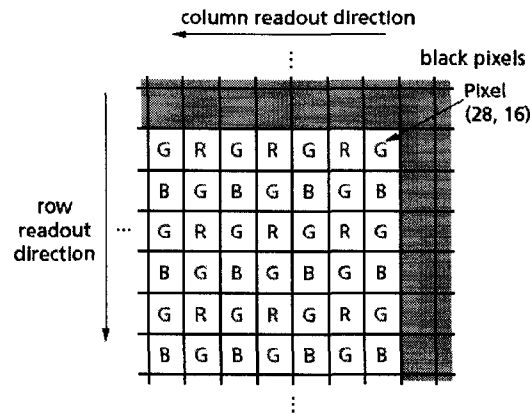


Figure 6.1: Bayer Pattern CFA on a CMOS Image Sensor

which is made up of alternating rows that expose red and green pixels and then green and blue pixels. The resulting image data from this type of sensor is not a full Red Green Blue (RGB) data, but rather three incomplete and phase shifted colour planes (seen in Fig. 6.1). The green channel appears redundantly to replicate the nature of the human eye, which has a greater ability to resolve green color information [2].

The process of constructing a full RGB image from CFA data is called demosaicking. There are currently a number of methods actively in use for performing this image reconstruction. The most basic of these was implemented in the camera application. In addition to this a more custom approach to image reconstruction was developed to perform more accurate reconstruction in real-time.

The first method implemented is called Nearest Neighbour. In this method, the missing colours at a pixel are simply copied from the neighbouring pixels. This is typically done in a set pattern, for instance, copying from the nearest pixel to the right and bottom of the current pixel being interpolated. On very high resolution images, this process may be acceptable, however when an image that has areas which exhibit sharp changes in contrast in a small region (a couple of pixels) is interpolated using this method, a “zipper” effect results. This method was implemented in the

camera design as an option for quick interpolation, mostly to verify the operation of the camera and its ability to successfully process data.

The next major set of demosaicking methods involve various types of interpolation, whereby the missing colors for a pixel are calculated as an average of neighbouring colours. The most common of these methods is bilinear interpolation, where only the nearest neighbours of similar colours are averaged. This interpolation was implemented into the camera as the predominate method for generating output data, however, the resulting image lacks accuracy at sharp edges.

Many more sophisticated methods have been proposed since, which attempt to identify regions or boundaries of objects in an image and interpolate along edges instead of across them, thus reducing the “zipper” effect and the appearance of colour mosaics (bleeding colours). These methods exploit image spacial or spectral correlation. Spacial correlation refers to the fact that within a homogeneous region of a natural image, neighbouring pixels share similar values. Spectral correlation refers to the fact that within these homogeneous regions, the ratio of colour planes (i.e. red to blue) are similar [9]. Two predominate methods involve the use of an edge indicator functions to form a weighted average of either the difference in neighbouring pixels [17] or the ratio of colours in neighbouring pixels [9]. Equations 6.1 and 6.2 show the general form of these methods respectively. For the purpose of demonstration, the calculation of the missing green at the 3rd row and 4th column is shown.

$$G_{34} = R_{34} + \left(\frac{\sum_{ij=24,33,44,35} e_{ij}(G_{ij} - R_{ij}^1)}{\sum_{ij=24,33,44,35} e_{ij}} \right) \quad (6.1)$$

$$G_{34} = R_{34} \left(\frac{\sum_{ij=24,33,44,35} e_{ij}(G_{ij}/R_{ij}^1)}{\sum_{ij=24,33,44,35} e_{ij}} \right) \quad (6.2)$$

Where e_{ij} is an edge weight function based on either gradients or other comparative criteria.

6.2 Hardware Implementations

The problem with methods such as [9] and [17] was that these processes of demosaicking required multiple iterations to attain accurate reconstruction, which these methods unsuitable for a real-time application. In addition, the edge weight functions used by these methods required complex operations, such as square root functions, which were difficult to implement in hardware. For these reasons, a novel demosaicking method was developed to improve the accuracy over simple bi-linear interpolation without the computational intensity and multiple iterations required by spacial and spectral correlation methods.

The objective of the new method was to improve the reconstruction accuracy over the bi-linear method, without the need to buffer much more data, and without introducing unnecessarily complex operations. Bilinear interpolation of a missing pixel color required that the four neighbouring pixel values be available. This required a 3x3 window of data to be available, which meant that at least two complete rows of image data need to be stored. This can be seen in Fig. 6.2.

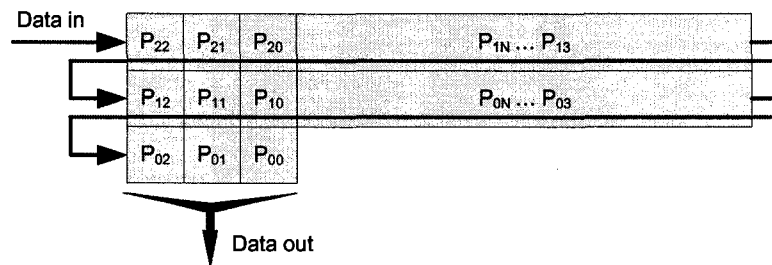


Figure 6.2: 3x3 Data Window for Bilinear Interpolation

The output from this block was complete RGB data and the input was single colour intensity depending on the current pixel being read in. The manner in which the missing values were computed depended on the colour of the center pixel P_{11} . The results for the missing colours were determined by the following formulae where

P'_{11} represented a missing colour value at the centered pixel:

$$P'_{11} = \frac{P_{12} + P_{21} + P_{10} + P_{01}}{4} \quad (6.3)$$

$$P'_{11} = \frac{P_{22} + P_{20} + P_{00} + P_{02}}{4} \quad (6.4)$$

$$P'_{11} = \frac{P_{12} + P_{10}}{2} \quad (6.5)$$

$$P'_{11} = \frac{P_{21} + P_{01}}{2} \quad (6.6)$$

The following table shows how the output is generated for 4 different cases:

Case	Missing Colour	Equation
Red Center	Green	6.3
	Blue	6.4
Blue Center	Green	6.3
	Red	6.4
Green Center on Red row	Red	6.5
	Blue	6.6
Green Center on Blue row	Red	6.6
	Blue	6.5

Table 6.1: Bilinear Output

The hardware implementation of such a method was quite straight forward. The window of data was implemented as a series of registers, connected to two line memories. The equations were computed on each cycle and the output was multiplexed depending on the current row and column which was determined by the frame controller block. The VHDL implementation for this method can be found in *ImageProcessor.vhd* in Appendix C.

However, in order to attain highly accurate image reconstructions, information on the changes in the colour planes as well as across colour planes needed to be used, and the direction of weighted interpolation was determined based on this gradient

information. For this, a larger 5x5 window of data was required. The architecture for implementing this in hardware was an expanded version of the 3x3 window (shown in Fig. 6.3).

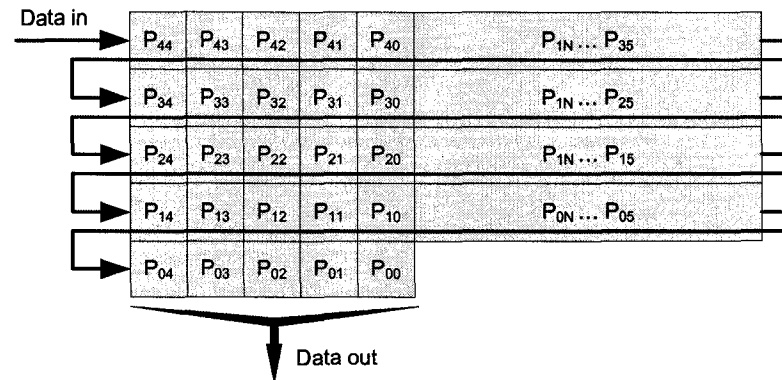


Figure 6.3: 5x5 Data Window for Edge Weight Function

6.3 Edge-Enhanced Real-Time Hardware Demosaicking

A literature review of real-time hardware demosaicking revealed very limited techniques. Some of these involved complex edge weight functions implemented in Look-Up Tables (LUTs). Other methods used slightly modified alternatives to bilinear interpolation. The techniques of [17] or [9] by simplifying the edge weight functions to be more cost-effective to implement in hardware as well as remove the requirement for multiple iterative processing, without substantially compromising accuracy. [17] was selected as a basis since it exploited spacial correlation which used subtraction to interpolate missing data. This was selected over spectral correlation (as in [9]) which used division or ratios, because subtraction was far more cost effective to implement in hardware.

The next step was to ascertain an effective edge weight function. In order to easily implement and test various functions, a series of MATLAB programs were developed to perform this task. First a function *RGB2RAW.m* was created to convert the full colour RGB data into its equivalent RAW data which appears from a CFA Bayer pattern. Nearest Neighbour as well as bilinear demosaicking were implemented in *demosaic.m* (found in Appendix D). Finally, the edge enhanced method found in *ee_demosaic.m* was iteratively developed and compared to Nearest Neighbour and bilinear in *demosaic_test.m*.

The first step in creating the Edge Enhanced demosaicking method was to fully recreate the green colour plane, as this plane comprises 50% of the incoming image data. In order to do this, an edge weight function was created by not only examining intensity changes between green pixels in a certain direction, but also considering changes in intensity of the red and/or blue channel. The final edge weight function can be seen in Equation 6.7 where $D_{i,j}(u, v)$ denotes the difference of the (i,j) pixel in the (u,v) direction.

$$D_{i,j}(u, v) = |P_{i+2u, j+2v} - P_{i,j}| + |P_{i+u, j+v} - P_{i-u, j-v}| + |2 * P_{i+u, j+v} - P_{i+v, j+u} - P_{i-v, j-u}| \quad (6.7)$$

where: $(u, v) \in \{(\pm 1, 0), (0, \pm 1)\}$

The four resulting edge weights are compared to a threshold value. If the edge weight in a given direction was less than the threshold value, then the interpolation was performed in that direction. Interpolation was performed using spacial correlation where the change in colour intensity in one colour plane was assumed to be the same in the other colour planes [9]. Equation 6.8 shows how a missing green value was calculated, where $P_{i,j}^G(u, v)$ denotes the interpolated value of the missing green pixel in the (u, v) direction.

$$P_{i,j}^G(u, v) = P_{i+u, j+v} + \frac{(P_{i,j} - P_{i+2*u, j+2*v})}{2} \quad (6.8)$$

where: $(u, v) \in \{(\pm 1, 0), (0, \pm 1)\}$

and : $(i, j) \in (i\%2 \neq j\%2)$ for missing green values.

The resulting interpolated values for each direction, which satisfied the threshold value, was then averaged to produce the final missing green value. If none of the edge weights satisfied the threshold criteria, then the missing green was computed as the average of the interpolation in all directions.

The missing red at a blue pixel or missing blue at a red pixel was then computed in a similar manner, except that instead of four directions of interpolation, the vertical directions were considered as one case and the horizontal directions were considered as another case so only two possible directions of interpolation exist. The same edge weight function in Equation 6.7 was used, but with both vertical directions $(u, v) = (\pm 1, 0)$ compared to another threshold value (*threshold2*) to see if either edge weight value satisfied the criteria. The same was done for the two horizontal directions $(u, v) = (0, \pm 1)$. Data for the missing red or blue pixel $P_{i,j}^x(u, v)$ was calculated according to Equation 6.9 using the newly calculated missing green value $P_{i,j}^G$ from Equation 6.8.

$$P_{i,j}^x(u, v) = \left[\left(\frac{P_{i+u+v, j+u+v} - P_{i-u-v, j+u+v}}{2} + P_{i,j}^G - P_{i+v, j+u} \right) + \left(\frac{P_{i+u+v, j-u-v} - P_{i-u-v, j-u-v}}{2} + P_{i,j}^G - P_{i-v, j-u} \right) \right] / 2 \quad (6.9)$$

$(u, v) \in \{(1, 0), (0, 1)\}$

and : $(i, j) \in (i\%2 \neq j\%2)$

and : $x = \begin{cases} \text{R for } i=2k; k \in \mathbb{N} \\ \text{B for } i=2k+1 \end{cases}$

Finally, the missing red and blue values at the known green values were computed. For this, a different edge weight function was used as criteria for the direction of interpolation. This edge weight function was computed similarly to the edge weight seen in Equation 6.7, however, with a slight variation to the last term as can be seen in Equation 6.10:

$$D_{i,j}(u, v) = |P_{i+2*u, j+2*v} - P_{i,j}| + |P_{i+u, j+v} - P_{i-u, j-v}| + \frac{|P_{i+u+v, j+u+v} - P_{i,j}| + |P_{i+u-v, j-u+v} - P_{i,j}|}{2} \quad (6.10)$$

where: $(u, v) \in \{(\pm 1, 0), (0, \pm 1)\}$

and : $(i, j) \in (i\%2 = j\%2)$ corresponding to missing Red and Blue values.

In this scenario, instead of averaging the interpolated pixel values in the direction that satisfies a threshold criteria, only interpolation in the direction of the minimum difference was performed. However, since interpolation, in this case, required information from pixels adjacent to the direction of interpolation, the minimum perpendicular edge with respect to the minimum edge weight was also considered. $P_{i,j}^x(u1, v1, u2, v2)$ denoted the interpolated pixel value in the $(u1, v1)$ direction where $(u2, v2)$ corresponded to the minimum $D_{i,j}(u, v)$ perpendicular to $(u1, v2)$.

$$P_{i,j}(u1, v1, u2, v2) = P_{i+u2, j+v2} + \left(P_{i,j} - \frac{P_{i+u1, j+v2} + P_{i-u1, j+v2}}{2} \right) \quad (6.11)$$

where: $(u1, v1) \in \{(\pm 1, 0), (0, \pm 1)\}$

and : $(i, j) \in (i\%2 = j\%2)$ for missing Red and Blue values.

$$\text{and : } x = \begin{cases} R \text{ for } i = 2k; k \in \mathbb{N} \\ B \text{ for } i = 2k+1; \end{cases}$$

The remaining missing colour not computed in Equation 6.11 was calculated according to Equation 6.9 in the direction corresponding to $(u1,v1)$ in Equation 6.11. The result was full RGB data for each pixel.

It was observed that all calculations were easily implemented in hardware as additions and shifts. The only multiplication that occurred was when three interpolated green values needed to be averaged. For instance, instead of dividing by 3, multiply by 21 and divide by 64 (shift by 6) since $21/64 \approx 0.328$.

6.4 Implementation and Results

The three methods of demosaicking described above were all implemented into a Spartan3E FPGA. A test bench *Image_Processor_TB.vhd* was created to verify these methods and can be found in Appendix C. The testbench reads a RAW data file generated in MATLAB and writes the output normally sent to the USB microcontroller, to a file. This output data was read in MATLAB to generate an image file and the results of the output were compared to the original image before it was converted to RAW data.

Table 6.2 shows the Y channel results of the three methods of interpolation as well as the Mean Square Error (MSE) and Peak Signal to Noise Ratio (PSNR) when compared to the original. The test image was selected based on its sharply defined edges that are not completely vertical or horizontal as these were difficult to accurately reconstruct.

The Edge Enhanced method showed very promising results, with the lowest MSE and most visually accurate reconstruction of the edges. There was a visible reduction

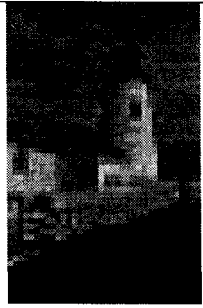
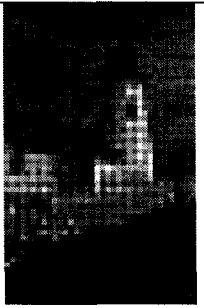

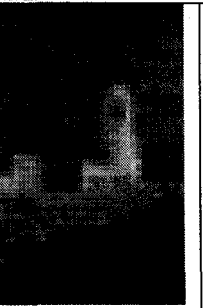
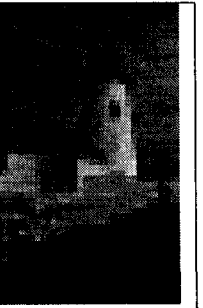
Original	RAW	NN	Bilinear	EE
				
MSE=0 PSNR= ∞	N/A N/A	60.7346 69.7601	25.3433 78.5001	9.2505 88.5785

Table 6.2: Demosaicking Results

in the jagged “zipper” edges present in the Bilinear and Nearest Neighbour results. Also, the Edge Enhanced image showed improved contrast over the other demosaicking methods.

Although each algorithm was successfully synthesised in the Spartan-3E FPGA, no definitive timing or utilization results have been included, as the VHDL code for the Edge Enhanced method has not been optimized. Optimizations must still be made to improve the critical path delay of the Edge Enhanced method, as it currently exceeds the required timing requirements.

Chapter 7

Conclusion

7.1 System Integration

The final design files for the camera PCB were sent to a California based company, Sierra Proto Express, for manufacturing. The company fabricated the PCBs according to the Gerber files provided and populated the components listed in Table B.1 in Appendix B. The final design can be seen in Fig. 7.1. The final cost to manufacture the cameras was \$300 CAD per camera including component cost, component population, PCB fabrication and shipping. The lead time for the manufacturing of these cameras was less than 10 days from order to delivery.

Before implementing the camera into the complete vision system, the camera boards were powered, programmed and tested with software developed by Neil Scott. If USB communication, serial slave programming of the FPGA and I²C communication were all successful, then the camera was connected to the triggers generated

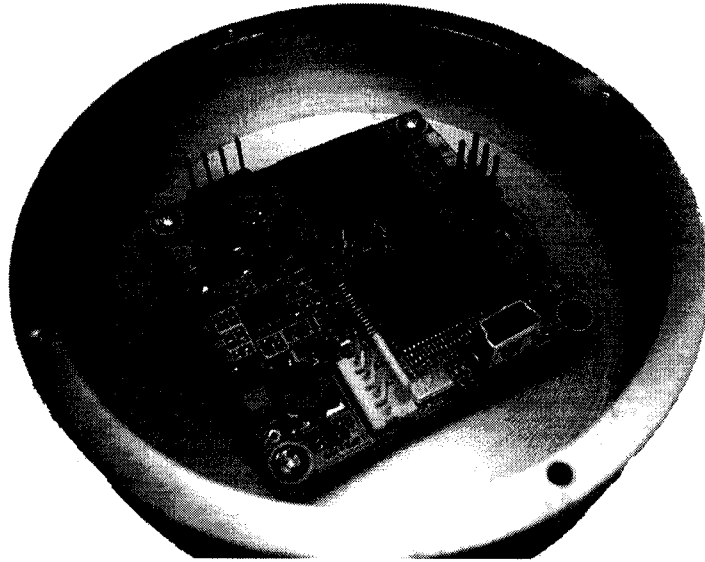


Figure 7.1: Camera PCB in an Enclosure

by the control board by Neil Scott and the output image data checked for errors. Cameras that operated correctly were mounted into the existing circular enclosures and integrated into a quadrant of the final MV system.

Fig. 7.2 shows the final images generate by the camera and imaging setup. Together, the four cameras capture 360° of the capsules body. The images in Fig. 7.2 were reconstructed using bilinear interpolation as the demosaicking method and the output is a gray scale image. This output format seems to suffice for the time being, however the possibility for enhanced demosaicking and YUV images is available. The image size seen in Fig. 7.2 is 750×245 pixels, covering approximately a 3 cm x 1 cm region in space. This corresponds to a 0.4 mm per pixel resolution, slightly lower than the desired output. This is because the current sensor configuration is set to 2x binning, where only every other row and every other column of the image sensor is exposed. This results in only $1/4$ of the total image resolution of the camera being used to capture the imaging region. This can be increased to the sensors full resolution, as the PC based image processing capabilities expand. The image resolution without

binning would then be 0.1 mm per pixel and satisfy the operational requirements of the system.

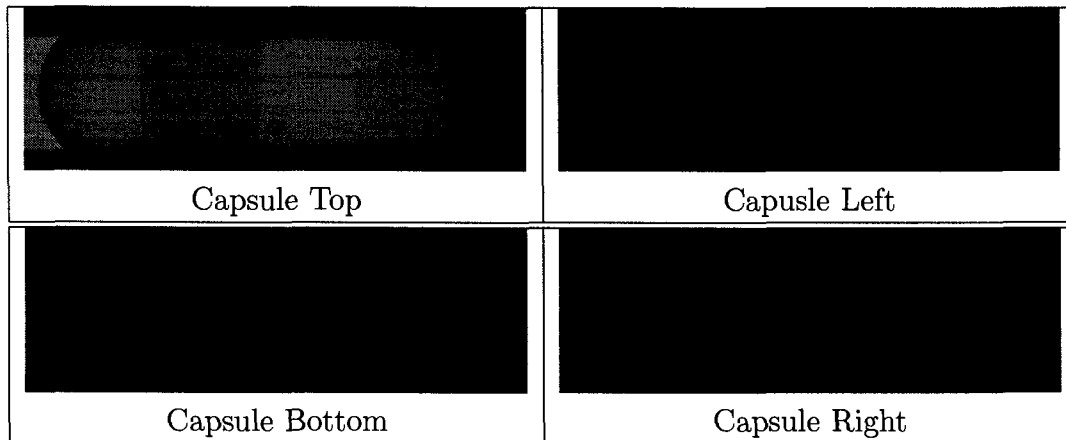


Figure 7.2: Final Capsule Images

At the moment, the MV system is operating stably at around 80% of the desired throughput, processing approximately 48 000 capsules per hour. This is due to the PC based image inspection algorithms taking place. The image processing algorithms are still in an experimental stage and will become faster and more efficient in the near future. The hardware designed in this thesis has been tested successfully for 60 000 capsules per hour at a resolution up to 1536x1024 which is more than sufficient for the required resolution. There is still much work to be done on the complete MV system before a commercialized product can exist. Still, the basis for a functional system that will meet the desired operating requirements and exceed the desired cost requirements was successfully created. This includes: designing and building a backplate to the camera enclosures with a suitable industrial I/O connector and improving the speed performance of the proposed edge enhanced image demosaicking method.

7.2 Summary

This thesis covered the specifications, development and design of a custom digital camera suitable for the application of pharmaceutical capsule inspection. This involved understanding the nature of the inspections to take place in order to determine the required operating specifications. The most important of these were: a throughput rate of 60 000 capsules per hour and an image resolution of 0.1 mm per pixel.

As a basis for the design, an existing, but out of date, MV system (for sorting capsules) was used. The objective of the project was to upgrade and retrofit the existing machine with custom electronics, in order to satisfy the outlined requirements. A custom hardware design approach was taken in order to minimize the system cost and provide maximum flexibility for future upgrades. This meant that suitable cameras and image processors needed to be selected, as well as an interface for the two. PCs were selected to perform the image inspection because they could be easily upgraded at a low cost. After close comparison of different possible cameras, a USB 2.0 interfaced custom camera was the desired solution, as this provided a cost effective solution with the desired data transfer rates. Using the operating and business specifications, the requirements for the camera and a general functional diagram were composed. The design methodology for creating a custom USB 2.0 digital camera was then outlined and the development and testing methods defined.

The first step in the bottom-up design of the camera was to select components, such as an appropriate image sensor and FPGA, that would meet the required specifications of resolution, flexibility and speed. After all the required components were selected, these were organized into functional groups and electrical schematics were created. This involved the use of Eagle CAD for designing the electrical schematics and PCB layout. Many design decisions were made when creating the schematics which include: Sizing the components of the TPS75003 triple supply to satisfy the

power requirements of the FPGA, setting up appropriate bypass capacitor networks, planning the digital and analog powering of the various devices and making all the appropriate I/O connections. The resulting schematics were then realized as a physical layout on a four layer PCB that was later manufactured and implemented into the MV system.

The other main design aspect of this project involved the development of VHDL code for the camera's FPGA. The FPGA acted as the main hardware controller on the device, interfacing the imaging sensor to the USB 2.0 microcontroller. The FPGA also was setup as a reconfigurable image processor, where the type of image demosaicking, color output and size of the incoming image could be changed through an I²C interface without reprogramming the device. The final VHDL coding included blocks for: Synchronising incoming image sensor data, reconstructing the CFA pattern into complete RGB data, converting this data to YUV format, and buffering the data in a FIFO before transferring it to the USB 2.0 device through an output controller block. The FPGA also had control of the distribution and delay of a trigger signal. This gave the flexibility for a single trigger to be delayed and outputted, allowing the multiple cameras of a quadrant to be daisy chained and operate off a single trigger from the control board by Neil Scott if necessary.

The VHDL code, specifically for the Image Processing block, had an associated test bench designed to read and write input/output data. Using this test bench, RAW image data in the form of a Bayer CFA pattern could be supplied to the image processing block and the processed output analyzed for accuracy. The test data was generated in a MATLAB function and the output data analyzed similarly. Intermediate test benches not included in this thesis were also used to verify the I²C write slave block as well as the output controller.

The final VHDL code was synthesized using Xilinx ISE 10.1 and a binary file was generated for programming the FPGA. Programming was ultimately done through

the USB microcontroller, as the FPGA was configured as a serial slave device for this purpose. The final device utilization for the FPGA was approximately 50% for LUTs and logic blocks and 90% for Block ram memories, with the Edge Enhanced demosaicking method included, indicating that the Spartan-3E500 device was an appropriate selection for this application.

The final manufactured cameras were tested as described above the implemented into the MV system.

7.3 Future Work

Although the current camera design proves the concept that a custom hardware approach can be an effective solution for the image acquisition component of the MV system, there is still some development required before the entire system can be sold as a commercial product. The two main areas for future work relevant to custom camera include: Design and build of a backplate for the camera enclosure and improving the synthesis of the edge enhanced demosaicking method.

The PCB camera boards were designed to optimise the area utilization of the PCB and thus was not overly concerned with the placement of I/O connectors on the PCB. The camera boards were mounted into the existing mechanical enclosures, however, the existing backplates for these enclosures do not provide the appropriate openings to allow the USB mini B connector, triggers and I²C connector access to the board. A new set of backplates must be designed in order to completely enclose the camera, while allowing for industrial standard connections into the enclosure for the connectors listed above. This may require that the PCB board level connectors be changed or moved, although modifications to the PCB design should be avoided.

In addition, the hardware synthesis of the edge enhanced demosaicking method needs to be refined. Currently, the synthesis of this image processing block does not

meet the timing constraints, as a number of operations are being sequentially executed. An investigation of the formulas used in this demosaicking method show that the maximum critical path for the algorithm includes 8 additions, 2 comparators and a LUT. These operations are performed on 8 bit data and should be able to be implemented to meet the 48MHz requirements of the Image Processor block. Modifications to the synthesis of the block is currently being performed.

Overall, the logical implementation of the device was successful. The desired image resolution and throughput was achieved. The architecture of the image processing block can successfully perform operations within a sliding 5x5 windows and can be reconfigured to suite the needs of the final MV system. The physical size of the board suites the existing enclosures, however a final back panel to this holder with the appropriate cable holes will still need to be designed. Ultimately, this custom USB camera design can be used as a strong foundation for the development of a cost effective commercial product suitable for a range of machine vision applications, specifically the inspection of pharmaceutical two-part gelatin capsules.

References

- [1] Mark Alexander. *Power Distribution System (PDS) Design: Using Bypass/Decoupling Capacitors*. Xilinx, February 2005. XAPP623 v2.1.
- [2] Bryce E. Bayer. U.s. pat. 3971065: Color imaging array, July 1976.
- [3] Compaq et al. *Universal Serial Bus Specification*, April 2000. Revision 2.0.
- [4] Cypress. *CY7C68013 EZ-USB FX2 USB Microcontroller High-Speed USB Peripheral Controller*, June 2002. 38-08012 Rev. *B.
- [5] Pharmaphil Inc. Requirements and specifications document. Internal document, November 2004.
- [6] IPC. *Generic Standard on Printed Board Design*, February 1998. IPC-2221.
- [7] Jean-Marc Irazabal and Steve Blozis. *Application Note: I2C Bus*. Philips Semiconductors, March 2003. AN10216-01.
- [8] A. Karloff, N. Scott, and R. Muscedere. A flexible design for a cost effective, high-throughput inspection system for pharmaceutical capsules. In *Proc. IEEE International Conference on Industrial Technology*, April 2008.
- [9] Ron Kimmel. Demosaicing: Image reconstruction from color CCD samples. *IEEE Transactin on Image Processing*, 7(3), 1999.
- [10] Micron Technology Inc. *1/2-Inch 3-Megapixel CMOS Digital Image Sensor*, June 2005. MT9T001_3100_DS_2.fm - Rev.D.
- [11] Jean P. Nicolle. <http://www.fpga4fun.com/FPGAinfo1.html>, January 2008.
- [12] NXP. *I2C-bus Extender: Product data sheet*, May 2008. P82B715 (rev. 07).
- [13] Christopher T. Robertson. *Printed Circuit Board Designer's Reference: Basic*. Prentice Hall, Upper Saddle River, NJ, 2004.

- [14] James Swarbrick. A history of dosage forms and basic preparations. In *Encyclopedia of Pharmaceutical Technology*, volume 7, pages 304–306. Informa Health Care, 1998.
- [15] Texas Instruments. *triple-Supply Power Management IC for Powering FPGAs and DSPs*, June 2007. SBVS052G.
- [16] D. Vernon. *Machine Vision: Automated Visual Inspection and Robot Vision*. Prentice Hall, 1991.
- [17] Ping Xue Xiaomeng Wang, Weisi Lin. Demosaicing with improved edge direction detection. *IEEE International Symposium on Circuits and Systems*, 3, 2005.
- [18] Xilinx. *Four-and Six-Layer, High-Speed PCB Design for the Spartan-3E FT256 BGA Package*, October 2006. XAPP489 v1.0.
- [19] Xilinx. <http://www.xilinx.com>, June 2008.
- [20] Xilinx. *Spartan 3E Family FPGA: Complete Datasheet*, April 2008. DS312-1 (v3.7).
- [21] Xilinx. *Spartan-3E FPGA Family: Complete Data Sheet*, April 2008. DS312.

Appendix A

System Requirements

Table A.1: Defect List and Tolerances[5]

	Defect	Maximum allowable size
Critical	Oil Hole	0.2 mm or larger
Critical	Scrape Hole	0.2 mm or larger
Critical	Cracked	0.2 mm or larger
Critical	Short Body	2.0 mm under spec
Critical	Double Dip	Large Defect
Critical	Telescoped	Large Defect
Critical	Mashed	Large Defect
Critical	Trims	Large Defect
Critical	Uncuts	Large Defect
Critical	Splits	1.0 mm or larger
Continued on next page		

Table A.1 – continued from previous page

	Defect	Maximum allowable size
Critical	Large Strings	5.0 mm or larger
Critical	Black Grease Mark	0.2 mm or larger
Critical	Closed Capsule	Large Defect
Critical	Long / Short Joining	within 1.0 mm of spec
Critical	Thin Spots	TBD
Critical	Rough Cuts	1.0 mm or larger
Critical	Short Cap	1.0 mm under spec
Critical	Long Body	1.0 mm over spec
Critical	Long Cap	1.0 mm over spec
Critical	Loose Pieces	Large Defect
Critical	Bad Join	Large Defect
Critical	Double Cap	Large Defect
Critical	Punched End	1.0 mm or larger
Critical	Side Corrugation	Any
Critical	Collet Pinches	0.2 mm or larger
Minor	Splits	0.2 mm or larger
Minor	Rough Cuts	0.5 mm - 1.0 mm
Minor	Collet Pinches	3.0 mm or larger
Minor	Bubbles	0.5 mm or larger
Minor	Wrinkles	2.0 mm or larger
Minor	Star Ends	3.0 mm or larger
Minor	Dye Spec	1.0 mm or larger
Minor	Dirt Marks	0.2 mm or lager
Minor	Strings	2.0 mm - 5.0 mm
Minor	Punched Ends	0.2 mm or larger
Continued on next page		

Table A.1 – continued from previous page

	Defect	Maximum allowable size
Minor	Grease Rings	0.2 mm or larger
Minor	Scrapes	1.0 mm or larger

Requirement	Acceptable	Marginal
1 Good capsules falsely rejected	<1%	<5%
2 Critical defects falsely accepted	0	0
2b Minor defects falsely accepted	0	0.5%
3 Speed of capable inspection	1000 caps/min	750 caps/min
4 Products inspected	All sizes and colours	Size 0 Natural
5 Vendor facility validation	Must pass	Must pass
6 In process validation	Must pass	Must pass

Table A.2: High Level Business Requirements[5]

Requirement	Description
1 Size	The system must recognise and inspect sizes (00, 0, 1, 2, 3, 4)
2 Colours	The system must recognize and inspect all colours
3 Defects	Recognize and count and eject all defects (See A.1)
4 Product Count	Count good capsules and display
5 Remove Defects	Identify and remove pieces exhibiting defects automatically
6 Operator Friendly	Must have intuitive and secure method of user operation
7 User Manual	Must have complete user manual with visual aids
8 Admin Manual	Must have complete administrator manual
9 Source Code	Must include source code
10 Self Testing	Detect setup problems such as camera focus or dirty pocket
11 Security	Systems must be secure and lockable
12 Markings	System must have all hoses, wires, switches labelled
13 Touch Screen	For first system to determine if ideally suited for the process
14 Drawings	System drawing (blue prints)
15 Validation	System driven validation process for lot and shift change
16 Count	Count to a preset quantity and close or redirect output

Table A.3: High Level Performance Requirements[5]

Scenario	
1	Train vision system to analyze product
2	Use the vision system to inspect product as it is being manufactured
3	Use the vision system to inspect product after the product has been manufactured
4	Computerized validation process for each item change (system prompted)
5	Computerized validation/challenge for each shift (system prompted)

Table A.4: Business Scenarios[5]

Appendix B

Camera Board Schematics

The following are the final circuit schematics used for the camera board design:

Table B.1: Bill of Materials

Ref	Description	Value	Manufacturer	Manufacturer Part No.	Qty	Package
C1,C2	16V Ceramic Cap, 10%	0.1u	Panasonic	ECJ-1VB1C104K	2	0603
C3	6.3V Ceramic Cap, 10%	1u	Panasonic	ECJ-1VB0J105K	1	0603
C4, C5	50V Ceramic Cap, 10%	1500p	Panasonic	ECJ-1VB1H152K	2	0603
C6, C44	16V Ceramic Cap, 10%	10000p	Panasonic	ECJ-1VB1C103K	2	0603
C7	6.3V Ceramic Cap, 20%	100u	Murata	GRM31CR60J107ME39L	1	1206
C8	6.3V Ceramic Cap, 20%	10u	Panasonic	ECJ-1VB0J106M	1	0603
C9	50V Ceramic Cap, +- 0.5p	10p	Panasonic	ECJ-1VC1H100D	1	0603
C10, C11	6.3V Tantalum Cap, 20%	100u	Kemet	T491B107M006AT	2	3528-21
C12-C43, C47- C57, C69-C74	10V Ceramic Cap, 10%	0.1u	Panasonic	ECJ-0EB1A104K	49	0402
C45, C46	50V Ceramic Cap, +- 0.25p	10p	Panasonic	ECD-G0E100C	2	0402
C58	16V Tantalum Cap, 10%	1u	AVX	TACL105K016XTA	1	0603
C59, C63, C66	6.3V Ceramic Cap, 10%	1u	Panasonic	ECJ-0EB0J105K	3	0402
C60, C64, C67, C75-C77, C79	16V Ceramic Cap, 10%	10000p	Panasonic	ECJ-0EB1C103K	7	0402
C61, C65, C68	6.3V Ceramic Cap, 20%	2.2u	Panasonic	ECJ-0EB0J225M	3	0402
C62, C78	16V Tantalum Cap, 20%	2.2u	AVX	TACL225M016XTA	2	0603
C80, C81	10V Tantalum Cap, 20%	10u	AVX	TACL106M010XTA	2	0603
CONN1	USB Mini-B, vertical	-	Molex	500075-0517	1	Mini-B
D1	20V Schottky Diode	2A	Vishay	SS22-E3/52T	1	DO214-AA
D2	20V Schottky Diode	1A	ON Semicon.	MBRM120ET3G	1	PowerMITE
JP1	36 pin breakable header	4pin	3M	929400-01-36	1	0.1" TH
JP2	18 pin, 2 row header	6pin	3M	929710-10-09	1	0.1" TH
JP3	4pin, friction pin header	4pin	3M	640454-4	1	0.1" TH
L1	1.4A Inductor, 30%	15u	Sumida	CDRH6D28NP-150NC	1	7x7x3mm
L2	2.4A Inductor, 30%	5u	Sumida	CDRH6D28NP-5R0NC	1	7x7x3mm
LED1	7.6 milicandela, 2.2V Green	20mA	Stanley	PG1112C-TR	1	0603
Q1, Q2	1.8V P-Channel MOSFET	2.4A	Fairchild	FDN304PZ	2	SOT-3
R1, R2	1/2W Resistor, 1%	0.033	Susumu	RL1632S-R033-F	2	1206
R3, R6	1/10W Resistor, 1%	61.9k	Panasonic	ERJ-3EKF61R9V	2	0603
R4	1/10W Resistor, 1%	15.4k	Panasonic	ERJ-3EKF15R4V	1	0603
R5	1/10W Resistor, 1%	36.5k	Panasonic	ERJ-3EKF36R5V	1	0603
R7, R9, R20	1/10W Resistor, 1%	4.7K	Panasonic	ERJ-2RKF4701X	3	0402
R8	1/10W Resistor, 1%	330	Panasonic	ERJ-2RKF3300X	1	0402

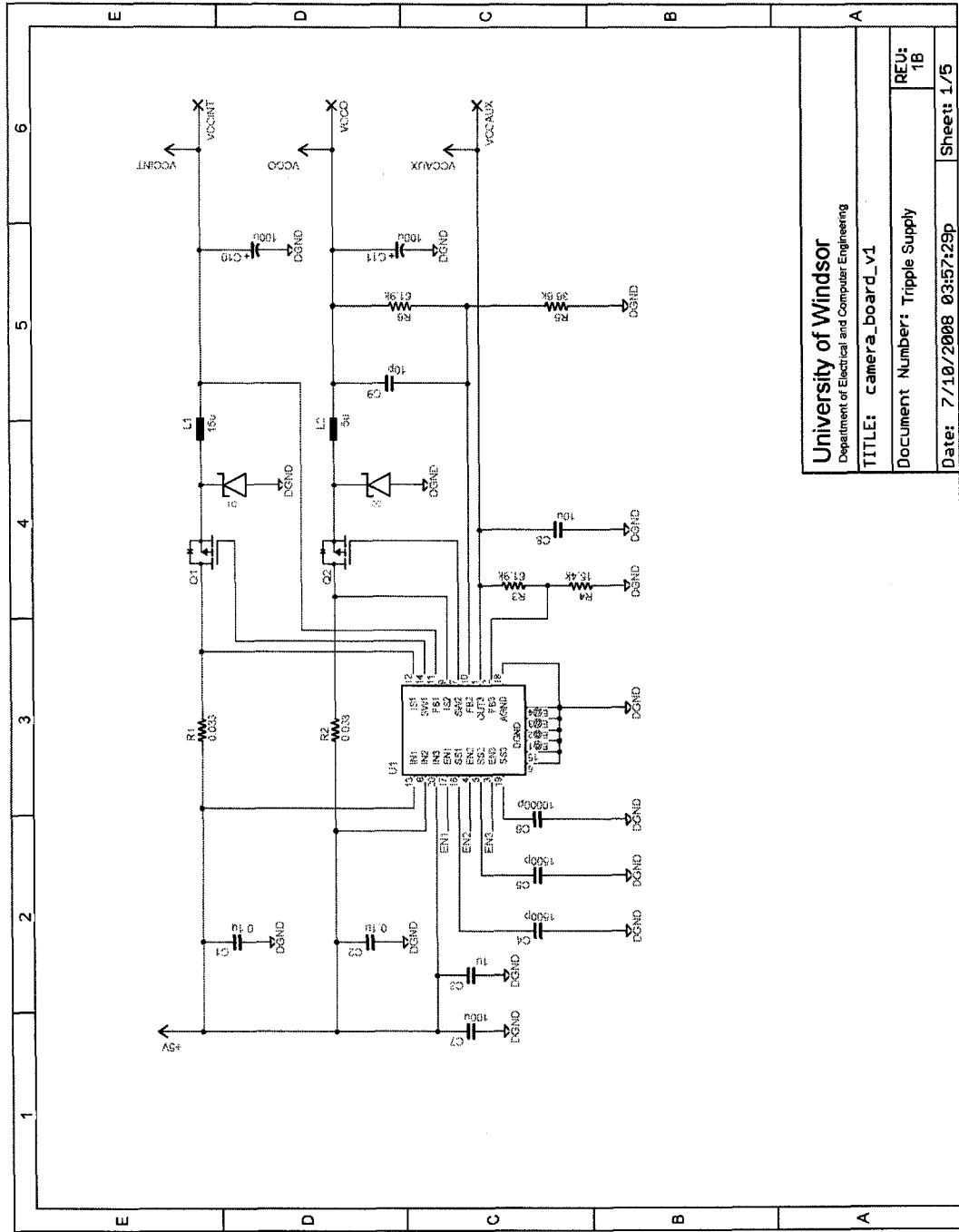
Continued on next page

B. CAMERA BOARD SCHEMATICS

Table B.1 – continued from previous page

R10	1/10W Resistor, 1%	10M	Panasonic	ERJ-3GEYJ106V	1	0603
R11	1/16W Resistor, 1%	100k	Panasonic	ERJ-2RKF1003X	1	0402
R12, R13	1/16W Resistor, 1%	10k	Panasonic	ERJ-2RKF1002X	2	0402
R14, R15	1/16W Resistor, 1%	1k	Panasonic	ERJ-2RKF1001X	2	0402
R16, R17	1/16W Resistor, 5%	680	Panasonic	ERJ-2GEJ681X	2	0402
R18	1/10W Resistor, 5%	0	Panasonic	ERJ-3GEY0R00V	1	0603
R21	1/16W Resistor, 5%	470	Panasonic	ERJ-2GEJ471X	1	0402
SW1	OFF-MOM switch, SMD	-	Omron	B3U-1000P	1	3x2.5mm
U1	Triple-Supply	3A	TI	TPS75003RHLLT	1	QFN-20
U2	Spartan 3E 500	-	Xilinx	XC3S500E-4FTG256C	1	FT256
U3	FX2 USB MCU	-	Cypress	CY7C68013A-100AXC	1	100-TQFP
U4	Micron 1/3" color CMOS	-	Micron	MT9T001P12STC	1	48-PLCC
U5	3.3V LDO Regulator	500mA	Catalyst Semicon.	CAT6219-330TD-GT3	1	SOT23-5
U6, U7	3.3V LDO Regulator	250mA	National Semi.	LP2992AIM5-3.3/NOPB	2	SOT23-5
U8	I2C Bus Extender	-	NXP	P82B715TD-T	1	8-SOIC
U9	I2C 128k EEPROM	-	Microchip	24LC128-1/ST	1	8-TSSOP
Y1	Crystal, series cap, SMD	24MHz	ECS	ECS-240-S-23B-TR	1	6x3.5mm

B. CAMERA BOARD SCHEMATICS



University of Windsor Department of Electrical and Computer Engineering	
TITLE: camera_board_v1	REV: 1B
Document Number: Tripple Supply	Sheet 1/5
Date: 7/10/2008 03:57:29p	

Figure B.1: TPS Triple Supply Schematic

B. CAMERA BOARD SCHEMATICS

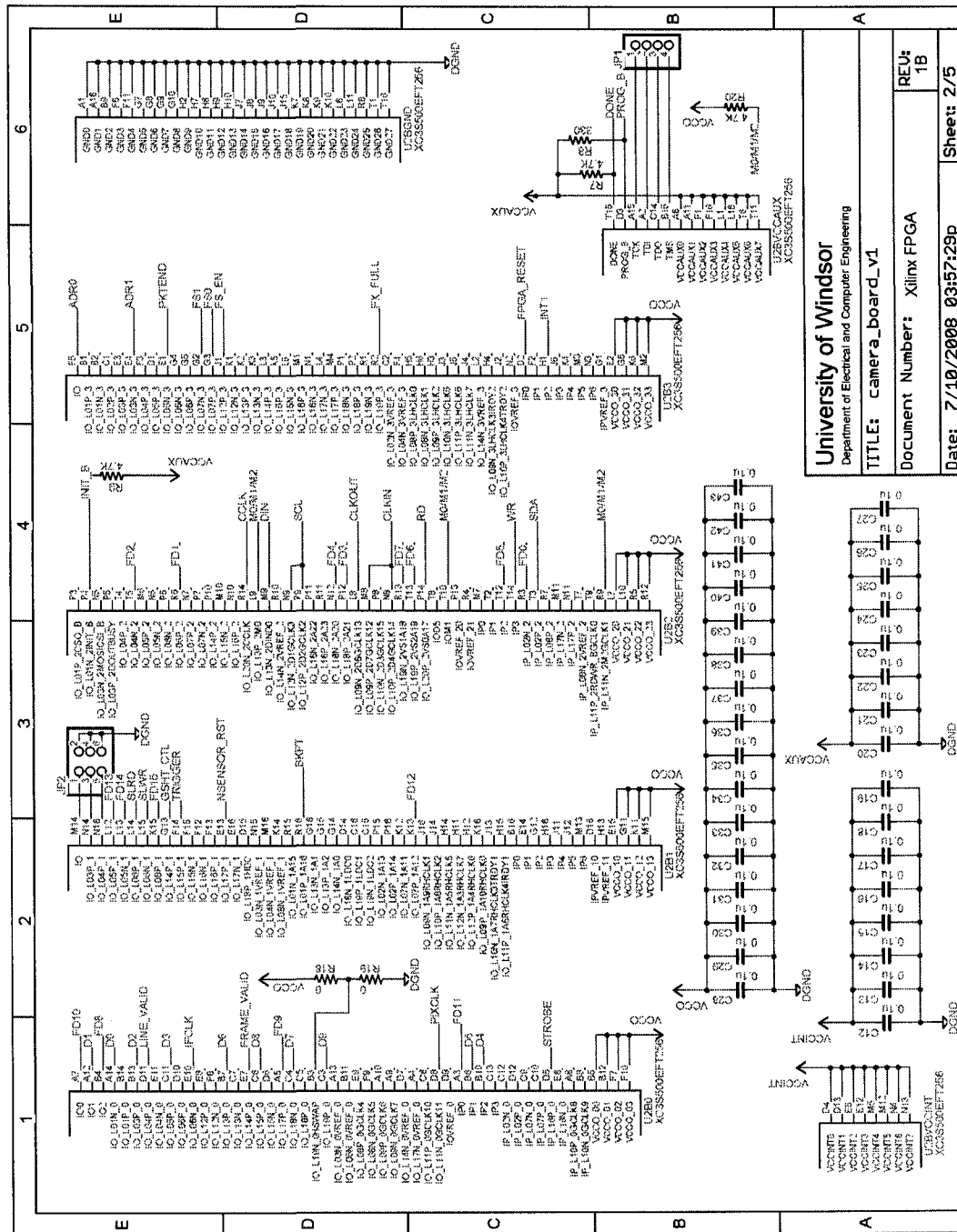
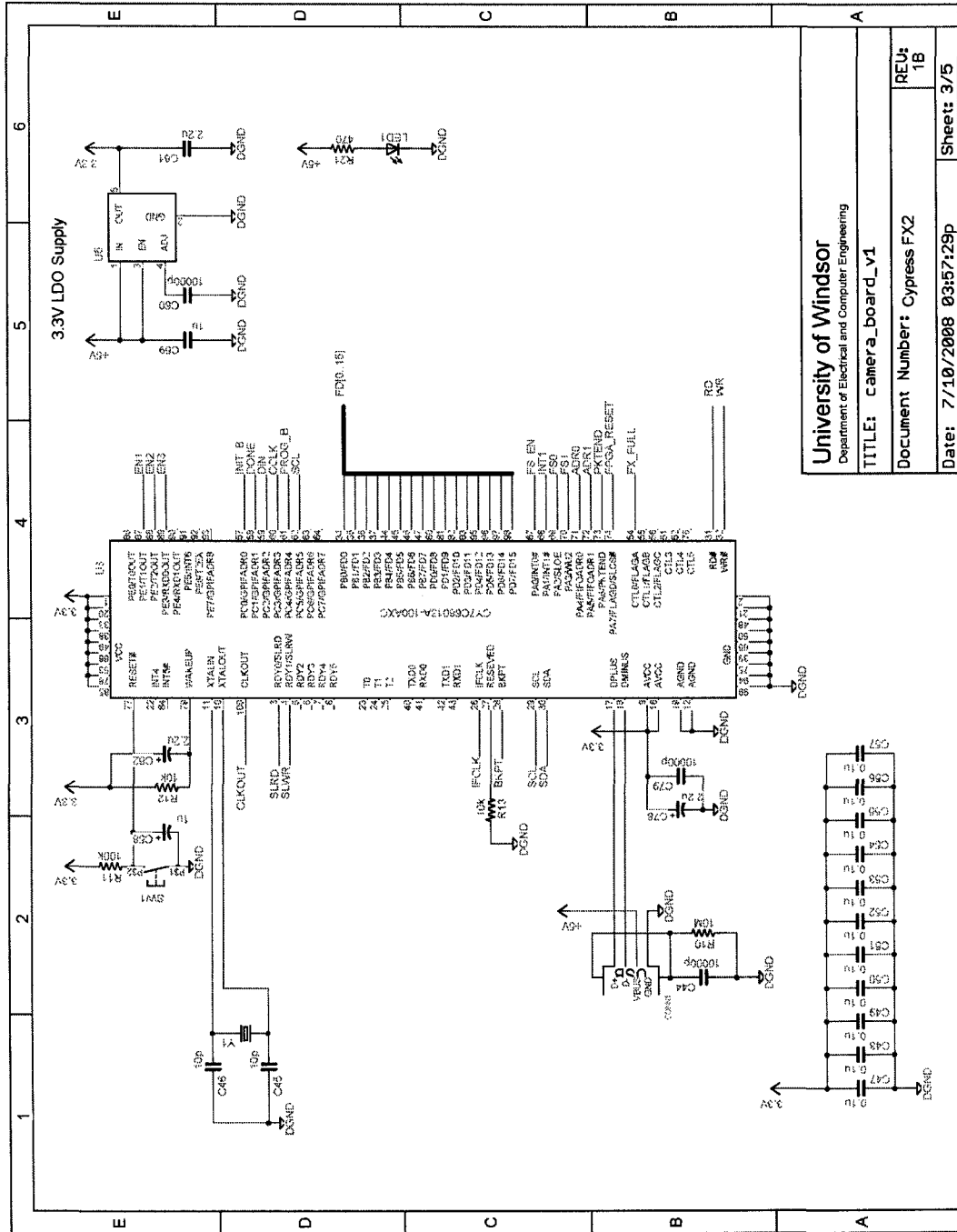
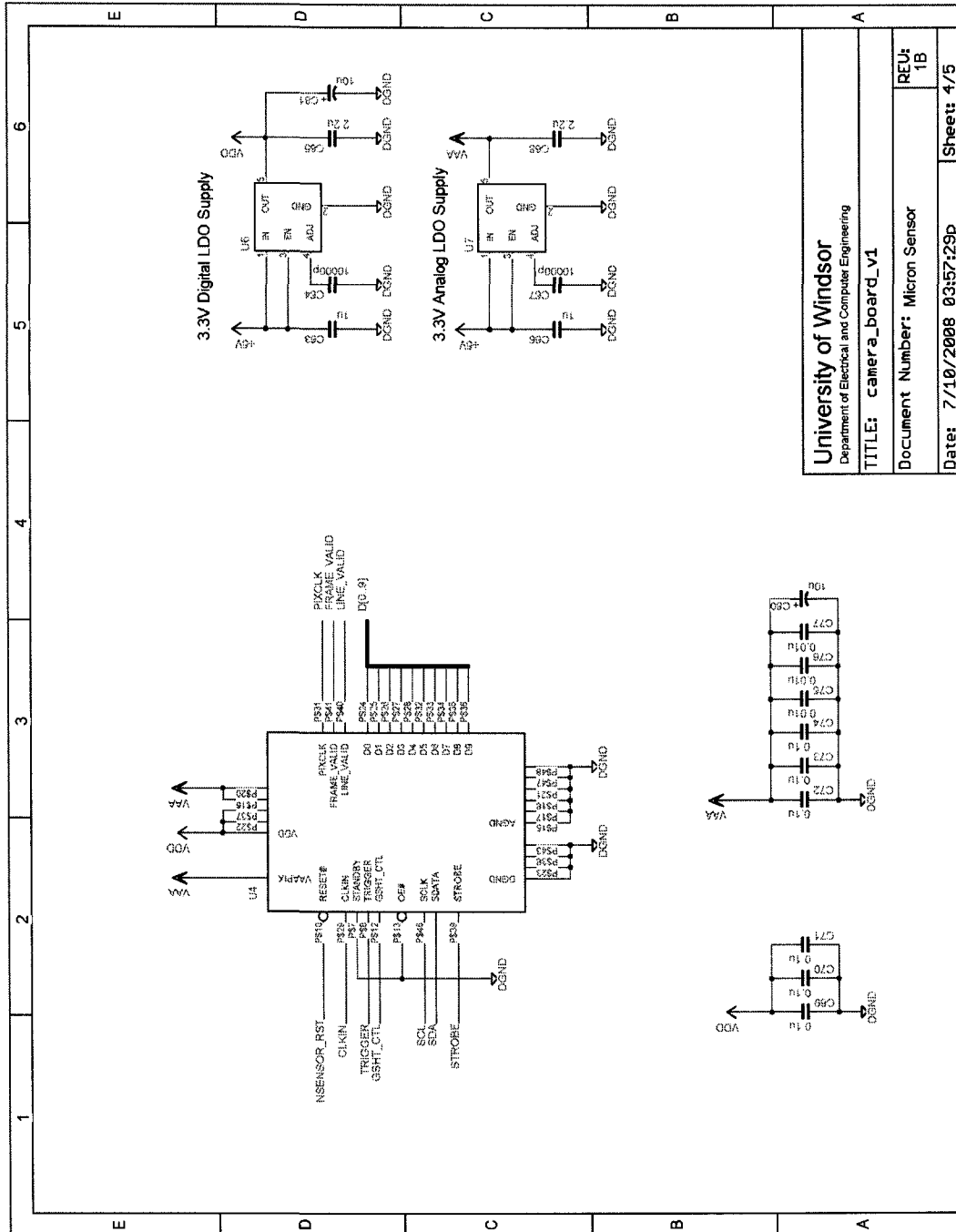


Figure B.2: Spartan-3E FPGA



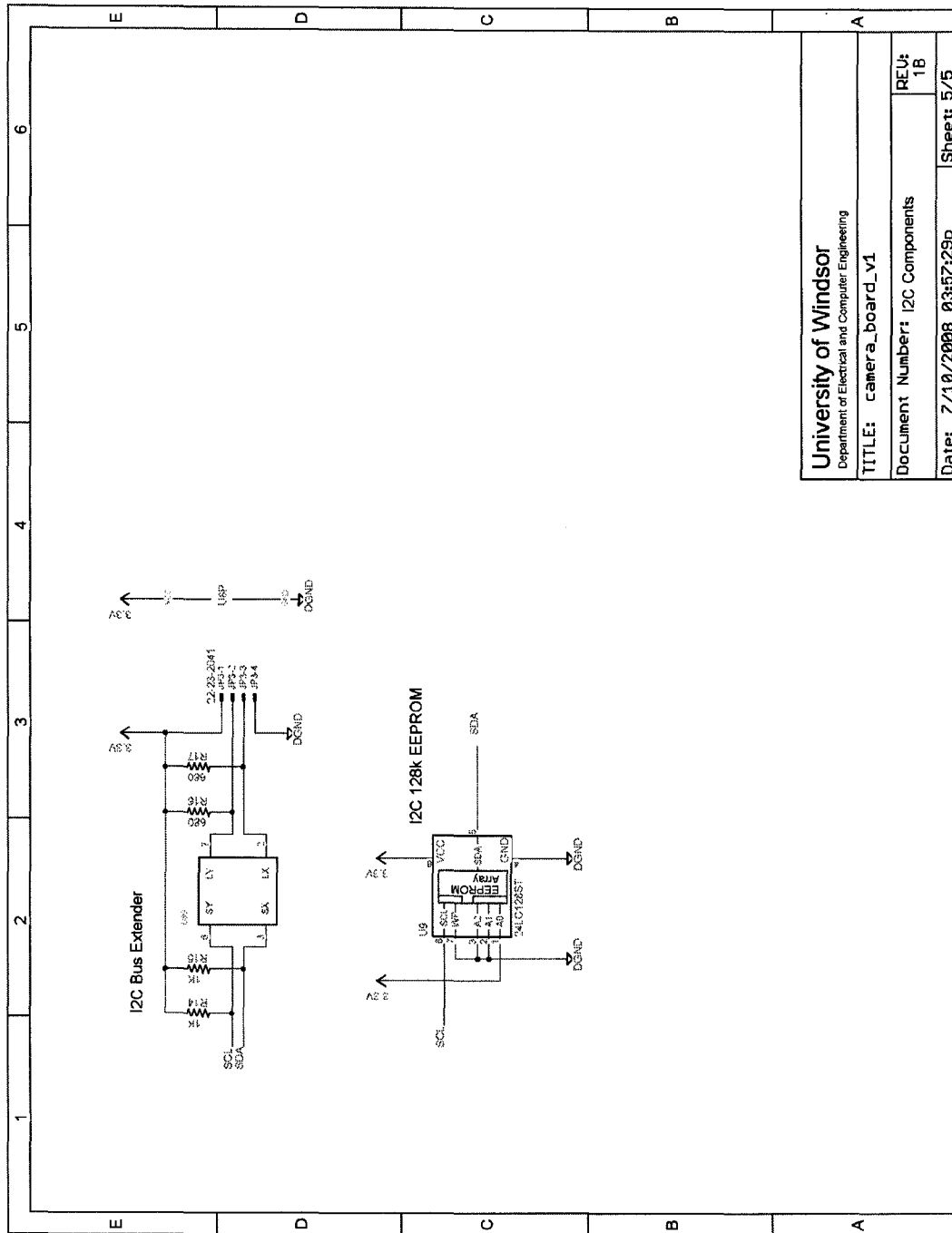
University of Windsor
 Department of Electrical and Computer Engineering
 TITLE: camera_board_v1
 Document Number: Cypress FX2
 Date: 7/10/2008 03:57:29p
 Sheet: 3/5
 REU: 1B

Figure B.3: Cypress FX2 USB Microcontroller



University of Windsor Department of Electrical and Computer Engineering	
TITLE: camera_board_v1	
Document Number: Micron Sensor	REV: 1B
Date: 7/10/2008 03:57:29p	Sheet: 4/5

Figure B.4: Micron Sensor Schematic



University of Windsor Department of Electrical and Computer Engineering	
TITLE: camera_board_v1	
Document Number: I2C Components	REV: 1B
Date: 7/10/2008 03:57:29p	Sheet: 5/5

Figure B.5: I2C Component Schematic

Appendix C

VHDL Code

Listing C.1: frame_grabber_v1.vhd

```
--- Company:
--- Engineer:
---
--- Create Date:    13:23:42 04/03/2007
--- Design Name:
--- Module Name:    frame_grabber_v1 - Behavioral
--- Project Name:
--- Target Devices:
--- Tool versions:
--- Description:
---
--- Dependencies:
---
--- Revision: 20071114
--- Revision 0.01 - File Created
--- Additional Comments:
---
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

----- Uncomment the following library declaration if instantiating
----- any Xilinx primitives in this code.
library UNISIM;
use UNISIM.VComponents.all;

library CCD_FRAME_GRABBER_LIB;
use CCD_FRAME_GRABBER_LIB.I2C_WRITE_SLAVE.ALL;

entity frame_grabber_v1 is
  Port (
    PIXCLK_IN : in  STD_LOGIC;
    IFCLK_IN  : in  STD_LOGIC;          -- FX2 Clock
    RESET_IN  : in  STD_LOGIC;
    LINE_VALID_IN : in  STD_LOGIC;
    FRAME_VALID_IN : in  STD_LOGIC;
    CCD_DATA_IN : in  STD_LOGIC_VECTOR (9 downto 0);
    GSHT_CTL_OUT : out STD_LOGIC;
```

```

        USB_FULLIN : in STD_LOGIC;
        USB_BUS_OUT : out STD_LOGIC_VECTOR (15 downto 0);
        USB_SLWR_OUT : out STD_LOGIC;
        USB_SLRD_OUT : out STD_LOGIC;
        USB_ADR0_OUT : out STD_LOGIC;
        USB_ADR1_OUT : out STD_LOGIC;
        USB_PCKTEND_OUT : out STD_LOGIC;

        NSENSOR_RST : out STD_LOGIC;
        --FRAME_ERROR : out STD_LOGIC;

        --FS_EN : out STD_LOGIC;
        --FS0 : out STD_LOGIC;
        --FS1 : out STD_LOGIC;

        MCLK : in STD_LOGIC;
        SCL : in STD_LOGIC;
        --SCL_FB : in STD_LOGIC;
        SDA : inout STD_LOGIC;

        P1 : in STD_LOGIC;
        N1 : out STD_LOGIC;

        CLOCK_IN : out STD_LOGIC;
        --CLOCK_IN_FB : in STD_LOGIC;
    );
end frame_grabber_v1;

-- Behavioral Declaration of frame_grabber_v1
architecture Behavioral of frame_grabber_v1 is

-- Signal Declaration

-- Constant Declaration
constant RESET_ACTIVE : std_logic := '0';

-- Clock signals
signal PIXCLK : std_logic;
signal PIXCLK_IBUFG : std_logic;
signal PIXCLK_LOCKED : std_logic;
signal PIXCLK_DV : std_logic;

signal USBCLK : std_logic;
signal USBCLK_IBUFG : std_logic;
signal USBCLK_LOCKED : std_logic;
signal USBCLK_cnt : std_logic_vector (3 downto 0);
signal USBCLK_DV : std_logic;

-- System Reset Signals
signal reset : std_logic;
signal sys_reset : std_logic;
signal fifo_reset : std_logic;
signal frame_reset : std_logic;
signal db_ctr : std_logic_vector (18 downto 0);

signal ig_reset : std_logic;
signal ip_reset : std_logic;

signal trigger_cnt : std_logic_vector (15 downto 0);
signal trigger_wait : std_logic_vector (15 downto 0);
signal trigger_b : std_logic;

-- Image Grabber Signals
signal byte_cnt : std_logic;
signal byte_0 , byte_1 : std_logic_vector (7 downto 0);
signal ram_data_in : std_logic_vector (15 downto 0);
signal ram_write : std_logic;
signal int_ram_write : std_logic;
signal ram_wen : std_logic;
signal ram_addr_en : std_logic;

```

```

signal ram_clr : std_logic;
signal ram_ren : std_logic;
signal ram_empty : std_logic;
signal ram_full : std_logic;
signal ram_reset : std_logic;
signal state : std_logic;
signal int_pxl_row : std_logic_vector (10 downto 0);
signal int_pxl_col : std_logic_vector (10 downto 0);
signal active_pxl_row : std_logic_vector (10 downto 0);
signal active_pxl_col : std_logic_vector (10 downto 0);
signal active_pxl_data : std_logic_vector (7 downto 0);
signal active_pxl_data_b : std_logic_vector (7 downto 0);
signal flush_frame : std_logic;
signal fs_ram_full : std_logic;
signal fs_cnt : std_logic_vector (3 downto 0);
signal pxl_drop_cnt : std_logic_vector (21 downto 0); -- enough to count
                    1 frame 2048x1536
signal pxl_write_cnt : std_logic_vector (21 downto 0);

signal int_fv : std_logic;
signal int_fv_b : std_logic;
signal int_lv : std_logic;
signal int_lv_b : std_logic;

-- Image Processor Signals
signal ip_en : std_logic;

-- Output Sync
signal usb_slwr : std_logic;
signal usb_pcktend : std_logic;
signal usb_bus : std_logic_vector (15 downto 0);

-- I2C Signals
signal i2c_enable : std_logic;
signal i2c_bus_busy : std_logic;
signal i2c_ack : std_logic;
signal i2c_data : std_logic_vector (7 downto 0);
signal i2c_debug : std_logic;

signal trigger : std_logic;

signal sys_regs : I2cDataReg;
signal ip_debug : std_logic;

signal fs_flag : std_logic;

-- Component Declaration
COMPONENT DCMLDLL
PORT(
    CLKIN_IN : IN std_logic;
    RST_IN : IN std_logic;
    CLKDV_OUT : OUT std_logic;
    CLKIN_IBUFG_OUT : OUT std_logic;
    CLK0_OUT : OUT std_logic;
    LOCKED_OUT : OUT std_logic
);
END COMPONENT;

COMPONENT ImageProcessor
PORT(
    CLK : IN std_logic;
    RESET : IN std_logic;
    EN : IN std_logic;
    DIN : IN std_logic_vector(7 downto 0);
    ROW : IN std_logic_vector (10 downto 0);
    COL : IN std_logic_vector (10 downto 0);
    ROW_SIZE : IN std_logic_vector (10 downto 0);
    COL_SIZE : IN std_logic_vector (10 downto 0);
    DATA_MODE : IN std_logic_vector (2 downto 0);
    COLOR_BIT : IN std_logic;
    FRAME : IN std_logic;
    DOUT : OUT std_logic_vector(15 downto 0);
    RAMWEN : OUT std_logic;
    IP_DEBUG : OUT std_logic
);
END COMPONENT;

```

```

COMPONENT ASYNC_FIFO_16 IS
port (
    din: IN std_logic_VECTOR(15 downto 0);
    rd_clk: IN std_logic;
    rd_en: IN std_logic;
    rst: IN std_logic;
    wr_clk: IN std_logic;
    wr_en: IN std_logic;
    dout: OUT std_logic_VECTOR(15 downto 0);
    empty: OUT std_logic;
    full: OUT std_logic);
END COMPONENT;

COMPONENT I2c_SlaveCtrl
PORT(
    Sys_clk : IN std_logic;
    Sys_Reset : IN std_logic;
    I2cEnable : IN std_logic;
    Scl : IN std_logic;
    Sda : INOUT std_logic;
    I2cBusBusy : OUT std_logic;
    I2cDataRegs : OUT I2cDataReg;
    I2cDebug : OUT std_logic
);
END COMPONENT;

----- BEGIN
begin

----- Component Instantiaization
-----
    Inst1.DCM_DLL: DCM_DLL PORT MAP(

    Inst2.DCM_DLL: DCM_DLL PORT MAP(
        CLKIN.IN => IFCLK.IN,
        RST.IN => reset,
        CLKDV.OUT => USBCLK.DV,
        CLKIN.IBUFG.OUT => USBCLK.IBUFG,
        CLK0.OUT => USBCLK,
        LOCKED.OUT => USBCLK.LOCKED
    );

    Inst1.DCM_DLL: DCM_DLL PORT MAP(
        CLKIN.IN => PIXCLK.IN,
        RST.IN => reset,
        CLKDV.OUT => PIXCLK.DV,
        CLKIN.IBUFG.OUT => PIXCLK.IBUFG,
        CLK0.OUT => PIXCLK,
        LOCKED.OUT => PIXCLK.LOCKED
    );

    Inst_ImageProcessor: ImageProcessor PORT MAP(
        CLK => pixclk,
        RESET => ip_reset,
        EN => ip_en,
        DIN => active.pxl.data_b,
        ROW => int.pxl.row,
        COL => int.pxl.col,
        ROW.SIZE => sys_regs (2) (10 downto 0),
        COL.SIZE => sys_regs (3) (10 downto 0),
        DATA.MODE => sys_regs (4) (3 downto 1),
        COLOR_BIT => sys_regs (4) (0),
        FRAME => FRAME.VALID.IN,
        DOUT => ram.data.in,
        RAM.WEN => ram.write,
        IP.DEBUG => ip.debug
    );

    Inst_ASYNC_FIFO: ASYNC_FIFO_16 PORT MAP (
        din => ram.data.in,
        rd_clk => usbclk,
        rd_en => ram.ren,
        rst => ram.reset,
        wr_clk => pixclk,
        wr_en => ram.wen,
        dout => usb.bus,
        empty => ram.empty,
        full => ram.full
    );

```



```

Inst_I2c_SlaveCtrl: I2c_SlaveCtrl PORT MAP(
    Scl => Scl,
    Sda => Sda,
    Sys_clk => usbclk,
    Sys_Reset => sys_reset,
    I2c_Enable => i2c_enable,
    I2c_Bus_Busy => i2c_bus_busy,
    I2c_Data_Regs => sys_regs,
    I2c_Debug => i2c_debug
);

-----
-- Signal Initialization
-----

-- Reset Signals
-----
reset <= (NOT reset_in) when (RESET_ACTIVE = '0') else reset_in;
ip_reset <= sys_reset; -- when (sys_reset = '1' or sys_regs(1)(1) = '1') else '0';
ram_reset <= sys_reset; -- or frame_reset;

-- Internal Signal Assignments
i2c_enable <= '1'; --sys_reset;
ram_wen <= ram_write AND NOT ram_full;

-- Output Assignments
-----
clock_in <= usbclk_dv;

GSHT_CTL_OUT <= trigger;
N1 <= trigger;

-- FX2 Configuration Pins
-----
USB_PCKTEND_OUT <= '0'; -- Active High
USB_SLRD_OUT <= '0'; -- Active Low
USB_ADR0_OUT <= '0';
USB_ADR1_OUT <= '0';

NSENSOR_RST <= '1'; -- Micron Sensor Enable

-----
-- Process: Debounce_Ctr
-- Desc: Handle Debounce on the RESET_IN pin.
-----
Debounce_Ctr : process (USBCLK, reset)
begin
    if (reset = '1') then
        db_ctr <= (others => '0');
    elsif (USBCLK'EVENT and USBCLK = '0') then
        if (USBCLK_LOCKED = '1' and db_ctr(18) = '0') then --should be 18
            -- Wait until DLL has locked onto the clock.
            db_ctr <= db_ctr + 1;
        end if;
    end if;
end process Debounce_Ctr;

-----
-- Process: System_Reset
-- Desc: Handle sys_reset based on state of RESET_IN pin and debounce counter.
-----
System_Reset : process (USBCLK, db_ctr(18))
begin
    if (USBCLK'Event and USBCLK = '1') then
        if (db_ctr(18) = '1') then
            sys_reset <= '0'; --sys_regs(1)(0);
        else
            sys_reset <= '1';
        end if;
    end if;
end process System_Reset;

-----
-- Process: TRIGGER_CTLR
-- Desc: Synchronize the data from the micron sensor.
-----
TRIGGER_CTLR : process (usbclk, sys_reset)
begin
    if (sys_reset = '1') then
        N1 <= '0';
    end if;
end process TRIGGER_CTLR;

```

```

        GSHT_CTLLOUT <= '0';
        trigger_wait <= (others => '0');
        trigger_cnt <= (others => '0');
    elsif (usbclk'Event and usbclk = '1') then
        trigger_b <= trigger;
        if trigger = '1' and trigger_b = '0' then
            trigger_wait <= data_regs(5);
            trigger_cnt <= (others => '0');
        elsif trigger = '1'
            GSHT_CTLLOUT <= trigger;
            trigger_cnt <= trigger_cnt + 1;
        end if;

        if trigger_wait /= 0 then
            trigger_wait <= trigger_wait - 1;
        end if;

        if trigger_wait = 0 and trigger_cnt /= 0 then
            N1 <= '1';
            trigger_cnt <= trigger_cnt - 1;
        end if;
    end if;
end process TRIGGERCLTR;

```

```

-- Process: SYNCIN
-- Desc: Synchronize the data from the micron sensor.

```

```

ip_en <= int_fv and int_lv;

SYNCIN : process (pixclk, sys_reset)
begin
    if (sys_reset = '1') then
        int_fv <= '0';
        int_fv_b <= '0';
        int_lv <= '0';
        active_pxl_data <= (others => '0');
        active_pxl_data_b <= (others => '0');

    elsif (pixclk'Event and pixclk = '1') then
        int_fv <= FRAME_VALID_IN;
        int_fv_b <= int_fv;
        int_lv <= LINE_VALID_IN;
        int_lv_b <= int_lv;
        active_pxl_data <= CCD_DATA_IN(9 downto 2);
        active_pxl_data_b <= active_pxl_data;

        if (int_fv = '1' and int_fv_b = '0') then
            fifo_reset <= '1';
        else
            fifo_reset <= '0';
        end if;
    end if;
end process SYNCIN;

FS_ERROR : process (pixclk, sys_reset)
begin
    if (sys_reset = '1') then
        fs_ram_full <= '1';
        fs_cnt <= (others => '1');
    elsif (pixclk'Event and pixclk = '1') then
        if (frame_valid_in = '1' and int_fv = '0') then
            fs_cnt <= (others => '1');
            --fs_ram_full <= '1';
        elsif (fs_cnt /= 0) then
            fs_cnt <= fs_cnt - 1;
        end if;

        if (ram_full = '1' and fs_cnt = 0) then
            fs_ram_full <= '0';
        else
            fs_ram_full <= '1';
        end if;
    end if;
end process FS_ERROR;

```

```

-- Process: FRAME_CTLR
-- Desc: Track frame status and report errors

```

```

FRAMECTRLR : process (pixclk , sys_reset)
begin
  if (sys_reset = '1') then
    int_pxl_row <= (others => '0');
    int_pxl_col <= (others => '0');
    frame_reset <= '1';
  elsif (pixclk'Event and pixclk = '1') then
    if (int_fv = '1') then
      if (int_fv_b = '0') then
        frame_reset <= '1';
      else
        frame_reset <= '0';
      end if;

      if (int_lv_b = '1') then
        if (int_lv = '1') then
          int_pxl_col <= int_pxl_col + 1;
        else
          int_pxl_col <= (others => '0');
          int_pxl_row <= int_pxl_row + 1;
        end if;
      end if;
    else
      int_pxl_row <= (others => '0');
      int_pxl_col <= (others => '0');
    end if;
  end if;
end process FRAMECTRLR;

```

```

-- USB DATA PROCESS

```

```

-- Process: SYNC_OUT

```

```

-- Desc: Synchronize output signals to the FX2

```

```

SYNC_OUT : process (usbclk , sys_reset)
begin
  if (sys_reset = '1') then
    usb_slwr_out <= '0';
    usb_bus_out <= (others=>'1');
  elsif (usbclk'EVENT and usbclk='1') then
    usb_slwr_out <= usb_slwr;
    if (pxl_write_cnt < "0000000000000000000011" ) then -- or (pxl_write_cnt >
      "000101111111111111101" and pxl_write_cnt < "0001100000000000000000")
      ) then
      usb_bus_out <= X"AAAA"; --(others => '1');
    else
      usb_bus_out <= usb_bus; --pxl_write_cnt (7 downto 0) &
      pxl_write_cnt (15 downto 8); --usb_bus;
    end if;
  end if;
end process SYNC_OUT;

```

```

-- Process: USB_Ctlr

```

```

-- Desc: State machine for sending data to FX2

```

```

USB_Ctlr : process (usbclk , sys_reset)
begin
  if (sys_reset = '1') then
    usb_slwr <= '0';
    state <= '0';
    ram_ren <= '0';
    pxl_write_cnt <= (others => '0'); --"0011000000000000000000";
  elsif (fifo_reset = '1') then
    pxl_write_cnt <= To_StdLogicVector(to_BitVector(sys_regs(6)(5 downto 0) &
      sys_regs(7)) srl 1);--"0001100000000000000000";
    --"0011000000000000000000";
  elsif (usbclk'EVENT and usbclk='1') then
    if (state = '0') then
      if (usb_full_in = '0' and ram_empty = '1' and frame_valid_in = '0'
        and pxl_write_cnt > 0) then
        ram_ren <= '0';
        usb_slwr <= '0';
        state <= '1';
        pxl_write_cnt <= pxl_write_cnt - 1;
      end if;
    end if;
  end if;
end process USB_Ctlr;

```

```

        elsif (usb_full_in = '0' and ram_empty = '0') then
            ram_ren <= '1';
            usb_slwr <= '0';
            state <= '1';
            pxl_write_cnt <= pxl_write_cnt - 1;
        else
            ram_ren <= '0';
            usb_slwr <= '0';
            state <= '0';
        end if;
    else
        usb_slwr <= '1';
        ram_ren <= '0';
        state <= '0';
    end if;
end if;
end process USB_Ctrlr;
end Behavioral;

```

Listing C.2: I2c_SlaveCtrl.vhd

```

-- Company:
-- Engineer:
--
-- Create Date:    11:15:26 07/25/2007
-- Design Name:
-- Module Name:    I2c_SlaveCtrl - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library CCD_FRAME_GRABBER_LIB;
use CCD_FRAME_GRABBER_LIB.I2C_WRITE_SLAVE.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity I2c_SlaveCtrl is
    Port ( Scl : in STD_LOGIC;
          Sda : inout STD_LOGIC;
          Sys_clk : in STD_LOGIC;
          Sys_Reset : in STD_LOGIC;
          I2c_Enable : in STD_LOGIC;
          I2c_Bus_Busy : out STD_LOGIC;
          I2c_Data_Regs : out I2c_Data_Reg;
          I2c_Debug : out STD_LOGIC);
end I2c_SlaveCtrl;

architecture Behavioral of I2c_SlaveCtrl is

    -- Constant Declaration
    constant SlaveAddress : std_logic_vector (6 downto 0) := "1010101"; -- 0x5D for MI
    Sensor

```

```

-- Signal Declaration

type I2cStateMach is (IDLE, ADDR, SND_ACK, RCV_DATA);
signal I2cState          : I2cStateMach;

signal reset : std_logic;
signal IntSda : std_logic;
signal IntSdaB : std_logic;
signal IntSdaCnt : std_logic_vector (4 downto 0); -- Sampling rate of Sda line for start
and stop.
signal IntStartDetected : std_logic; -- Start signal
signal IntScl : std_logic;
signal IntSclB : std_logic;
signal IntStartDetectedB : std_logic; -- Delayed start signal
signal IntStopDetected : std_logic; -- Stop signal
signal IntI2cAck : std_logic; -- Trigger an Acknowledge
signal IntBusBusy : std_logic; -- Set on start signal and cleared on stop
signal.
signal IntI2cBitCnt : std_logic_vector (3 downto 0); -- Counts received bits.
signal IntBitCntEn : std_logic; -- Counts incoming bits
signal IntBitCntClr : std_logic; -- Clears bit counter
signal IntI2cData : std_logic_vector (7 downto 0); -- I2C data
signal IntShiftRegEn : std_logic; -- Enable the shift register
signal IntShiftRegClr : std_logic; -- Clear the shift register
signal IntByteCnt : std_logic_vector (3 downto 0);

--type I2cDataReg is array (4 downto 0) of std_logic_vector (15 downto 0);
signal IntDataRegs : I2cDataReg;
signal IntRegAddr : std_logic_vector (7 downto 0);

signal IntI2cDebug : std_logic;

-- Component Declaration
component I2c_ShiftReg
  PORT (
    CLK: IN std_logic;
    SDIN: IN std_logic;
    Q: OUT std_logic_VECTOR(7 downto 0);
    CE: IN std_logic;
    ACLR: IN std_logic);
END COMPONENT;

component I2c_BitCounter
  PORT (
    clk: IN std_logic;
    ce: IN std_logic;
    aclr: IN std_logic;
    q: OUT std_logic_VECTOR(3 downto 0));
END COMPONENT;

begin

-- Process Declaration

reset <= '1' when (sys_reset = '1' or I2cEnable = '0') else '0';
I2cDebug <= '1' when (IntDataRegs(3) = x"07FF") else '0';

-- Input and Output
-- Sda is a bidirectional signal.

Sda <= '0' when (I2cState = SND_ACK) else 'Z'; --OVER HERE!!!!!!

WriteI2cDataRegs : process (Scl, Reset)
begin
  if (Reset = '1') then
    IntDataRegs (0) <= "0000000000000000"; -- DEFAULT_REG
    IntDataRegs (1) <= "0000000000000110"; -- FPGA RESETS:
    IntDataRegs (2) <= "0000001011111111"; -- ROW_SIZE: default 767
    IntDataRegs (3) <= "0000001111111111"; -- COL_SIZE: default 1023
    IntDataRegs (4) <= "0000000000000010"; -- DATA_MODE: default 2 (RAW data) (8-bit)
    IntDataRegs (5) <= "0000000000000000"; -- TRIGGER_DELAY: default 0
    IntDataRegs (6) <= "0000000000000110"; -- FRAME_SIZE_UPPER: default 1024x768 =
    786432
    IntDataRegs (7) <= "0000000000000000"; -- FRAME_SIZE_LOWER: ^
    IntDataRegs (8) <= "0100000010101010"; -- FRAME_MARKER : xxDDDDDDxxxxxxx
    IntDataRegs (9) <= "0000001111111101"; -- IP_ROW_BUFFER_SIZE : default 1023-2 =
    1021
  end if;
end process;

```

```

IntDataRegs (10) <= "0000010000000100"; -- IP_OUT_COUNT : default 1023 + 5 = 1028
IntDataRegs (11) <= "0000000000000000";
IntDataRegs (12) <= "0000000000000000";
IntDataRegs (13) <= "0000000000000000";
IntDataRegs (14) <= "0000000000000000";
IntDataRegs (15) <= "0000000000000000";
IntDataRegs (16) <= "0000000000000000";
IntDataRegs (17) <= "0000000000000000";
IntDataRegs (18) <= "0000000000000000";
IntDataRegs (19) <= "0000000000000000";
IntDataRegs (20) <= "0000000000000000";
IntDataRegs (21) <= "0000000000000000";
IntDataRegs (22) <= "0000000000000000";
IntDataRegs (23) <= "0000000000000000";
IntDataRegs (24) <= "0000000000000000";
IntDataRegs (25) <= "0000000000000000";
IntDataRegs (26) <= "0000000000000000";
IntDataRegs (27) <= "0000000000000000";
IntDataRegs (28) <= "0000000000000000";
IntDataRegs (29) <= "0000000000000000";
IntDataRegs (30) <= "0000000000000000";
IntDataRegs (31) <= "0000000000000000";

elsif (IntScl'Event and IntScl = '0') then
  if (IntI2cBitCnt = "0111") then
    if (IntByteCnt = "001") then
      IntRegAddr <= IntI2cData (6 downto 0) & IntSda;
    elsif (IntByteCnt = "010") then
      IntDataRegs (conv_integer(IntRegAddr)) (15 downto 8) <= IntI2cData
        (6 downto 0) & IntSda;
    elsif (IntByteCnt = "011") then
      IntDataRegs (conv_integer(IntRegAddr)) (7 downto 0) <= IntI2cData (
        6 downto 0) & IntSda;
    end if;
  end if;
end if;
end process;

I2cDataRegs <= IntDataRegs;

SignalSync : process (Reset, sys_clk)
begin
  if (reset = '1') then
    IntSda <= '0';
    IntSdaB <= '0';
    IntScl <= '0';
    IntSclB <= '0';
    IntSdaCnt <= (others => '0');
  elsif (sys_clk'Event and sys_clk = '1') then
    IntSdaCnt <= IntSdaCnt + 1;
    if (IntSdaCnt = "00000") then
      IntSda <= Sda;
      IntSdaB <= IntSda;
      IntScl <= Scl;
      IntSclB <= IntScl;
    end if;
  end if;
end process SignalSync;

DetectStartStop : process (Reset, sys_clk)
begin
  if (reset = '1') then
    IntStartDetected <= '0';
    IntStopDetected <= '0';
  elsif (sys_clk'Event and sys_clk = '1') then
    if (IntSda = '0' and IntSdaB = '1') then
      if (IntScl = '1') then
        IntStartDetected <= '1';
      else
        IntStartDetected <= '0';
      end if;
    elsif (IntSda = '1' and IntSdaB = '0') then
      if (IntScl = '1') then
        IntStopDetected <= '1';
      else
        IntStopDetected <= '0';
      end if;
    end if;
  end if;

  if (I2cState = ADDR) then

```

```

        IntStartDetected <= '0';
    end if;

    if (IntStartDetected = '1') then
        IntStopDetected <= '0';
    end if;
end if;
end process;

-- Indication that the device is busy. Set when a Start is detected and released when a
-- Stop is found. This bit is take outside as status bit for connecting logic.
-- A one indicates, Bus Busy.

BusBusyReg : process (Sys_clk, Reset)
begin
    if (Reset = '1') then
        IntBusBusy <= '0';
        --IntBusBusyB <= '0';
    elsif (Sys_clk'event and Sys_clk = '1') then
        if (IntStartDetected = '1') then
            IntBusBusy <= '1';
        end if;
        if (IntStopDetected = '1') then
            IntBusBusy <= '0';
        end if;
        --IntBusBusyB <= IntBusBusy;
    end if;
end process;

I2cBusBusy <= IntBusBusy;           -- Ctrl output as indication the the controller has the
    bus.

-- Main I2C state machine.
-- The following process contains the main I2C slave state machine.
-- This state machine is clocked on the falling edge of Scl.

MainStateMach : process (IntScl, Reset, IntStopDetected)
begin
    if (Reset = '1' or IntStopDetected = '1') then
        I2cState <= IDLE;
        IntByteCnt <= (others=>'0');
        --IntMainStatMachStop <= '0';
    elsif (IntScl'event and IntScl = '0') then
        case I2cState is
            ----- IDLE I2cState -----
            when IDLE =>
                if (IntStartDetected = '1') then
                    I2cState <= ADDR;
                    IntByteCnt <= (others=>'0');
                end if;

            ----- ADDR I2cState -----
            when ADDR =>
                if (IntI2cBitCnt = "0111") then
                    if (IntI2cData (6 downto 0) & IntSda = SlaveAddress & '0')
                        then
                            I2cState <= SND_ACK;
                        else
                            I2cState <= IDLE;
                        end if;
                    end if;

            ----- SND_ACK I2cState -----
            when SND_ACK =>
                IntByteCnt <= IntByteCnt + 1;
                IntI2cAck <= '1';
                I2cState <= RCV_DATA;

            ----- RCV_DATA I2cState -----
            when RCV_DATA =>
                IntI2cAck <= '0';
                if (IntStartDetected = '1') then
                    I2cState <= ADDR;
                elsif (IntI2cBitCnt = "0111") then
                    I2cState <= SND_ACK;
                end if;
        end case;
end process;

```

```

        end if;
    end process;

-- I2C shift register
ShiftReg : I2c_ShiftReg PORT MAP (
    CLK => NOT IntScl,
    SDIN => IntSda,
    Q => IntI2cData,
    CE => IntShiftRegEn,
    ACLR => IntShiftRegClr
);

IntShiftRegEn <= '1' when (I2cState = ADDR) or (I2cState = RCV_DATA) else '0';
IntShiftRegClr <= '1' when (Reset = '1' or IntStopDetected = '1' or I2cState = SND_ACK) else '0';

-- I2C bit counter.
BitCounter : I2c_BitCounter PORT MAP (
    CLK => NOT IntScl,
    CE => IntBitCntEn,
    aclr => IntBitCntClr,
    q => IntI2cBitCnt
);

IntBitCntEn <= '1' when (I2cState = ADDR
                        or (I2cState = RCV_DATA)
                        else '0';

IntBitCntClr <= '1' when (I2cState = IDLE)
                        or (I2cState = SND_ACK)
                        or (IntStartDetected = '1')
                        else '0';

--
end Behavioral;

```

Listing C.3: ImageProcessor.vhd

```

-- Company:
-- Engineer:
--
-- Create Date:    11:10:59 04/17/2007
-- Design Name:
-- Module Name:    ImageProcessor - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ImageProcessor is
    Port ( CLK : in STD_LOGIC;
          RESET : in STD_LOGIC;
          EN : in STD_LOGIC;
          DIN : in STD_LOGIC_VECTOR (7 downto 0);
          ROW : in STD_LOGIC_VECTOR (10 downto 0);
          COL : in STD_LOGIC_VECTOR (10 downto 0);
          FRAME : in STD_LOGIC;

```



```

        ROW_SIZE : in STD_LOGIC_VECTOR (10 downto 0);
        COL_SIZE : in STD_LOGIC_VECTOR (10 downto 0);
        DATA_MODE : in STD_LOGIC_VECTOR (2 downto 0);
        COLOR_BIT : in STD_LOGIC;
        DOUT : out STD_LOGIC_VECTOR (15 downto 0);
        RAMWEN : out STD_LOGIC;
        IP_DEBUG : out STD_LOGIC);
end ImageProcessor;

architecture Behavioral of ImageProcessor is

    constant ROW_COUNT : integer := 5;
    constant COL_COUNT : integer := 5;
    -- Type Definition
    type reg_array is array (integer range <>, integer range <>) of std_logic_vector (7 downto 0);

    -- Signal Declarations
    -- Row Memory Signals
    -- signal row_in: reg_array (ROW_COUNT-1 downto 0, COL_COUNT-1 downto 0);
    signal row_in_00, row_in_01, row_in_02, row_in_03, row_in_04 : std_logic_vector (7 downto 0);
    signal row_in_10, row_in_11, row_in_12, row_in_13, row_in_14 : std_logic_vector (7 downto 0);
    signal row_in_20, row_in_21, row_in_22, row_in_23, row_in_24 : std_logic_vector (7 downto 0);
    signal row_in_30, row_in_31, row_in_32, row_in_33, row_in_34 : std_logic_vector (7 downto 0);
    signal row_in_40, row_in_41, row_in_42, row_in_43, row_in_44 : std_logic_vector (7 downto 0);

    signal row_addr : STD_LOGIC_VECTOR (10 downto 0);
    signal row_wen : STD_LOGIC;
    signal row_en : STD_LOGIC;
    signal pxl_cnt : STD_LOGIC_VECTOR (2 downto 0);
    signal row_addr_max : STD_LOGIC_VECTOR (10 downto 0);
    signal row_wen_in : std_logic;

    -- RGB construction signals
    shared variable R, G, B : integer;
    shared variable R1, G1, B1 : integer;
    shared variable R2, G2, B2 : integer;
    shared variable Ri, Gi, Bi : integer;
    shared variable Yi, Ui, Vi : integer;
    shared variable Yt, Ut, Vt : std_logic_vector (31 downto 0);
    signal tmp_Y : STD_LOGIC_VECTOR (9 downto 0);
    signal Y : STD_LOGIC_VECTOR (9 downto 0);
    signal U : STD_LOGIC_VECTOR (3 downto 0);
    signal V : STD_LOGIC_VECTOR (3 downto 0);

    -- Edge Enhanced Interpolation signal
    shared variable D1, D2, D3, D4 : integer;
    shared variable Dp1, Dp2, Dp3, Dp4 : integer;
    shared variable P1, P2, P3, P4 : integer;
    shared variable P5, P6, P7, P8 : integer;
    shared variable tP1, tP2, tP3, tP4 : integer;
    shared variable tP5, tP6, tP7, tP8 : integer;
    shared variable c1, c2, c3, c4 : integer;
    shared variable c5, c6, c7, c8 : integer;
    shared variable threshold1 : integer := 25;
    shared variable threshold2 : integer := 55;
    shared variable sum1, sum2 : integer;
    shared variable sum3, sum4 : integer;

    -- Output Control signals
    signal byte_cnt : STD_LOGIC;
    signal tmp_byte : STD_LOGIC_VECTOR (7 downto 0);
    signal out_en : STD_LOGIC;
    signal out_wait : STD_LOGIC;
    signal out_cnt : STD_LOGIC_VECTOR (14 downto 0);
    signal in_cnt : STD_LOGIC_VECTOR (14 downto 0);
    signal in_en : STD_LOGIC;
    signal int_en : STD_LOGIC;
    signal int_din : STD_LOGIC_VECTOR (7 downto 0);
    signal row_d1, row_d2, row_d3, row_d4 : STD_LOGIC;
    signal col_d1, col_d2, col_d3, col_d4 : STD_LOGIC;
    signal row_cnt, col_cnt : STD_LOGIC_VECTOR (10 downto 0);

    signal int_ip_debug : STD_LOGIC;

    -- signal start_row : STD_LOGIC_VECTOR (10 downto 0);

```

```

--signal start_col : STD_LOGIC_VECTOR (10 downto 0);

-- Component Definition
COMPONENT RowBuffer IS
  port (
    addr: IN std_logic_VECTOR(10 downto 0);
    clk: IN std_logic;
    din: IN std_logic_VECTOR(7 downto 0);
    dout: OUT std_logic_VECTOR(7 downto 0);
    en: IN std_logic;
    we: IN std_logic);
END COMPONENT;

begin

-- Row Buffers
-- Generate the Row Buffers based on the number of rows and cols required to process a pixel.

-- Row-Buffer-GEN:
-- for N in ROW_COUNT - 2 downto 0 generate
-- begin
--
--                               Inst_Row : RowBuffer Port Map (
--                               addr => row_addr,
--                               clk => CLK,
--                               din => row_in(N,4),
--                               dout => row_in(N+1,0),
--                               en => EN,
--                               we => row_wen
--                               );
-- end generate;

Inst_Row1 : RowBuffer Port Map (
  addr => row_addr,
  clk => CLK,
  din => row_in_04,
  dout => row_in_10,
  en => row_en,
  we => row_wen
);

Inst_Row2 : RowBuffer Port Map (
  addr => row_addr,
  clk => CLK,
  din => row_in_14,
  dout => row_in_20,
  en => row_en,
  we => row_wen
);

Inst_Row3 : RowBuffer Port Map (
  addr => row_addr,
  clk => CLK,
  din => row_in_24,
  dout => row_in_30,
  en => row_en,
  we => row_wen
);

Inst_Row4 : RowBuffer Port Map (
  addr => row_addr,
  clk => CLK,
  din => row_in_34,
  dout => row_in_40,
  en => row_en,
  we => row_wen
);

-- Row Shift
-- Handle the addressing of the Row Buffers

row_wen <= EN or out_en;
row_en <= EN or out_en;
row_addr_max <= conv_std_logic_vector(conv_integer(col_size) - 5, 11);

ROW_SHFT : process (CLK, RESET)
begin
  if (RESET = '1') then
    row_in_00 <= (others => '0');
    row_in_01 <= (others => '0');
  end if;
end process;

```

```

row_in_02 <= (others => '0');
row_in_03 <= (others => '0');
row_in_04 <= (others => '0');

--row_in_10 <= (others => '0');
row_in_11 <= (others => '0');
row_in_12 <= (others => '0');
row_in_13 <= (others => '0');
row_in_14 <= (others => '0');

--row_in_20 <= (others => '0');
row_in_21 <= (others => '0');
row_in_22 <= (others => '0');
row_in_23 <= (others => '0');
row_in_24 <= (others => '0');

--row_in_30 <= (others => '0');
row_in_31 <= (others => '0');
row_in_32 <= (others => '0');
row_in_33 <= (others => '0');
row_in_34 <= (others => '0');

--row_in_40 <= (others => '0');
row_in_41 <= (others => '0');
row_in_42 <= (others => '0');
row_in_43 <= (others => '0');
row_in_44 <= (others => '0');

row_addr <= (others => '0');

row_cnt <= (others => '0');
col_cnt <= (others => '0');

elsif (CLK'EVENT and CLK = '1') then
  if (EN = '1' or out_en = '1') then

    if (not((row <= "0000000011") or (row = "00000000100" and col <= "
      00000000100")) or out_en = '1') then
      col_cnt <= col_cnt + 1;
      if col_cnt = col_size then
        col_cnt <= (others => '0');
        row_cnt <= row_cnt + 1;
        if row_cnt = row_size then
          row_cnt <= (others => '0');
        end if;
      end if;
    end if;

    if (row_addr = row_addr_max) then
      row_addr <= (others => '0');
    else
      row_addr <= row_addr + 1;
    end if;

    if DATA_MODE = 0 then
      row_in_00 <= COL(7 downto 0);
    else
      row_in_00 <= DIN;
    end if;

    -- Delay row count signal by 5 shifts
    row_d1 <= row(0);
    row_d2 <= row_d1;
    row_d3 <= row_d2;
    row_d4 <= row_d3;

    col_d1 <= col(0);
    col_d2 <= col_d1;
    col_d3 <= col_d2;
    col_d4 <= col_d3;

    row_in_01 <= row_in_00;
    row_in_02 <= row_in_01;
    row_in_03 <= row_in_02;
    row_in_04 <= row_in_03;

    row_in_11 <= row_in_10;
    row_in_12 <= row_in_11;
    row_in_13 <= row_in_12;
    row_in_14 <= row_in_13;

```

```

        row.in_21 <= row.in_20;
        row.in_22 <= row.in_21;
        row.in_23 <= row.in_22;
        row.in_24 <= row.in_23;

        row.in_31 <= row.in_30;
        row.in_32 <= row.in_31;
        row.in_33 <= row.in_32;
        row.in_34 <= row.in_33;

        row.in_41 <= row.in_40;
        row.in_42 <= row.in_41;
        row.in_43 <= row.in_42;
        row.in_44 <= row.in_43;

        tmp.Y <= Y;
    end if;
end process ROW_SHFT;

in_en <= '1' when ((row <= "00000000011") or (row = "00000000100" and col <= "00000000101")) else
'0';
out_en <= '1' when (out_cnt /= 0) else '0';
ROW_SHFT_DELAY : process (RESET, CLK)
begin
    if (RESET = '1') then
        out_cnt <= (others => '0');
    elsif (CLK'Event and CLK = '1') then
        if (row = row_size and col = col_size) then
            out_cnt <= conv_std_logic_vector(4*conv_integer(col_size) + 10, 15); --
                conv_std_logic_vector (16, 15);
        end if;
        if (out_cnt /= 0) then
            out_cnt <= out_cnt - 1;
        end if;
    end if;
end process ROW_SHFT_DELAY;

-- Synchronize Output
-- Latch 16 bit words with a signal to trigger a WEN for the ram fifo.

SYNC_OUT : process (CLK, RESET)
begin
    if (RESET = '1') then
        DOUT <= (others => '1');
        byte_cnt <= '0';
        RAMWEN <= '0';
        int_en <= '0';
    elsif (CLK'EVENT and CLK = '1') then
        int_en <= EN;
        if ((EN = '1' and in_en = '0') or out_en = '1') then
            byte_cnt <= NOT byte_cnt;
            -- Build a 16 bit word with 2-8bits for raw data output.
            if (color_bit = '0') then
                if (byte_cnt = '1') then
                    DOUT <= tmp.Y(7 downto 0) & Y (7 downto 0); --
                        SWITCH FOR HARDWARE SYNTHESIS
                    RAMWEN <= '1';
                else
                    RAMWEN <= '0';
                end if;
            else
                DOUT <= Y(7 downto 0) & U(3 downto 0) & V(3 downto
0);
                RAMWEN <= '1';
            end if;
        else
            --byte_cnt <= '0';
            RAMWEN <= '0';
        end if;
    end if;
end process SYNC_OUT;

```

```

-- Get RGB
-- Bilinear Interpolation of the the RAW image data to 3- 8bit color channels.

GET_RGB : process (RESET, CLK, EN)
begin
  if (RESET = '1') then
    Y <= (others => '0');
    U <= (others => '0');
    V <= (others => '0');
  elsif (CLK'Event and CLK = '1' and (EN = '1' or out_en = '1')) then
    -- case OUTPUTFORMAT is
    case DATA_MODE is
      when "000" => -- TEST DATA MODE
        R := conv_integer (row_in_44);
        G := conv_integer (row_in_44);
        B := conv_integer (row_in_44);
      when "001" => -- RAW DATA MODE
        R := conv_integer (row_in_44);
        G := 0;
        B := 0;
      when "010" => -- Nearest Neighbour Interpolation
        if (row_cnt(0)='0') then
          if (col_cnt(0)='0') then -- Green Center on
            Red Row
              R := conv_integer (row_in_43);
              G := conv_integer (row_in_44);
              B := conv_integer (row_in_34);
            else
              -- Red Center
              R := conv_integer (row_in_44);
              G := conv_integer (row_in_34);
              B := conv_integer (row_in_33);
            end if;
          else
            if (col_cnt(0) = '0') then -- Blue Center
              R := conv_integer (row_in_33);
              G := conv_integer (row_in_43);
              B := conv_integer (row_in_44);
            else
              -- Green Center on Blue Row
              R := conv_integer (row_in_34);
              G := conv_integer (row_in_33);
              B := conv_integer (row_in_43);
            end if;
          end if;
        when "011" => -- Bilinear Interpolation
          if (row_cnt(0)='0') then
            if (col_cnt(0)='0') then -- Green Center on
              Red Row
                R := (conv_integer (row_in_21) + conv_integer (
                  row_in_21))/2;
                G := conv_integer (row_in_22);
                B := (conv_integer (row_in_12) + conv_integer (
                  row_in_32))/2;
              else
                -- Red Center
                R := conv_integer (row_in_22);
                G := (conv_integer (row_in_21) + conv_integer (
                  row_in_21) + conv_integer (row_in_12) +
                  conv_integer (row_in_32))/4;
                B := (conv_integer (row_in_11) + conv_integer (
                  row_in_13) + conv_integer (row_in_31) +
                  conv_integer (row_in_33))/4;
              end if;
            else
              if (col_cnt(0) = '0') then -- Blue Center
                R := (conv_integer (row_in_11) + conv_integer (
                  row_in_13) + conv_integer (row_in_31) +
                  conv_integer (row_in_33))/4;
                G := (conv_integer (row_in_21) + conv_integer (
                  row_in_21) + conv_integer (row_in_12) +
                  conv_integer (row_in_32))/4;
                B := conv_integer (row_in_33);
              else
                -- Green Center on Blue Row

```

```

R := (conv_integer (row_in_12) + conv_integer (
row_in_32))/2;
G := conv_integer (row_in_22);
B := (conv_integer (row_in_21) + conv_integer (
row_in_21))/2;
end if;
end if;

when "100" => -- Edge Enhanced
if not(row_cnt(0) = col_cnt(0)) then -- Red of Blue Center Pixel

-- North Edge Detect
D1 := abs(conv_integer (row_in_02) - conv_integer (
row_in_22)) +
abs(conv_integer (row_in_12) -
conv_integer (row_in_32)) +
abs(2*conv_integer (row_in_12) -
conv_integer (row_in_21) -
conv_integer (row_in_23))/2;
-- East Edge Detect
D2 := abs(conv_integer (row_in_24) - conv_integer (
row_in_22)) +
abs(conv_integer (row_in_23) -
conv_integer (row_in_21)) +
abs(2*conv_integer (row_in_23) -
conv_integer (row_in_12) -
conv_integer (row_in_32))/2;
-- South Edge Detect
D3 := abs(conv_integer (row_in_42) - conv_integer (
row_in_22)) +
abs(conv_integer (row_in_32) -
conv_integer (row_in_12)) +
abs(2*conv_integer (row_in_32) -
conv_integer (row_in_23) -
conv_integer (row_in_21))/2;
-- West Edge Detect
D4 := abs(conv_integer (row_in_20) - conv_integer (
row_in_22)) +
abs(conv_integer (row_in_21) -
conv_integer (row_in_23)) +
abs(2*conv_integer (row_in_21) -
conv_integer (row_in_32) -
conv_integer (row_in_12))/2;

tP1 := conv_integer (row_in_12) + (conv_integer (row_in_22
) - conv_integer (row_in_02))/2;
if D1 < threshold1 then
P1 := tP1;
c1 := 1;
else
P1 := 0;
c1 := 0;
end if;

tP2 := conv_integer (row_in_23) + (conv_integer (row_in_22
) - conv_integer (row_in_24))/2;
if D2 < threshold1 then
P2 := tP2;
c2 := 1;
else
P2 := 0;
c2 := 0;
end if;

tP3 := conv_integer (row_in_32) + (conv_integer (row_in_22
) - conv_integer (row_in_42))/2;
if D3 < threshold1 then
P3 := tP3;
c3 := 1;
else
P3 := 0;
c3 := 0;
end if;

tP4 := conv_integer (row_in_21) + (conv_integer (row_in_22
) - conv_integer (row_in_20))/2;
if D4 < threshold1 then
P4 := tP4;

```

```

else      c4 := 1;
else      P4 := 0;
          c4 := 0;
end if;

sum1 := P1 + P2 + P3 + P4;
sum2 := c1 + c2 + c3 + c4;

-- Fill in the missing Green Pixel
if sum2 = 0 then
  G := (tP1 + tP2 + tP3 + tP4)/4;
elsif sum2 = 1 then
  G := sum1;
elsif sum2 = 2 then
  G := sum1/2;
elsif sum2 = 3 then
  G := sum1*21/64;
else
  G := sum1/4;
end if;

-- Find missing Red or Blue Pixel

tP5 := (((conv_integer (row_in_11) + conv_integer (
row_in_31))/2 + G - conv_integer (row_in_21)) +
        ((conv_integer (row_in_13) +
conv_integer (row_in_33))/2
+ G - conv_integer (
row_in_23))) / 2;

if (D1 < threshold2) or (D3 < threshold2) then
  P5 := tP5;
  c5 := 1;
else
  P5 := 0;
  c5 := 0;
end if;

tP6 := (((conv_integer (row_in_11) + conv_integer (
row_in_13))/2 + G - conv_integer (row_in_12)) +
        ((conv_integer (row_in_31) +
conv_integer (row_in_33))/2
+ G - conv_integer (
row_in_32))) / 2;

if (D2 < threshold2) or (D4 < threshold2) then
  P6 := tP6;
  c6 := 1;
else
  P6 := 0;
  c6 := 0;
end if;

sum3 := P5 + P6;
sum4 := c5 + c6;

if sum4 = 0 then
  if (D1 <= D2 and D1 <= D4) or (D3 <= D2 and D3 <=
D4) then
    P7 := tP5;
  else
    P7 := tP6;
  end if;
elsif sum4 = 1 then
  P7 := sum3;
else
  P7 := sum3/2;
end if;

if row_cnt(0) = '0' then -- Red Center
  R := conv_integer (row_in_22);
  B := P7;
else -- Blue Center
  R := P7;
  B := conv_integer (row_in_22);
end if;

```

```

--
else -- Green Center Pixel

```

```

-- North Edge Detect
D1 := abs(conv_integer (row_in_02) - conv_integer (
    row_in_22)) +
    abs(conv_integer (row_in_12) -
        conv_integer (row_in_32)) +
    (abs(conv_integer (row_in_11) -
        conv_integer (row_in_22)) +
    abs(conv_integer (row_in_13) -
        conv_integer (row_in_22)))/2;

-- East Edge Detect
D2 := abs(conv_integer (row_in_24) - conv_integer (
    row_in_22)) +
    abs(conv_integer (row_in_23) -
        conv_integer (row_in_21)) +
    (abs(conv_integer (row_in_13) -
        conv_integer (row_in_22)) +
    abs(conv_integer (row_in_33) -
        conv_integer (row_in_22)))/2;

-- South Edge Detect
D3 := abs(conv_integer (row_in_42) - conv_integer (
    row_in_22)) +
    abs(conv_integer (row_in_32) -
        conv_integer (row_in_12)) +
    (abs(conv_integer (row_in_33) -
        conv_integer (row_in_22)) +
    abs(conv_integer (row_in_31) -
        conv_integer (row_in_22)))/2;

-- West Edge Detect
D4 := abs(conv_integer (row_in_20) - conv_integer (
    row_in_22)) +
    abs(conv_integer (row_in_21) -
        conv_integer (row_in_23)) +
    (abs(conv_integer (row_in_31) -
        conv_integer (row_in_22)) +
    abs(conv_integer (row_in_11) -
        conv_integer (row_in_22)))/2;

if (D1 <= D2) and (D1 <= D3) and (D1 <= D4) then -- D1 is
    minimum
    P1 := conv_integer (row_in_12) + (conv_integer (
        row_in_22) - conv_integer (row_in_02))/2;
    if D2 <= D4 then
        P2 := conv_integer (row_in_23) +
            (conv_integer (row_in_22) - (
                conv_integer (row_in_13) +
                conv_integer (row_in_33)))/2;
    else
        P2 := conv_integer (row_in_21) +
            (conv_integer (row_in_22) - (
                conv_integer (row_in_11) +
                conv_integer (row_in_31)))/2;
    end if;
elseif D2 <= D3 and D2 <= D4 then -- D2 in minimum
    P2 := conv_integer (row_in_23) + (conv_integer (
        row_in_22) - conv_integer (row_in_24))/2;
    if D1 <= D3 then
        P1 := conv_integer (row_in_12) +
            (conv_integer (row_in_22) - (
                conv_integer (row_in_11) +
                conv_integer (row_in_13)))/2;
    else
        P1 := conv_integer (row_in_32) +
            (conv_integer (row_in_22) - (
                conv_integer (row_in_31) +
                conv_integer (row_in_33)))/2;
    end if;
elseif D3 <= D4 then
    P1 := conv_integer (row_in_32) + (conv_integer (
        row_in_22) - conv_integer (row_in_42))/2;
    if D2 <= D4 then
        P2 := conv_integer (row_in_23) +
            (conv_integer (row_in_22) - (
                conv_integer (row_in_13) +
                conv_integer (row_in_33)))/2;

```



```

else
    P2 := conv_integer (row_in_21) +
          (conv_integer (row_in_22) - (
            conv_integer (row_in_11) +
            conv_integer (row_in_31)))/2);
end if;
else
    P2 := conv_integer (row_in_21) + (conv_integer (
    row_in_22) - conv_integer (row_in_20))/2;
    if D1 <= D3 then
        P1 := conv_integer (row_in_12) +
              (conv_integer (row_in_22) - (
                conv_integer (row_in_11) +
                conv_integer (row_in_13)))/2);
    else
        P1 := conv_integer (row_in_32) +
              (conv_integer (row_in_22) - (
                conv_integer (row_in_31) +
                conv_integer (row_in_33)))/2);
    end if;
end if;
G := conv_integer (row_in_22);
if row_cnt(0) = '0' then -- Red row
    R := P2;
    B := P1;
else
    R := P1;
    B := P2;
end if;
end if;
when others =>
    R := 0;
    G := 0;
    B := 0;
end case;
if data_mode = "000" or data_mode = "001" then
    Yt := conv_std_logic_vector(R, 32);
    Ut := conv_std_logic_vector(G, 32);
    Vt := conv_std_logic_vector(B, 32);
    Y <= "00" & Yt(7 downto 0);
    U <= Ut(7 downto 4);
    V <= Vt(7 downto 4);
else
    Yi := (((66*R)+(129*G))+((25*B)+ 4224));
    Ui := (((-38*R) + (-74*G))+((112*B)+ 32896));
    Vi := (((112*R) + (-94*G))+ ((18*B)+ 32896));
    Yt := conv_std_logic_vector(Yi, 32);
    Ut := conv_std_logic_vector(Ui, 32);
    Vt := conv_std_logic_vector(Vi, 32);
    Y <= "00" & Yt(15 downto 8);
    U <= Ut(15 downto 12);
    V <= Vt(15 downto 12);
end if;
end if;
end process GET_RGB;
end Behavioral;

```

Listing C.4: Image_Processor_TB.vhd

```

-- Company:
-- Engineer:
--
-- Create Date: 12:42:31 03/12/2008
-- Design Name: ImageProcessor
-- Module Name: /home/akarloff/pill_machine/hdl/Spartan_FG_v2/Image_Processor_TB.vhd
-- Project Name: Spartan_FG_v2
-- Target Device:
-- Tool versions:
-- Description:

```

```

--
-- VHDL Test Bench Created by ISE for module: ImageProcessor
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test. Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.

```

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE std.textio.ALL;

library textutil;      -- Synposys Text I/O package
use textutil.std_logic_textio.all;
use textutil.txt_util.all;

ENTITY Image_Processor_TB_vhd IS
END Image_Processor_TB_vhd;

ARCHITECTURE behavior OF Image_Processor_TB_vhd IS

    -- Component Declaration for the Unit Under Test (UUT)
    COMPONENT ImageProcessor
    PORT(
        CLK : IN std_logic;
        RESET : IN std_logic;
        EN : IN std_logic;
        DIN : IN std_logic_vector(7 downto 0);
        ROW : IN std_logic_vector(10 downto 0);
        COL : IN std_logic_vector(10 downto 0);
        FRAME : IN std_logic;
        ROW_SIZE : IN std_logic_vector(10 downto 0);
        COL_SIZE : IN std_logic_vector(10 downto 0);
        DATA_MODE : IN std_logic_vector(2 downto 0);
        COLOR_BIT : IN std_logic;
        DOUT : OUT std_logic_vector(15 downto 0);
        RAMWEN : OUT std_logic;
        IP_DEBUG : OUT std_logic
    );
    END COMPONENT;

    -- Simulation parameters
    constant clk_high : time := 10 ns; -- ns
    constant clk_low : time := 10 ns; -- ns
    constant row_pxl_size : integer := 60; --768; --pxls
    constant col_pxl_size : integer := 40; --512; --pxls
    constant h_blank : integer := 4; --pxls
    constant v_blank : integer := 12; --pxls
    constant reset_delay : time := 100 * (clk_high + clk_low);
    constant line_time_0 : time := col_pxl_size * (clk_high + clk_low);
    constant line_time_1 : time := line_time_0 + h_blank * (clk_low + clk_high);
    constant frame_time_0 : time := line_time_1 * row_pxl_size + h_blank * (clk_low + clk_high
    );
    constant frame_time_1 : time := line_time_1 * 2; -- !!! Times number of row buffers !!!!
    constant frame_time_2 : time := frame_time_0 + frame_time_1;

    --Inputs
    SIGNAL PIXCLK_IN : std_logic := '0';
    SIGNAL USBCLK_IN : std_logic := '0';
    SIGNAL RESET_IN : std_logic := '1';
    SIGNAL LINE_VALID_IN : std_logic := '0';
    SIGNAL FRAME_VALID_IN : std_logic := '0';
    SIGNAL USB_FULL_IN : std_logic := '0';
    SIGNAL CCD_DATA_IN : std_logic_vector(9 downto 0) := (others=>'0');

    --Inputs
    SIGNAL EN : std_logic := '0';
    SIGNAL FRAME : std_logic := '1';
    SIGNAL COLOR_BIT : std_logic := '0';

```

```

SIGNAL ROW : std_logic_vector(10 downto 0) := (others=>'0');
SIGNAL COL : std_logic_vector(10 downto 0) := (others=>'0');
SIGNAL ROW_SIZE : std_logic_vector(10 downto 0) := conv_std_logic_vector (row_pxl_size -
1, 11); --"00000001011"; --"10111111111";
SIGNAL COL_SIZE : std_logic_vector(10 downto 0) := conv_std_logic_vector (col_pxl_size -
1, 11); --"00000001011"; --"11111111111";
SIGNAL DATAMODE : std_logic_vector(2 downto 0) := "011";

-- Outputs
SIGNAL USB_BUS_OUT : std_logic_vector(15 downto 0);
SIGNAL USB_SLWR_OUT : std_logic;
SIGNAL IP_DEBUG : std_logic;

signal inc_data : std_logic := '0';

-- File IO
shared variable VALUE_in, VALUE_out : character; --BIT VECTOR (7 downto 0);
shared variable iVALUE_in : integer;
shared variable L_in, L_out : LINE;
shared variable GOOD : boolean;
file FIN : TEXT is in "tb.raw"; --"image.raw";
file FOUT : TEXT is out "output.raw";

-- Test Bench Variables
shared variable pxl_data : std_logic_vector(9 downto 0) := (others => '0');
shared variable pxl_row : std_logic_vector(10 downto 0) := (others => '0');
shared variable pxl_col : std_logic_vector(10 downto 0) := (others => '0');
BEGIN

-- Instantiate the Unit Under Test (UUT)
 uut: ImageProcessor PORT MAP(
    CLK => PIXCLK_IN,
    RESET => RESET_IN,
    EN => EN,
    DIN => CCD_DATA_IN(7 downto 0),
    ROW => ROW,
    COL => COL,
    FRAME => FRAME,
    ROW_SIZE => ROW_SIZE,
    COL_SIZE => COL_SIZE,
    DATA_MODE => DATA_MODE,
    COLOR_BIT => COLOR_BIT,
    DOUT => USB_BUS_OUT,
    RAMWEN => USB_SLWR_OUT,
    IP_DEBUG => IP_DEBUG
 );

clock : PROCESS
begin
    PIXCLK_IN <= '1';
    wait for clk_low;
    PIXCLK_IN <= '0';
    wait for clk_high;
end process clock;

data_out : process (USB_SLWR_OUT)
begin
    if (USB_SLWR_OUT'Event and USB_SLWR_OUT = '1') then
        --hwrite(L_out, USB_BUS_OUT); --to_bitVector(USB_BUS_OUT), left,
        write(L_out, str(conv_integer(USB_BUS_OUT(15 downto 8))) & "-" &
            str(conv_integer(USB_BUS_OUT(7 downto 0))));
        --write(L_out, string'(CONV_INTEGER(USB_BUS_OUT(7 downto 0))));
        --write(L_out, string'(CONV_INTEGER(USB_BUS_OUT(15 downto 8))));
        writeline(FOUT, L_out);
    end if;
end process;

EN <= FRAME_VALID_IN and LINE_VALID_IN;

tb : PROCESS
BEGIN
    -- Wait 100 ns for global reset to finish
    wait for 100 ns;
    RESET_IN <= '0';
    wait for 98 ns;
    FRAME_VALID_IN <= '1';
    wait for 100 ns; --(v_blank*(clk_high+clk_low));
    for pxl_rows in 0 to row_pxl_size - 1 loop
        wait for (h_blank * (clk_high+clk_low));
    end loop;

```

```

        line_valid_in <= '1';
        readline (FIN, L.in);
        for pxl_cols in 0 to col_pxl_size - 1 loop
            --read(L.in, VALUE.in, GOOD);
            read(L.in, iVALUE.in, GOOD);
            CCD_DATA_IN <= conv_std_logic_vector(iVALUE.in, 10);
            --CCD_DATA_IN <= conv_std_logic_vector(character'pos(VALUE.in),
                10);
            --COL <= COL + 1;
            ROW <= conv_std_logic_vector(pxl_rows, 11);
            COL <= conv_std_logic_vector(pxl_cols, 11);
            wait for (clk_high + clk_low);
        end loop;
        --ROW <= ROW + 1;
        --COL <= (others => '0');
        line_valid_in <= '0';
        wait for (h_blank * (clk_high + clk_low));
    end loop;
    wait for (v_blank * (clk_high + clk_low));
    frame_valid_in <= '0';
    ROW <= (others => '0');
    COL <= (others => '0');
    wait for 20000 ns;
    RESET_IN <= '1';

    wait; -- will wait forever
END PROCESS;

```

END;

Listing C.5: frame_grabber_v1.ucf

```

# Timing Constraints
NET "PIXCLK" PERIOD = 20ns;
NET "USBCLK{" PERIOD = 20ns;

#####
## Version 2 PCB I/O Pin Assignments
#####
NET "PIXCLK.IN" LOC = "D8" ;
NET "IFCLK.IN" LOC = "E10" | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
NET "RESET.IN" LOC = "D2" | IOSTANDARD = LVCMOS33 ;
NET "LINE.VALID.IN" LOC = "D11" | IOSTANDARD = LVCMOS33 ;
NET "FRAME.VALID.IN" LOC = "E7" | IOSTANDARD = LVCMOS33 ;
NET "CCD_DATA_IN<0>" LOC = "A14" | IOSTANDARD = LVCMOS33 ;
NET "CCD_DATA_IN<1>" LOC = "A12" | IOSTANDARD = LVCMOS33 ;
NET "CCD_DATA_IN<2>" LOC = "B13" | IOSTANDARD = LVCMOS33 ;
NET "CCD_DATA_IN<3>" LOC = "C11" | IOSTANDARD = LVCMOS33 ;
NET "CCD_DATA_IN<4>" LOC = "B10" | IOSTANDARD = LVCMOS33 ;
NET "CCD_DATA_IN<5>" LOC = "B6" | IOSTANDARD = LVCMOS33 ;
NET "CCD_DATA_IN<6>" LOC = "B7" | IOSTANDARD = LVCMOS33 ;
NET "CCD_DATA_IN<7>" LOC = "C4" | IOSTANDARD = LVCMOS33 ;
NET "CCD_DATA_IN<8>" LOC = "C6" | IOSTANDARD = LVCMOS33 ;
NET "CCD_DATA_IN<9>" LOC = "C3" | IOSTANDARD = LVCMOS33 ;
NET "GSHT.CTL.OUT" LOC = "G13" | IOSTANDARD = LVCMOS33 | DRIVE = 16 | SLEW = FAST ;
NET "USB_FULL.IN" LOC = "R2" | IOSTANDARD = LVCMOS33 ;
NET "USB_BUS.OUT<0>" LOC = "J1" | IOSTANDARD = LVCMOS33 ;
NET "USB_BUS.OUT<1>" LOC = "R6" | IOSTANDARD = LVCMOS33 ;
NET "USB_BUS.OUT<2>" LOC = "T5" | IOSTANDARD = LVCMOS33 ;
NET "USB_BUS.OUT<3>" LOC = "P12" | IOSTANDARD = LVCMOS33 ;
NET "USB_BUS.OUT<4>" LOC = "N12" | IOSTANDARD = LVCMOS33 ;
NET "USB_BUS.OUT<5>" LOC = "G3" | IOSTANDARD = LVCMOS33 ;
NET "USB_BUS.OUT<6>" LOC = "T13" | IOSTANDARD = LVCMOS33 ;
NET "USB_BUS.OUT<7>" LOC = "R13" | IOSTANDARD = LVCMOS33 ;
NET "USB_BUS.OUT<8>" LOC = "G2" | IOSTANDARD = LVCMOS33 ;
NET "USB_BUS.OUT<9>" LOC = "A5" | IOSTANDARD = LVCMOS33 ;
NET "USB_BUS.OUT<10>" LOC = "A7" | IOSTANDARD = LVCMOS33 ;
NET "USB_BUS.OUT<11>" LOC = "B4" | IOSTANDARD = LVCMOS33 ;
NET "USB_BUS.OUT<12>" LOC = "K13" | IOSTANDARD = LVCMOS33 ;
NET "USB_BUS.OUT<13>" LOC = "L12" | IOSTANDARD = LVCMOS33 ;
NET "USB_BUS.OUT<14>" LOC = "L13" | IOSTANDARD = LVCMOS33 ;
NET "USB_BUS.OUT<15>" LOC = "K15" | IOSTANDARD = LVCMOS33 ;
NET "USB_ADR0.OUT" LOC = "F5" | IOSTANDARD = LVCMOS33 ;
NET "USB_ADR1.OUT" LOC = "E4" | IOSTANDARD = LVCMOS33 ;
NET "USB_SLRD.OUT" LOC = "L14" | IOSTANDARD = LVCMOS33 ;
NET "USB_SLWR.OUT" LOC = "L15" | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
NET "USB_PCKTEND.OUT" LOC = "E1" | IOSTANDARD = LVCMOS33 ;
NET "NSENSOR_RST" LOC = "E13" | IOSTANDARD = LVCMOS33 ;
#NET "FS.EN" LOC = "J1" | IOSTANDARD = LVCMOS33 ;

```

C. VHDL CODE

```
#NET "FS0" LOC = "G3" | IOSTANDARD = LVCMOS33 ;  
#NET "FS1" LOC = "G2" | IOSTANDARD = LVCMOS33 ;  
NET "MCLK" LOC = "L8" | IOSTANDARD = LVCMOS33 ;  
NET "SCL" LOC = "P9" | IOSTANDARD = LVCMOS33 ;  
NET "SDA" LOC = "N16" | IOSTANDARD = LVCMOS33 ;  
NET "CLOCKIN" LOC = "N8" | IOSTANDARD = LVCMOS33 ;  
NET "N1" LOC = "N14" | IOSTANDARD = LVCMOS33 | DRIVE = 16 | SLEW = FAST ; # Trigger Out T2  
NET "P1" LOC = "M14" | IOSTANDARD = LVCMOS33 ; # Trigger In T1
```

Appendix D

MATLAB Code

Listing D.1: demosaic_test.m

```
=====  
% Camera Board Demosaicing  
=====  
% FILENAME: demosaic_test.m  
% DESCRIPTION: Computes the output for an image interpolated using the  
%               Edge Enhanced method and compares to Nearest Neighbour and  
%               Bilinear.  
% DATE: July 24, 2007  
% AUTHOR: Anthony Karloff, University of Windsor  
=====  
  
filename = 'lighthouse.jpg';  
  
% Read the image file  
im = double(imread(filename));  
M = size(im,1);           %height (rows - y)  
N = size(im,2);           %width (cols - x)  
  
% Create the RAW data  
bayer = RGB2RAW(im);  
raw_data = bayer(:, :, 1).*repmat([0 1; 0 0], M/2, N/2) + bayer(:, :, 2).*repmat([1 0; 0 1], M/2, N/2)  
          + bayer(:, :, 3).*repmat([0 0; 1 0], M/2, N/2);  
  
% Process as Nearest Neighbour and Bilinear for comparison.  
imNN = DEMOSAIC(bayer, 'nearest-neighbour');  
imBilinear = DEMOSAIC(bayer, 'bilinear');  
% Demosaic using the Edge Enhanced Method  
imEE = ee_demosaic(raw_data);  
  
% Compute Peak Signal to Noise Ratio (PSNR)  
error = imRRR(3:M-2, 3:N-2,:) - im(3:M-2, 3:N-2,:);  
im_MSE = MSE(imEE(3:M-2, 3:N-2,:), im(3:M-2, 3:N-2,:));  
im_PSNR = 10 * log10(65025/im_MSE)  
  
% Create a Grayscale for comparison  
yuv = RGB2YUV(imEE);  
  
% Display images.  
imtool(uint8(imBilinear));  
imtool(uint8(imNN));  
imview(uint8(imEE));  
imtool(uint8(yuv(:, :, 1)))
```

Listing D.2: ee_demosaic.m

```

%=====
% Camera Board Demosaicing
%=====
% FILENAME: ee_demosaic.m
% DESCRIPTION: A cheap hardware demosaic
% DATE: June 3, 2008
% AUTHOR: Anthony Karloff, University of Windsor
%=====

function [im_out] = ee_demosaic (im_in)

M = size(im_in,1); %rows
N = size(im_in,2); %cols

im_data = im_in;

threshold1 = 25; %95;
threshold2 = 55;
threshold3 = 0;
threshold4 = 0;

% Initialize the Output Matrix
im_demosaic(:, :, 1) = im_data.*repmat([0 1; 0 0], M/2, N/2); % Red Channel
im_demosaic(:, :, 2) = im_data.*repmat([1 0; 0 1], M/2, N/2); % Green Channel
im_demosaic(:, :, 3) = im_data.*repmat([0 0; 1 0], M/2, N/2); % Blue Channel

% Define Vectors for interpolating neighbouring pixels
v(1,:) = [-1 0];
v(2,:) = [0 1];
v(3,:) = [1 0];
v(4,:) = [0 -1];
v(5,:) = [-1 1];
v(6,:) = [1 1];
v(7,:) = [1 -1];
v(8,:) = [-1 -1];

for i=3:M-2
    for j=3:N-2
        % Red or Blue Center
        if mod(i,2) ~= mod(j,2) % RED or BLUE
            for k=1:4
                D(k) = abs(im_data(i+2*v(k,1), j+2*v(k,2)) - im_data(i, j)) + abs(im_data(i+v(k,1), j+v(k,2)) - im_data(i-v(k,1), j-v(k,2))) + (abs(2*im_data(i+v(k,1), j+v(k,2)) - im_data(i-v(k,2), j-v(k,1)) - im_data(i+v(k,2), j+v(k,1))))/2;
            end

            % Complete the missing GREEN value
            sum = 0;
            sum2 = 0;
            cnt = 0;

            for k=1:4
                sum2 = sum2 + im_data(i+v(k,1), j+v(k,2)) + (im_data(i, j) - im_data(i+2*v(k,1), j+2*v(k,2)))/2;
                if D(k) < threshold1
                    sum = sum + im_data(i+v(k,1), j+v(k,2)) + (im_data(i, j) - im_data(i+2*v(k,1), j+2*v(k,2)))/2;
                    cnt = cnt + 1;
                end
            end

            if cnt == 0
                im_demosaic(i, j, 2) = sum2/4;
            else
                im_demosaic(i, j, 2) = sum/cnt;
            end

            % Complete the missing RED or BLUE value
            sum = 0;
            sum2 = 0;
            cnt = 0;
            sum2 = (((im_data(i-1, j-1) + im_data(i+1, j-1))/2 + im_demosaic(i, j, 2) - im_data(i, j-1)) + ...
                ((im_data(i-1, j+1) + im_data(i+1, j+1))/2 + im_demosaic(i, j, 2) - im_data(i, j+1)))/2 + ...
                (((im_data(i-1, j-1) + im_data(i-1, j+1))/2 + im_demosaic(i, j, 2) - im_data(i-1, j)) + ...
                ((im_data(i+1, j-1) + im_data(i+1, j+1))/2 + im_demosaic(i, j, 2) - im_data(i+1, j)))/2;

            if (D(1) < threshold2 || D(3) < threshold2)

```

```

sum = (((im_data(i-1,j-1) + im_data(i+1,j-1))/2 + im_demosaic(i,j,2) - im_data(i,j
-1)) + ...
      ((im_data(i-1,j+1) + im_data(i+1,j+1))/2 + im_demosaic(i,j,2) - im_data(i,j
+1))) / 2;
cnt = cnt + 1;
end

if (D(2) < threshold2 || D(4) < threshold2)
sum = sum + ...
      ((im_data(i-1,j-1) + im_data(i-1,j+1))/2 + im_demosaic(i,j,2) - im_data(i
-1,j)) + ...
      ((im_data(i+1,j-1) + im_data(i+1,j+1))/2 + im_demosaic(i,j,2) - im_data(i
+1,j))/2;
cnt = cnt + 1;
end

if cnt == 0
[Y1 I1] = min(D(1:4));
if I1 == 1 || I1 == 3
im_demosaic(i,j,1+2*mod(i,2)) = (((im_data(i-1,j-1) + im_data(i+1,j-1))/2 +
im_demosaic(i,j,2) - im_data(i,j-1)) + ...
      ((im_data(i-1,j+1) + im_data(i+1,j+1))/2 +
im_demosaic(i,j,2) - im_data(i,j+1))) / 2;
else
im_demosaic(i,j,1+2*mod(i,2)) = (((im_data(i-1,j-1) + im_data(i-1,j+1))/2 +
im_demosaic(i,j,2) - im_data(i-1,j)) + ...
      ((im_data(i+1,j-1) + im_data(i+1,j+1))/2 +
im_demosaic(i,j,2) - im_data(i+1,j)))/2;
end
else
im_demosaic(i,j,1+2*mod(i,2)) = sum/cnt;
end

else %GREEN Pixel
for k=1:4
Dp(k) = im_data(i+v(k,1)+v(k,2),j-v(k,1)+v(k,2)) - im_data(i,j);
end

for k=1:4
D(k) = abs(im_data(i+2*v(k,1),j+2*v(k,2)) - im_data(i,j)) + abs(im_data(i+v(k,1),j
+v(k,2))-im_data(i-v(k,1),j-v(k,2))) + (abs(Dp(k)) + abs(Dp(mod(k+2,4)+1)))/2;
end

[Y1 I1] = min(D(1:4));
if I1 == 1
[Y2 I2] = min([D(2) D(4)]);
elseif I1 == 2
[Y2 I2] = min([D(1) D(3)]);
elseif I1 == 3
[Y2 I2] = min([D(2) D(4)]);
else
[Y2 I2] = min([D(1) D(3)]);
end

if I1 == 1
im_demosaic(i,j,1+2*mod(i,2)) = im_data(i+v(I1,1),j+v(I1,2)) + (im_data(i,j) -
im_data(i+2*v(I1,1),j+2*v(I1,2)))/2;

if I2 == 1
im_demosaic(i,j,3-2*mod(i,2)) = im_data(i,j+1) + (im_data(i,j) - (im_data(i-1,
j+1) + im_data(i+1,j+1))/2);
else
im_demosaic(i,j,3-2*mod(i,2)) = im_data(i,j-1) + (im_data(i,j) - (im_data(i-1,
j-1) + im_data(i+1,j-1))/2);
end

elseif I1 == 3
im_demosaic(i,j,1+2*mod(i,2)) = im_data(i+v(I1,1),j+v(I1,2)) + (im_data(i,j) -
im_data(i+2*v(I1,1),j+2*v(I1,2)))/2;

if I2 == 1
im_demosaic(i,j,3-2*mod(i,2)) = im_data(i,j+1) + (im_data(i,j) - (im_data(i-1,
j+1) + im_data(i+1,j+1))/2);
else
im_demosaic(i,j,3-2*mod(i,2)) = im_data(i,j-1) + (im_data(i,j) - (im_data(i-1,
j-1) + im_data(i+1,j-1))/2);
end

elseif I1 == 2
im_demosaic(i,j,3-2*mod(i,2)) = im_data(i+v(I1,1),j+v(I1,2)) + (im_data(i,j) -
im_data(i+2*v(I1,1),j+2*v(I1,2)))/2;

```



```

        if I2 == 1
            im_demosaic(i,j,1+2*mod(i,2)) = im_data(i-1,j) + im_data(i,j) - (im_data(i-1,j
            -1) + im_data(i-1,j+1))/2;
        else
            im_demosaic(i,j,1+2*mod(i,2)) = im_data(i+1,j) + im_data(i,j) - (im_data(i+1,j
            -1) + im_data(i+1,j+1))/2;
        end
    else
        im_demosaic(i,j,3-2*mod(i,2)) = im_data(i+v(I1,1),j+v(I1,2)) + (im_data(i,j) -
        im_data(i+2*v(I1,1),j+2*v(I1,2)))/2;
        if I2 == 1
            im_demosaic(i,j,1+2*mod(i,2)) = im_data(i-1,j) + im_data(i,j) - (im_data(i-1,j
            -1) + im_data(i-1,j+1))/2;
        else
            im_demosaic(i,j,1+2*mod(i,2)) = im_data(i+1,j) + im_data(i,j) - (im_data(i+1,j
            -1) + im_data(i+1,j+1))/2;
        end
    end
end
end
end
im_out = im_demosaic;

```

Listing D.3: demosaic.m

```

=====
% Camera Board Demosaicing
%
% FILENAME: DEMOSAIC.m
% DESCRIPTION: Performs Various Interpolations in a RAW Bayer Pattern image
% DATE: March 25, 2008
% AUTHOR: Anthony Karloff, University of Windsor
=====

function [im_demosaic] = DEMOSAIC (im_raw, mode)

if size(size(im_raw),2) == 3
    R = im_raw(:,:,1);
    G = im_raw(:,:,2);
    B = im_raw(:,:,3);

    switch mode
        case 'nearest_neighbour'
            im_demosaic(:,:,1) = R + circshift(R,[0 -1]) + circshift(R,[1 0]) + circshift(R,[1
            -1]);
            im_demosaic(:,:,2) = G + circshift(G,[0 1]);
            im_demosaic(:,:,3) = B + circshift(B,[0 1]) + circshift(B,[-1 0]) + circshift(B,[-1
            1]);

        case 'bilinear'
            %Interpolate Green Channel
            G = G + imfilter(G,[0 1 0; 1 0 1; 0 1 0]/4);

            %Interpolate Blue at Reds
            Br = imfilter(B,[1 0 1; 0 0 0; 1 0 1]/4);
            %Interpolate Blue at Greens
            Bg = imfilter(B+Br,[0 1 0; 1 0 1; 0 1 0]/4);

            %Interpolate Red at Blues;
            Rb = imfilter(R,[1 0 1; 0 0 0; 1 0 1]/4);
            %Interpolate Red at Greens;
            Rg = imfilter(R+Rb,[0 1 0; 1 0 1; 0 1 0]/4);

            im_demosaic(:,:,1) = R + Rb + Rg;
            im_demosaic(:,:,2) = G;
            im_demosaic(:,:,3) = B + Br + Bg;

        otherwise
            im_demosaic = 0;
    end
end
end

```

Listing D.4: RGB2RAW.m

```

%=====
% Camera Board Demosaicing
%=====
% FILENAME: RGB2RAW.m
% DESCRIPTION: Converts the 3 channels of an RGB image to RAW image format
%               along with the 3 separate color channels.
% DATE: July 24, 2007
% AUTHOR: Anthony Karloff, University of Windsor
%=====

function [im_raw] = RGB2RAW (input)

if size(size(input),2) == 3
    img_in = double(input);

    R = img_in(:,:,1);
    G = img_in(:,:,2);
    B = img_in(:,:,3);

    M = size(img_in,1);
    N = size(img_in,2);

    im_raw(:,:,1) = R.*repmat([0 1; 0 0], M/2, N/2);
    im_raw(:,:,2) = G.*repmat([1 0; 0 1], M/2, N/2);
    im_raw(:,:,3) = B.*repmat([0 0; 1 0], M/2, N/2);

else
    img_raw = 0;
end

```

Listing D.5: RGB2YUV.m

```

%=====
% Camera Board Demosaicing
%=====
% FILENAME: RGB2YUV.m
% DESCRIPTION: Converts the 3 channels of an RGB image to YUV image format
%               along with the 3 separate color channels.
% DATE: July 24, 2007
% AUTHOR: Anthony Karloff, University of Windsor
%=====

function [img_yuv] = RGB2YUV (input)

if size(size(input),2) == 3
    img_in = double(input);

    R = img_in(:,:,1);
    G = img_in(:,:,2);
    B = img_in(:,:,3);

    Y = (66.*R+129.*G+25.*B+128 + 4096)./256;
    U = (-38.*R-74.*G+112.*B+128 + 32768)./256;
    V = (112.*R-94.*G+18.*B+128 + 32768)./256;

    img_yuv(:,:,1) = Y;
    img_yuv(:,:,2) = U;
    img_yuv(:,:,3) = V;

else
    img_yuv = 0;
end

```

VITA AUCTORIS

Anthony Christopher Karloff was born in Windsor, Ontario, Canada on April 1st, 1982. He received his B.A.Sc. Degree in Electrical Engineering in 2006 from the University of Windsor, Windsor, ON. He is currently a candidate in the department of Electrical and Computer Engineering at the University of Windsor. His research interest include Image and Signal Processing, Reconfigurable System design and custom hardware design.