

University of Windsor

## Scholarship at UWindor

---

Electronic Theses and Dissertations

Theses, Dissertations, and Major Papers

---

2008

# An Integrated Control and Data Acquisition System for Pharmaceutical Capsule Inspection

Neil E. Scott

*University of Windsor*

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

---

### Recommended Citation

Scott, Neil E., "An Integrated Control and Data Acquisition System for Pharmaceutical Capsule Inspection" (2008). *Electronic Theses and Dissertations*. 8120.

<https://scholar.uwindsor.ca/etd/8120>

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](mailto:scholarship@uwindsor.ca)) or by telephone at 519-253-3000ext. 3208.

# **An Integrated Control and Data Acquisition System for Pharmaceutical Capsule Inspection**

by

**Neil E. Scott**

A Thesis

Submitted to the Faculty of Graduate Studies through the  
Department of 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: 978-0-494-70596-4*  
*Our file* *Notre référence*  
*ISBN: 978-0-494-70596-4*

#### 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.

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

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

#### 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.

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

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

  
**Canada**

© 2008 Neil E. Scott

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*

---

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

This thesis incorporates the outcome of joint research undertaken in collaboration with Anthony C. Karloff and Mohammed J. Islam under the supervision of Dr. Roberto Muscedere. The collaboration contributions are outlined in Chapter 1. The personal contributions, design work and development performed by the author are the focus of this thesis.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contributions of other researchers to my thesis, and have obtained written permission from the co-authors to include the above materials 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.

---

# *Abstract*

---

Pharmaphil Inc. manufactures two-part gelatin capsules for the pharmaceutical industry. Their current methods of quality control of their product is by performing manual inspection of every carton of capsules prior to shipment. In today's modern manufacturing world, more efficient and cost-effective means of quality control exist. It is Pharmaphil's desire to develop a custom machine vision system to replace manual inspection with a potential opportunity in the capsule manufacturing quality control market. In collaboration with the Electrical and Computer Engineering Department at the University of Windsor, a novel system was developed to achieve this goal. The objective was to develop a system capable of inspecting 1000 capsules per minute with the ability to detect holes, cracks, dents, bubble, double caps and incorrect colour or size.

Using an antiquated machine vision system for capsule inspection from the mid-nineties as a base, a modern inspection system was developed that performed faster and more thorough inspections. As a measure to minimize the overall system cost as well as to increase flexibility, a full custom design was undertaken. The resulting system follows a traditional machine vision system whereby the main components include an image acquisition component, a processing unit and machine control. The designed system uses custom USB2.0 cameras to acquire images, a standard desktop PC to process image data and a custom machine control board to perform machine control and timing. The system operates with four identical quadrants operating in parallel to increase throughput.

The final system developed provided a proof-of-concept for the approach taken. The machine control and image acquisition component of the system yielded a maximum throughput of 1200 capsules per minute. After incorporating image inspection, the final result was a system that was capable of inspecting capsules at a rate of about 800 capsules per minute with high accuracy. With optimizations, the system throughput can be further improved. The findings throughout the development of the prototype system provide an excellent basis from which the first generation commercial unit can be designed.

To my family for their perpetual support and to Brianne for her patience, support and advice.

---

## *Acknowledgments*

---

I would like to express my gratitude towards my generous and supportive supervisor Dr. Roberto Muscedere for his guidance and devotion to the project. I would like to thank my colleagues Anthony Karloff and Mohammed Islam for their contributions to the project. I would also like to thank my committee members Dr. Mohammed Khalid and Dr. Walid Abdul-Kader.

# Contents

<b>Declaration of Co-Authorship</b>	<b>iv</b>
<b>Abstract</b>	<b>v</b>
<b>Dedication</b>	<b>vi</b>
<b>Acknowledgments</b>	<b>vii</b>
<b>List of Figures</b>	<b>xiv</b>
<b>List of Tables</b>	<b>xvi</b>
<b>List of Abbreviations</b>	<b>xvii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Current Quality Control Methods . . . . .	1
1.2 Automated Vision Solution . . . . .	2
1.3 Typical Manufacturing Defects . . . . .	2
1.4 State of the Art . . . . .	3
1.5 Motivation . . . . .	3
1.6 Design Strategy . . . . .	3
<b>2 Background</b>	<b>5</b>
2.1 OptiSorter . . . . .	5
2.1.1 Mechanical Design . . . . .	8
2.1.2 Electrical Design . . . . .	9
2.2 Universal Serial Bus (USB) . . . . .	11
2.2.1 Overview . . . . .	11
2.2.2 Enumeration . . . . .	12

---

2.2.3	Host Controller Interface . . . . .	13
2.2.4	Device Classes . . . . .	15
2.2.5	Data Flow Types . . . . .	16
2.2.6	USB Connectors and Cabling . . . . .	17
2.2.7	Bus Power . . . . .	18
2.2.8	USB Device Drivers . . . . .	18
2.3	I <sup>2</sup> C . . . . .	19
2.4	Machine Vision . . . . .	21
<b>3</b>	<b>Design Methodology</b>	<b>24</b>
3.1	Design Approach . . . . .	25
3.2	Design Considerations . . . . .	26
3.2.1	Data Transfer Medium . . . . .	26
3.2.2	Image Sensor . . . . .	27
3.2.3	Inspection Environment . . . . .	27
3.2.4	System Control . . . . .	28
3.2.5	Power Supplies . . . . .	29
3.2.6	Inspection PCs . . . . .	29
<b>4</b>	<b>USB 2.0 Camera</b>	<b>31</b>
4.1	Hardware Level Design . . . . .	34
4.2	Cypress EZ-USB FX2 . . . . .	35
4.2.1	ReNumeration . . . . .	35
4.3	FX2 Firmware . . . . .	37
4.3.1	Universal Serial Radio Project FX2 Library . . . . .	37
4.3.2	Bulk Transfers . . . . .	37
4.3.3	MI Sensor Configuration . . . . .	38
4.3.4	FPGA Register Configuration . . . . .	39
4.3.5	FPGA Loader Firmware . . . . .	39
4.3.6	Slave-Serial FPGA Loading Technique . . . . .	39
4.4	Control Board Communication . . . . .	40
4.5	EEPROM Memory Map . . . . .	41
4.6	USB2.0 Camera Linux Device Driver . . . . .	42
4.6.1	LibUSB . . . . .	42
4.6.2	pm_cam Class . . . . .	42
4.6.3	imgUSB . . . . .	46

---

---

<b>5</b>	<b>System Control Board</b>	<b>48</b>
5.1	Hardware Design . . . . .	48
5.1.1	Isolation Circuits . . . . .	49
5.1.2	Driving Circuits . . . . .	51
5.1.3	MCU Selection . . . . .	51
5.1.4	Power Regulation Circuit . . . . .	54
5.1.5	Electrically Controlled Pneumatic Valve Control Circuit . . . . .	54
5.1.6	Stepper Motor Controller Control Circuit . . . . .	56
5.1.7	LED Lighting Control Circuit . . . . .	57
5.1.8	Proximity Sensor Input Circuit . . . . .	58
5.1.9	Camera Triggering Circuit . . . . .	59
5.1.10	I <sup>2</sup> C Expansion Circuit . . . . .	59
5.1.11	PC Soft Power and Sense Circuits . . . . .	61
5.1.12	I <sup>2</sup> C Bus Switch Circuit . . . . .	61
5.1.13	Emergency Stop Input Circuit . . . . .	64
5.1.14	RS-232 Communication . . . . .	64
5.2	System Control Board PCB Layout . . . . .	65
5.2.1	gEDA Open-Source Tools . . . . .	66
5.2.2	PCB Fabrication . . . . .	66
5.2.3	PCB Population . . . . .	67
5.3	I <sup>2</sup> C I/O Expansion Board . . . . .	67
5.4	Firmware Development . . . . .	71
5.4.1	Functional Requirements . . . . .	73
5.4.2	Motor Control . . . . .	73
5.4.3	Motor Ramping Control . . . . .	76
5.4.4	Camera Trigger Control . . . . .	77
5.4.5	I <sup>2</sup> C Master . . . . .	78
5.4.6	I <sup>2</sup> C Slave . . . . .	80
5.4.7	I <sup>2</sup> C Bus Switch . . . . .	80
5.4.8	Job Queue . . . . .	81
5.4.9	I <sup>2</sup> C I/O Expansion Board Control . . . . .	84
5.4.10	I <sup>2</sup> C Expansion Interrupt Control . . . . .	85
5.4.11	I <sup>2</sup> C LCD Control . . . . .	86
5.4.12	UART Handler . . . . .	87
5.4.13	Soft PC Power Control . . . . .	89

---

---

5.4.14	Capsule Tracking and Counts . . . . .	90
5.4.15	Heartbeat Timer . . . . .	90
5.4.16	Debug Mode Timer . . . . .	90
<b>6</b>	<b>Host PC</b>	<b>92</b>
6.1	Operating System Selection . . . . .	92
6.2	Hardware Selection . . . . .	93
6.3	Software . . . . .	94
6.4	<i>inspect</i> . . . . .	94
6.4.1	POSIX Threads . . . . .	95
6.5	<i>fpga_loader_ss</i> . . . . .	101
6.6	<i>cam_init</i> . . . . .	102
6.7	<i>pyWindowConfig</i> . . . . .	103
6.8	<i>pyCamCal</i> . . . . .	104
6.9	W32 Control Panel Application . . . . .	104
6.10	Data Collection . . . . .	105
6.11	File Organization . . . . .	106
6.12	Start Up . . . . .	107
6.12.1	inittab . . . . .	107
6.12.2	Firmware Loading Script . . . . .	107
<b>7</b>	<b>Assembling the Prototype</b>	<b>109</b>
7.1	Wiring . . . . .	109
7.1.1	USB2.0 Camera . . . . .	111
7.2	Firmware . . . . .	111
7.3	Host PCs . . . . .	111
7.3.1	Operating System . . . . .	111
7.3.2	Software . . . . .	113
<b>8</b>	<b>Recommendations and Conclusions</b>	<b>114</b>
	<b>References</b>	<b>119</b>
<b>A</b>	<b>Control Board Design Reference</b>	<b>122</b>
A.1	Control Board Schematics . . . . .	122
A.2	I <sup>2</sup> CI/O Expansion Board Schematics . . . . .	138

---



---

<b>B</b>	<b>USB2.0 Camera FX2 Firmware</b>	<b>140</b>
B.1	Cypress EZ-USB FX2 Vendor Requests . . . . .	140
B.2	Micron Image Sensor Register Definitions . . . . .	141
B.2.1	mi_regs.h . . . . .	141
B.3	FX2 Firmware . . . . .	142
B.3.1	Makefile . . . . .	142
B.3.2	fx2cam_common.h . . . . .	143
B.3.3	fx2cam_ids.h . . . . .	144
B.3.4	fx2cam_i2c_addr.h . . . . .	145
B.3.5	fx2cam_usb.h . . . . .	145
B.3.6	_startup.a51 . . . . .	146
B.3.7	usb_descriptors.a51 . . . . .	147
B.3.8	vectors.a51 . . . . .	151
B.3.9	eprom_regs.h . . . . .	151
B.3.10	fx2cam_common.c . . . . .	151
B.3.11	fx2cam_main.c . . . . .	153
<b>C</b>	<b>USB2.0 Camera Linux Driver</b>	<b>164</b>
C.1	IMGUSB Fast USB Class . . . . .	164
C.1.1	imgusb.h . . . . .	164
C.1.2	imgusb.cc . . . . .	165
C.2	PM_CAM USB Primitives . . . . .	167
C.2.1	pm_prims.h . . . . .	167
C.2.2	pm_prims.cc . . . . .	168
C.3	PM_CAM USB2.0 Camera Driver Class . . . . .	170
C.3.1	pm_cam.h . . . . .	170
C.3.2	pm_cam.cc . . . . .	172
<b>D</b>	<b>System Control Board Firmware</b>	<b>185</b>
D.1	common.h . . . . .	185
D.2	job_ids.h . . . . .	190
D.3	main.c . . . . .	190
D.4	i2c_slave.h . . . . .	220
D.5	i2c_slave.c . . . . .	221
D.6	i2c_commands.h . . . . .	223
D.7	i2c_2.h . . . . .	223

---

---

D.8	i2c_2.c . . . . .	223
D.9	i2c_io_exp.h . . . . .	227
D.10	i2c_io_exp.c . . . . .	227
D.11	lcd_i2c.h . . . . .	228
D.12	lcd_i2c.c . . . . .	229
D.13	uart_commands.h . . . . .	233
D.14	uart2.h . . . . .	234
D.15	uart2.c . . . . .	234
<b>E</b>	<b>Host PC Software</b>	<b>237</b>
E.1	inspect . . . . .	237
E.1.1	Makefile . . . . .	237
E.1.2	inspect.h . . . . .	238
E.1.3	inspect.cc . . . . .	239
E.1.4	inspect.conf . . . . .	261
E.2	test_ip . . . . .	261
E.2.1	test_ip.cc . . . . .	261
E.3	cam_init . . . . .	263
E.3.1	Makefile . . . . .	263
E.3.2	cam_init.cc . . . . .	263
E.4	fpga_loader_ss . . . . .	271
E.4.1	Makefile . . . . .	271
E.4.2	fpga_loader_ss.cc . . . . .	271
E.5	pyWindowConfig . . . . .	279
E.5.1	pyWindowConfig.py . . . . .	279
E.6	pyCamCal . . . . .	285
E.6.1	pyCamCal.py . . . . .	285
E.7	Human Machine Interface (w32) . . . . .	290
E.7.1	frmMain.frm . . . . .	290
E.7.2	modGUIConsts.bas . . . . .	312
E.7.3	modCSVParser.bas . . . . .	312
E.7.4	modUARTCommands.bas . . . . .	313
<b>VITA AUCTORIS</b>		<b>315</b>

---

# List of Figures

2.1	OptiSorter . . . . .	6
2.2	OptiSorter Quadrant Birds-Eye View . . . . .	7
2.3	USB Bus Topology . . . . .	12
2.4	USB Standard Connectors . . . . .	17
2.5	I <sup>2</sup> C Sample Schematic . . . . .	20
2.6	I <sup>2</sup> C Timing Diagram . . . . .	21
3.1	Conceptual block-diagram of the PharmaSorter . . . . .	25
4.1	Micron Evaluation Board Block Diagram . . . . .	33
4.2	USB2.0 Camera High Level Block Diagram . . . . .	35
5.1	System Controller Block Diagram . . . . .	49
5.2	Typical Optocoupler Circuit Symbol . . . . .	50
5.3	Common Isolation Circuit . . . . .	50
5.4	Isolation Driving Circuit - Sinking . . . . .	52
5.5	Isolation Driving Circuit - Sourcing . . . . .	52
5.6	Counter Electromotive Force Protection Circuit . . . . .	55
5.7	Pneumatic Valve Control Circuit . . . . .	55
5.8	Motor Control Circuit . . . . .	56
5.9	LED Lighting Control Circuit . . . . .	58
5.10	Proximity Sensor Input Circuit . . . . .	58
5.11	Camera Trigger Driver Circuit . . . . .	59
5.12	I <sup>2</sup> C Buffered Expansion Circuit . . . . .	60
5.13	I <sup>2</sup> C Buffered Expansion Interrupt Circuit . . . . .	60
5.14	Inspection PC Soft Power Circuit . . . . .	61
5.15	Inspection PC Power Sense Circuit . . . . .	62

---

5.16 I <sup>2</sup> C Bus Switch Circuit . . . . .	63
5.17 E-Stop Input Circuit . . . . .	64
5.18 RS-232 Transceiver Circuit . . . . .	65
5.19 Populated System Control Board PCB . . . . .	68
5.20 I <sup>2</sup> C I/O Expansion Board Input Circuit . . . . .	69
5.21 I <sup>2</sup> C I/O Expansion Board Output Circuit . . . . .	69
5.22 I <sup>2</sup> C I/O Expansion Board I <sup>2</sup> C Address Select Circuit . . . . .	70
5.23 I <sup>2</sup> C I/O Expansion Board Input Circuit . . . . .	71
5.24 I <sup>2</sup> C I/O Expansion Board . . . . .	72
5.25 Motor Pulse Control flow diagram . . . . .	75
5.26 Proximity Sensor Interrupt flow diagram . . . . .	76
5.27 Flow Diagram of Bus Switch Interrupt Service Routine . . . . .	82
6.1 PC Mounting Scheme . . . . .	94
6.2 Inspect Software Scheduling Scheme . . . . .	95
6.3 Inspect Main Flow Diagram . . . . .	97
6.4 Inspect Image Acquisition Flow Diagram . . . . .	98
6.5 Inspect Image Processing Flow Diagram . . . . .	100
6.6 Inspect HTML Output File . . . . .	101
6.7 pyWindowConfig Screen Shot . . . . .	104
6.8 pyCamCal USB Device Selection Screen Shot . . . . .	104
6.9 pyCamCal Main Window Screen Shot . . . . .	105
6.10 W32 Control Panel Application Screen Shot . . . . .	106
7.1 Panel Layout (Front) . . . . .	110
7.2 Panel Layout (Rear) . . . . .	110
7.3 High-Level Wiring Diagram . . . . .	112
8.1 Camera Views of Bubble Defect . . . . .	115
8.2 System Control Board Soft PC Power Fix . . . . .	118

---

# List of Tables

2.1	USB Control Request Setup Packet . . . . .	13
2.2	USB Control Request bmRequestType BitMap . . . . .	13
2.3	USB Control Requests . . . . .	14
2.4	USB Device Classes . . . . .	15
2.5	USB pin out . . . . .	17
2.6	Frequently Uses libUSB Functions . . . . .	19
3.1	Digital Interface Comparison . . . . .	26
4.1	Cypress C0 Load - Descriptor Values Only . . . . .	36
4.2	Cypress C2 Load - Descriptor Values and Firmware . . . . .	36
4.3	USB2.0 Camera EEPROM Memory Map . . . . .	41
4.4	pm_usb Class Functions . . . . .	44
4.5	pm_prims Primitive USB Functions . . . . .	45
4.6	imgUSB Class Functions . . . . .	47
5.1	System Control Board MCU Requirements . . . . .	53
5.2	dsPIC I <sup>2</sup> C Master Functions . . . . .	79
5.3	UART Command Format . . . . .	88
5.4	UART Commands . . . . .	89
6.1	Host PC Hardware . . . . .	93
6.2	USB2.0 Camera Position Identifiers . . . . .	103
6.3	Host PC IP Addresses . . . . .	105

---

## *List of Abbreviations*

---

CCD	Charge Coupled Device.
CMOS	Complimentary Metal Oxide Semiconductor.
DID	Device ID.
FOV	Field of View.
FPGA	Field Programmable Gate Array.
GNU	GNU's Not Linux.
GPL	General Public License.
I <sup>2</sup> C	Inter-Integrated Circuit.
LCD	Liquid Crystal Display.
LED	Light Emitting Diode.
Mbps	Mega-bits Per Second.
MCU	Microcontroller.
OS	Operating System.
PAL	Phase Alternating Line.
PC	Personal Computer.
PCB	Printed Circuit Board.
PID	Product ID.
URB	USB Request Block.
USB	Universal Serial Bus.
VID	Vendor ID.

---

# Chapter 1

---

## *Introduction*

---

The manufacturing of two-part gelatin capsules requires a highly controlled process to ensure the resulting product is of optimal quality. The customer expectation is that the product is free of functional as well as of aesthetic defects. It is not possible however, to have absolute control over every aspect of the manufacturing process and as a result, defects are inevitable. To ensure the end-product is free of flaws, some means of quality control are required. Although, one of the most influential facets for quality control is cost. The profit made from the sale of an individual capsule is very small and thus the cost and time devoted to the inspection of each individual capsule should reflect this.

### 1.1 Current Quality Control Methods

Pharmaphil Inc. currently uses manual inspection to ensure the quality of their product. This involves a worker monitoring a conveyor belt of capsules and removing any defective product. Human inspection provides an immediate solution for quality control as humans can be trained to look for a set of various flaws in products deeming them unacceptable. A human can also quickly adapt to changing environments and different products. However, manual inspection has several drawbacks: First, human inspection is not consistent. The inevitability of human error eliminates the certainty of an ideal inspection. Also, it is certainly not reasonable to expect that congruency exists between workers. Second, the attention span of human workers is limited, making it impossible for a worker to provide accurate results for the entire duration of a given shift. Although a human worker may have a low initial cost and produce reasonable results, human inspection is ultimately exceedingly

more expensive than a high-quality, reliable automated inspection system.

## 1.2 Automated Vision Solution

In contrast, automated vision systems require significant start-up costs and development time. A vision system needs to be planned out, built and tested before it can be used in a quality control situation with confidence. Once built and verified, however, an automated vision system can provide consistent, objective inspection. It can also run indefinitely without breaks, aside from regular scheduled maintenance or downtime for repairs. However, a vision system does require a consistent environment to perform inspection, unlike humans that can easily adapt to changes in lighting or product. Even slight environmental changes can be detrimental in the performance of an automated vision system. For the inspection of two-part gelatin capsules, which are produced in various sizes and colours, specific inspection algorithms and parameters must be tailored to each specific capsule type. Although, once developed, an automated inspection system is a much more cost effective and accurate means of quality control than manual inspecting.

The cost devoted to the inspection of each individual capsule is very small and this must be considered for a machine vision system to replace a human worker, this must be considered. Developing nations clearly have an advantage in ensuring their product is of high quality since the cost of manual inspection is a very small component of the overall cost of the product. Thus for North America to compete, where labour costs are substantially higher, an automated inspection system is a practical solution to quality control. Automated inspection systems exist for the inspection of two-part gelatin capsules, however the expense of such a unit cannot always be warranted.

## 1.3 Typical Manufacturing Defects

Inconsistencies in the manufacturing of two-part gelatin capsules and equipment problems result in several common flaws. This includes dents, bubbles, holes, strings, cracks, dirt, double caps and incorrect colour or size. Some defects are classified as more severe over others. For example, a hole or a crack is a functional flaw as once the capsule is filled, it is possible for product to spill out. This is a critical defect for if received by an end-user, but also can cause machine failure at the filling stage. Other flaws such as bubble do not pose immediate functional failure, although the region of a bubble is a much thinner than that of the rest capsule and can easily turn into a hole. Dents and double caps can cause failure at the filling stage since they have an irregular shape. Incorrect colour or sized capsules can cause serious problems if they reach an end-user, resulting in possible product recall. Dirt, grease and strings are cosmetic flaws and do not affect the capsule, however imply poor quality control or manufacturing.



## 1.4 State of the Art

A machine vision system requires a single setup cost as well as possible maintenance costs. Typical systems available on the market cost in the range of \$500,000 [1]. Two big names in the hard capsule inspection business are Daiichi Jitsugyo Viswill Co. and Eisai Machinery USA Inc. Each of these companies offer inspection systems for two-part hard gelatin capsules, along with other pharmaceutical products. The Daiichi Jitsugyo Viswill CVIS-SXX-E system is capable of inspecting number 1 through 5 sized uni-colour and bi-colour hard capsules, excluding transparent and dark capsules [5]. Their system performs an inspection on the circumference surface of the cap and body with a inspection capacity of 1700 to 2500 capsules per minute [5]. The CVIS-SXX-E uses high-resolution CCD line sensor cameras and is capable of detecting a minimum size flaws of 100 $\mu$ m [5]. Eisai Machinery offers three models of capsule inspection systems, rated at 800, 1600 and 2500 capsules per minute. The Eisai system uses CCD cameras for inspection, but the capsule sizes and colours supported are not mentioned. Therefore, while the current current state-of-art capsule inspection systems do exist they are quite expensive.

## 1.5 Motivation

The overall goal of this project is to design and develop a working prototype for a full custom capsule inspection system. The system developed must be capable of a target inspection rate of 1000 capsules per minute. For the prototype system, the focus was on only the inspection of size-0 natural two-part hard gelatin capsules, which is the most popular capsule sold by Pharmaphil Inc. After the completion of the proof-of-concept stage, sound evidence that such a system is plausible permitting further development towards a commercial system. Thus the objective of the project was to complete the proof-of-concept stage with a working prototype.

## 1.6 Design Strategy

Pharmaphil Inc. purchased several antiquated inspection systems developed in the mid 1990s with the intent of upgrading them with modern technology. The existing system, called the OptiSorter was developed by an unknown German company. The OptiSorter provides a decent mechanical basis for inspection but with outdated electronics and processing capabilities. Using this existing mechanical frame and some existing hardware such as motors, actuators and sensors, the electronics and processing equipment could be updated to provide a more thorough and accurate inspection. In collaboration with Pharmaphil Inc., the department of electrical and computer engineering compiled a team of three graduate students to achieve this goal. This thesis refers to the

project as the PharmaSorter, although a formal name for the system has not been decided upon by Pharmaphil.

The approach taken in solving this problem was to develop a full-custom design in order to reduce cost. By recycling as many components of the OptiSorter as possible, the end cost of the system was reduced significantly. The full-custom design involved the design of custom USB2.0 cameras with the development of the associated drivers and software and the design of a system controller to interface with the mechanics of the system including motor, actuators and sensors. Custom software running on standard desktop PCs was developed to facilitate analysis of images acquired. The full custom approach deemed successful in reducing cost and the result was a proof-of-concept prototype from which the first generation commercial unit can be designed.

The research project involved the collaboration of three students in the Department of Electrical and Computer Engineering at the University of Windsor. The development of the prototype system was divided into three sections, namely: machine control and data acquisition, camera hardware and real-time image processing, and image analysis using image processing. The research team was comprised of myself, responsible for machine control and data acquisition, Anthony C. Karloff, responsible for camera hardware design and real-time image processing and Mohammed J. Islam who developed the image analysis algorithms using image processing.

This thesis outlines the personal contributions made to the project. This includes the detailed design of various components of the system including detailed hardware design, firmware and software development. Supplemental documents including technical reference manuals of various hardware are companion to this thesis and are referenced accordingly. Source code listings for the firmware and software developed for the prototype system are included in Appendix Chapters B through E.

This thesis is organized into seven chapters. The second chapter provides essential background information regarding the various technologies used in the project. This is not intended to be a comprehensive discussion of these technologies, but rather an introduction. The third through seventh chapters cover detailed design methodology of the various components of the system. Chapter 3 provides the basic design approach and considerations for the system. Chapter 4 details the design of the USB2.0 cameras in particularly the contributions made in terms of firmware and software drivers. In Chapter 5, a detailed design specification of the system control board outlines the hardware design, circuit board layout and fabrication and firmware development. The host PC hardware requirements, operating system selection and software development is discussed in Chapter 6. Chapter 7 provides a reference to how each component of the system interacts and the required interconnections of the various hardware components in a high-level manner. The final chapter, Chapter 8 provides some concluding remarks of the project including results and recommendations for future development towards a commercial unit.

---

## Chapter 2

---

### *Background*

---

This chapter is intended to provide the reader with some background information on some of the technologies used in the development of the system. The development of the PharmaSorter prototype was accelerated by reusing an the mechanics of an existing capsule inspection system called the OptiSorter. Background information regarding the OptiSorter is presented in this section including the capsule loading and ejection mechanism, and electrical hardware that was reused in the prototype design. For the prototype system, custom USB2.0 cameras were developed and custom firmware and drivers were created. USB2.0 is a fairly involved technology although many semiconductor vendors offer devices to reduce development time and complexity. The basics of USB2.0 are presented here. Although I<sup>2</sup>C is not a complicated communication interface, it does deserve some recognition due to the abundant use in the prototype system. I<sup>2</sup>C is a two-wire serial communication interface intended for chip-to-chip communication.

### 2.1 OptiSorter

The inspection of any object using a computer vision requires the development of a mechanical system to load, fixture and eject the object in addition to the electrical hardware and software. For the development of the PharmaSorter, an existing mechanical structure was used to facilitate this requirement. The OptiSorter, shown in Figure 2.1, is a machine vision system developed in the early-to-mid nineties for purpose of inspecting two-part pharmaceutical capsules.

This system facilitates individual capsule loading and ejection. The capsule rests in a holder and is inspected using a series of four cameras which acquire images of all surfaces of the capsule. Pneumatic

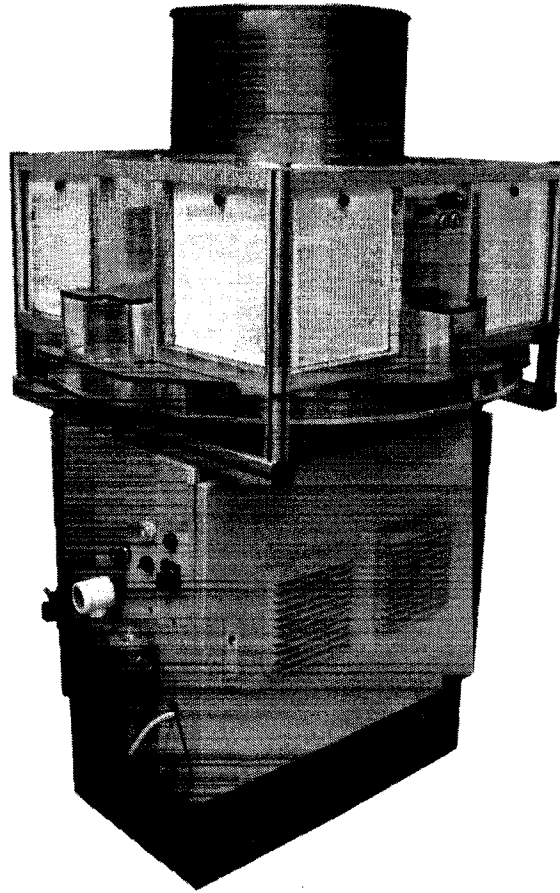


Figure 2.1: OptiSorter

valves are used to eject the capsule into either a pass or fail discharge mechanisms. The OptiSorter is comprised of four identical inspection quadrants operating in parallel. The components of a quadrant are labelled in Figure 2.2 accompanied by a detailed description of the OptiSorter's components including the electrical and mechanical systems.

1. **Rotating Disc** The rotating disc is the base of the loading hopper with slots to allow capsules to be queued in each arm (4) which are fixed to the disc.
2. **Capsule Intake Lever** The capsule intake lever controls the flow of capsules entering the holder (5). Two holders are used to allow a single capsule to be loaded onto the holder at a given time.
3. **Lifters** The lifters are fixed metal blocks used to lift the intake levers (2) to allow a capsule to be loaded in the *Loading Stage*. The lifters are positioned such that one capsule is loaded onto

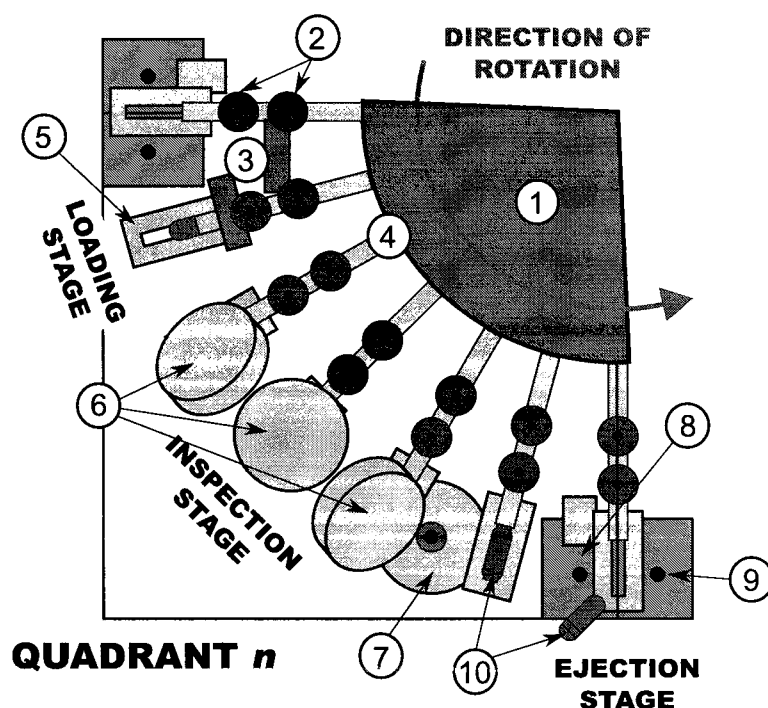


Figure 2.2: OptiSorter Quadrant Birds-Eye View

the holder and another held in the waiting position while the rest are queued single file in the arm (4).

4. **Arm** The arms are hollow metal shafts that hold the capsules in the inspection queue. The arms are the link between the hopper and the holder (5) and rotate to move the capsule through the various stages of the inspection process.
5. **Capsule Holder** The capsule holder is a machined steel block with a groove to hold the capsule and a slot allowing for the bottom portion of the capsule to be visible. The capsule rests on the holder while it passes through the various stages.
6. **Top View Cameras** The top view cameras, comprised of a left, center and right camera, acquire images of the top surface of the capsule. The cameras are triggered when the holder enters their field of view.
7. **Bottom View Camera** A single camera is positioned to acquire an image of the bottom of the capsule through the slot in the holder (5). This image is acquired after the top view cameras (6) acquire images of the top surface of the capsule.

8. **Accept Ejection** An electrically controlled pneumatic valve is used to eject a passed capsule using a burst of air.
9. **Reject Ejection** A constant supply of air is used to reject all capsules *not* passed (assumed fail).
10. **Two-Part Capsule** The capsule is the object being inspected. A given machine can only support a single size capsule but by modifying inspection algorithms and machine parameters, various different coloured capsules can be inspected.

The OptiSorter provides a suitable base for performing inspection with the ability to replace the electronics and incorporate more advanced inspection methods. The existing mechanical structure is suitable for the throughput requirement and provides an adequate inspection environment. By re-using the OptiSorter, the upgraded system is cost effective and provides a reasonable inspection.

### 2.1.1 Mechanical Design

#### Capsule Loading

The OptiSorter has a relatively simple, yet elegant mechanical design. The mechanics of the system rely on a single stepper motor which rotates a series of 24 arms which load and fixture the capsule for inspection. Each mechanical arm is hollow with a diameter slightly larger than a capsule. The arm queues capsules in single-file from a large hopper atop the machine. As each arm rotates, two levers are used to load an individual capsule into the capsule holder and the waiting position. The levers are activated by lifters located prior to the inspection stage. As the arm enters an inspection area, the first lever is lifted to allow a single capsule to be queued between the two levers in the waiting position. After a short distance, the second lever is lifted to allow the queued capsule to rest in the holder. The dual plunger system is in place to ensure only a single capsule is loaded at a given time.

#### Capsule Holder

As mentioned above, the capsules rest in a capsule holder. The capsule holder is a piece of machined stainless steel with a groove in which the capsule rests. A slot in the holder allows the capsule to be viewed from all angles by cameras, thereby maximizing the viewable surface area. A proximity sensor is used to trigger the cameras for image acquisition once the capsule enters the inspection stage. The rotating arms are attached to a steel disc with notches machined out. Each notch represents the entry of a new capsule into the inspection area. This signal is used as a feedback mechanism

so the position of the arm is always known. This is essential to properly organize the sequence of events required for inspection.

### **Capsule Ejection**

After the capsule passes the inspection stage it enters the ejection stage where the capsule is ejected into one of two ejection chutes. The ejection of a passed capsule is performed by an electrically controlled pneumatic valve that uses a burst of air to eject a capsule. For a rejected capsule, a constant stream of air is used to eject the capsule into the reject chute. Following ejection, the capsules rest on a metal disc that rotates at a reduced speed proportional to the speed of the arms. The capsules collect on the disc in one of two slots, accepted or rejected. As the disc rotates, the capsules are finally discharged at one of the two exits.

### **2.1.2 Electrical Design**

The OptiSorter's electrical system is quite dated by today's standards, however some components of the existing electrical system can be reused in the re-design. The OptiSorter was mostly comprised of custom electronics including cameras, acquisition boards and an input/output board as well as sensors, actuators and a stepper motor. There were several power supplies used in the system to provide the power requirements of different components. Finally, a simple HMI was designed using several push-buttons and a character LCD.

#### **Power Supplies**

To adequately meet the requirements of the system, several power supplies were used. A 24VDC supply was used to operate the motor, the electrically controlled pneumatic valves and proximity sensor. A set of standard PC power supplies were used to provide power for the acquisition boards and cameras. A set of custom designed 8VDC and +/- 15VDC were also used in the system. It is presumed that these were used for an analog portion of the image acquisition system.

#### **Illumination**

A matrix of LEDs were used to provide backlighting illumination. The LEDs were soldered to a small circuit board and attached to a diffusing plastic block. The voltage used to operate them is not known, but assumed to be 8-12VDC. The backlighting was created using high-intensity red LEDs. It is assumed that these were pulsed to preserve the LEDs lifespan and the provided exposure control for the image sensor.

### **Cameras and Acquisition**

The OptiSorter is equipped with 16 cameras, four for each of the inspection quadrants. The cameras, presumably black-and-white, have a custom circuit board with a custom enclosure and a commercial lens. The wiring of the cameras infers that the cameras produce an analog signal, most probably PAL. For each camera, an acquisition and processor board is used to acquire the images and perform inspection. Each of the acquisition boards is equipped with an equivalent 80286 processor and other signal processing semiconductor devices. Each acquisition board is connected to a large bus where the input/output board also resides. It is presumed that following an inspection, the pass/fail signal is sent to the input/output board which signals the pneumatic valves to accept or reject a given capsule.

### **Stepper Motor and Controller**

The OptiSorter uses a five-pole stepper motor to rotate the capsule holder arms. The stepper motor is controlled by a motor controller. The motor controller accepts a direction signal and steps on the rising edge of the input step pulse. This makes motor control rather straightforward and requires little feedback to track the position of the arms. The motor controller operates at 24VDC and requires a 24V pulse to step.

### **Proximity Sensor**

A single proximity sensor is used as feedback of the position of the holder arms. When the holder enters the inspection area, a pulse is seen from the proximity sensor. This, along with the knowledge of the number of pulses applied to the stepper motor, is used to track the position of the holders.

### **Electrically Controlled Pneumatic Valves**

Electrically controlled pneumatic valves are used to eject the accepted capsules. The valve is engaged (opened) by a 24VDC supply voltage. A single valve exists in each inspection quadrant and is opened according to the pass/fail result of the inspection by the input/output board.

### **Human Machine Interface**

An very simple HMI used to be used by a machine operator is located at the front of the machine. It is assumed that this interface is used to start and stop the machine and retrieve simple feedback from the system. The HMI is comprised of a set of two push buttons, a switch, a lamp and a 16x2 character LCD. It is presumed that the HMI is controlled by the I/O board using a parallel interface.



## Monitor

A television unit placed atop the machine is presumed to be used for system calibration. A series of switches used to select a camera exists on the side of the machine. A particular camera can be selected and viewed on the television unit. It is assumed that a PAL signal directly from each camera is displayed and used for the calibration of lens focus, camera position, etc.

## 2.2 Universal Serial Bus (USB)

Universal Serial Bus (USB) is a hi-speed interface used for connecting peripheral devices to a computer, or any host system. USB is abundantly available and most modern PCs are equipped to support it. USB devices are also widely available in commercial electronics and commonly used in industry. It is a preferred solution due to its hot-swap capability, supply capabilities and high speed transfer ability. Developing a USB device requires a working knowledge of the physical, protocol and software levels of the interface.

USB replaces legacy interfaces including serial and parallel ports on PCs. Standard devices like printers, scanners, mice and keyboards now connect using USB. Among these device, many other devices are being supported such as PDAs, mobile phones and other hand-held devices. USB is a serial bus interface first introduced in November 1995 with the initial release USB 1.0 by Microsoft, Intel, Philips and US Robotics [39]. USB 1.1 was later released in 1998 to rectify the adoption problem from the initial release [39]. The most current release of USB is version 2.0, which was led by Hewlett-Packard, Lucent, Microsoft, NEC and Philips [36].

### 2.2.1 Overview

The Universal Serial Bus is a star topology with a single master on the bus, the host. A device on a USB bus cannot initiate a transfer to a host; it must be requested by the host prior to transmission. USB supports the addition of HUBs to the topology to support additional devices. Up to five tiers of HUBs are permitted supporting, up to 127 devices on a single bus. There is a single root hub in any given topology which is integrated directly into the USB host controller device of the host system. A example USB bus topology is shown in Figure 2.3.

Each USB device, or peripheral, is typically comprised of one or more logical sub-devices that perform a single function. Each of the logical sub-devices is referred to as an *interface*. Communication with an interface is performed through *pipes* which are channels that link to the *endpoints* of a given interface. Each interface can be made up of one or more endpoints that are used to transfer data. Each endpoint is unidirectional, meaning it can only be used to send or receive data. USB supports up to 32 endpoints, 16 IN endpoints and 16 OUT endpoints, with IN and OUT referring

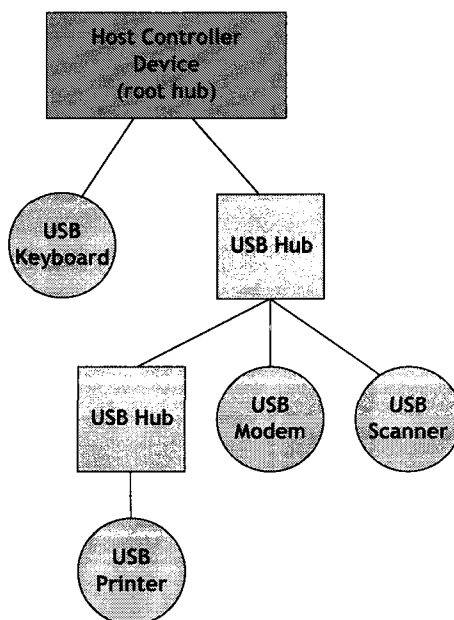


Figure 2.3: USB Bus Topology

to the direction of data flow with respect to the host. There is an exception however, with endpoint zero (EP0), which is a required endpoint used for device configuration. Endpoint zero is a bi-directional control endpoint required in all devices. This is used for device configuration during the enumeration process and is also often used for vendor specific requests.

### 2.2.2 Enumeration

Enumeration is the process of device identification to the host system upon being attached. The enumeration process begins with the host initiating a device reset followed by a request for device configuration information (USB standard device requests). If the device is recognized by the host operating system, the corresponding device driver is loaded. Device identification is typically based on three device descriptor values namely VendorID, ProductID and DeviceID. Since USB devices cannot initiate communication, the host controller polls bus traffic in a round-robin fashion to detect devices added to the bus. Each USB hub has a status bit to report the attachment or removal of a device. In the case of a device being attached, the host enables the port and assigns the device an address on the bus. The device is enumerated using a pipe to the device's control endpoint zero where the device and endpoint descriptors are requested by the host.

## Endpoint Zero

It is essential that all devices support a common mechanism for accessing device information. This is accomplished using the default control endpoint which exists on all devices. This endpoint is bi-directional (IN and OUT) with endpoint address 0. During enumeration, this endpoint is queried for standard device information including vendor and product identification, device class and power requirements. This is accomplished by standard descriptor requests from a device, it's interfaces and the endpoints of each interface. This is performed using the default control pipe. A control request is formatted as described in Table 2.1 where the *bmRequestType* bitmap is detailed in Table 2.2. The specific configuration data is acquired through a series of requests as described in Table 2.3.

Offset	Field	Size	Value	Description
0	bmRequestType	1	BitMap	Description of Request (Type, Direction & Recipient)
1	bRequest	1	Value	Specific Request
2	wValue	2	Value	Specific to Request
4	wIndex	2	Index	Used by request to pass and index
6	wLength	2	Count	Number of bytes to transfer in data stage (if present)

Table 2.1: USB Control Request Setup Packet

Bit(s)	Description
D7	Data Transfer Direction 0 - Host to Device 1 - Device to Host
D6..5	Type 0 - Standard 1 - Class 2 - Vendor 3 - Reserved
D4..0	Recipient 0 - Device 1 - Interface 2 - Endpoint 3 - Other 4..31 - Reserved

Table 2.2: USB Control Request bmRequestType BitMap

### 2.2.3 Host Controller Interface

The host system uses a host controller to facilitate the USB protocol at the physical level. A USB host controller device has several standards for the releases of USB. The UHCI and OHCI (Universal Host Controller Interface and Open Host Controller Interface) were both released along with USB

<b>bmRequestType</b>	<b>bRequest</b>	<b>wValue</b>	<b>wIndex</b>	<b>wLength</b>	<b>Data</b>
0x00, 0x01, 0x02	CLEAR_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None
0x80	GET_CONFIGURATION	Zero	Zero	One	Configuration Value
0x80	GET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
0x81	GET_INTERFACE	Zero	Interface	One	Alternate Interface
0x80, 0x81, 0x82	GET_STATUS	Zero	Zero Interface Endpoint	Two	Device, Interface, or Endpoint Status
0x00	SET_ADDRESS	Device Address	Zero	Zero	None
0x00	SET_CONFIGURATION	Configuration Value	Zero	Zero	None
0x00	SET_DESCRIPTOR	Descriptor Type and Descriptor Index	Zero or Language ID	Descriptor Length	Descriptor
0x00, 0x01, 0x02	SET_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None
0x01	SET_INTERFACE	Alternate Setting	Interface	Zero	None
0x82	SYNCH_FRAME	Zero	Endpoint	Two	Frame Number

Table 2.3: USB Control Requests

1.1. The OHCI standard is more hardware dependant and faster than the more software dependant UHCI. OHCI however was more expensive to implement. Regardless of the standard followed, the same functionality was supported by both interfaces. To reduce the complexity, the release of USB 2.0 permitted only a single interface, EHCI (Enhanced Host Controller Interface). Host controller devices were required to be backwards compatible and support one of either UHCI or OHCI. The host operating system uses the host controller interface to communicate directly with the host controller device using a register file. The host controller device follows the USB protocol to communicate with peripheral devices.

### 2.2.4 Device Classes

In an effort to reduce the dependence of devices on individually specific drivers, USB device classes alleviate this by creating standard classes to which a device can belong. By belonging to a given device class, the device must follow the standards specified by said class. A device class can be supported by a single device driver without the need for a fully custom driver for each specific device. Thus the same driver can be used for many unique devices. Commonly known device classes include HID (Human Interface Device) and Mass Storage device used for keyboards and flash disks respectively. A more thorough list of device classes and examples are listed in Table 2.4.

Base Class	Descriptor Usage	Description
0x00	Device	Indicate that interface descriptors are to be used for the device.
0x01	Interface	Used for audio compatible devices. This can be used in the audio interface of a webcam device.
0x02	Both	Communication and CDC Control device used in devices such as ethernet adaptors, modems, etc.
0x03	Interface	Human Interface Device. This device class is reserved for devices such as mice and keyboards.
0x05	Interface	Physical device class. This class is used in force feedback joysticks.
0x06	Interface	Still Imaging device class. Sometimes used for digital cameras, but most use Mass Storage Device.
0x07	Interface	Printer device class is used for printers and multi-function printer/scanners.
0x08	Interface	Mass Storage Device is used for USB flash drives, external DVD Drives, etc.
0x09	Device	USB Hub class is used for full and high speed USB hubs.
0x0A	Interface	CDC-Data class is used with the communication device class
0x0B	Interface	Smart Card Class is reserved for USB smart card readers
0x0D	Interface	Content Security interface class
0x0E	Interface	Video interface class - used for video devices like web cams
0x0F	Interface	Personal Health care class
0xDC	Both	Diagnostic Device - used in USB 2.0 Compliance testing apparatus
0xE0	Interface	Wireless controller - used for WiFi and Bluetooth adapters
0xEF	Both	Miscellaneous - Such as Active Sync devices or Palm Sync devices.
0xFE	Interface	Application Specific - use for Device Firmware Upgrade, IRDA bridge device, etc.
0xFF	Both	Vendor Specific

Table 2.4: USB Device Classes

### 2.2.5 Data Flow Types

USB supports four distinct data transfer types: control, bulk, interrupt and isochronous. Each transfer type is suitable for a particular application. A USB transfer occurs between the device driver and an endpoint and thus each endpoint is defined as one of the four flow types in a single direction (IN or OUT). The transfer of data between the host software and the device endpoint occurs over a logical channel, often referred to as a pipe. An endpoint may be defined as one of the following data flow types.

#### **Control Transfer**

The control endpoint is used to configure the device during enumeration and also for controlling other device-specific functionality. Control endpoints are usually reserved for control and status operations where data is non-periodic. The maximum data size of a control endpoint is 64 bytes for USB 2.0 (high-speed) and full-speed devices, but only 8 bytes for low-speed devices. This transfer type is guaranteed and therefore occurs without loss of data.

#### **Bulk Transfers**

Bulk endpoints are used when large amounts of data are to be transferred, as for printers, scanners or external hard-disks. Bulk transfers are guaranteed by built-in error detection to ensure reliable data transfer. This is accomplished by using error detection via CRC and invoking hardware retries if required due to delivery failure or bus errors. A bulk transfer is guaranteed for integrity but not for transfer rate. The bandwidth utilization is dependent on the bus activity and will vary. For USB 2.0, the maximum bulk transfer size is 512 bytes and 64 bytes for full-speed endpoints.

#### **Interrupt Transfers**

An interrupt endpoints is designed for devices that do not frequently send or receive data. The interrupt endpoint supports bounded service periods meaning the service time is guaranteed. The maximum payload size of an interrupt endpoint is 1024 bytes for USB 2.0 and 64 bytes for full-speed.

#### **Isochronous Transfers**

Isochronous transfers are typically used for periodic and continuous data transfer between the host and the device. It is typically used when timing is critical but data integrity is not.

### 2.2.6 USB Connectors and Cabling

So far, the protocol level has been introduced. The USB specification [36] also contains the connector requirements, cabling, isolation, etc. This information is relevant to the overall layout and design of a system from a hardware design viewpoint and the logistics of the cabling. USB connectors are standard and well designed. They were specifically designed to permit easy insertion and removal with the inability to incorrectly attach. The connectors are robust unlike some predecessors with pins that can be bent or broken if not properly inserted. There are several standard types of USB connectors divided into two classes, A-type connectors and B-type connectors. A-type connectors are reserved for the host side and B-type connectors are reserved for peripherals. Figure 2.4 illustrates the common USB connectors used [39].

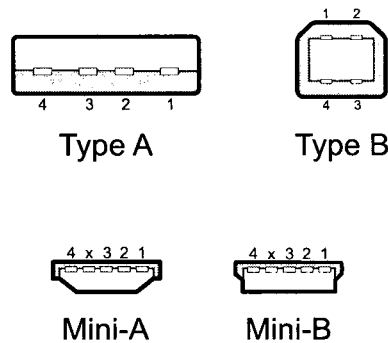


Figure 2.4: USB Standard Connectors

USB cabling is restricted to a maximum of 5 meters (16.4 feet) without the use of hubs. If hubs are used, the maximum distance is about 25 meters since the maximum cable length between hubs is 5 meters with a maximum of 5 tiers of hubs. If the USB cable is too long, the host will not receive the data packet on time and the command will be lost. The cable length is based on the round trip delay of an electrical signal in the length of copper wiring and is also dependant on the quality of the cable. A USB cable is comprised of four wires (sometimes five) with a ground shielding. The data lines are a twisted differential pair to reduce crosstalk. The wiring pin out of a USB cable is shown in Table 2.5.

Pin	Name	Colour	Description
1	$V_{BUS}$	Red	+5V Bus Power
2	D-	White	Differential Data Negative
3	D+	Green	Differential Data Positive
4	GND	Black	Ground

Table 2.5: USB pin out

### 2.2.7 Bus Power

The USB cable provides 5V power (with a 5% tolerance) to peripheral devices. When a device is initially connected, the host reserves 100mA supply which can be later increased after enumeration for a maximum current of 500mA. There is an exception in battery charging devices whereby the USB host can provide up to 1.5A worth of current. Devices that require more power must be externally powered.

### 2.2.8 USB Device Drivers

A device driver is software that is responsible for interfacing with hardware. It is operating system dependent and is designed for specific hardware. The driver is responsible for sending commands to hardware and the interchange of data.

For the Linux operating system, several types of drivers can be developed. A driver can be part of the kernel or a loadable kernel module. These types of drivers require an extensive knowledge of the operating system. However, a user-space driver can be developed to interface with a device that requires less understanding of the operating system kernel. A user-space driver can be safer in that a poorly written driver cannot affect the kernel as severely, or cause it to panic. LibUSB is a cross-platform library of user-space routines for controlling USB devices without the need for kernel drivers. LibUSB is currently available for Linux, MS-Windows, Mac-OSX, and FreeBSD.

#### libUSB

LibUSB is an open-source project licensed under the GNU GPL intended to provide a library for user level applications to access USB devices regardless of the operating system. LibUSB provides a relatively simple method of interacting with USB hardware. Routines were developed so a dedicated kernel driver is not required to communicate with a USB device. Common functions for scanning the USB bus, for retrieving descriptor data and for claiming interfaces are rather straight-forward using libUSB. Of the four data transfer types, control, bulk, interrupt and isochronous, the first three are supported by libUSB. Although not perfect for all scenarios, libUSB provides many useful functions for finding and controlling USB devices in user space. Some of the more commonly used functions are listed in Table 2.6.



libUSB Function	Description
usb_init()	Initialize the libUSB library for the application.
usb_find_busses()	Searches the system for all USB busses. Returns the number of changes since last call.
usb_find_devices()	Searches all busses to find all devices on each bus. Returns number of changes since last call.
usb_get_busses()	Returns the list of busses as type usb_bus.
usb_open(usb_device *dev)	Open a device for use. This must be called before any operations can be performed. Returns a handle to the device, or an error code.
usb_close(usb_dev_handle *dev)	Used to close an opened device. Returns 0 on success
usb_set_configuration(usb_dev_handle *dev, int configuration)	Sets active configuration of a device. This is specified in the descriptor field bConfigurationValue. Returns 0 on success.
usb_set_altinterface(usb_dev_handle *dev, int alternate)	Set the active setting of the current interface as described in descriptor field bAlternateSetting. Returns 0 on success.
usb_claim_interface(usb_dev_handle *dev, int interface)	Claims the interface with the operating system. The interface number is specified in the descriptor bInterfaceNumber. Returns 0 on success.
usb_release_interface(usb_dev_handle *dev, int interface)	Releases a previously claimed interface. Returns 0 on success.
usb_control_msg(usb_dev_handle *dev, int requesttype, int request, int value, int index, char *bytes, int size, int timeout)	This function performs a control request as defined in Table 2.1. Returns 0 on success, or a negative error code.
usb_bulk_write(usb_dev_handle *dev, int ep, char *bytes, int size, int timeout)	This function performs a bulk write to a bulk-OUT endpoint specified by <i>ep</i> . Returns number of bytes written on success, or a negative error code.
usb_bulk_read(usb_dev_handle *dev, int ep, char *bytes, int size, int timeout)	This function performs a bulk read from a bulk-IN endpoint specified by <i>ep</i> . Returns number of bytes read on success, or a negative error code.

Table 2.6: Frequently Uses libUSB Functions

## 2.3 I<sup>2</sup>C

I<sup>2</sup>C (Inter-Integrated Circuit) is a serial bus developed by Philips for connecting low-speed peripherals of an embedded system. The I<sup>2</sup>C bus is a bi-directional two-wire multi-master bus used for connecting microcontrollers or processors to devices or other processors.

The I<sup>2</sup>C bus uses two open-drain lines, a serial data line (SDA) and a serial clock line (SCL) thus each of these lines must be pulled high using pull-up resistors. Typically 5V or 3.3V voltage levels are used in a system, although higher and lower voltages can be used. A schematic of an

example configuration of a single master I<sup>2</sup>C bus with several slave devices is shown in Figure 2.5.

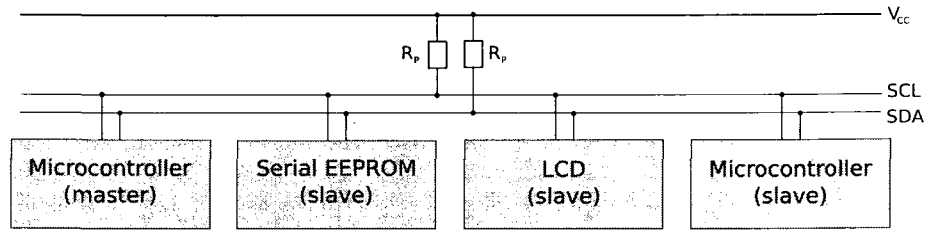


Figure 2.5: I<sup>2</sup>C Sample Schematic

The addressing scheme for I<sup>2</sup>C slave devices consists of a 7-bit address space with 16 reserved addresses resulting in a maximum of 112 nodes for a given bus. A typical slave device has fixed high address bits and several inputs to select the lower address bits (typically three bits). This allows the designer to use multiples of the same device on a single bus with distinct addresses.

The I<sup>2</sup>C bus most commonly operates at 100 kbit/s, known as standard mode. More recent revisions of the I<sup>2</sup>C specification include faster operating speeds including Fast Mode at 400 kbit/s, Fast Mode Plus at 1 Mbit/s and High Speed Mode at 3.4Mb/s. The High speed mode of operation typically requires current sources rather than simple pull-up resistors to permit faster rise times.

Since the lines of the I<sup>2</sup>C bus are driven by open-drain drivers, the pull-up resistors are needed to pull the line voltage to  $V_{CC}$ . If a device wishes to communicate on the bus, the line is pulled to ground to communicate a logical zero, and allowed to float to represent a logical one. The master(s) on the bus are responsible for providing a clock, however with clock-stretching, slaves are able to momentarily pause the transfer of information if required. Clock-stretching occurs when the slave holds the SCL line low for longer than the clocked frequency. The master must read the clock line after releasing it to ensure it has been pulled high, if not, the master must wait until the clock line is high before reading off the data line (SDA).

Since I<sup>2</sup>C supports multi-master busses, arbitration is required to ensure multiple masters do not attempt to use the bus at the same time. Arbitration in I<sup>2</sup>C is quite simple and does not give priority to a particular master. The process of arbitration is required if two masters start a transmission around the same time, each transmitter checks the level of the data line (SDA) and compares it to what is expected. If the value on the bus does not match, the particular transmitter loses arbitration and ceases transmission. If two masters are sending messages to two different slaves, the slave device with the lower slave address will win arbitration due to the nature of the bus.

In an I<sup>2</sup>C transaction, the master initiates transmission by sending a START bit, followed by the 7-bit address of the target slave device, followed by a read/write bit. The read/write bit indicates the direction of data flow with respect to the master: 0 for write and 1 for read. If the slave exists

on the bus, it will respond with an ACK (acknowledge) for the address. An acknowledge by a slave is initiated by pulling the SDA line low for the expected clock which is recognized by the master. The master will then continue in either transmit or receive mode. After the transmission, the master will send a STOP bit indicating the transaction is complete. A master will acknowledge when receiving data by holding the SDA line low after each byte received, except for the last, to indicate the expected amount of data has been received.

The START bit (S) is recognized by a high to low transition of SDA while SCL is high. A STOP bit (P) is recognized by a low to high transition of SDA while SCL is high. A typical I<sup>2</sup>C transaction timing diagram is shown in Figure 2.6.

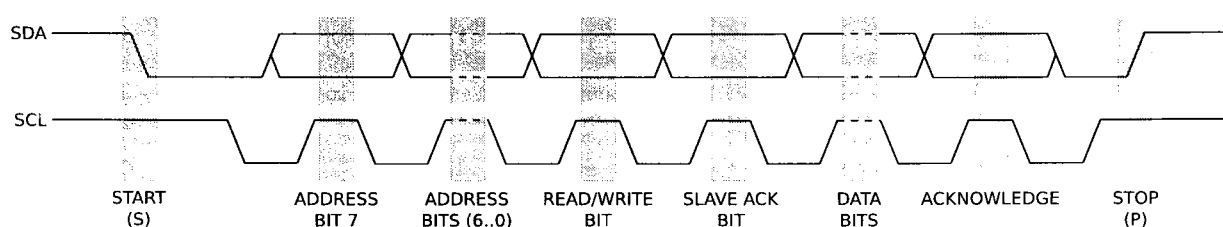


Figure 2.6: I<sup>2</sup>C Timing Diagram

I<sup>2</sup>C is commonly used in semiconductor devices where simplicity and cost are key factors. Since I<sup>2</sup>C is a two-wire interface, the pin cost is minimal, resulting in a simple and low cost solution. I<sup>2</sup>C is common found in serial EEPROMs, low speed ADC and DACs, LCD displays and real-time clock chips. Some manufacturers use what they refer to as a two-wire interface (TWI) that is an adaptation of I<sup>2</sup>C but not completely conforming to the I<sup>2</sup>C specification.

## 2.4 Machine Vision

Machine vision is the encapsulation of computer vision, image processing and machine control to typically perform inspection on manufactured goods in industry. A traditional machine vision system is comprised of cameras, computers and digital inputs and outputs or computer networks for machine control. Machine vision systems are intended to replace human inspectors working on assembly lines to automate the inspection process. Although human inspection is more flexible and adaptive than machine classification, human inspection is subject to inaccuracies due to human attention span and circumstance. Although the human eye is more versatile than any vision system as it can adapt to various illumination conditions, a machine vision system can replace the need for human inspection by producing high quality, objective inspection.

Since machine vision systems do not “see” the way humans do, images acquired can be analyzed to look for certain irregularities or characteristic of an object. This is typically accomplished by

analyzing individual pixels in regions of an image. This portion of machine vision falls under the image processing section. Image processing is a crucial portion of any machine vision system since it is essentially the brains of the analysis. Image processing used in machine vision will typically perform some pre-processing on an image including de-noising and image enhancement. Typically, the majority of the image processing during inspection is performed on binary images following the pre-processing. Some of the common processing methods include pixel counting, thresholding, segmentation, edge detection and template matching. However, it is entirely dependent on the application and will change depending on the object and the environment. An inspection system may check for surface flaws and take measurements to determine the quality of the part.

A typical machine vision system is comprised of one or more digital cameras, typically black and white although colour cameras are gaining popularity, along with the appropriate optics such as lenses and mounts. The majority of any inspection is typically most dependent on the grayscale information and thus a grayscale camera is typically preferred since it enables the highest resolution and faster speed. Some applications, however, require colour information. In this case, a colour sensor is required and the grayscale information is extracted from the RGB colour space. The system must also have an interface to digitize and acquire the image data, typically a frame-grabber or other computer interface such as USB. A processor is required to analyze the image(s) and is often a PC or embedded DSP processor. A device gaining popularity is a smart camera which incorporates all of the above mentioned components into a single camera. The smart camera has a lens, sensor, acquisition hardware and processor all within a single device. Input/Output hardware is required to interface with the machine portion of the system. Sometimes a communication link like an Ethernet or RS-232 connection is used rather than dedicated I/O to report inspection results.

Illumination is a crucial element of any machine vision system in order to produce a consistent environment that is conducive to high-quality inspection. Since the cameras are not as versatile as the human eye, various illumination techniques can be employed to enhance details of an object. High-intensity LEDs are commonly used for illumination, however fibre optic, laser, fluorescent and halogen lamps are also used. Many different lighting schemes exist that provide suitable lighting for different applications. The backlighting technique is commonly used for measuring objects. This method provides even illumination and casts silhouettes of object which are suitable for edge detection of solid or transparent objects [14]. Ringlights are another commonly used lighting source comprised of one or more rings of LEDs placed around the camera. The ring light provides uniform front light for matte surfaces. Many other illumination techniques exist and are suitable for different applications. One must consider the object being inspected, mechanical constraints and speed of inspection when selecting lighting since some lighting solutions require more space and produce varying light intensity.

In order to perform inspection, software must be developed to perform the image processing and yield a pass or fail result of the inspection. This is platform dependent and can be accomplished in many different ways. The machine must be equipped with sensors and actuators that when combined are used to locate parts, trigger image acquisition and sort accepted and rejected parts. The system must incorporate means of obtaining the result of inspection from the software in order to properly eject the object.

The design of a consistent and high-quality machine vision system requires a well planned approach. The design of each system is dependent on the part being analyzed and the rate at which the inspection must occur. The cost of the system is another component that must be considered. The value of the part being inspected and the throughput of the machine vision system must correlate to it's manufacturing and running cost.

---

## Chapter 3

---

### *Design Methodology*

---

The design of any system requires a substantial amount of planning and research before any type of development can begin. The design and development of the PharmaSorter began with a list of high-level requirements and an existing mechanical structure, namely the Optisorter. In order to meet the system requirements, many considerations had to be taken into account: the required inspection rate of the system, the inspection detail requirements (minimum flaw size), the cost constraints and the ability to maximize the reuse of existing hardware.

The final conceptual design of the system involved the use of USB 2.0 cameras and PCs to perform the capsule inspection with a system controller for the operation of the controls and mechanics of the OptiSorter. The overall block diagram of the proposed system is shown in Figure 3.1. From the block diagram, it is evident that there are three main components of the system. The USB 2.0 cameras which are used to acquire images, the inspection PCs for processing the images and a system controller to control the hardware and to synchronize timing. Chapter 4 through Chapter 6 discuss the design of each component in detail.

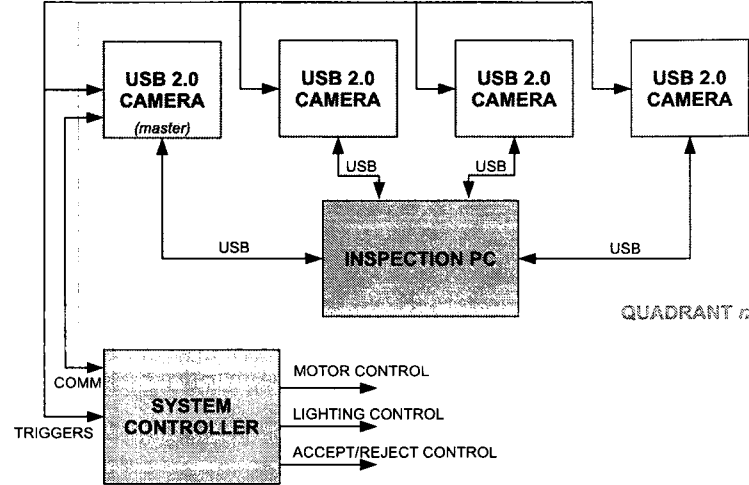


Figure 3.1: Conceptual block-diagram of the PharmaSorter

### 3.1 Design Approach

The design of the PharmaSorter involved a great deal of initial planning and research. This included assessing the requirements and constraints of the system, and creating a development strategy based on these design criterion. The design approach involved the development of the small components of the project and putting the pieces together until a quality finished product was achieved. This involved building and testing circuits at the breadboard level with the aid of development boards and writing and debugging firmware to verify that the selected devices were suitable for their purpose. This was accompanied by the development of drivers and software to test data rates and ensure the selected hardware was suitable for the application. Building small circuits and writing custom firmware and software was a practical way of testing and verifying designs. Although the initial concepts for each of the design stages were based on educated predictions, it was imperative that they were thoroughly tested and verified. As each component was finalized, they were incorporated into the final prototype.

Due to the requirements and constraints of the system, a fully custom design strategy was taken. This involved designing custom circuits, developing custom firmware and writing many software applications. This was essential to reduce the cost of the system, and reuse many of existing hardware components of the OptiSorter. By undertaking a fully custom approach, the cost of the prototype was minimized, although the design time increased substantially.

During every stage of design and development, many key considerations were constantly being taken into account. Focusing on these reduced design errors and improved flexibility.

## 3.2 Design Considerations

The primary goal of the design of the system was to upgrade the electronics of the OptiSorter, the driving force for this being the requirement of a low-cost solution. In addition to this constraint, the required system throughput of 1000 capsules per minute was important as was the ability to detect a certain degree of flaws. This means the camera must be able to obtain a minimum amount of detail in the capsule in a certain amount of time.

### 3.2.1 Data Transfer Medium

A low cost camera solution that met the desired system throughput was sought out. After comparing available technologies in transfer mediums, the USB2.0 interface was selected. A comparison of interfaces is shown in Table 3.1 [6].

	FireWire 1394.a	FireWire 1394.b	Camera Link	USB 2.0	Gigabit Ethernet
<b>Data Transfer Rate</b>	400 Mbps	800 Mbps	up to 3.6 Gbps	480 Mbps	1000 Mbps
<b>Max. Cable Length</b>	4.5m	up to 100m	10m	5m	100m
<b>Max. Devices</b>	up to 63	up to 63	1	up to 127	unlimited
<b>Connector</b>	6 pin	9 pin	26 pin	4 pin (USB)	RJ45 / CAT5e
<b>Capture Board</b>	Optional	Optional	Required	Optional	Not Required
<b>Cost</b>	Moderate	Moderate	High	Low	High

Table 3.1: Digital Interface Comparison

Table 3.1 clearly shows that CameraLink is the fastest transfer medium with transfer rates of up to 3.6Gbps. However, the CameraLink interface requires a dedicated capture board for each camera which can be expensive. It has the most expensive cabling in comparison to the other interfaces listed, but could be rather straightforward to implement. The Gigabit Ethernet interface has a more than adequate transfer rate, is available in every PC, should be straightforward enough to implement in software but has a more complicated hardware/firmware design. Finally, FireWire and USB are technologies that are almost comparable in terms of transfer rate, with FireWire being slightly faster. Both, however, are adequate for the application. The major difference between these two technologies from a designer's point of view is availability and complexity. Although most PCs are equipped with FireWire controllers, almost all are equipped with USB. Due to the popularity of USB, more peripheral controllers are available for USB than FireWire and usually at a lower cost. Although either is suitable for the application, USB2.0 was selected because it was a low-cost interface that meets the data transfer rate requirements. Also, since USB is a star topology with a single master compared to FireWire which is a peer-to-peer based system, the complexity of USB devices is significantly less than that of FireWire.



### 3.2.2 Image Sensor

As a cost reduction measure, and to reduce mechanical changes, a custom USB2.0 camera was designed. For the design of the camera it was important that an appropriate sensor was selected. Some of the considerations for the image sensor include the decision between CCD or CMOS, colour or monochrome, resolution requirements and sensor size. After determining the minimum feature size flaw detection requirement, and considering transfer rate and image processing constraints, a colour, 3.1 mega-pixel, 1/2 inch CMOS image sensor was selected. A colour sensor was preferred since the ability to detect capsule colour was important. Although from analyzing the grayscale images, a colour classification can be estimated based on a subset of known colours, a colour sensor will provide more accurate information. The sensor chosen has a maximum resolution of 3.1 mega-pixels with a 4:3 aspect ratio resulting in an image size of 2048x1536 which significantly exceeds the minimum feature size requirement. The physical size of the sensor determines the sensor field of view (FOV) which was selected to roughly match the sensor used in the camera native to the OptiSorter. It was also desirable to have a low-cost sensor that suited the application.

### 3.2.3 Inspection Environment

One of the most important components of a quality machine vision system is the inspection environment. This includes the illumination, camera angle, immunity to external noise and maximizing viewable area of the capsule. Considerations such as illumination type and wavelength, obstructions in the inspection environment and motion of the capsule are all important in acquiring a quality image. From a machine vision standpoint, the better quality and consistency of the image results in a more accurate and faster inspection.

#### **Illumination**

Illumination is one of the most important aspects in achieving an ideal inspection environment. An abundance of lighting scenarios exist for which an appropriate selection is highly dependent on the object being inspected. Some lighting techniques include ring lights, back-lights, dark field ring-light, dome-lights, light-line and coaxial. Variations of these can be used and the inclusion of polarizers and filters can improve the image quality. Also, modifying the wavelength of the light source and sensor will affect the image quality. The object size, lustre, shape and colour must be considered in selecting an illumination technique in order to obtain the best quality image. For a transparent gelatin capsule which is rather small and highly reflective with a consistent shape, it is best suited to back-lighting which will enhance edges of the perimeter of the capsule and any flaws within the body of the capsule. Due to the required inspection rate and the fact the capsules are in motion during inspection, a measure to reduce blur is required. This is achieved by strobing

the light thus reducing exposure time and minimizing the effect of motion blur. From this, the light source, wavelength and intensity must be determined. Many lighting sources exist, yet LED is one of the most popular in machine vision applications. LEDs are compact, low-power, rugged, can be strobed and possess a long-life. LEDs come in a variety of wavelengths. The most commonly used in machine vision are red, green, blue and white. Since the recognition of colour is important for the inspection of the capsules, a white LED was selected. Comparing the spectral content of a white LED to the quantum efficiency of the selected sensor, a relatively close match exists between the spectral content of a white LED and the sensitivity of the sensor.

### **Capsule Holder**

The importance of the consistency of the inspection environment must be emphasized. In order for an inspection algorithm to yield consistent inspection results, a uniform environment must exist. The OptiSorter was originally equipped with solid holder which provided a consistent environment for inspection. A clear holder concept was attempted, however due to the sensitivity to scratches and inconsistent machining, they did not provide the consistency required for the application. The initial thought was that the use of a clear holder would permit more viewable area of the capsule through the transparent plastic. After experimental trials however, the inconsistency of the clear holder and the marginal improvement in viewable area was not compelling enough to pursue the idea further. After setting up a reasonable inspection environment, the solid holder provided a consistent environment suitable for performing inspection.

### **3.2.4 System Control**

Another integral component of the system is the system controller. The system controller is essentially an input/output board with communication interfaces such as I<sup>2</sup>C and UART. By taking a custom approach in designing the system controller, the cost can be drastically reduced over selecting a generic PLC, or other controller, with much more flexibility. The downside of a full custom approach is the extra design and development time. Starting with the requirements of the system controller, component selection began. The speed at which the system must operate, the required outputs and inputs and the information required after inspection were all considerations that were taken into account during the design. In addition, the selection of components had to be reasonable to reduce overall board cost and meet the performance requirements. In selecting the microcontroller of the system controller board, many criteria were considered including speed, size and features. A microcontroller was required that met the system requirements at a reasonable cost.

In addition to the system controller MCU selection were the other hardware components responsible for operating mechanical controls and isolation between different components of the system.

For this, many devices were researched, purchased and tested to verify their suitability for the desired application. One key consideration was the component cost and complexity.

### 3.2.5 Power Supplies

Any electrical system requires at least one power supply. For the PharmaSorter, it was desired to maximize the recycling of existing components of the OptiSorter. Unfortunately, the only reusable power supply was a 24V 1A Siemens supply. It was determined that the motor controller and the electrically controlled pneumatic valves required a 24V supply. The LCD panel required a 5V supply, however it appeared that the switches and lamp operated at 24V. The proximity sensor and back-light LEDs required 12VDC to properly function. The system controller board must control the motor, pneumatic valves, lights, proximity sensor and HMI and thus requires 24VDC and 12VDC supplies. The selected MCU operates at 3.3V and thus must be accounted for. Due to the current supply requirements of the back-light LEDs, a second supply of 12V at 3.6A was used in the PharmaSorter. The 3.3V required for the system controller MCU and other ICs on the system controller board was regulated using a linear regulator from the 12V line. Each of the inspection PCs required an individual standard ATX power supply and thus four 500W power supplies were used for that purpose. The complete hardware for the inspection PCs are detailed in Chapter 6. This power scheme provides adequate power to the separate areas of the system and is more than sufficient for the power demand.

### 3.2.6 Inspection PCs

With flexibility as a primary goal, many features of the inspection PCs had to be determined. Since the inspection PCs are not fully functional desktop PCs, merely motherboards with RAM and hard disks, the goal was to find the best value for the performance requirements. Since a single PC manages four cameras, with a single PC per quadrant, a quad-core processor was desirable. After careful research of available components, a quality motherboard and CPU combination were achieved that would be compatible with the host OS. Fortunately with the speed and performance of PCs constantly improving, future generations of the PharmaSorter will benefit from this. Considering the stability of the system, the Windows operating system was not a feasible option. As far as a practical and reasonable OS for the application, Linux was an attractive choice since it is an open-source, fairly reliable, stable and secure operating system. After selecting Linux as the host-OS, the remaining question was which distribution would be the best choice. Also, how well did it function with the selected PC hardware? After evaluation of various Linux distributions, Debian was selected for it's stability and reputable package management system. One of the most prominent features of Debian is the APT package management system. This is a well maintained package system that

allows for easy automated installations and updates. Debian allows one to install a base system without an X-Server or extraneous applications from which the user can install packages they deem necessary. Since the inspection PCs do not need to display any information graphically, they do not require an X-Server to function. The flexibility and mature package management system were the most influential components in the selection of Debian.

The design considerations and constraints were not initially fixed, but rather adapted and evolved as the project progressed. With the initial constraints in place however, the development of the various phases and components of the PharmaSorter project was started.

---

## Chapter 4

### *USB 2.0 Camera*

---

USB 2.0 is a modern protocol used to interface between computers and external devices. USB is a replacement for the legacy parallel and serial ports which are being phased out of desktop PCs by the computer industry. The main advantages of USB are the ability for multiple devices to be connected to a single USB network and the faster data transfer rates that be achieved with a greater deal of flexibility over serial and parallel ports.

USB devices require much more planning and development than serial and parallel port devices. Each USB device must have a unique identifier, handle a set of standard requests and must adhere to a strict set of rules defined by the USB2.0 specification [36]. It is fortunate however, that many semiconductor vendors provide USB2.0 microcontrollers that ease device development by incorporating the particulars of the USB2.0 specification directly in hardware. An example of one of these devices is the Cypress EZ-USB FX2 series of microcontrollers. The FX2 is a high-speed USB microcontroller with seven user endpoints which allows great flexibility and expandability during the design of peripherals. The selected USB microcontroller for the PharmaSorter USB2.0 cameras is the Cypress EZ-USB FX2LP.

Although there are vendors offering commercial USB still cameras, it was desired to take a full custom approach and design a USB2.0 camera from the ground up. The most prominent factor swaying this decision is cost. Taking a custom approach by designing the USB2.0 camera is beneficial for many reasons: First and foremost, the ability to reuse existing mechanical components. The OptiSorter was equipped with full-custom cameras from the original design with aluminum enclosures and lenses. By reusing these elements, the cost of the system is reduced substantially. Also, a fully custom approach permits use of application specific hardware components. A custom

design minimizes component count and board size that even in small fabrication runs, can be achieved at a relatively low cost.

The image acquisition system for the PharmaSorter involved a full-custom hardware design. A full custom approach is a major cost reduction technique since current USB2.0 still cameras for machine vision applications start above \$1000USD. A comparable camera from Silicon Imaging, the SI-1300-U, is a USB2.0 colour 3.2 megapixel camera with a cost of around \$1300. A full custom design begins with the conceptual design of the system, schematic level design and PCB layout and fabrication. When a hardware device is being designed, it is often easier and cheaper for the designer to find an evaluation system or development kit with the hardware they are considering. Many development kits provide schematics which serve as a validated reference design. For the PharmaSorter, this was indeed a situation where this could be taken advantage of. One of the most significant components of the camera hardware is the image sensor. There are many vendors and many sensors available that vary in image resolution (megapixels), speed and quality. After evaluating the products offered by various vendors, the Micron MT9T001 3.1 megapixel CMOS images sensor was selected [22]. The MT9T001 is low cost, low noise, high quality CMOS image sensor capable of 2048x1536 pixel images with a frame rate of 12 to 93 fps (frame rate depends on window size and resolution) [22].

After selecting the image sensor, a Micron evaluation board was purchased that was equipped with the MT9T001 sensor, a Cypress FX2 USB 2.0 microcontroller and a Xilinx VirtexII FPGA. This provided a good base for developing a custom camera. With the development kit, the various components could be evaluated based on the requirements of the project. It also provides a base design for custom hardware since all development board schematics and component selection were provided. A high level block diagram of the Micron evaluation camera is shown in Figure 4.1.

The Micron evaluation board was equipped with more than required to begin the development of custom firmware and software for the PharmaSorter. By designing custom firmware, the abilities of the evaluation board could be observed and a new hardware design could be tailored to meet the requirements of the project by minimizing component count, overall cost and complexity.

In many cases, full source code for evaluation boards is provided by the vendor. This was not the case however for the Micron camera development board. All firmware and MS-Windows drivers were closed source. Micron did provide documentation for the API they shipped with the device and sample code on using it with various programming languages. This was not terribly useful for the PharmaSorter since the host PCs would be running Linux, not Windows, and would be using custom firmware. However, some aspects of the Micron design posed useful as a reference design.

A useful application for MS-Windows that monitors traffic on the USB port known as USB Snoopy [40] was used to “reverse engineer” the driver and firmware developed by Micron for their

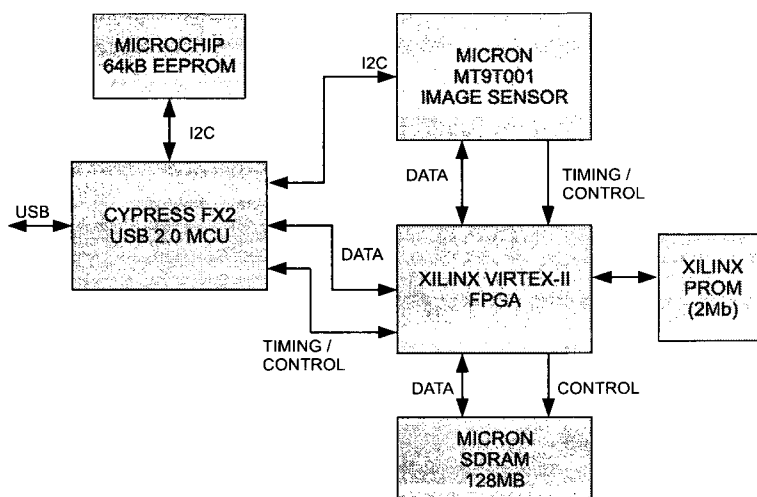


Figure 4.1: Micron Evaluation Board Block Diagram

evaluation kit. This was useful in determining how the sensor and FPGA were initialized. By using the API documentation and examining the control transfers it was evident that I<sup>2</sup>C write and read requests were used to configure the image sensor registers. When it came to actually transferring image data, Micron used 1024-byte blocks of data in a USB bulk transfer [23]. In accordance to the USB 2.0 specification, bulk transfers are limited to a maximum transfer size of 512-bytes [36]. The fact that Micron is using a 1024-byte bulk transfer permits lower overhead according to their findings [23].

In addition to the Micron evaluation board, a Cypress FX2 USB development board was also purchased. This aided in the development of the USB firmware and Linux driver. Cypress provided sample firmware programs (with source), written in C, for the development board. The firmware was developed using Keil Development tools, in particular the C51 compiler [13]. Keil develops compilers and assemblers for various families of microcontrollers including 8051 MCUs. Since the core of the Cypress FX2 is an enhanced 8051 structure, the Keil C51 compiler is a suitable tool to use, however it is quite expensive. Considering that the host PCs will run Linux, an open-source operating system, it was not unreasonable to use open-source tools to develop firmware for the USB camera. A compiler known as SDCC (Small Device C Compiler) is an open-source tool used to compile firmware for various embedded architectures including 8051 core MCUs [38]. A search on-line yielded several sample projects that targeted the Cypress FX2 using SDCC and open-source tools. These posed useful in the firmware development for the USB2.0 camera.

## 4.1 Hardware Level Design

After developing initial firmware and software using the Micron development board, which is described in detail in the subsequent sections, a finalized hardware design of the camera was developed. From a block diagram level description of the USB2.0 camera board, schematics were designed based on the specifics of the devices used in the design. After thorough analysis of the requirements of the system, a final camera board design was achieved based on the reference design of the Micron evaluation board. The design included a Xilinx Spartan 3E FPGA to replace the Xilinx Virtex II from the Micron reference design. It was determined that the Virtex II was exceedingly powerful for the application. The Spartan 3E provided the more than required logic elements for the firmware developed with adequate block-RAMs for image buffering. The finalized design excluded a SDRAM which was included in the Micron reference design for image buffering. It was determined that this was unnecessary and the overhead would compromise the target inspection rate. By excluding the SDRAM, the camera operates in real-time. It was determined that if the transfer rate could not meet the requirements, the image acquisition time would be exceedingly long to meet the timing requirement of the inspection throughput.

An addition to the camera hardware design is the inclusion of a NXP I<sup>2</sup>C bus extender [29]. The NXP I<sup>2</sup>C bus extender is an IC that extends the I<sup>2</sup>C bus by increasing the total system capacitive load to around 3000pF [29]. This permits longer transmission lines between devices on an I<sup>2</sup>C bus. I<sup>2</sup>C is used to transfer pass/fail messages to the system controller following an inspection. Although not intended for long distance communication, use of I<sup>2</sup>C bus extenders buffer the I<sup>2</sup>C line with higher driving currents that permit long distance communication [29].

The finalized design block diagram of the USB2.0 camera board is shown in Figure 4.2. This block diagram is a high-level representation of the USB2.0 camera developed. The important components including the sensor, FPGA, MCU, EEPROM and bus extender are included excluding power connections. The USB2.0 interface permits 500mA current draw at 5VDC. The USB bus power supply is used to power the camera, however is stepped down to 3.3V for all the ICs on the board with the exception of the FPGA. The Spartan3E series FPGA requires specific power ramping [41] and thus an application specific power IC is used to provide the 1.2V, 2.5V and 3.3V supply voltages that meets the required power-up ramping. The Texas Instruments TPS75003 is a specialized power management IC designed for powering the Spartan-3, Spartan-3E and Spartan-3L with the required start-up profile [33].



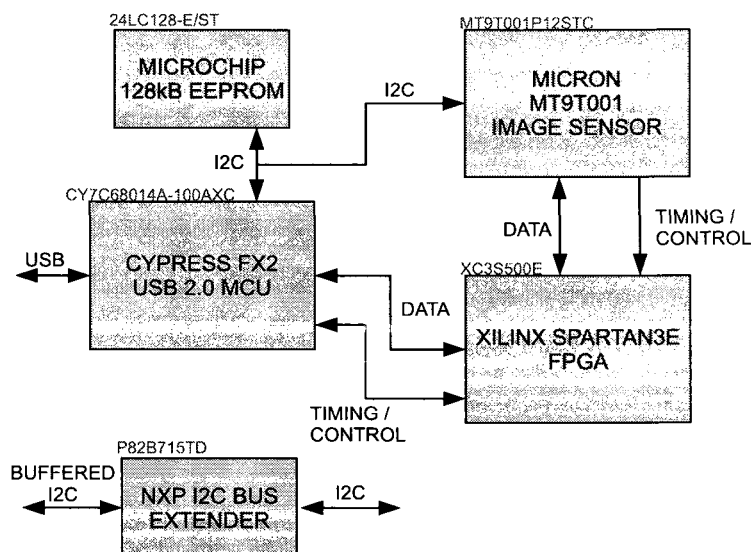


Figure 4.2: USB2.0 Camera High Level Block Diagram

## 4.2 Cypress EZ-USB FX2

The Cypress EZ-USB FX2 is a well featured MCU designed as a single chip solution for high-speed USB peripherals. The FX2 complies entirely with the USB2.0 specification and provides a set of registers and interrupts to handle USB communications at the hardware level. This abstracts some of the complexities of the USB protocol away from the designer. The FX2 is equipped with an enhanced-8051 core, based on the Intel 8051 architecture, with additional features including reduced instruction cycle (4 clocks compared to 12), faster clock (up to 48 MHz), additional timers and interrupts and an I<sup>2</sup>C controller [4]. This MCU is suitable for the USB2.0 camera because it permits soft firmware updates, is a single chip solution, has a built-in slave FIFO, a built-in I<sup>2</sup>C controller for setting registers of the image sensor and has an 8051 core compatible with an open-source compiler.

### 4.2.1 ReNumeration

Cypress uses a trademark technique for device identification and programming known as ReNumeration. The FX2 is a “soft” configured device that can take on multiple distinct USB devices [4]. When a device is connected to a PC, it is enumerated with the OS (see Section 2.2 for more detailed information). The FX2 is able to enumerate itself initially as a generic Cypress device that can be loaded with user firmware and without disconnecting and reconnecting from the USB. It can then ReNumerate itself through register settings (RENUM and DISCON) as the newly programmed device. The FX2 supports three ways of loading firmware. Firstly, an external EEPROM can hold

the entire firmware which can be loaded on power-up including vendorID, productID and deviceID information. Secondly, an external EEPROM can only hold the device vendor ID, product ID and device ID. In this scenario the PC can load the firmware over USB. This allows for easy firmware updates or “soft-firmware” loading. Or finally, if no EEPROM exists, the FX2 will appear as a generic Cypress device which can be loaded with firmware over USB. Each of the EEPROM contents requirements for the above scenarios is listed in the following tables Table 4.1 and Table 4.2 respectively.

EEPROM Address	Contents
0x00	0xC0
0x01	VendorID Low (VID_L)
0x02	VendorID High (VID_H)
0x03	ProductID Low (PID_L)
0x04	ProductID High (PID_H)
0x05	DeviceID Low (DID_L)
0x06	DeviceID High (DID_H)
0x07	Configuration Byte

Table 4.1: Cypress C0 Load - Descriptor Values Only

EEPROM Address	Contents
0x00	0xC2
0x01	VendorID Low (VID_L)
0x02	VendorID High (VID_H)
0x03	ProductID Low (PID_L)
0x04	ProductID High (PID_H)
0x05	DeviceID Low (DID_L)
0x06	DeviceID High (DID_H)
0x07	Configuration Byte
0x08	Length High
0x09	Length Low
0x0A	Start Address High
0x0B	Start Address Low
-	Data Block
...	...
	0x80
	0x01
	0xE6
	0x00
last	00000000

Table 4.2: Cypress C2 Load - Descriptor Values and Firmware

The C2 loading technique permits more than one data blocks for program data in different memory locations and is terminated by a unique identifying signature. The EEPROM loading

techniques of the FX2 are rather simple. The USB2.0 camera firmware uses the C0 loading technique to permit simple in-the-field firmware updates.

## 4.3 FX2 Firmware

One of the most vital components of the camera system is the ability to reliably and consistently transfer image data to the PC. USB offers several ways in which a computer can interact with a peripheral, each with deliberate yet versatile intentions. The USB 2.0 camera uses the default control endpoint 0 along with a single bulk endpoint to interact with the PC. The control endpoint is used to configure the MI sensor registers, configure the FPGA registers along with other various functions. The bulk endpoint is used to transfer the image data to the PC. A bulk endpoint is chosen because of the built-in error checking and guaranteed data integrity which is important for this application. Using a bulk endpoint for data transfer ensures that there is never any invalid data and that all the data requested will arrive to the PC.

### 4.3.1 Universal Serial Radio Project FX2 Library

The USB 2.0 camera firmware (FX2 firmware), was developed using SDCC. After researching on-line for existing FX2 firmware libraries with register definitions and USB interrupt handlers, the USRP (Universal Serial Radio Project) was discovered [37]. The USRP project is a high-speed data acquisition board developed by the open-source community. The USRP uses an Altera FPGA and a Cypress FX2 MCU to transfer waveform data to a PC running Linux, Windows or OS X. The FX2 framework developed for the USRP project was coded in C specifically for the SDCC compiler. The USRP project firmware code was used as a base for the firmware development of the camera. This framework included all register definitions, handled USB interrupt and responded to all standard requests defined by the USB 2.0 specification [36]. By using this existing library, the camera firmware development could begin immediately.

### 4.3.2 Bulk Transfers

Due to the nature of the image data acquired from the sensor, the best USB transfer method to use is bulk transfers. Due to the built-in error checking and guaranteed data integrity of the transfer method, it is the most suitable for the application. The Cypress FX2 has a 4kB slave FIFO that can be used to hold data to send to a PC or to store data that has been received from the host. The size of the FIFO for a given endpoint is dependent on the number of endpoints used. Since transferring packets of bulk data is quite common among USB peripherals, for example scanners and flash drives, the FX2 has the ability to automatically commit packets when the FIFO begins to

fill. The auto commit feature of the FX2 allows data to be automatically submitted as a USB bulk packet of data when it reaches a given amount [4]. To maximize data transfer ability, the maximum bulk packet size (512 bytes) is used with the auto commit feature to permit image data to be sent to the PC as the FIFO is filling. Since the entire slave FIFO of the FX2 is dedicated to endpoint 2 (bulk-in endpoint), the FIFO is able to buffer up to eight packets of data.

### 4.3.3 MI Sensor Configuration

The Micron image sensor uses a two-wire communication scheme to configure registers. Micron does not explicitly use the term  $I^2C$ , however this is essentially what is being used.  $I^2C$  is used to set and read registers of the Micron sensor. When writing a register of the MI sensor, an  $I^2C$  message is sent to the MI slave address with the R/W flag set to 0 indicating a write request. The next byte of data is the register, followed by two bytes containing the register data. Similarly, to read a register from the sensor, an  $I^2C$  message is sent to the MI slave address with the R/W flag set to 0 indicating a write request, followed by the register address. Following this, a bus restart is initiated and the master sends an  $I^2C$  message to the MI slave address with the R/W flag set to 1 indicating a read request. The sensor responds with two bytes containing the register data.

The FX2 library framework from the USRP project has routines for  $I^2C$  read and write requests. These functions are used to interact with the MI sensor and FPGA. Two USB vendor control requests are created for  $I^2C$  transfers. A vendor IN request is used to read data from an  $I^2C$  slave device and a vendor OUT request is used to write data to an  $I^2C$  slave device. The vendor requests associated with the camera firmware can be found in Appendix B.1. The framework of the vendor request for  $I^2C$  write and read requests is shown below.

#### $I^2C$ write request:

```
usb_control_msg:
    bRequestType:  VENDOR_REQUEST_OUT
    bRequest:      VRQ_I2C_WRITE
    wIndex:        Not Used
    wValue:        I2C Slave Address
    data:          Data to be written
    length:        Length of Data
```

#### $I^2C$ read request:

```
usb_control_msg:
    bRequestType:  VENDOR_REQUEST_IN
    bRequest:      VRQ_I2C_READ
    wIndex:        Not Used
    wValue:        I2C Slave Address
    data:          Data to store result
    length:        Length of Data
```

#### 4.3.4 FPGA Register Configuration

Similar to the method of setting and reading registers from the Micron sensor, the FPGA was coded to support register manipulation using I<sup>2</sup>C. The same I<sup>2</sup>C read and write USB vendor requests are used. The FPGA however does not support reading registers at the current time, only writing is implemented.

#### 4.3.5 FPGA Loader Firmware

To reduce the part count and cost of the camera, a memory device is not included in the design. The Micron evaluation board does include a Xilinx XC18V02VQ44 2Mb configuration PROM. This device is used to store the FPGA configuration information. Upon power-on, the FPGA loads the data from the configuration PROM in master serial mode. FPGAs are very flexible however in the ways they can be configured. Some configuration techniques include JTAG, master serial, slave-serial, master-parallel and slave-parallel. Each method has pros and cons, but the most convenient technique to load the device using an intelligent processor, such as a MCU, is the slave-serial method for this application.

#### 4.3.6 Slave-Serial FPGA Loading Technique

Slave-serial loading technique for FPGAs involves the use of an external processor to load the FPGA configuration at any given time. To facilitate loading the FPGA, the following lines are used:

- **CCLK** - Configuration Clock (input)
- **PROG** - Asynchronous Reset to configuration logic (input/output)
- **INIT** - Indicate when device is ready to receive configuration data. Also used to flag errors (input/output)
- **DONE** - Indicates when configuration is in startup sequence (input/output)
- **M[2:0]** - Mode select, selects configuration mode to use (input)
- **DIN** - Serial configuration data input (input)
- **DOUT** - Serial configuration data output for daisy chaining (output)

Loading of the FPGA configuration bit file is facilitated through a set of vendor requests using the FX2 control endpoint zero. A total of three requests are required for loading the FPGA. The vendor requests used are VRQ\_FPGA\_LOAD\_SS which is used for both IN and OUT request. The IN requests are FPGA\_LOAD\_START and FPGA\_CHECK\_DONE. The OUT request is

FPGA\_LOAD\_DATA. When loading the FPGA, the FPGA\_LOAD\_START is used to put the FPGA in configuration mode. The firmware pulses the PROG bit until the INIT bit goes high. When INIT is high, the device is ready to be loaded with the configuration data. If for some reason INIT does not go high, the host is notified using the endpoint zero data buffer of the IN request. If a 1 is returned, the FPGA was successfully put into configuration mode. If a 0 is returned, an error occurred while putting the FPGA in configuration mode. Once in configuration mode, the configuration data is bit-banged to the FPGA using the CCLK and DIN bits. The OUT request FPGA\_LOAD\_DATA is used with the 64-byte endpoint zero buffer to load the device. The host parses through the BIT file and sends the data to the FX2 in 64-byte blocks. The FX2 then processes the received data bit by bit and sends it to the FPGA. The DIN bit is set high or low depending on the bit value and the CCLK line is pulsed high then low. This is repeated for the entire BIT file worth of data. After each packet is loaded to the device, the INIT bit is checked to see if the INIT line has gone LOW indicating a configuration error. If there is an error during configuration, the host will receive a broken pipe error indicating an error during loading. After the entire BIT file has been bit-banged to the FPGA, the host sends the request FPGA\_CHECK\_DONE. This request pulses the CCLK line and continually checks the status of the DONE bit. If the DONE bit goes high, the configuration was successful and complete. Otherwise there was an error during configuration. The excess of CCLKs is to allow any digital clock managers to lock which typically only takes a few microseconds.

## 4.4 Control Board Communication

In order to facilitate communication with the system control board, the I<sup>2</sup>C master module of the FX2 is used. The control board MCU I<sup>2</sup>C module is in slave mode and responds to requests from the camera. To facilitate multiple masters connected to a single slave device, an I<sup>2</sup>C bus switch was designed for the system control board that switches between masters until a START bit is seen. When a channel is active, the SCL (serial clock) lines of the other cameras is held low. Only the active I<sup>2</sup>C channel is permitted to communicate with the system control board MCU. Before any I<sup>2</sup>C transactions can be made for a camera, the bus must be activated by the system control board I<sup>2</sup>C bus switch. To ensure that the bus is active, the SCL line of the camera I<sup>2</sup>C bus is connected to a GPIO of the FX2. This line is checked before an I<sup>2</sup>C request is made. This allows the master to wait until the I<sup>2</sup>C bus channel is active before starting a transaction. The reason all buses are not directly connected together is due to the fact that the Micron Image sensor I<sup>2</sup>C slave address is fixed and this would incur severe bus problems.

## 4.5 EEPROM Memory Map

The most convenient way to store information regarding the location of a particular camera in the system is to use the camera EEPROM. Using the memory map listed in Table 4.3, location information and specific camera window parameters are stored. Due to unique calibration requirements of each camera because of mechanical imperfections, windowing information stored in the EEPROM is used to provide consistent images to the image processing software. The same EEPROM that is used for USB identification is used for storing the camera specific settings. The FX2 startup requires the first 8-bytes of data using the 0xC0 loading scheme, the camera specific settings are stored in the range 0x10 to 0x28.

EEPROM Address	Contents
0x00-0x07	Reserved for USB Identification (see Section 4.2)
0x08-0x0F	Not Used
0x10	Quadrant
0x11	Position
0x12	Master Flag
0x13-0x1F	Reserved for Future Use
0x20	Window Width High
0x21	Window Width Low
0x22	Window Height High
0x23	Window Height Low
0x24	Window Column Start High
0x25	Window Column Start Low
0x26	Window Row Start High
0x27	Window Row Start Low
0x28	Window Column Skip
0x29	Window Row Skip

Table 4.3: USB2.0 Camera EEPROM Memory Map

In the initialization of the FX2, a routine called *load\_camera\_config()* is used to load these parameters from the EEPROM to the RAM of the FX2. This way, simple USB control requests can be made to obtain this information in software without excess I<sup>2</sup>C traffic or latency.

A simple utility was created for loading this camera specific information. The utility *cam\_init* (see Section 6.6) is designed to set the camera location information and is able to read and write data of specific EEPROM memory locations. To make the configuration of window parameters easier, a simple GUI application written in Python was developed that is a front-end for *cam\_init* (see Section 6.7).

## 4.6 USB2.0 Camera Linux Device Driver

In order for the inspection PC to acquire image data from the image sensor, a driver had to be developed. In most operating systems, two types of drivers typically exist, user-mode drivers and kernel-mode drivers. A user-mode driver runs entirely in user-space, where applications run. Kernel-mode drivers operate in kernel space and have a closer relationship with hardware. Kernel-mode drivers typically require a solid understanding of the operating system and proficiency in C programming. It is not difficult to write a poor device driver that may cause the entire system to crash. User-mode drivers on the other hand are much safer in the sense that it is unlikely for them to cause a system to crash. In user-mode drivers, a layer exists between the user space and kernel space. Due to the transfer of data between the two, a user-mode driver is slower than a kernel-mode driver. The overall trade off is between performance, stability and development time.

The USB2.0 camera driver is a hybrid driver that operates mainly in user-space but uses low-level calls to improve image transfer performance.

### 4.6.1 LibUSB

LibUSB is a library of routines for manipulating USB devices. This is a convenient API for user-mode USB device driver development. LibUSB is equipped with functions for finding, claiming and interacting with devices. It is used extensively in the USB2.0 camera driver for obtaining the device handles, claiming device interfaces and simple control requests. LibUSB does support bulk read/write functions, however it suffers performance loss due to the constant switching between user and kernel space.

To alleviate this, a class was developed to support low level calls for retrieving data over the bulk-IN endpoint of the USB2.0 camera.

### 4.6.2 `pm_cam` Class

A class was developed that holds the USB2.0 device driver called *pm\_cam*. This class is used to encapsulate all functions related to the camera. This allows an application to create instances of the class for each device found and manipulate the device in a well organized fashion.

Using a set of functions created for the *pm\_cam* class, manipulation of devices is rather straightforward. The important functions of the *pm\_cam* object are listed below (see Table 4.4).



In addition to these functions, a set of primitive functions were created for finding devices and acquiring device handles. The primitive functions are encapsulated in a library called *pm\_prims*. The primitive functions are listed with descriptions in Table 4.5.

A typical example, using pseudo-code, of finding devices and initializing them in host software using the library *pm\_cam* is listed in Listing 4.1.

Function	Description
pm_cam(struct usb_dev_handle *udh, int block_size, int n_blocks)	Constructor, creates an instance of the pm_cam class for the device handle specified with block_size for bulk transfers. Also creates an instance of <i>imgUSB</i> class for specified device handle for high speed bulk transfer.
pm_cam()	Destructor, free up any allocate memory and deletes instance of <i>imgUSB</i> object.
write_cmd(int request_type, int request, int value, int index, char *data, int len)	Perform a standard control request to device (using libUSB).
write_reg(unsigned char reg, short value)	Write a value to a specific register of the Micron image sensor.
read_reg(unsigned char reg, short *dat)	Read data from a specific register of the Micron image sensor.
write_fpga_reg(unsigned char reg, short value)	Write a value to a register of the FPGA. FPGA does not support reading at this time.
set_window_width(int _width)	Set the width of the window, also write value to image sensor register.
set_window_height(int _height)	Set the height of the window, also write value to image sensor register.
set_window_width(int _width, int _skip)	Set the width of the window with horizontal skip factor. Will compute appropriate image width and window width depending on skip value.
set_window_height(int _height, int _skip)	Set the height of the window with vertical skip factor. Will compute appropriate image height and window height depending on skip value.
get_window_width()	Returns the window width value.
get_window_height()	Returns the window height value.
get_image_width()	Returns the image width value (from window width and horizontal skip factor).
get_image_height()	Returns the image height value (from window height and vertical skip factor).
set_window_col_start(int _col_start)	Set the image sensor window column start pixel.
set_window_row_start(int _row_start)	Set the image sensor window row start pixel.
read_window_params()	Load from camera the window parameters stored in the EEPROM.
get_eeprom_window_width()	Returns the EEPROM value for window width of camera.
get_eeprom_window_height()	Returns the EEPROM value for window height of camera.
get_eeprom_window_col_start()	Returns the EEPROM value for window column start pixel of camera.
get_eeprom_window_row_start()	Returns the EEPROM value for window row start pixel of camera.
get_eeprom_window_col_skip()	Returns the EEPROM value for column skip value of camera.
get_eeprom_window_row_skip()	Returns the EEPROM value for row skip value of camera.
get_cam_location()	Read camera location information stored in EEPROM of camera. This include quadrant, position (left, right, center, bottom) and a master flag.
get_cam_position()	Returns the EEPROM value for camera position.
get_cam_quadrant()	Returns the EEPROM value for camera quadrant.
get_cam_master()	Returns the EEPROM value for camera master flag.
imgusb_allocate_urbs()	Allocate memory for <i>imgUSB</i> object URBs (depends on image size). This function uses the <i>image_width</i> and <i>image_height</i> variables of the class.
grab_frame(unsigned char *buf)	Retrieve image data to <i>buf</i> using <i>imgUSB</i> class. Returns number of bytes received.
cam_fpga_reset(int state)	Put the FPGA in or out of reset state.
cam_fpga_power(int state)	Enable or disable power to FPGA.
bayer2gray(unsigned char *bayer, unsigned char *buf, int width, int height)	Use a simple software nearest-neighbour interpolation to generate grayscale image from bayer pattern output of image sensor.
write_tiff(unsigned char *buf_in, char *filename, int width, int height)	Write the data in <i>buf_in</i> to a grayscale TIFF image to <i>filename</i> with dimension specified by <i>width</i> and <i>height</i> respectively.

Table 4.4: pm\_usb Class Functions

Function	Description
pm_init_usb()	Required to initialize libUSB.
pm_get_device_count()	Returns the number of devices found in the entire USB system that match the specified USB VID and PID in the header file <i>pm_ids.h</i>
pm_find_camera (int n.th)	Returns device pointer to the $n^{th}$ instance of the device in the USB system.
pm_camera_configured (struct usb_device *d)	Returns TRUE if the device specified is configured (firmware loaded).
pm_open_interface(struct usb_device *d, int if_num, int alt_if_num)	Returns the device handle to the interface specified for the device specified. If the interface does not exist, NULL is returned.
pm_close(struct usb_dev_handle *udh)	Releases the interface and closes the device. Returns TRUE on success.

Table 4.5: pm\_prims Primitive USB Functions

```

...
initialize libusb

dev_count = get device count

IF no devices found THEN
    OUTPUT: error message
    EXIT
END IF

allocate memory for pointers to devices
allocate memory for pointers to device handles
allocate memory for pm_cam class objects

FOR all devices in dev_count
    obtain device handle

    IF unable to open device THEN
        OUTPUT: error message
        EXIT
    END IF

    IF device not loaded with correct firmware THEN
        OUTPUT: error message
        EXIT
    END IF

    claim device interface and obtain device handle

    IF unable to obtain handle THEN
        OUTPUT: error message
        EXIT
    END IF

    create an instance of the pm_cam object for handle found
NEXT device
...

```

Listing 4.1: Finding Devices with pm\_usb Class

Once the device handles are acquired, the *imgUSB* objects can be created. The *imgUSB* class contains low-level IOCTL calls to improve the performance of USB bulk transfers. From experimentation, the data transfer rate is more than 30% faster using the *imgUSB* class over the libUSB *usb\_bulk\_read()* function. Not to mention the reduced CPU utilization with the *imgUSB* class.

### 4.6.3 *imgUSB*

It was determined early in the project that the transfer rates capable of libUSB were inadequate for the throughput requirement of the system. The theoretical maximum transfer of the high-speed USB standard is 480Mbps [36]. This translates to 60MB/s, however when testing the transfer rate using the *usb\_bulk\_read()* function in the libUSB library, the maximum achievable transfer rate with a continuous data stream was around 31 MB/s. After researching for ways to improve this in user-space drivers, the Universal Serial Radio Project was discovered [37]. The USRP project uses low level IOCTL calls to perform bulk transfers. After making appropriate changes and incorporating the *fusb* library from the USRP project into a test application, transfer rates of 42MB/s were realizable. The USRP project *fusb* class however had some extraneous overhead. The *imgUSB* class was modeled after the USRP *fusb* class and the *usb\_bulk\_read()* function in libUSB. The resulting *imgUSB* class was more suitable for the application of periodic, but not continuous, data acquisition.

The *imgUSB* class is not terribly complex. The process of acquiring data from the bulk endpoint of the camera involves submitting URBs using a series of IOCTL calls to the USBDEVFS (USB Device File System) for the particular file descriptor of the USB device (acquired from libUSB). The URBs request blocks of data which is defined in the constructor of the *imgUSB* class. The maximum block size is 16kB, although it was found that 8kB provides reasonable performance and puts less of a constraint on the image size. *imgUSB* was not programmed to handle data transfer sizes that are not even multiples of the block size specified. After submitting the URBs, the URBs are reaped in a blocking IOCTL call. This blocking call essentially waits for all URBs to finish, or fail. Once all submitted URBs have been reaped, the function returns TRUE for a successful transfer and FALSE if an error occurred.

The functions contained in the *imgUSB* class are described in the following table (see Table 4.6).

Although the image size could be specified in the descriptor, but for convenience an additional function is used to allocate the memory used for the URBs.

Function	Description
<code>imgusb(struct usb_dev_handle *dev_hdl, int ep, int block_size)</code>	Constructor, initialized the <code>imgusb</code> class by specifying the device handle (from <code>libUSB</code> ), the endpoint to use (must be a bulk-IN endpoint) and the block size to use for the URBs.
<code>imgusb()</code>	Destructor, free occupied memory.
<code>allocate_urbs(int image_size)</code>	Used to allocate memory for URBs for the image size (or data size). This must be evenly divisible by the block size specified.
<code>get_image_size()</code>	Returns the image size.
<code>get_image(char *buf)</code>	Used to acquire an image from the camera using a series of URBs. The resulting data is stored in the referenced variable <i>buf</i> . Returns <code>TRUE</code> on success and <code>FALSE</code> on fail.

Table 4.6: `imgUSB` Class Functions

---

## Chapter 5

---

### *System Control Board*

---

The system control board is an essential piece of hardware for machine control and capsule tracking. This includes controlling the motor speed, capsule ejection hardware, providing camera triggers, controlling lighting, operating the HMI, and monitoring system health. Added functionality required for the upgraded system includes the soft PC power control, camera communication interface and RS-232 support.

Starting from a conceptual design of the requirements of the system control board, a block diagram representing the various interfaces was created, shown in Figure 5.1. Using a microcontroller (MCU) is the most obvious and practical means of controlling the various hardware and thus is required. Selecting an appropriate MCU is not trivial though. An essentially unlimited supply of devices exist to choose from. Selecting an architecture with adequate capabilities that does not exceed cost constraints is desired. A typical modern MCU suitable for this application operates at 3.3V or 5V and consumes a very modest amount of power. Interfacing with the motor controller, operating ejection hardware and switching lighting requires significantly more power than what is capable from the outputs of a typical MCU. Also, considering that many of the existing hardware components operate at 12V or 24V and the processor operates at 3.3V or 5V, isolation circuits must be designed to bridge the gap.

#### 5.1 Hardware Design

The hardware design process for the system control board was a fully custom design. Hardware components were researched, tested and incorporated into the final system control board. The

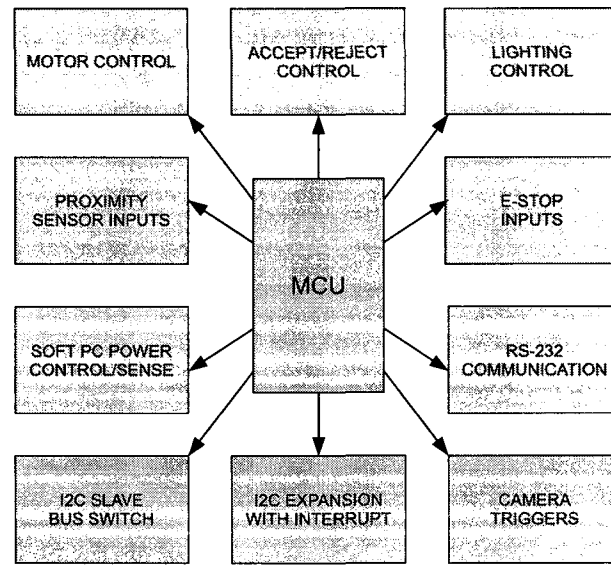


Figure 5.1: System Controller Block Diagram

design of the system control board involved the design of many smaller circuits that interface with various hardware components of the system with additional circuits to support specific functionality. In addition to basic machine control tasks, it was desired to incorporate flexibility into the design to permit expandability.

### 5.1.1 Isolation Circuits

Isolation circuits are quite common in the field of electronics, especially when interfacing different voltage levels and impedance ranges. An optocoupler or optoisolator, is a device that permits full electrical isolation between circuits. An optocoupler uses a short optical path to transfer a signal between circuits. Because light is used, there exists no electrical connection between the two sides of the device. Thus, the potential level on either side of the optocoupler may differ. A typical optocoupler is comprised of a LED (light emitting diode) and a photo-transistor. When the LED is “on”, the photo-transistor will conduct. A typical circuit symbol for an optocoupler is shown in Figure 5.2.

Optocouplers are used extensively in the design of the system controller board. Thus, a brief explanation of the operation of a typical isolation circuit will be discussed. The design of each isolation circuit varies slightly depending on the required response time of the circuit, the voltage levels involved and the driving requirements of the circuit. Although power consumption is not a primary concern in the system control board design, the circuit parameters used impact the power consumed by the device. The goal was to maintain a modest current draw while exceeding timing

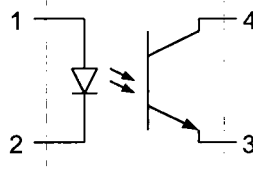


Figure 5.2: Typical Optocoupler Circuit Symbol

requirements. A typical isolation circuit used in the design of the system control board is shown in Figure 5.3.

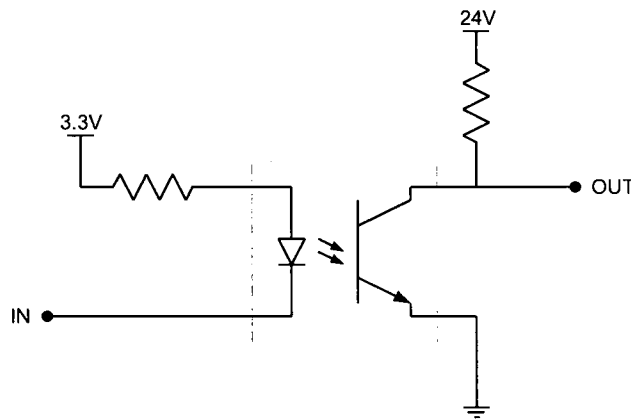


Figure 5.3: Common Isolation Circuit

For this particular circuit, when the IN pin is at 0V, the LED is ON, and thus the photo-transistor is in conduction. This results in the voltage seen at OUT to be pulled close to 0V. Similarly, when the input IN is 3.3V, the LED is off and the photo-transistor is not in conduction. This results in the output voltage seen at OUT to be pulled close to 24V by the pull-up resistor. The above circuit demonstrates that isolation can exist between two potential levels. The above configuration is a common theme throughout the design of the system control board. Many hardware elements of the Optisorter operate at 12V and 24V requiring isolation circuits. Many circuits also need a substantial current supply. The photo-transistor of the optocoupler is quite weak and not intended for driving large loads. For this, driving circuits are required. The driving circuits use the output of the isolation circuit to control the power delivered to the load.

Without optocouplers, interfacing a MCU with other hardware elements would involve the design of complex circuits. Optoisolators simplify this task greatly. It will be evident just how significant isolation circuits are in the design of the system control board.



### 5.1.2 Driving Circuits

Considering that the driving capabilities of a typical MCU is limited to only tens of milliamperes and the driving capabilities of the photo-transistor of an optocoupler is also quite low, driving circuits are required to drive many of the hardware elements of the PharmaSorter. The use of power MOSFETs (metal oxide semiconductor field effect transistors) allow large loads to be driven by lower input signals. Power MOSFETs can operate like switches and can be used to turn on and off hardware of the PharmaSorter with relatively high current sinking or sourcing abilities depending on the type of power MOSFET used. Like any transistor family, N and P channel devices exist for power MOSFETs. The N channel FETs are typically used when sinking loads, and P channel FETs are typically used when sourcing loads. Both N and P channel devices are used in the design of the system control board. Another desirable property of power MOSFETs is the input current required to turn the device “ON” is very low. Also, the “ON” resistance of the device is very low. This results in efficient switching with very low heat generation. Typical driving circuits are shown in Figure 5.4 and Figure 5.5. Both sinking and sourcing configurations are illustrated.

When used in conjunction with an isolation circuit, devices can be switched on or off with substantial driving capabilities. For example, the N-Channel power MOSFET used in many areas of the design is the International Rectifier IRF7103. The IRF7103 is a dual N-channel power MOSFET with a maximum drain current ( $I_D$ ) of 3A capable of 50V  $V_{DSS}$  and an  $R_{DS(on)}$  of only 0.130  $\Omega$ . The IRF7103 is a suitable device for the system control board. It is an extremely efficient, fast switching device that is used to control solenoids of the electrically controlled pneumatic valves. The P-Channel sibling to the IRF7103 is the IRF7306. The IRF7306 is a dual P-Channel power MOSFET with a maximum drain current of 3.6A capable of -30V  $V_{DSS}$  and a  $R_{DS(on)}$  of 0.10  $\Omega$ . The IRF7306 is used in sourcing circuits and the IRF7103 is used in sinking circuits. The initial state of devices, before the MCU has been initialized, can be controlled by the selection of either P or N channel drivers. When considering the input to the LED of the optoisolator in a floating state, the LED will not be on and thus the phototransistor will not be conduction. If an N-channel device is selected with the given configuration, the MOSFET will be on. Likewise, for a P-channel device and the same configuration, the “default” state of the device is off.

The circuits described above are common in the design of the system control board. The system control board circuits are however more diverse and are described in the following sections.

### 5.1.3 MCU Selection

Selecting an appropriate MCU for the system control board deemed to be a more exhaustive task than one might expect. When selecting a device for a somewhat specific application can be daunting. Semiconductor manufacturers offer a vast selection of devices for designers to select from. The

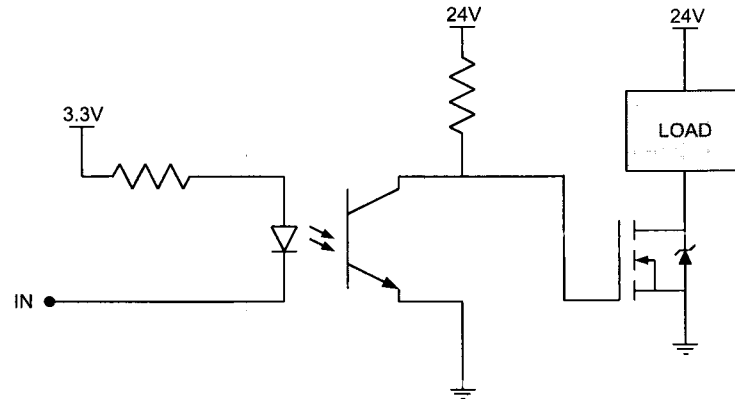


Figure 5.4: Isolation Driving Circuit - Sinking

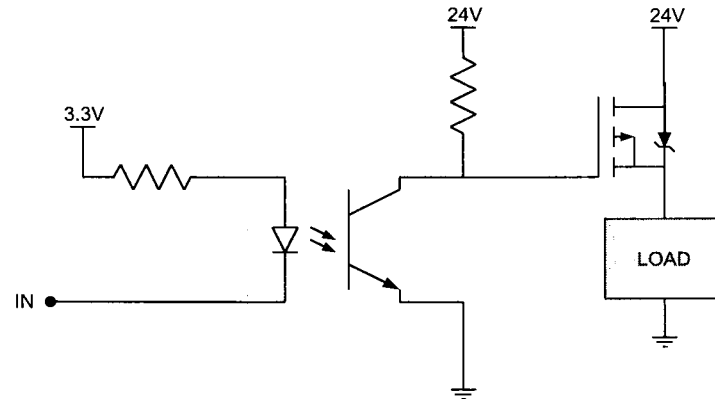


Figure 5.5: Isolation Driving Circuit - Sourcing

characteristics considered for the system control board MCU were speed, peripherals, cost and I/O capabilities. The device functionality wish list is described in the table below (Table 5.1).

MCU Function / Category	Requirement
<b>GPIO (General Purpose I/O)</b>	<b>24</b>
Pneumatic Valve Control	6
E-Stop Source	4
External I <sup>2</sup> C I/O Expander Interrupt Source	4
I <sup>2</sup> C Slave Multiplexer Control Signals	2
PC Soft Power Control	4
PC Soft Power Sense	4
<b>Input Capture</b>	<b>4</b>
Proximity Sensors	4
<b>Output Compare</b>	<b>7</b>
Motor Pulse Signal	1
Camera Triggers	2
Back/Front Lighting (LED Matrix)	4
<b>External Interrupts</b>	<b>2</b>
E-Stop	1
External I <sup>2</sup> C I/O Expander Interrupt Signal	1
<b>I<sup>2</sup>C Slave Bus (Camera Communication)</b>	<b>1</b>
<b>I<sup>2</sup>C Master Bus</b>	<b>1</b>
<b>UART (RS232 Communication)</b>	<b>1</b>

Table 5.1: System Control Board MCU Requirements

Although many semiconductor manufacturers offer devices that meet and exceed the requirements listed, it was desired to find a development board with the selected MCU that is readily available. It was also desired to find a device that had an inexpensive or free C cross-compiler. Also, a 16-bit device was preferred over an 8-bit device. It was eventually decided that the Microchip dsPIC33 family of MCUs met the requirements in the metrics listed above. The dsPIC33FJ256GP710 MCU is the highest pin count device with the most program memory offered in the dsPIC33 series. Microchip offers a C cross-compiler for the dsPIC33 family, namely the C30 Tool suite. Microchip offers a student version of the compiler for free. They also offer a commercial version which has more optimizations but with a hefty price tag. One of the requirements when searching for MCUs was the availability of a development board. Microchip offers a development board for the dsPIC33 known as the Explorer 16. The Explorer 16 is readily available through Digi-key and other prominent electronics suppliers. The Explorer 16 evaluation kit is shipped with the Explorer 16 development board, an ICD2 in-circuit programmer/debugger, Microchip MPLAB and the C30 compiler. The ICD2 in-circuit programmer is a programmer solely for Microchip MCUs. MPLAB is an IDE for managing projects targeting Microchip MCUs and handles the linking of object files.

#### 5.1.4 Power Regulation Circuit

Power regulation is very common in all electronic devices since digital circuits require stable DC power. The dsPIC33 operates at 3.3V and thus a 3.3V line is required to power it along with the other devices used in the system control board. Since the system control board controls hardware at higher levels, these voltages must also be present on the board. The lighting control circuit operates the LED back and front lights by providing 12V pulsed strobes. The electrically controlled pneumatic valves require 24V to activate. The stepper motor control board operates at 24V and requires a 24V pulse signal to step the motor. The system is equipped with a Siemens S5-100U 24V supply capable of delivering 1A. Considering the requirements of the system, this single supply would not suffice. An additional 12V supply was purchased to operate the 12V hardware along with powering the control board. The V-Infinity VOF-45-12 is a 12V supply capable of supply 3.7A. The system control board has connections for both the 12V and 24V supplies and are fused for over-current protection using 1A fuses on each line. The 12V line is regulated down to 5V using a ST Microelectronics linear regulator, and further regulated down to 3.3V using a Linear Technologies low-noise, low-dropout (LDO) regulator (LT1763). The reason for the cascading regulators is that the LT1763 is capable of an absolute maximum input voltage of 20V. If there was a short between the 12V and 24V lines, this could potentially destroy the 3.3V regulator and possibly other components of the board. The ST-Micro 5V regulator is capable of a maximum input voltage of 35V and would not be damaged in the event of a short or improper connection.

#### 5.1.5 Electrically Controlled Pneumatic Valve Control Circuit

The Optisorter was equipped with a series of electrically controlled pneumatic valves. These valves are used to control the ejection of capsules with a main valve control on the main air supply. The valves require 24V to open, allowing air to flow.

When the valve is activated, an electro-mechanical solenoid engages allowing air to pass through. Since the solenoid is a purely inductive load, the phenomenon of back-EMF (electro-motive force), or counter-EMF (CEMF) must be taken into account. CEMF is caused by a changing electromagnetic field, like that of a solenoid. When the solenoid is released, a reverse voltage is developed that can potentially damage sensitive devices. In order to handle the CEMF, a diode is connected across the load to short the CEMF voltage across the load, as shown in Figure 5.6.

In the above circuit, when the solenoid is switched off, a large counter EMF appears across the terminals. This voltage can be several thousands of volts which could potentially incur severe damage to the MOSFET. The forward biased diode forces this voltage to be suppressed through the solenoid. The diode used in the system control board to protect against CEMF from the pneumatic valves are fast-recovery, 100V, Fairchild 14N002FSCT. These were selected for their fast recovery,



shown in the previous section, this configuration requires that the dsPIC MCU sources the LED of the optoisolator because of its output current capabilities. The schematic shown includes resistor values, part number and reference designators used in the actual design.

### 5.1.6 Stepper Motor Controller Control Circuit

The Optisorter system was equipped with a 5-pole stepper motor and a corresponding stepper motor controller. By using a stepper motor, the position of the rotor is always known. However, controlling a stepper motor is not necessarily trivial. Each pole of the motor has independent coils that must be energized in a correct sequence. Designing and building a circuit for this would be time consuming and unnecessary considering the availability of motor controllers. The stepper motor controller uses control signals from an intelligent device to control the direction and speed of the stepper motor. The controller is designed to be connected to an industrial logic level device which operate at 24V. Thus, 24V control signals are required to operate the stepper motor controller. The direction of rotation of the Optisorter can only be in a single direction. Thus, the direction input is permanently set using a hardwired connection. The only available control is the motor speed. This is controlled by a pulse-train at the desired frequency. Each pulse steps the motor a single step. By applying a train of pulses at a set frequency, the speed of the motor can be controlled while the position of the motor is always known.

The stepper motor control circuit must step a 3.3V pulse supplied by the dsPIC MCU to 24V using an isolation circuit. The stepper motor controller does not draw a significant amount of power and thus a power MOSFET driver is not required. Rather, a smaller transistor can be used to switch the voltage, as shown in Figure 5.8.

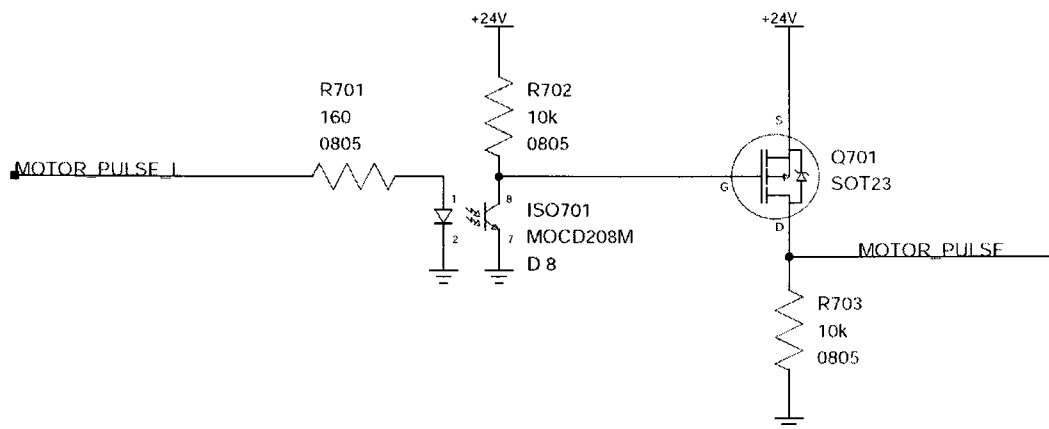


Figure 5.8: Motor Control Circuit

The switching transistor in this configuration is a P-Channel MOSFET capable of driving 130mA current. The transistor used is the NXP BSS84 T/R. This device is connected to a 10k  $\Omega$  load resistor. This resistance in combination with the MOSFET provides a fast enough switching time to exceed the maximum frequency of the motor.

An output capture module of the dsPIC is used to allow hardware to generate pulses for the motor. By adjusting the timer reset value, the hardware will automatically generate a stream of pulses at a given frequency.

### 5.1.7 LED Lighting Control Circuit

The lighting system of the Optisorter was a matrix of high-intensity red LEDs diffused through an opaque block of plastic. This provided lighting from the rear of the capsule holder and illuminated the capsule for the cameras. The existing cameras were assumed to be grayscale sensors where colour information was not required. Thus using red light was acceptable. For the upgraded system, white LEDs are used to achieve the a wider spectrum of light. This allows more accurate colour information to be acquired. The white light system is a matrix of high-intensity white LEDs that are strobed when the image sensor of the camera is being exposed. The entire inspection environment is enclosed in a dark enclosure and therefore this technique of strobing the light mimics a mechanical shutter. This way, the exposure of the sensor can be tightly controlled to ensure the images acquired are free of blur with sufficient light to gather quality images.

The lighting system operates at 12V. This is supplied by the VOF45-12 supply and switched through the system control board. The control board consists of a total of four lighting control channels with two outputs per channel. This allows for more outputs to be connected to a single channel. With four channels, the lighting requirements can be met with room for potential upgrades. It was initially assumed that the only lighting required would be from the rear of the holder, however it may be beneficial to illuminate the front of the capsule to acquire more accurate colour information for the inspection of coloured capsules.

The LED control uses P-Channel power MOSFETs to drive the LED matrix banks. The isolation circuits introduced earlier are used here, see Figure 5.9.

In this configuration, each power MOSFET is capable of driving up to 3.7A per, although it is unlikely that the LED arrays will load the driver this much. The current configuration of the PharmaSorter uses two of the possible four outputs and solely for back-lighting of two camera locations.

To tightly control the duration of the strobe of light, an output compare module of the dsPIC is used. This is a hardware method of timing the pulse of an output. By setting a start and end register with a base timer, the hardware will automatically enable and disable the output. This

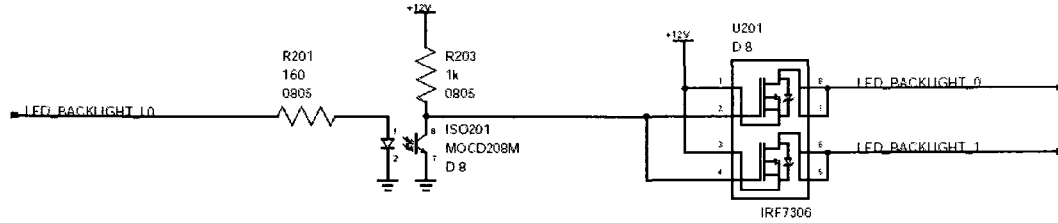


Figure 5.9: LED Lighting Control Circuit

ensures that the pulse duration of the light strobe is consistent.

### 5.1.8 Proximity Sensor Input Circuit

To provide feedback of the holder arm location, a series of notches have been machined into a metal disk located in the machine. A fixed proximity sensor provides a pulse every time a notch is encountered. This feedback can be used to track the position of the rotating arms. The proximity sensors operate at 12V and output a high signal when close to a dense object and a low signal otherwise. The proximity sensors are inputs to the dsPIC that must be dropped down to a 3.3V logic level. To manage this, isolation circuits are used. The actual circuit used in the design is shown in Figure 5.10.

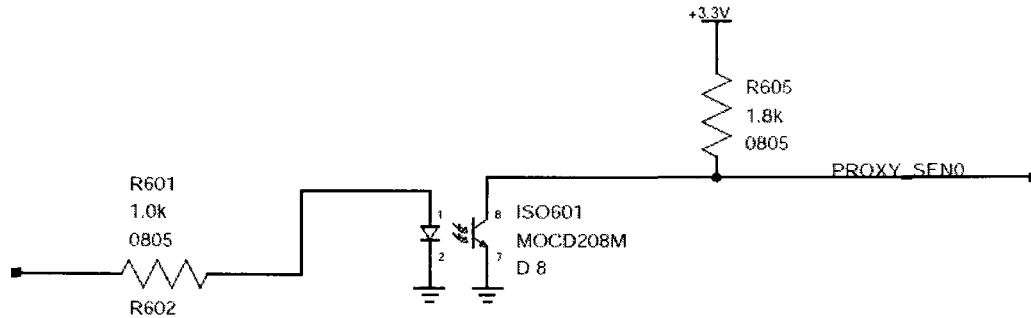


Figure 5.10: Proximity Sensor Input Circuit

The system control board was designed to handle up to four proximity sensors for additional feedback. Only a single proximity sensor is used in the prototype since the number of motor pulses applied to the stepper motor can be used to determine the exact position of the arm.

The proximity sensor inputs are connected to input capture modules of the dsPIC. Input capture allow an interrupt to be generated when the pin changes with a timer value recorded upon the event.



### 5.1.9 Camera Triggering Circuit

To synchronize the cameras with the position of the arm, triggers are provided to the cameras at appropriate times to begin image acquisition. The image sensor of the camera accepts a 3.3V trigger. A triggering method referred to as global shutter control [22] is used to control exposure time of the sensor. This is an input to the camera and requires specific timing. Using an output compare of the dsPIC, the camera trigger timing is precisely controlled. Although both the camera and the dsPIC MCU operate at 3.3V, the output from the dsPIC is required to drive a number of cameras, for this reason, it is buffered using a Texas Instruments CD74HCT126M line buffer. This device acts as a current driver allowing a larger fan-out. The schematic is shown in Figure 5.11.

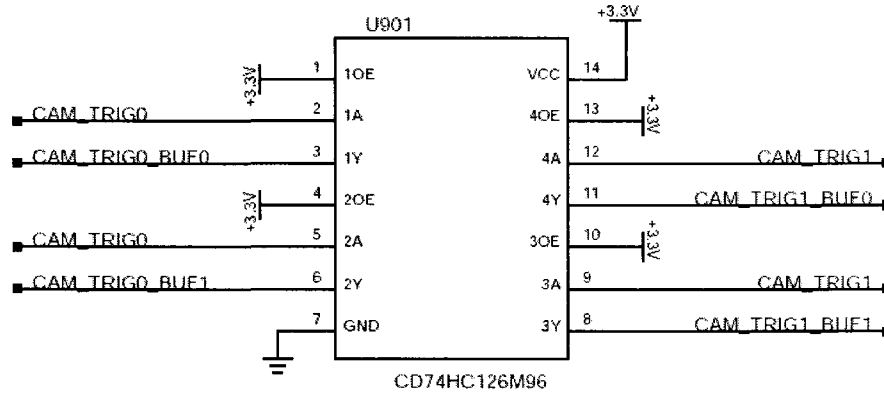


Figure 5.11: Camera Trigger Driver Circuit

### 5.1.10 I<sup>2</sup>C Expansion Circuit

Although the I/O capability of the dsPIC was not exhausted, the I<sup>2</sup>C bus was used to promote future system upgrades. In addition to providing extra I/O, I<sup>2</sup>C can be used for devices such as sensors and memories. These can be beneficial to the system for sensing things like temperature and storing setup information. To improve the distance of the transmission line between the devices and the system control board, and also to increase noise immunity, an I<sup>2</sup>C line buffer is used to increase the current of the signals. The NXP P82B715 I<sup>2</sup>C bus extender chip buffers the I<sup>2</sup>C transmission lines permitting long distance cabling. This circuit is shown in Figure 5.12. Splitting the I<sup>2</sup>C bus into four buffered buses with eight board connectors allows for a great deal of expansion. Each of these buffered buses is connected to the local bus of the dsPIC operating in master mode potentially many slave devices to be used. Four of the eight I<sup>2</sup>C connectors have an independent interrupt line that can be used to trigger the dsPIC on an external event. For example, an I/O expansion chip

can generate an interrupt on pin change. This interrupt can be detected and appropriate actions can be taken. The I<sup>2</sup>C buffering interrupt circuit is shown in Figure 5.13.

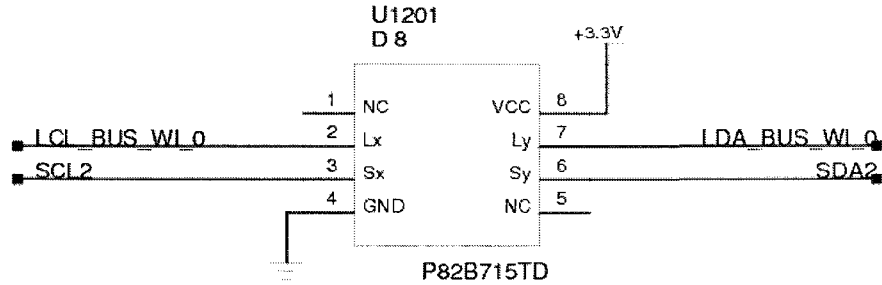


Figure 5.12: I<sup>2</sup>C Buffered Expansion Circuit

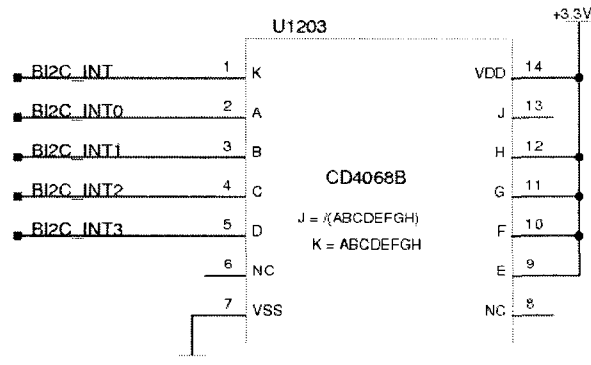


Figure 5.13: I<sup>2</sup>C Buffered Expansion Interrupt Circuit

The buffered I<sup>2</sup>C buses with interrupt lines use an active-low interrupt signal. The intent for these expansion buses is for I/O expansion where a NXP PCA8575D I/O expander chip is used. These chips have an active-low interrupt output that uses an open-drain configuration. Open-drain circuits require an external pull-up resistor, however the system control board does not pull these lines high. For buffered buses not used, the interrupt line should be pulled high using a jumper across the 3.3V line and the interrupt input. To reduce the number of interrupts used in the dsPIC, the buffered I<sup>2</sup>C interrupts are connected to an AND gate. The output of the AND gate is connected to an interrupt of the dsPIC. Each of the buffered I<sup>2</sup>C line is also connected to the dsPIC so the source of interrupt can be determined. On an interrupt event, the dsPIC firmware can check the buffered I<sup>2</sup>C interrupt input lines to determine the source of the interrupt from which the it can then takes appropriate actions.

### 5.1.11 PC Soft Power and Sense Circuits

To ensure the system is completely autonomous, the power of the inspection PCs motherboards is controlled via the system control board. This is accomplished by using the motherboard power switch and motherboard power LED connections. The system control board was designed to provide power control for up to four inspection PCs, one for each quadrant. When the system starts up, it was intended that a power on signal is sent to each motherboard from the system control board. The power state of each PC can be monitored using the power LED connector of the motherboard. In the event of a severe software error in a given quadrant, the ability to restart the PC exists using this soft power interface. The PC soft power circuit is a simple isolation circuit with the photo-transistor connected to the SW connector of the motherboard. The positive side is connected to the collector of the photo-transistor and the emitter is connected to the negative side of the connector. This was verified on an ATX motherboard to ensure it would work as expected, however it was later determined that the powering input of ATX motherboards is not standardized. For the motherboards used for the prototype, the powering circuit required the PWR.SW\_+ pin be pulled to ground to signal the power circuitry of the motherboard. Thus, modifications to the power signal cable were made to facilitate this requirement. The PC soft power sense circuit also uses an optocoupler isolation circuit. The power LED connection of the motherboard is connected to the LED of the optocoupler. When the PC is ON, the photo-transistor of the optocoupler will be conduction, pulling the input to the dsPIC to ground. GPIO of the dsPIC are used for these circuits since timing and events are not critical. The PC soft power circuits are shown in Figure 5.14 and Figure 5.15.

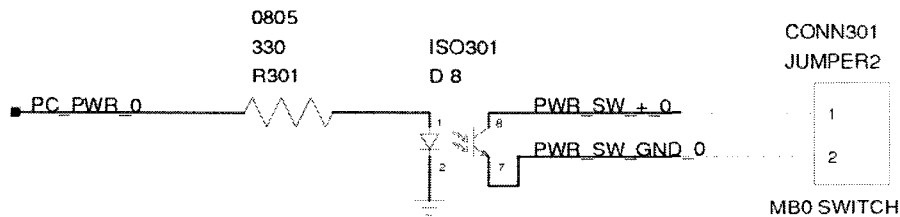


Figure 5.14: Inspection PC Soft Power Circuit

### 5.1.12 I<sup>2</sup>C Bus Switch Circuit

To facilitate communication between the master camera of each quadrant and the system control board, I<sup>2</sup>C is used. Four input I<sup>2</sup>C buses from each quadrant are connected to a custom designed bus switch where the active bus can be selected using two control signals, A0 and A1 respectively.

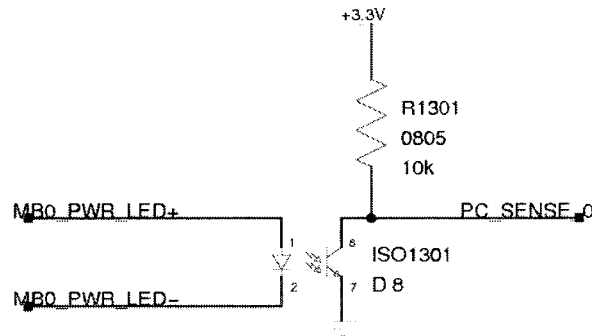
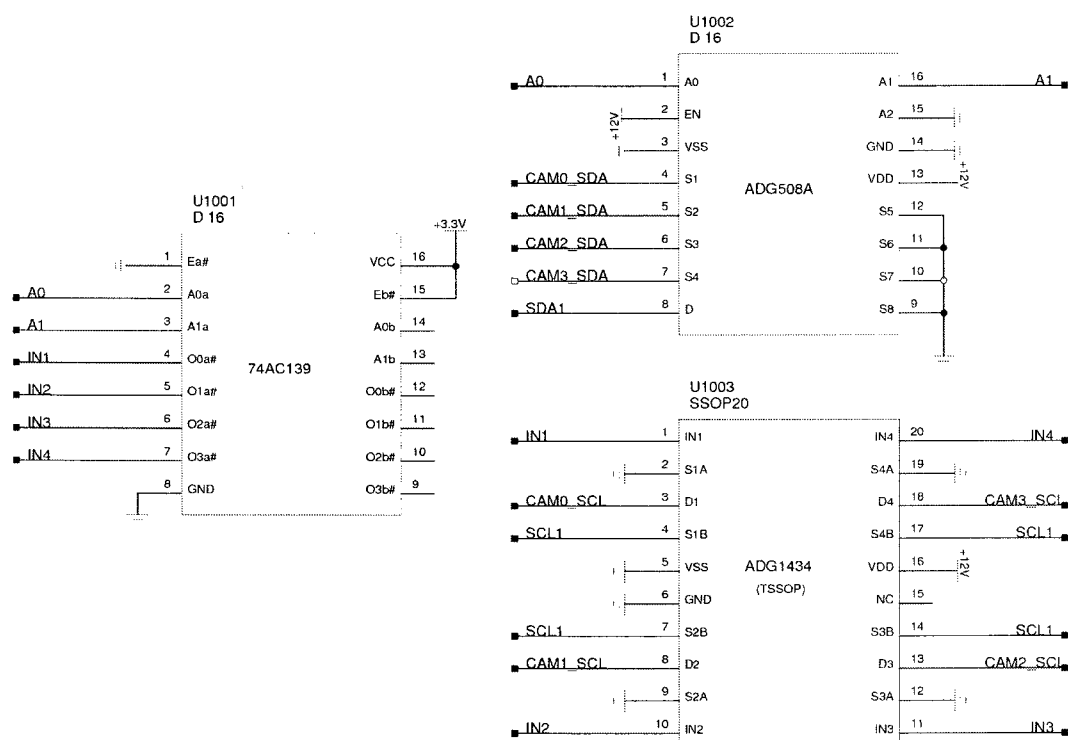


Figure 5.15: Inspection PC Power Sense Circuit

This is achieved by using an analog switch and an analog multiplexer. The design scheme involves one I<sup>2</sup>C module of the dsPIC that is operating in slave mode and the manipulation of the control signals, an individual camera I<sup>2</sup>C bus can be connected to the dsPIC I<sup>2</sup>C slave module. The serial-data lines (SDA) of the cameras are connected to an analog multiplexer (ADG508A). The A0 and A1 pins of the ADG508A are used to select one of the four channels to connect to the dsPIC SDA line. In order for the cameras to honour the multiple master, single slave system, the serial-clock (SCL) line of each of the cameras is pulled low, except for the one switched to the dsPIC. This is accomplished using a double pole, single throw analog switch (ADG1434). The DPST analog switch has four inputs, one for each of the analog inputs. Using an decoder (74AC139), the A0 and A1 lines can be used to select a single line of the DPST switch (IN1 through IN4). The circuit is shown in Figure 5.16.

The supply voltage of both the analog multiplexer and the analog switch is at 12V. These devices will not operate properly at 3.3V and thus must be powered using the 12V supply. Since the analog switch is used for a digital signal, any signal degradation will be negligible considering the application. Fortunately, the system control board was equipped with a 12V supply that could be used to power these devices.

The dsPIC slave I<sup>2</sup>C module input channel is switched between the four cameras at timed intervals until communication by one of the devices is initiated. If communication is initiated by the active channel, the dsPIC will listen exclusively to the given camera until the transaction is complete. While the dsPIC is listening to a given camera, the SCL line of the others is pulled low to indicate to the other cameras not to initiate communication.

Figure 5.16: I<sup>2</sup>C Bus Switch Circuit

### 5.1.13 Emergency Stop Input Circuit

Although using a soft emergency stop is not permitted in many industrial settings for obvious safety reasons, the inclusion of soft emergency stop in the system control board can be used to halt the system for other safety events such as the opening of panel doors during operation. The E-Stop circuit is a series of four isolation circuit, each connected to the input of an AND gate. The emergency stop inputs are active high, and isolated using optocouplers (MOCD208M). The isolation circuit is rather straightforward, see Figure 5.17. When an E-Stop input goes high (assuming all others are low), the LED of the respective optocoupler causes the photo-transistor to conduct pulling the input to the AND gate low. The output of the AND gate, which is connected to an interrupt of the dsPIC, will go low and generate an interrupt event on the negative edge triggered interrupt. Since the isolated E-Stop inputs are also connected to the dsPIC GPIOs, the source of the interrupt can be determined. Upon an interrupt, the dsPIC firmware can read the individual inputs to determine the source of the interrupt and appropriate actions can be taken.

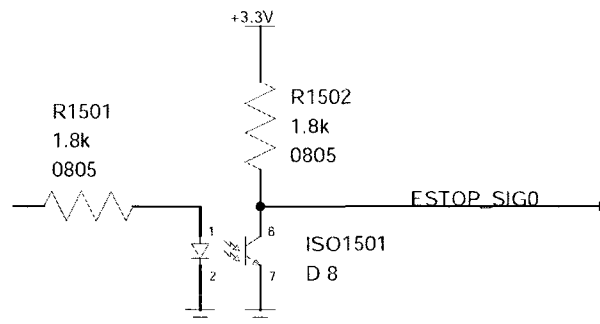


Figure 5.17: E-Stop Input Circuit

### 5.1.14 RS-232 Communication

RS-232 is standard for serial communication between computers and peripherals and is primarily used for low-speed communication systems. Although PC serial ports are being phased out, they are still quite commonly used and are attractive because of the maturity and ease of integration of the protocol. The dsPIC33 has two UART modules which support RS-232 serial communication. The use of a RS-232 transceiver is required to interface between the dsPIC's 3.3V supply level and the required  $\pm 25V$  for RS-232. The transceiver used on the system is a Linear Technology LTC1386 low-power TIA562 transceiver. This device operates at 3.3V and has an internal charge pump to boost the voltage to the required  $\pm 25V$ . The 3.3V level lines are connected to UART module of the dsPIC and the RS-232 side is connected to a DB9 female connector (RS-232 standard).

The RX (receive) and TX (transmit) are the only pins of the RS-232 wires used since the overall communication scheme developed is quite elementary. The transceiver circuit is shown in Figure 5.18.

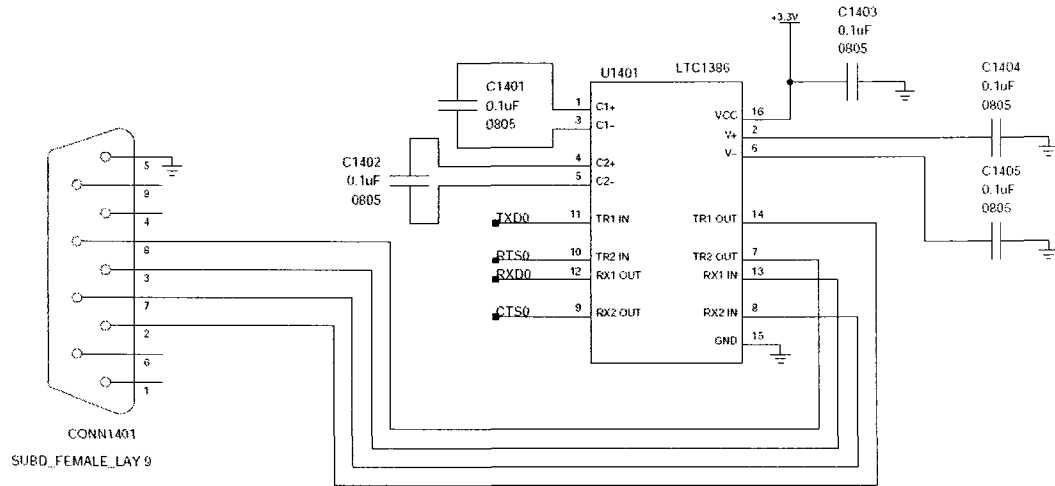


Figure 5.18: RS-232 Transceiver Circuit

## 5.2 System Control Board PCB Layout

After the design of the system control board was verified using breadboard circuits and the dsPIC Explorer 16 evaluation board, the design of a printed circuit board (PCB) could begin. The design of the PCB was not constrained by size or features although it had to promote easy integration with the existing hardware of the PharmaSorter. To achieve this, easy to manage connectors were selected to allow easy wiring. Many screw terminal blocks were used for the higher power hardware such as the pneumatic valves, LED lighting and proximity sensors. For lower voltages or I<sup>2</sup>C communication channels, polarized header connectors were used.

PCB layout can be done using any of a large range of tools. The tools available range from free shareware and open-source tools to high-end professional tools such as Cadence. After evaluating the available tools, gEDA was selected for the design. Since the system control board PCB was not of great complexity, gEDA would be a suitable free software tool that has a relatively easy learning curve. Although Cadence, a professional and high quality tool, was available, it was not used due to its complexity and steep learning curve.

### 5.2.1 gEDA Open-Source Tools

gEDA is an open-source project for electronic design automation. The available tools include a schematic capture application, PCB layout tool, simulation interface and more. The tools used for the design of the system control board were *gschem* for creating the schematic, and *pcb* for creating the layout. The gEDA tools are not terribly difficult to understand and use, and contain many of the required component footprints used in the design. The tool did however require some manual configuration of text files. The PCB layout tool can use custom footprints which must be entered in text format. Some custom footprints had to be created for the PCB layout of the system control board. When creating the schematics using *gschem*, the footprint of the component is specified. After the reference designators are set for the components, an application called *gsch2pcb* can be used to generate a *pcb* layout file which uses the footprint defined in the schematic editor for the object. When the PCB file is opened using *pcb*, the component footprints are loaded without connections. The netlist file which is created by *gsch2pcb* must be loaded from which the nets can be connected manually or using the auto-route tool.

Manual routing was used for the layout of the system control board since it provides greater control. The initial floor-planing of the board required the general placement of the components to maintain a neat and organized appearance. This should be done in an intelligent manner to help ease the task of routing. Since the system control board has many small localized sub circuits, the routing for these sections was done in isolation, and then connected to the rest of the circuit. Some other considerations regarding noise were taken into account during layout design. Potential noisy circuits were isolated from the dsPIC MCU as much as possible. The potentially noisy voltage regulation circuits and the high current switching circuits were placed as far away from the dsPIC as possible.

### 5.2.2 PCB Fabrication

After the design was complete and verified, Gerber files were created and sent to be manufactured. Gerber files are the standard used by PCB manufactures. These files contain all the information pertaining to traces, vias, holes and land patterns. The standard Gerber format is RS-274X. The PCB fabricator PCB Express of Mulino Oregon was selected to fabricate the system control board. The automated system made submission seamless. A quantity of four was ordered to allow for human error and backup boards.

PCB Express, like most other board manufactures, offer different board finishing. Although slightly more expensive, the professionally finished PCB with a solder-mask and silkscreen was ordered. This not only is more aesthetically pleasing, it has the benefit of making soldering easier. The solder mask helps prevent solder bridging during population and the silkscreen provides labels



of the location of the various components. Since the system control board is has a fairly large BOM, this was beneficial. The cost per unit for an order of four circuit boards with dimensions of 6.65"x5" was \$68.75.

### 5.2.3 PCB Population

For the quantity of PCBs ordered, automated population could not be warranted considering the soldering was manageable in-house. A high-end Weller soldering iron was purchased for the project that was suitable for surface mount work. The digital variable temperature control was useful in ensuring the soldering temperature maximums were not exceeded.

The components of the system control board were available and ordered exclusively from Digikey. The most difficult component to populate was the dsPIC MCU. The dsPIC device selected was available in a 100-pin TQFP (Thin Quad Flat Pack). This particular package has a 0.5mm pitch. The technique used to solder this device was to first align the device, apply flux to the leads of one side (using a flux dispensing pen), and touch the iron to a corner pin using the solder already tinned on the footprint. This will fix the component for the rest to be soldered. Following the same technique for the rest of the pins are soldered to the board. This technique does not require additional solder and is likely the simplest method. After complete, the joints are verified using magnifying equipment to ensure no cold solder joints or poorly connected pins exist.

During the population process, circuits were verified to ensure everything was properly seated. The dsPIC was first device tested after the required components for it to operate were soldered, ie. crystal, decoupling capacitors, 3.3V regulator, etc. The Microchip programmer was used to ensure it could program the dsPIC. Following this, the other circuits were populated accordingly and tested during this process.

The final populated system control board PCB is shown below in Figure 5.19.

## 5.3 I<sup>2</sup>C I/O Expansion Board

A highly desirable characteristic of the system control board was for it to possess expandability. However, the extent of expansion is limited by the processor used. One of the most prominent deficiencies for a device such as the system control board is limited I/O. Since the system designed is interfacing with components that typically require higher voltages and significant driving capabilities and are of unpredictable size, incorporating additional generic I/O on the system control board would be impractical. Thus, an I/O expansion board was designed to facilitate this. When considering the design of the I/O expansion board, a practical number of I/O available was considered, along with the desired voltage range. The I<sup>2</sup>C I/O Expansion board operates at voltages ranging from 5V to

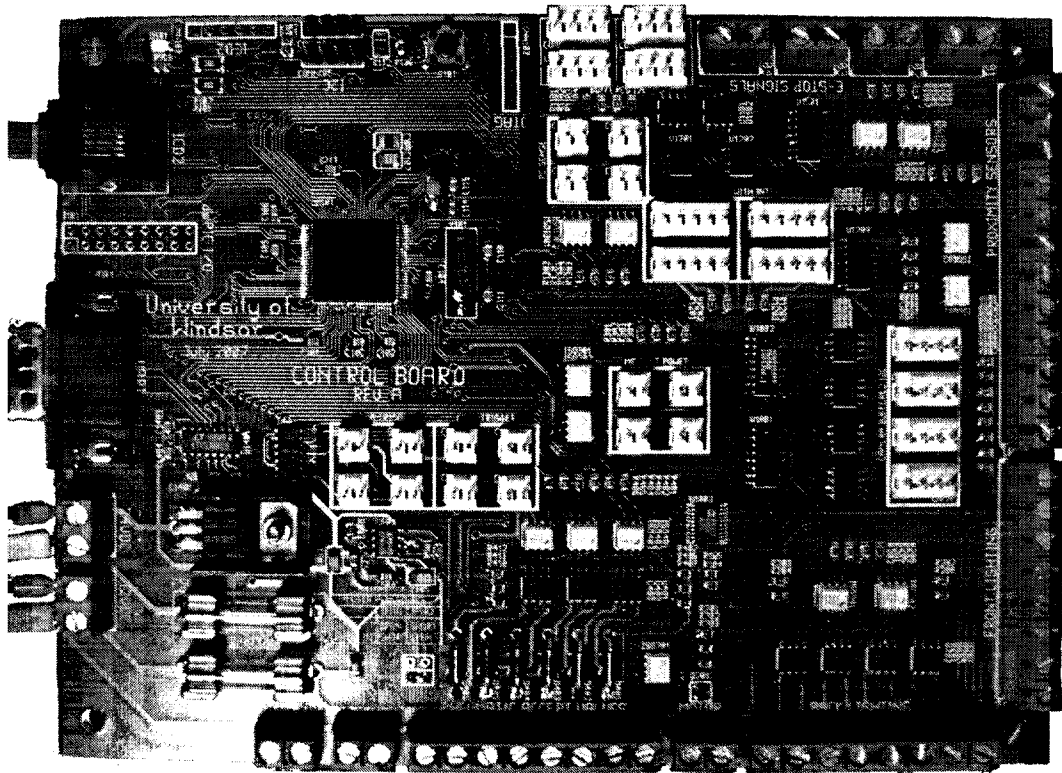


Figure 5.19: Populated System Control Board PCB

35VDC with a 3.3V I<sup>2</sup>C communication interface.

The I/O expansion board uses an I<sup>2</sup>C I/O expander from NXP semiconductors. The NXP PCA8575D is comprised of 16-bit quasi-bidirectional I/O that can be read and set over an I<sup>2</sup>C interface. The PCA8575D also has an interrupt line which can be connected to the interrupt line of the microcontroller. This allows for event driven code which is much more efficient over polling techniques. The NXP I/O expander chip is the heart of the I/O expansion board. Surrounding it are isolation and driving circuits to permit a wide range of voltage levels with high driving capabilities. The PCA8575D operates between 2.3V and 5.5V, and thus since the system control board microcontroller operates at 3.3V, the I/O expansion board does also. The I<sup>2</sup>C line of the I/O expansion board is buffered using an NXP P82B715 I<sup>2</sup>C bus extender chip. This promotes improved signal integrity which is preferred for long distance transmission lines.

The inputs to the I/O expansion board turn on the LEDs of opto-isolators, as shown in Figure 5.20. As a result, the photo-transistor will conduct pulling the quasi-bidirectional pin of the NXP I/O expander chip low. The inputs of the I/O expander chip used are P00 through P07. Any change on an input will cause the I/O expander chip to generate an interrupt from which the

MCU can read the state of the I/O. The interrupt of the NXP I/O expander chip is an open-drain configuration and thus requires an external pull-up resistor. This is pulled high via a 2.2k  $\Omega$  resistor on the I2C I/O expansion board, *not* the system control board.

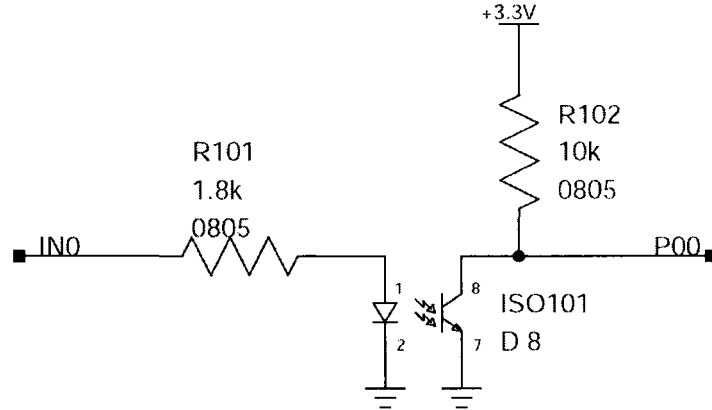


Figure 5.20: I<sup>2</sup>C I/O Expansion Board Input Circuit

The outputs of the I/O expansion board are generated from the NXP I/O expansion chip quasi-bidirectional I/O pins P10 through P17. These outputs are best for sinking and thus are connected to the cathode of the LED of the opto-isolator through a current limiting resistor. Thus, when the output of the I/O expander chip is set LOW, the LED is on, and thus the photo-transistor is conducting. This low voltage is seen at the gate of the P-Channel power MOSFET causing it to conduct thus driving the load at the output. The default state of the I/O of the I/O expander chip is all high. Thus, the initial state of the outputs is OFF. For safety reasons, this is desirable. The typical output circuit is shown in Figure 5.21.

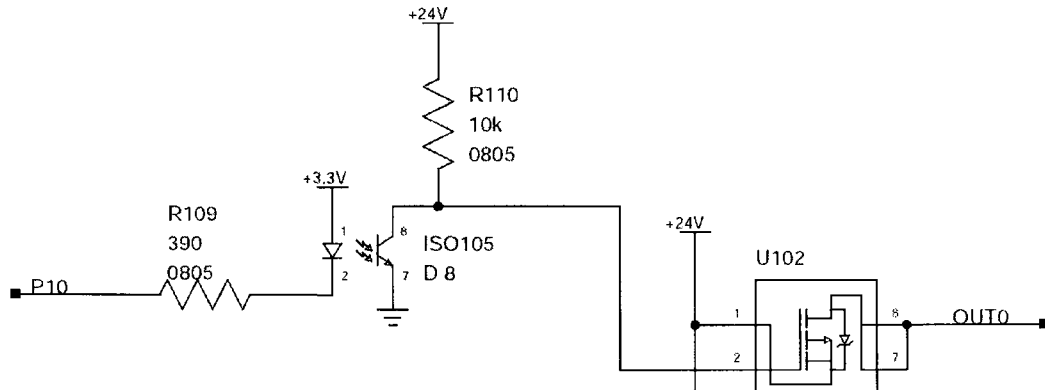


Figure 5.21: I<sup>2</sup>C I/O Expansion Board Output Circuit

To facilitate easy integration and connectivity of the I/O expander board to the PharmaSorter, screw terminal blocks were used to connect the inputs and outputs to the I/O expansion board. The I/O power supply connector and the optional external 3.3V supply connectors are also screw terminal blocks, but of different colour as to not be confused with I/O. A 5-pin connector is used to connect the buffered I<sup>2</sup>C bus, the interrupt line and ground to the system control board. A polarized connector is used to prohibit incorrect connection.

Since it may be desired to have several I/O expansion boards co-existing in the same system, a set of jumpers is used to select the I<sup>2</sup>C slave address of the I/O expansion board. The NXP I/O expander chip allows for up to eight addressable devices to exist on the same bus, resulting in three address inputs, A0, A1 and A2. These inputs can be set high or low by setting the jumpers accordingly. The address pins are each pulled low via a 10k $\Omega$  resistor, and shorting the jumper shorts the input high. This circuit is shown in Figure 5.22.

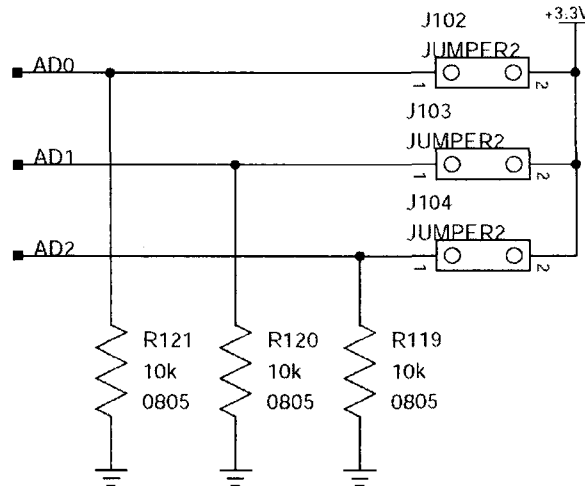


Figure 5.22: I<sup>2</sup>C I/O Expansion Board I<sup>2</sup>C Address Select Circuit

To promote flexibility with the I/O expander board in terms of I/O voltage levels, the optional 3.3V regulator on the board allows a maximum input of 24V. Thus, if a voltage greater than this is required, a DPAK<sup>2</sup> footprint exists for an optional 24V linear regulator. If this is not used, the input and output pads of the DPAK<sup>2</sup> can conveniently fit a 0 $\Omega$  2010 resistor.

The 3.3V supply can be from an external source either on the 5-pin I<sup>2</sup>C connector, or from the 2-terminal screw block connector. It can also be regulated on the I/O expansion board using the Linear Technologies LT1121CS8-3.3 3.3V low-drop out regulator. This particular regulator allows for a maximum input voltage of 30VDC, capable of supplying 150mA.

An optional footprint exists on the I/O expansion board for a Texas Instruments TMP175

I<sup>2</sup>C temperature sensor. This sensor can be used to monitor ambient temperature in the vicinity. The I<sup>2</sup>C slave address of this sensor shares the address pins used for the NXP I/O expander chip. Additional two-row headers exist on the board to allow for expansion if the existing isolation circuits are inadequate for a particular application. A separate header exists for P00 through P07 and for P10 through P17. Each header has single pin removed in different locations, commonly referred to as a key, to prevent incorrect connection and to polarize the connector. The headers also have 3.3V and ground connections. The header connectors are shown in Figure 5.23.

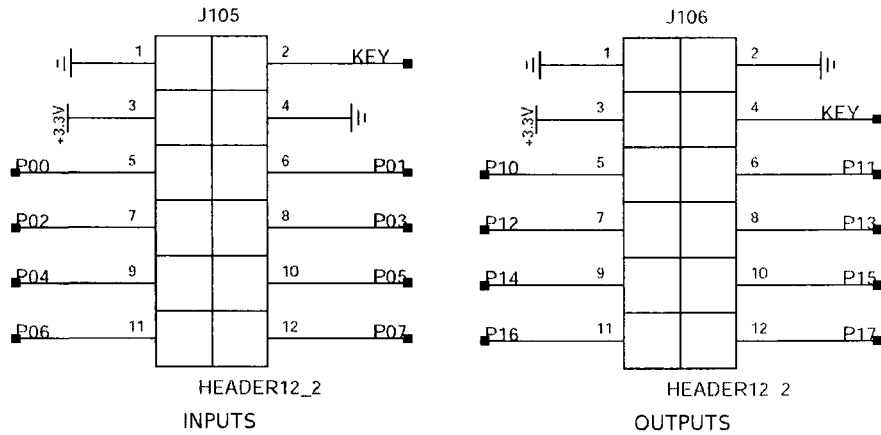


Figure 5.23: I<sup>2</sup>C I/O Expansion Board Input Circuit

A PCB was designed for the I<sup>2</sup>C I/O expansion board and submitted for fabrication. Like the layout of the system control board, the I/O expansion board PCB was designed using gEDA. The design involved surface mount components with the smallest IC package being SOIC, and the smallest resistor size being 0805. The intent in the layout was to achieve the smallest board size while maintaining reasonable component sizes. The terminal block connectors largely defined the size of the board. The resulting PCB size was a 4.1 in. x 2.2 in. (104.14mm x 55.8mm). The final I<sup>2</sup>C I/O expansion board PCB is shown in Figure 5.24. For detailed operating instructions of the I<sup>2</sup>C I/O expansion board refer to Appendix ??.

## 5.4 Firmware Development

Firmware is a computer program the runs a dedicated application for given hardware. The dsPIC33 microcontroller is the only programmable device on the system control board, and thus is the only component for which firmware was required. Firmware development for the system control board was developed primarily in C. Occasionally, assembly was used for timing dependent tasks and for

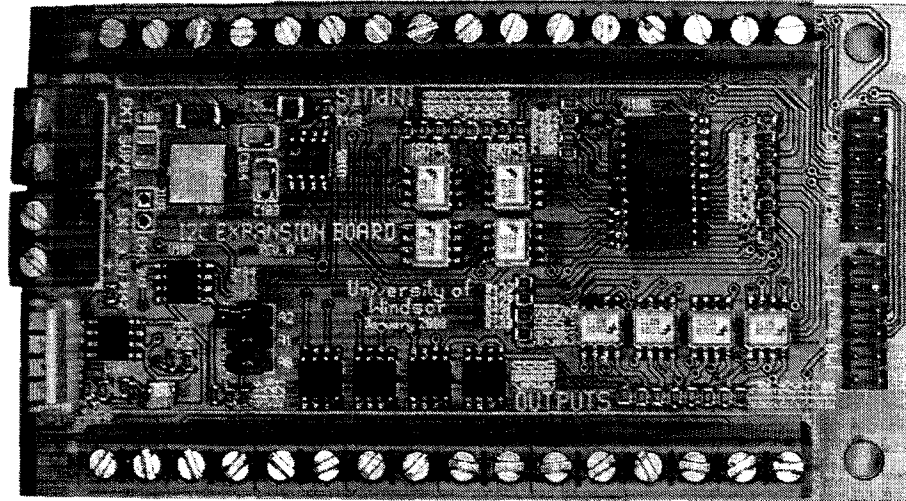


Figure 5.24: I<sup>2</sup>C I/O Expansion Board

optimization. The Microchip C30 compiler is a fully ANSI compliant C compiler for 16-bit Microchip MCUs. The system control board firmware was developed using the MPLAB integrated development environment (IDE) for managing project files and the C30 compiler was used to generate the machine code.

Firmware development can be a daunting task if preliminary planning stages are neglected. The firmware design of the system control board was an iterative process broken into sections. The majority of the firmware is event driven using interrupts, timers, output compare and input capture modules of the dsPIC. Event driven design increases performance vastly over polling techniques, however, to reduce complexity and development time, some functions use polling. The overall goal of the system control board firmware was a design that was a good compromise between performance and moderate development time.

The system control board firmware development was an iterative process. By taking small steps and ensure the functionality of each block individually, it simplified the implementation of the entire application. When a given function was to be developed, it was isolated from the main program, debugged and verified that it satisfied the expectations. As each block was completed, it was added to the main program where it was verified once again. Developing for embedded systems can be more complicated than developing on a workstation because debugging becomes more involved. Because of this, it is extremely important that things are well thought out to minimize debug time.

### 5.4.1 Functional Requirements

The system control board is required to handle various tasks including: controlling motor speed, lighting, camera triggers, capsule ejection, communication with cameras, RS-232 communication and operate a simple HMI. For each of these tasks, the interaction between hardware and software had to be distinguished. For many of the required tasks, a hardware module of the dsPIC was used to reduce MCU overhead and thus improve overall performance. For example, the motor speed is controlled by the frequency of the square wave generated by an output of the dsPIC. Exploiting an output capture pin, the pulse train is automatically generated with no processor overhead.

### 5.4.2 Motor Control

The PharmaSorter uses a 5-pole stepper motor with a stepper motor controller. This allows for easy operation of the motor. Using a stepper motor along with a proximity sensor feedback, the position of the capsule is always known based on the number of steps applied to the motor after a transition of the output of the proximity sensor. The proximity sensor is located near a disk with grooves for each capsule holder. Thus, each time a groove passes the proximity sensor, a pulse is seen by the dsPIC and an interrupt is generated. The motor control method is quite simple. Using an output compare module of the dsPIC, and a timer, the frequency and width of the pulses applied to the motor controller can be controlled.

Setting up the output compare module is rather straightforward and is performed during system initialization. The system initialization initializes all timers, output compares, input compares, interrupts and I/O in a function called *init\_sys()*. The code listing for the initialization of the motor pulse control portion is shown below (see Listing 5.1).

```
/* Configure Timer2 for Output Capture */
T2CONbits.T32 = 0;
PR2 = (unsigned int) (((double) Fcy) / (double) motor_speed) - 1.0) / T2PF;
T2CONbits.TON = 1;
T2CONbits.TCKPS = T2TCKPS;

/* Enable Output Compare to generate PWM for motor control */
OC_MTR_CTRL_CONbits.OCM = OCM_DISABLED; /* Initially OFF */
OC_MTR_CTRL_CONbits.OCTSEL = OCTSEL_TIMER2; /* Use Timer2 */
OC_MTR_CTRL_R = 0;
OC_MTR_CTRL_RS = MTR_PULSE_WIDTH;
OC_MTR_CTRL_IF = DEASSERTED;
OC_MTR_CTRL_IE = TRUE;
```

Listing 5.1: Motor Control Initialization

The code listing also includes the configuration of Timer2. Timer2 is the base frequency of the output compare module and is thus used to control the frequency of pulses applied to the stepper motor controller. The points of interest here are the PR (period register) and the TCKPS (timer

clock pre-scaler) registers. The period register controls the frequency and the pre-scaler is used to select a divider for the system clock. For simplicity, the pre-scaler is fixed to a value that achieves a suitable range for the system. The pre-scaler for Timer2 (T2TCKPS) is set to 0x03, which translates to a pre-scale factor (PF) of 256. Thus, the global clock is divided by 256 for Timer2, resulting in an effective timer clock frequency of 156.25 kHz, for a system clock of 40MHz. The following formula is used to determine a period value for a given frequency, this is shown below where *motor\_speed* is in Hz.

$$PR2 = \frac{f_{cy}}{(motor\_speed - 1.0) \times TCKPS}$$

Note the frequency of the dsPIC is  $f_{cy} \approx 40MHz$ . This frequency is based on the input crystal (Y1), and the PLL (phase-lock-loop) registers. The external oscillator operates at 8MHz, and using the selected PLL parameters, a system clock of roughly 40MHz is achieved. With the selected parameters for the output compare module, the maximum pulse rate is about 3.9kHz, and the minimum is about 3Hz. The maximum pulse rate depends on the pulse width of the motor pulse,  $t_{on}$ . After experimental trials it was determined that a pulse duration of about  $250\mu s$  was suitable. The parameters selected fulfill the speed range of the system and limits the maximum speed. It should also be noted that the output compare mode (OCM) is initially disabled, this will start the system with the motor disabled, which is desirable for obvious safety reasons.

The motor control function is related to the lighting and camera trigger controls. From experimentation, it was determined that there are about 495 to 505 motor pulses applied between adjacent holders. In order to properly calibrate camera triggers and accept ejection, various positions are recorded. By using a counter, *pulse\_counter*, that increments each time a pulse is applied to the stepper motor controller, the exact location of the capsule holder is known. The counter is reset once the proximity sensor edge is detected, and hence the next capsule is entering the inspection area. The flowchart below illustrates the motor pulse counter interrupt service routine (ISR) and the ISR generated by the proximity sensor, as shown in Figure 5.25 and Figure 5.26 respectively.

From the flow diagrams, the overall motor control is rather straightforward. Being entirely interrupt driven promotes efficiency, however constantly checking the *pulse\_counter* value each step is not the most efficient, it is however the most convenient method of coding this section. Also, the required CPU time is rather insignificant. Take the system operating at a motor pulse rate of 2200Hz, to achieve 4.2 capsules/sec (target inspection rate), and assuming checking the *pulse\_counter* value each step requires 20 clocks, the relative CPU time is calculated below.

$$CPU\% \approx \frac{\Delta t_{PULSES}}{\Delta t_{CHECK\_PULSE\_COUNT}}$$



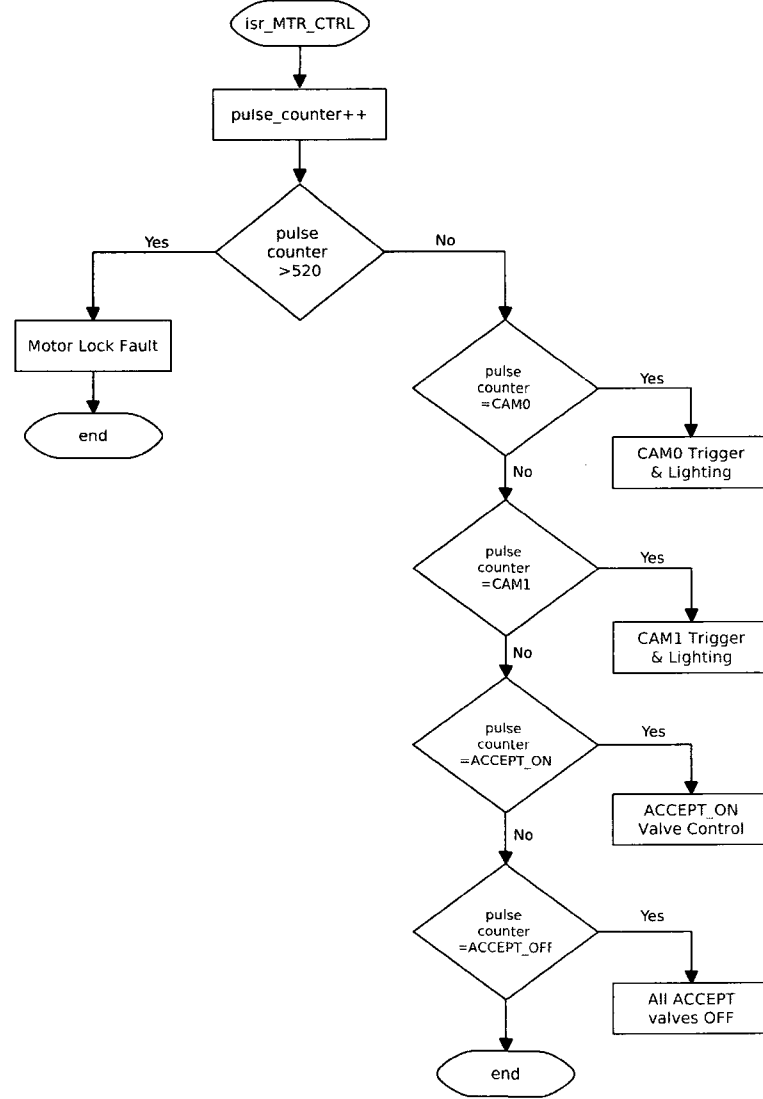


Figure 5.25: Motor Pulse Control flow diagram

where,

$$\Delta t_{PULSES} = \frac{1}{2200Hz} = 455\mu s$$

and,

$$\Delta t_{CHECK\_PULSE\_COUNT} = \frac{1}{40MHz} \times 20clocks = 500ns$$

thus,

$$CPU\% \approx \frac{500ns}{455\mu s} = 0.1098\%$$

From this quick calculation, it is evident that the impact of this operation is quite negligible.

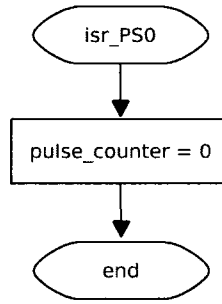


Figure 5.26: Proximity Sensor Interrupt flow diagram

### Proximity Sensor Compensation

The proximity sensor used in the PharmaSorter is a capacitive sensor and thus is not ideal for high-speed sensing. Thus, a delay is introduced which is proportional to the speed of the motor. To alleviate this problem, a compensation factor is used to adjust for this. The compensation factor is based on experimental results solely. The compensation factor is based on the frequency of the motor and the calculation is shown below.

$$PS_{comp} = (0.12 \times \omega_{motor}) - 34$$

### 5.4.3 Motor Ramping Control

It was obvious that once the motor pulse code was tested that something had to be done about the sudden starting and stopping. Due to the inertia of the machine when rotating, the motor had to be ramped up and down to prevent the stepper motor from slipping. The stepper motor slipping not only sounded unpleasant but it could potentially cause damage to the controller or the motor itself. The ramping technique was accomplished rather easily using a timer, namely Timer5, and a few control variables. The *ramp\_mode* variable is set to either *RAMP\_UP* or *RAMP\_DOWN* and Timer5 is enabled. Each timer event, the motor speed is either incremented or decremented accordingly until the target speed is met. The timer interrupt service routine is shown below in Listing 5.2.

```

/* Clear Interrupt Flag */
IFS1bits.T5IF = DEASSERTED;

/* If Ramp Up */
if (ramp_mode == RAMP_UP) {
    motor_speed++;
    PR2 = (unsigned int) (((((double) Fcy) / (double) motor_speed) - 1.0) / (double)
        T2PF);
    PNEU_MAIN = DEASSERTED;
}
  
```

```

/* If Ramp Down */
if (ramp_mode == RAMP_DOWN) {
    motor_speed--;
    PR2 = (unsigned int) (((double) Fcy) / (double) motor_speed) - 1.0 / (double)
        T2PF);
}

/* If desired speed reached - stop this timer */
if (motor_speed == motor_speed_target) {
    T5CONbits.TON = FALSE;
    enable_count = 0;
    if (motor_stop_flag) {
        disable_count = 0;

        OC_MTR_CTRL_CONbits.OCM = OCM_DISABLED;
        PNEU_MAIN = ASSERTED;
    }
}

```

Listing 5.2: Motor Ramp Timer Routine

By employing the motor ramping technique, the motor will not slip as it accelerates or decelerates preventing wear on the machine and preventing damage to the stepper motor and controller. Although the constant calculation of the period register of Timer2 (PR2) is being performed, this is only done during motor start and stop and has an insignificant affect on the normal operation of the system.

#### 5.4.4 Camera Trigger Control

Because the cameras are operating in Global Shutter Control Mode (see Chapter 4), the width of the trigger pulse must be tightly controlled. This is accomplished by using an output compare module. The required pulse width of the camera trigger is  $392\mu\text{s}$  for the given PXCLK frequency of 48MHz [22]. To achieve this, the following values are setup for the output capture. Note that the camera and lighting output compare modules use Timer3 of the dsPIC, and Timer2 is reserved for motor control since the frequency is varying. The initialization of Timer3 and the output compare module for the camera trigger is shown below. Note that there are two separate camera triggers (CAM\_TRIG0 and CAM\_TRIG1) used in the system and the code for only a single trigger is shown below (see Listing 5.3).

```

/* Configure Timer3 for Output Compare */
PR3 = 0xffff;
T3CONbits.TCKPS = 2;
T3CONbits.TON = TRUE;

/* Camera Trigger0 Output Compare */
OC_CAM_TRIG0_CONbits.OCM = OCM_DISABLED;
OC_CAM_TRIG0_CONbits.OCTSEL = OCTSEL_TIMER3;
OC_CAM_TRIG0_R = 0;
OC_CAM_TRIG0_RS = CAM_TRIG0_PULSE_WIDTH;

```

## Listing 5.3: Camera Trigger Output Compare Initialization

Since Timer3 is used for both the camera triggers and lighting control, a suitable pre-scaler of 2 is selected. This will divide the clock by 64, for an effective frequency of 625kHz for a resolution of  $1.6\mu s$ .

When a camera trigger pulse is to be applied to the camera(s), the following code sequence restarts Timer3 and sets the output compare mode of the given trigger into single pulse mode (OCM\_SINGLE\_PULSE). The code listing is shown in Listing 5.4.

```
/* Enable Trigger1 and BL1 strobe */
/* Reset Timer 3 for Backlight control and Camera Trigger Control */
T3CONbits.TON = FALSE;
TMR3 = 0;

/* Enable Single Pulse Mode for LED Backlight Output Compare */
OC_LED_BL1_CONbits.OCM = OCM_SINGLE_PULSE;
OC_CAM_TRIG1_CONbits.OCM = OCM_SINGLE_PULSE;

T3CONbits.TON = TRUE;
```

## Listing 5.4: Trigger and Backlight Strobe Routine

The code listing also enables the back-light output compare for camera 1. Obviously when the camera is triggered, the light strobe is to be initiated to illuminate the object. The pulse width of each can be different, although they are dependent on the same timer. The R and RS registers are used to set the on and off value corresponding to Timer3.

For a typical inspection of a clear capsule, a back-light strobe duration of about 1.25ms is required for adequate illumination. This translates to a value of about 800 in the output compare secondary register, if the primary register is 0. The strobe duration is an important parameter to ensure that motion blur is eliminated. Thus, a very intense strobe is required for a very short duration. In addition, by using amplification techniques in the image sensor, light intensity can be compensated for.

### 5.4.5 I<sup>2</sup>C Master

The system controller I<sup>2</sup>C master module is used to manipulate external I<sup>2</sup>C slave devices such as I/O expanders and temperature sensors. One of the primary reasons for including I<sup>2</sup>C expandability is to reduce the I/O requirement of the system controller MCU. Considering the PharmaSorter is a prototype, it is desirable to avoid restricting the design. The available expansion options allow for additions to the system with relative ease. The dsPIC33 controller used has two I<sup>2</sup>C modules that can operate in either master or slave mode. The I2C2 module is used for the master and the I2C1 module is used for the slave functions. By exploiting the built-in I<sup>2</sup>C controller in the

dsPIC, there is a reduction in processor overhead as well as a simplification in development. The I<sup>2</sup>C master module was coded in a polling fashion to reduce programming time. This is chiefly due to the fact the the communication over the I<sup>2</sup>C master bus is quite minimal. I<sup>2</sup>C is merely used for updating the LCD, and attending to events from the I<sup>2</sup>C I/O expansion boards (which are interrupt driven). The control of the I<sup>2</sup>C master module is accomplished with several key functions: *init\_i2c2()*, *i2c2\_write()* and *i2c2\_read()*. The initialization of the I<sup>2</sup>C module in master mode involves enabling the module and enabling the master interrupt. The I<sup>2</sup>C frequency is also set via the I2CBRG register whereby the frequency of the SCL (serial clock) is determined from the following formula.

$$I2CxBRG = \left( \frac{f_{CY}}{f_{SCL}} - \frac{f_{CY}}{1,111,111} \right) - 1$$

Thus, for a  $f_{CY} = 40MHz$  and a desired I<sup>2</sup>C clock frequency of  $f_{SCL} = 100kHz$ , a I2C2BRG value of 363 is required.

The *init\_i2c2()* function will set the I<sup>2</sup>C clock frequency based on the above formula and set other registers to configure the I<sup>2</sup>C master interrupt. The interrupt routine sets a flag (*jDone*) to indicate that the previous request was completed, or that an event has occurred. As mentioned earlier, the I<sup>2</sup>C read and write functions, *i2c2\_read()* and *i2c2\_write()*, were programmed in a polling fashion. Thus, the MCU is “stuck” in the particular function until the sequence has completed. The *i2c2\_read()* function returns an integer upon completion. A zero is returned on success and a -1 on failure. Similarly, the *i2c2\_write()* function follows the same return method. The *i2c2\_read()* function takes an address of the I<sup>2</sup>C slave address, a buffer to store the data and the length of data expected. Using timeouts, the function is guaranteed to return even if there is a severe bus failure. This is similar in the *i2c2\_write()* function. A detailed description of the three functions used for engaging I<sup>2</sup>C communication is listed in Table 5.2.

Function	Description
<i>init_i2c2()</i>	Initialize the I2C2 module of the dsPIC33 in master mode with a SCL frequency of 100kHz
<i>i2c2_read(unsigned char addr, unsigned char *buf, unsigned char len)</i>	Perform an I <sup>2</sup> C read request to the slave device at address <i>addr</i> for <i>len</i> bytes and stores the result in <i>*buf</i> . Returns 0 on success, or -1 on error.
<i>i2c2_write(unsigned char addr, unsigned char *buf, unsigned char len)</i>	Performs an I <sup>2</sup> C write request to the slave device at address <i>addr</i> for <i>len</i> bytes with the data in <i>*buf</i> . Returns 0 on success, or -1 on error.

Table 5.2: dsPIC I<sup>2</sup>C Master Functions

### 5.4.6 I<sup>2</sup>C Slave

The I<sup>2</sup>C slave module is used for retrieving messages from the four quadrants and responding to requests. This is accomplished using a time multiplexed I<sup>2</sup>C bus switch system whereby each of the I<sup>2</sup>C buses is allocated a certain amount of time to initiate communication with the system controller. This is detailed in the Section 5.4.7. The dsPIC33 I<sup>2</sup>C slave module requires an I<sup>2</sup>C slave address and must acknowledge the master when addressed. The I<sup>2</sup>C slave module requires an initialization function whereby the slave-mode interrupts are set for the modules. The slave address is set via the I2C1ADD register and was arbitrarily set to 0x44, since no other device in the system use this address. The slave module code is stored directly in the interrupt routine. The interrupt routine must be serviced every time a byte of data is retrieved by the dsPIC33 where the slave address of the message matches the I2C1ADD register. This occurs immediately after retrieving the slave address and after each byte subsequent data. The routine must acknowledge this event and either send data or be prepared to receive data based on the read/write flag. Because the requirements of the system controller, the pass/fail feedback mechanism is not terribly complex and therefore the routine only needs to respond to three requests: get capsule ID request, set pass/fail result and set PC ready flag as described below.

- **I2CCMD\_GET\_CAP\_ID** - The master is requesting the capsule ID of the current capsule. The slave responds with a 8-bit value that for the 4-bit capsuleID padded with zeros.
- **I2CCMD\_SET\_PF** - The master will set a specific capsule ID pass/fail flag. The received byte contains a 4-bit capsule ID and a 2-bit pass/fail code. This result is stored in an array of pass/fail results for the given quadrant.
- **I2CCMD\_SET\_PC\_READY** - Indicate when PC is ready to begin inspecting. This message is sent by the host when the PC software is loaded and the software is ready to begin acquiring images and to begin inspection.

### 5.4.7 I<sup>2</sup>C Bus Switch

Due to the limited number of I<sup>2</sup>C modules in the dsPIC33 MCU, a time-multiplexed system was devised to retrieve messages from the various quadrant master cameras. Although other approaches for this could have been pursued, the time multiplexed bus seems to be one of the simpler solutions primarily since I<sup>2</sup>C was the desired communication bus and that typical MCUs have no more than two I<sup>2</sup>C modules. General purpose I/O could have been used such that dedicated buses existed for each quadrant, however this would require much more complex code and which would not have even been practical. The time-multiplexed I<sup>2</sup>C bus switch system incorporates the use

of an analog multiplexer and an analog SPDT switch as described in Section 5.1.12. The active I<sup>2</sup>C communication channel is selected by switching the SDA and SCL lines of the active I<sup>2</sup>C bus to the dsPIC33 MCU SDA and SCL lines. The inactive channel SCL lines are switched to GND to prohibit communication on those channels. This technique does not interfere with the I<sup>2</sup>C protocol since as long as the I<sup>2</sup>C bus is inactive at the time that the SCL line is pulled to ground, there is no chance of generating a false bus event such as a START or STOP. A total of four pins are occupied in the dsPIC33 for the I<sup>2</sup>C bus switch, two GPIOs and the I<sup>2</sup>C pins SDA2 and SCL2. The bus switch active channel is controlled by the GPIOs labeled A0 and A1.

The firmware controlling the bus switch is comprised of a single timer. When enabled, this timer will check the status of the I<sup>2</sup>C bus to verify it is inactive. If the bus is free, the next channel will be switched for a period of time and continuing in a cyclic fashion, see the flowchart below in Figure 5.27.

When an I<sup>2</sup>C message is received, the I<sup>2</sup>C timer register is decremented by a factor of the time splice allocated. This is accomplished by subtracting the value from the timer TMR register, defined as TMR\_BUS\_SWITCH\_TMR.

In order to calculate a reasonable switching frequency, the time required to satisfy an I<sup>2</sup>C request must be considered. For an I<sup>2</sup>C SCL operating at 400kHz, as in the case of the I<sup>2</sup>C bus from the cameras, with a message containing a 7-bit address, a R/W bit, 8-bits of data, two bits for Start and Stop and two ACKs, a total of 20 clocks are required. Thus, we can approximate the absolute minimum time requirement as shown below.

$$\begin{aligned}\Delta t_{I2C\_TRANS} &\approx \frac{20}{400 \times 10^3} \\ &= 50\mu s\end{aligned}$$

The amount of time between capsules should be considered, as a maximum time requirement. If operating at the target inspection rate of 1000 capsules/minute, the amount of time between capsules is 200ms. With this wide window, a reasonable switching time must be decided upon, it would not be unreasonable to switch once every 1 ms. This way, a full cycle will occur every 4ms, or at a frequency of 250kHz. With the code extending the time splice if an attempt to switch during a transaction, meeting the time requirements should not pose any problems. From experimental tests, this switching frequency exceeds the requirements of the system.

#### 5.4.8 Job Queue

In a real-time firmware running a dedicated application, it is essential that the most important events are attended to first. Scheduling in software is required for multi-tasking applications and important

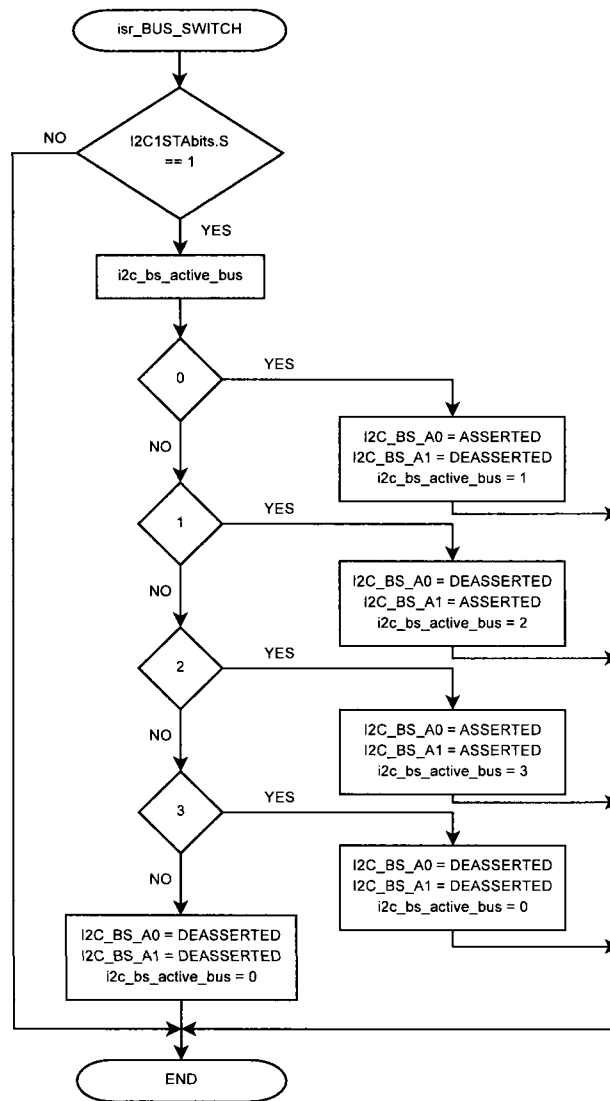


Figure 5.27: Flow Diagram of Bus Switch Interrupt Service Routine

in real-time applications. The priority of a process is an important parameter in scheduling. For the firmware design of the system control board, an rudimentary scheduling technique was devised whereby time critical events are attended to immediately upon their occurrence, and low-priority events being added to a low priority job queue that is serviced in the application main loop. The job queue is used solely for tasks of low importance such as attending to the HMI events, setting LCD text, fulfilling UART requests and managing the soft-power PC control. The job queue system is comprised of two variables, a job structure and a set of four functions associated functions along list of defined job IDs. The job structure is organized as follows (see Listing 5.5).



```

struct JOB {
    unsigned char job_id;           /* Job Identification number */
    unsigned char data[32];        /* Data for associated job */
    unsigned char bc;              /* Number of bytes of data (if used) */
};

```

Listing 5.5: Job Structure

This structure holds all relevant information of a particular job including an identification number from a list of pre-defined IDs, an array of data for holding specific information for a job and a byte count of the number of bytes of the array used. An array of JOB structures was created to hold the queue of jobs. Two variables, namely *curr\_job* and *last\_job* were used to track the jobs in the queue and the ones that have been serviced. A set of four functions, namely *get\_next\_job()*, *get\_next\_curr\_job()*, *add\_job()* and *complete\_job()* were created for adding and removing jobs from the queue. These functions are described in detail below.

- **get\_next\_job()** - Returns the next available job index number.
- **get\_next\_curr\_job()** - Returns the next unserviced job index number in the list.
- **add\_job(unsigned char job\_id)** - Will add a job to job queue and increment the last\_job counter.
- **complete\_job()** - Will remove the last job from the list and increment curr\_job counter.

By manipulating these functions and directly modifying data in the structure array, jobs can be added to the queue rather conveniently. An example of adding a job to write a message to the HMI is shown below (see Listing 5.6).

```

unsigned char tjob = get_next_job();           /* Get next available job index */
job_list[tjob].data[16] = 1;                  /* Select LCD Line to write message to */
sprintf (job_list[tjob].data, "Hello World!"); /* Write message to data array */
job_list[tjob].bc = 12;                       /* Set message size */
add_job(LCD_WRITE);                          /* Add job with JobID code LCD_WRITE */

```

Listing 5.6: Adding HMI Message Job

This is a rather straightforward method of adding jobs without a great deal of processing overhead. The job queue structure allows for flexibility in the types of jobs that can be serviced using this technique. An example of servicing a job from the job queue is shown in the next code snippet. The servicing of jobs is accomplished directly in the main loop as shown in Listing 5.7.

```

while (1) {
    if (curr_job != last_job) {
        switch (job_list[get_next_curr_job()].job_id) {
            ...
            case LCD_WRITE:
                lcd_cursor_to (job_list[get_next_curr_job()].data[16], 0); /* Go to
                    designate line number */

```

```

        lcd_print (job_list[get_next_curr_job()].data, job_list[get_next_curr_job
        ()].bc); /* print message */
        complete_job(); /* remove job from queue */
        break;
...
    }
}

```

Listing 5.7: Executing LCD Write Job

The designed job queue system was a convenient method of servicing low-priority routines of the system control board MCU. It was suitable for all HMI handling, soft PC power control and UART requests. It is flexible in the sense that the addition of new jobs can be integrated with ease.

#### 5.4.9 I<sup>2</sup>C I/O Expansion Board Control

To facilitate control of I<sup>2</sup>C I/O expansion boards (see Section 5.3), several specific routines were created to handle and organize these devices. In particular, a structure was created to hold information pertaining to a specific I<sup>2</sup>C expansion board attached to the system and is described below (see Listing 5.8).

```

struct I2C_IO_EXP {
    unsigned char i2c_addr;          /* I2C Slave Address */
    unsigned char bi2c_channel;      /* Buffered I2C Interrupt Channel */
    unsigned char inp;               /* Input data */
    unsigned char outp;              /* Output data */
    unsigned char ts_i2c_addr;       /* Temperature Sensor I2C Address */
    unsigned char ts_conf_reg;       /* Temperature Sensor Configuration Register */
    unsigned char ts_temp_reg[2];    /* Temperature Sensor Temperature Data */
};

```

Listing 5.8: I<sup>2</sup>C I/O Expander Structure

Several specific functions were created for manipulating the I<sup>2</sup>C I/O expansion boards. These functions are used for reading and setting I/O values, configuring the temperature sensor and retrieving temperature sensor data. The functions created for I<sup>2</sup>C I/O board manipulation are listed below.

- **i2c\_io\_exp\_write** (struct I2C\_IO\_EXP \*io\_exp) - Used to write data stored in specified I/O expander structure to device.
- **i2c\_io\_exp\_read** (struct I2C\_IO\_EXP \*io\_exp) - Used to read data to I/O expander structure of specified device.
- **i2c\_io\_exp\_set\_ts\_conf** (struct I2C\_IO\_EXP \*io\_exp) - Used to set the configuration register of the temperature sensor of the I/O expander specified.

- **i2c\_io\_exp\_get\_ts\_conf** (**struct I2C\_IO\_EXP \*io\_exp**) - Used to read the configuration register of the temperature sensor of the I/O expander specified.
- **i2c\_io\_exp\_get\_ts\_temp** (**struct I2C\_IO\_EXP \*io\_exp**) - Used to read the temperature from the temperature sensor of the I/O expander specified.

Manipulation of I/O on I<sup>2</sup>C I/O expanders is made rather straightforward with the abstraction techniques used here. For example, operation of the HMI LCD is accomplished through an I<sup>2</sup>C I/O expander board and the buttons and switches of the HMI are inputs to the I<sup>2</sup>C I/O expander. This can be adapted however to a device for any purpose, as long as timing is not critical.

#### 5.4.10 I<sup>2</sup>C Expansion Interrupt Control

The system controller board was designed to support I<sup>2</sup>C expansion with a total of eight buffered I<sup>2</sup>C connectors. Four of the eight connectors have an additional interrupt line for event detection. The I<sup>2</sup>C I/O expansion board uses this interrupt line to inform the dsPIC that an input change event has occurred. At this point, the dsPIC33 can query the particular device that generated the interrupt. The interrupt lines are connected to an AND gate and to GPIOs of the dsPIC, and the output of the AND gate is connected an external interrupt of the dsPIC33. On an event, the interrupt service routine will determine the source of the interrupt and create a job that will update the register of the I2C\_IO\_EXP structure instance. A code snippet of the interrupt service routine is shown below (see Listing 5.9).

```
void
__attribute__((interrupt, no_auto_psv)) isr_BI2C (void)
{
    unsigned char tjob;

    INT_BI2C_IF = DEASSERTED;
    if (!bi2c_lock) {
        bi2c_lock = TRUE;
        tjob = get_next_job();
        job_list[tjob].bc = 1;
        job_list[tjob].data[0] = 0xff;

        if (!BI2C_INT0)
            job_list[tjob].data[0] = 0;
        if (!BI2C_INT1)
            job_list[tjob].data[0] = 1;
        if (!BI2C_INT2)
            job_list[tjob].data[0] = 2;
        if (!BI2C_INT3)
            job_list[tjob].data[0] = 3;

        add_job (BI2C_EVENT);
    }
}
```

Listing 5.9: Buffered I<sup>2</sup>C Interrupt Routine

By using an interrupt to detect events of an I<sup>2</sup>C slave devices, the firmware does not need to poll these devices which in turn frees up controller resources. The *bi2c\_lock* variable is used as a software debounce since all the inputs used in the system are from push buttons and hence suffer from jitter. This variable is reset in the system heartbeat timer after a set amount of time.

#### 5.4.11 I<sup>2</sup>C LCD Control

As mentioned in the previous section, the control of the HMI is performed over I<sup>2</sup>C using an I/O expansion board. This permits the 5V LCD to be controlled using the I<sup>2</sup>C master module bus of the system controller at 3.3V. The OptiSorter was equipped with an 16x2 character display with a Hitachi HD44780 controller [10]. Using the I/O expander structure (see Section 5.4.9), the specific I/O control sequence required to set data on the LCD is accomplished through a set of various functions. The specific functions used for controlling the LCD are listed below. Because of the limited I/O available on the I<sup>2</sup>C expansion board, the LCD is operated in 4-bit mode [10].

- **lcd\_init ()** - Initialize the LCD in 4-bit data mode
- **lcd\_push\_nibble (char nibble)** - Put a 4-bit nibble of data on the data lines and toggle EN to load data
- **lcd\_write\_cmd (char cmd)** - Send a command to the LCD
- **lcd\_write\_data (char data)** - Write a character to the LCD
- **lcd\_set\_bl (char state)** - Set HMI panel LED
- **lcd\_cursor\_to (char line, char x)** - Set LCD cursor to *line* and *x* position
- **lcd\_clear()** - Clear the display
- **lcd\_print (char \*msg, char len)** - Write a series of characters (*msg*) to the display of length *len*

Using these routines, writing messages to the LCD is rather straightforward to manipulate the LCD. These routines use software delays to meet the timing requirements of the LCD. This is not the most efficient way of accomplishing this, however the development time is reduced significantly. Due to the fact that messages are infrequently updated on the HMI panel, it is not critical that an event driven LCD driver is developed.

### 5.4.12 UART Handler

The system controller support a simple RS232 interface for more advanced machine control over the simple HMI panel. The RS232 interface is used to set motor speed, control motor state, set back-light strobe duration, retrieve counters and other system state information. Since a simple HMI cannot possible fulfill all of these requirements in an elegant fashion, it was intended that an additional control panel PC could be used for this purpose. The dsPIC33 has two built-in UART modules one of which was used for machine control and status. The entire UART control was coded in two functions and two interrupt service routines along with a couple of data buffers (one for receive and one for transmit) and counters to track the index position of the buffer. The functions used and interrupt service routines are described in detail below.

- **init\_uart2()** - Initialize the UART2 module using TX and RX line only at desired baud rate (57.6kbps).
- **uart\_hdlr()** - Handler called after six-byte UART command has been received. This function performs data integrity check and fulfills the request.
- **\_U2RXInterrupt()** - ISR for UART received data. This function will load data into the cyclic receive buffer *uart\_rx\_buf*.
- **\_U2TXInterrupt()** - ISR for UART transmit data, this routine is called after a byte has been successfully transmitted. This function will automatically load remaining data to U2TXREG from the transmit buffer *uart\_tx\_buf* until complete.

Two structures were created to handle the data pointers for the transmit and receive buffers. This structure is listed in Listing 5.10.

```

struct UART_Rx {
    unsigned char wr;
    unsigned char rd;
};

struct UART_Tx {
    unsigned char wr;
    unsigned char rd;
};

```

Listing 5.10: UART Transmit and Receive Counter Structures

The received data format of dsPIC33 is a six-byte command where the first byte is the request, followed by two bytes of optional data. The last three bytes are an XOR of the first three for simple error checking (see Table 5.3).

The error checking employed is quite straightforward and uses minimal computational time to verify the data. A code listing of the data verification is shown below. The error counter, *err*, is used

Byte	Description
0x00	UART command request
0x01	UART command data high
0x02	UART command data low
0x03	XOR of byte 0x00
0x04	XOR of byte 0x01
0x05	XOR of byte 0x02

Table 5.3: UART Command Format

to count the number of errors found in the data. If the error counter is not zero, an error has been detected and the dsPIC33 will respond with an NACK, otherwise it will respond with an ACK. The host PC will expect this response and can continue accordingly. The routine for handling incoming UART requests is shown below (see Listing 5.11).

```

/* Perform Error Check on data */
for (c = uart_rx.rd; c < uart_rx.rd + 3; c++) {
    if ((uart_rx_buf[c] ^ uart_rx_buf[c+3]) != 0xff)
        err++;
}

/* Respond accordingly */
if (err) {
    U2TXREG = 0x15; /* NACK */

    /* Reset buffer pointers equal */
    uart_rx.rd = uart_rx.wr = 0;
    uart_cmd_flag = 0;
    return;
}
else {
    U2TXREG = 0x06; /* ACK */
    /* Wait for ACK msg to be sent */
    while (!U2STAbits.TRMT);
}

```

Listing 5.11: UART Data Check and Response

The supporting UART commands are described in 5.4. The list of UART commands can easily be expanded if required in future revisions. Since one byte is used for the UART command, 255 possible commands can be created. The UART commands were separated into **get** and **set** with respect to the PC. The **get** commands are used to retrieve data from the system control board, and the **set** commands are used to send data.

Using these commands, all machine control can be operated by an external PC or device with RS232 availability. This permits a highly customizable front-end for a finalized commercial product. For the prototype, a simple 32-bit Windows application was developed as a demonstration application on interfacing with the system controller. This application was developed in Microsoft Visual Basic 6 and is described in detail in Section 6.9.

UART Command / Request	Description
UARTCMD_SET_MOTOR_STAT	Enable or disable the motor
UARTCMD_SET_MOTOR_FREQ	Set motor frequency (in Hz)
UARTCMD_SET_BL <sub>x</sub> .WIDTH	Set the pulse duration of the backlight pulse, where x is the backlight 0 or 1
UARTCMD_SET_FL <sub>x</sub> .WIDTH	Set the pulse duration of the frontlight pulse, where x is the frontlight 0 or 1
UARTCMD_SET_CAM <sub>x</sub> .PULSE.POS	Set the position of the camera trigger 0, or 1 pulse with respect to motor pulses
UARTCMD_SET_ACCEPT_ON.PULSE.POS	Set the position of the accept start position (for pneumatic valves) with respect to motor pulses
UARTCMD_SET_ACCEPT_OFF.PULSE.POS	Set the position of the accept end position with respect to motor pulses
UARTCMD_GET_MOTOR_STAT	Get state of motor (running or stopped)
UARTCMD_GET_MOTOR_FREQ	Get motor pulse frequency (in Hz)
UARTCMD_GET_BL <sub>x</sub> .WIDTH	Get the pulse duration of the backlight 0 or 1, pulse
UARTCMD_GET_FL <sub>x</sub> .WIDTH	Get the pulse duration of the frontlight 0 or 1, pulse
UARTCMD_GET_CAM <sub>x</sub> .PULSE.POS	Get the position of the camera trigger 0, or 1 pulse with respect to motor pulses
UARTCMD_GET_ACCEPT_ON.PULSE.POS	Get the position of the accept start position (for pneumatic valves) with respect to motor pulses
UARTCMD_GET_ACCEPT_OFF.PULSE.POS	Get the position of the accept end position with respect to motor pulses
UARTCMD_GET_PC.PWR.STATE	Get the power state of the PCs (two bits per PC)
UARTCMD_GET_GOOD_COUNT	Get the accept capsule count value for the specified quadrant
UARTCMD_GET_BAD_COUNT	Get the reject capsule count value for the specified quadrant
UARTCMD_GET_TOTAL_COUNT	Get the total capsule count value for the specified quadrant
UARTCMD_RESET_COUNTERS	Reset all counters to 0s
UARTCMD_DEBUG.MODE	Put the system controller in debug mode or normal mode

Table 5.4: UART Commands

#### 5.4.13 Soft PC Power Control

In order to facilitate a completely autonomous system, the power of the host PCs is controlled via the system controller. This is accomplished by manipulating the power switch input of the motherboard and monitoring the power LED output. A rather straightforward circuit was designed to simulate clicking a power button as described in Section 5.1.11. The state of the four host PCs is read from the input pins and is stored in a structure `PC.PWR.STATUS` where two bits hold the state of the power. Three distinct power states were created, off, on and ready. Two bit masks are used to define the power state as described below.

- **PC\_POWER\_ON (bit 0)** - Logic-one indicates the host PC is running, otherwise the power

is off

- **PC\_POWER\_READY (bit 1)** - This bit is used to indicate if the ready signal has been received from the host PC software.

This is used to track the power status of the various PCs in the system. When the system starts, the PCs are turned ON, and the system waits until a ready signal is received from all PCs. Once all PCs are running and the inspection software is ready, the system is ready for inspection.

The state of the PC power is checked every heartbeat timer event, and the ready signal flag is updated through an I<sup>2</sup>C message and reset on if an OFF state of a given PC is detected.

#### 5.4.14 Capsule Tracking and Counts

A highly important measure in the design of the PharmaSorter, is that the pass/fail result of a given capsule is correct. Thus, a capsule tracking system was developed for storing a pass/fail result for each individual capsule. This is achieved using a two-dimensional array of pass/fail data for a buffer of sixteen capsules for each quadrant. The array *capsule\_passfail* along with a capsule ID tracking variable *cap\_count* is used to track and store inspection results for each capsule. The current capsule *cap\_count* is incremented every time a new capsule enters the inspection stage, ie. when a pulse from the proximity sensor is seen. The host PC uses the master camera of each quadrant to request the identity the capsule being inspected, and can use this ID to return a pass/fail result following inspection.

A set of capsule counters is used to record the pass and fail results received from each quadrant host PC. These values are stored in long type variables with a good, bad and total counter for each quadrant. These counter variables are labelled *capsule\_total\_count*, *capsule\_good\_count* and *capsule\_bad\_count*.

#### 5.4.15 Heartbeat Timer

In order to perform occasional tasks in the system controller, a dedicated heartbeat timer is used. For the heartbeat timer, Timer8 in the dsPIC33 is used. The heartbeat timer is used to flash the HMI LED based on the state of the system, handle system shutdown, delay inputs for software debounce, and update host PC power status. The interval at which the heartbeat timer is called is approximately 205ms.

#### 5.4.16 Debug Mode Timer

During the development it was desirable to have a debug mode within the system controller board. The debug mode merely generates timed triggers for the cameras without running the motor or



monitoring the feedback from the proximity sensor. The debug mode timer is set to a fixed interval and toggles between triggering the top-view and bottom-view cameras. When enabled, the debug timer effectively triggers every 192ms. The debug mode timer is initially off, but can be enabled through the RS232 control interface.

---

## Chapter 6

---

### *Host PC*

---

The host PCs are an integral part of the PharmaSorter that are responsible for acquiring images from the USB2.0 cameras and performing inspection on the images. The host PCs are standard desktop computers without the user essential portions such as monitor, keyboard and mouse. The host PCs run autonomously without human intervention but can be access remotely for maintenance and configuration. Many considerations were taken into account when selecting the host PC hardware and operating system. For the system to be complete, several host PC software applications were developed. The main application is *inspect*, which is responsible for the image acquisition and processing. Several smaller applications were required including an application for loading the firmware to the FPGA (*fpga\_loader.ss*) and a camera initialization/calibration tool called *cam\_init*.

### 6.1 Operating System Selection

Early in the project, it was decided upon that Linux was the best choice for the host PC operating system. There are several significant advantages to using a UNIX-like OS, like Linux, for the application. Linux can run an various architecture and is compatible with most desktop PCs. Some of the desirable traits of Linux are listed below.

- Stable and secure
- Not dependent on a window manager (less resources used)
- Open source and free
- Reliable (long up-times)

Once it was established that Linux was to be used as the host operating system, the selection of an appropriate distribution was the next step. Hundreds of distributions are based around the Linux kernel and there is no singular distribution that is most appropriate for the application. However, Debian was selected for its popularity, renowned stability and polished package management system.

## 6.2 Hardware Selection

One of the most influential constraints on the entire project is the cost factor. Thus, the selection of host PC hardware must reflect this requirement, however performance is also an important factor. A compromise between performance and cost was sought out and an Intel Core 2 Quad system was a reasonable trade off for the time at which the systems were purchased. The desktop PC market is constantly changing and evolving. As a result, future generations of the PharmaSorter will benefit from faster processors and higher data rates. The components used for the host PC are listed in Table 6.1 with the cost at the time they were purchased (May 2008).

Part Number	Description	Cost
ASUS P5K ATX	Asus P5K ATX LGA775 P35 DDR2 2PCI-E16 1PCI-E1 SATA2 motherboard	\$142.03
OCZ2P8001GK	OCZ PC2-6400 1GB (2x512) Platinum XTC Dual Channel RAM	\$34.50
BX80562Q6600	Intel Core 2 Quad Q6600 / 2.4 GHz (1066MHz) -L2 8MB	\$249.00
ST3250410AS	Seagate 250GB SATA 3GB 7200RPM 16MB Hard Disk Drive	\$69.99
W0100RU	ThermalTake PurePower 500W ATX 2.0 Power Supply	\$63.00
	<b>Total</b>	<b>\$558.52</b>

Table 6.1: Host PC Hardware

For a grand total (excluding applicable taxes and delivery fees), the cost of each inspection PC is under \$600. Each system is powerful enough for the application. To fixture the inspection PCs in the system, a minor modification to the existing circuit board holder was required as cases were not purchased.

The OptiSorter was originally equipped with a circuit board rack that held a total of 17 boards vertically in the system. This circuit board rack was retrofit to hold the four host PC motherboards. The motherboards were significantly larger than the existing circuit boards and thus two new metal plates were fabricated to meet the dimensional requirements of the motherboards. The new plates also had mounting holes to mount a total of four standard hard disk drives, two on each plate. This modification made it possible to mount all four host PCs inside the PharmaSorter. The power supplies required for powering the motherboards were placed vertically underneath the motherboard rack. They can be easily attached inside the machine by adding brackets to the lower support. The PC mounting scheme is shown in Figure 6.1.

The cooling of the host PCs is accomplished by a set of two high throughput fans already existent

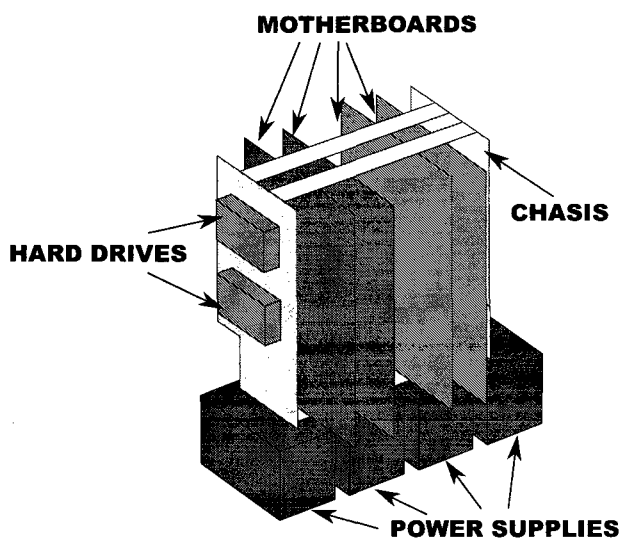


Figure 6.1: PC Mounting Scheme

in the OptiSorter along with the CPU cooling fans on the motherboards. The cooling fans operate at 240VAC and a simple 120V to 240V transformer was used to step up the voltage.

### 6.3 Software

The most liquid component of the PharmaSorter project is the software. The majority of the software developed for the PharmaSorter was written in C and C++. Small GUI applications intended for calibration were written in Python and simply provide a front-end to the C/C++ applications. C/C++ are high level languages capable of low-level calls and thus are powerful for software development. Developing in C/C++ yields high performance applications without the extraneous overhead in other languages like Java or Python. All C/C++ applications were compiled using the GNU gcc / g++ compiler and intended to run on a Linux x86 system. The PharmaSorter software applications include: *inspect*, *fpga\_loader\_ss*, *cam\_init*, *py\_write\_window* and *py\_cam\_calibrate*. An additionally application for advanced control of the system control board was written for Microsoft Windows using Microsoft Visual Basic 6.0. This is a simple GUI example application and is intended for a touch-screen HMI or the like.

### 6.4 *inspect*

*inspect* is the main software application used in the PharmaSorter. This application is responsible for acquiring images from the cameras of the respective quadrant, performing the inspection using

image processing techniques and return a pass/fail result to the system controller (through the master camera of the quadrant). *inspect* is a threaded application that uses the *pm\_cam* driver (see Section 4.6.2) to interact with the connected cameras. An external library contains all the image processing functions used (libIP). Several Linux libraries are used by *inspect* including pthreads, libtiff and libusb.

### 6.4.1 POSIX Threads

Threads allow applications to parallelize operations [15]. For the user-mode driver created, it is essential to use threads in order to ensure no data is lost during the transfer from the camera and that data can be acquired from multiple cameras simultaneously. There is limited buffering in the camera hardware and thus if images were acquired sequentially, only the first image acquired would be complete. With modern processors that have multiple cores, parallelism in software can take advantage of more processing capabilities in a dedicated application such as the PharmaSorter. The basic scheduling scheme of the *inspect* is shown in Figure 6.2.

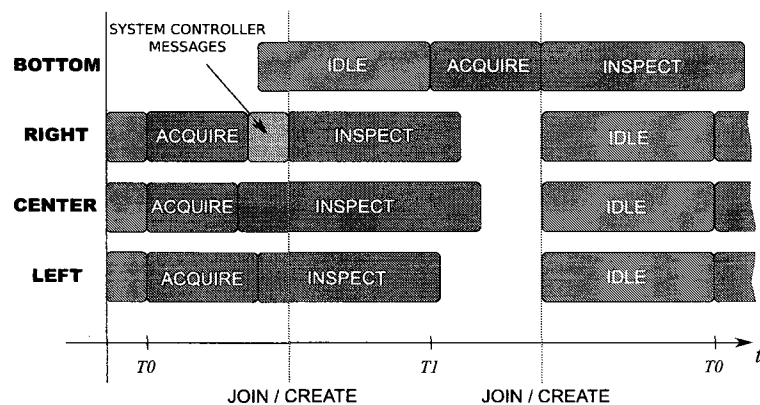


Figure 6.2: Inspect Software Scheduling Scheme

There are essentially a total of eight threads created for every capsule inspected. After finding and initializing the USB2.0 cameras in *inspect*, the main loop is started where a set of image acquisition threads are created for all top view cameras (ie. left, center and right). After created, these threads are joined. Before each acquisition thread is finished, an image processing thread is created to process the image acquired. Following the join, an image acquisition thread is created for the bottom camera which is also joined in the main loop. After the bottom camera acquisition thread completes, the cyclic buffer index is incremented and the capsule counter is also incremented. The image processing threads are not joined. They are restricted to timing requirements but can finish at any point. Once complete, the inspection result is stored in a global array. The master camera is responsible for

communication with the system controller and thus must obtain a capsuleID of the current capsule from the system controller. It must also send a pass/fail result of the previous inspection. This is accomplished in the acquisition thread for the master camera only. Due to the mechanics of the machine, a capsule inspection result must be determined by at about the time the next capsule is enters the inspection stage. Because of this constraint, the previous capsule inspection result is sent to the system controller at the same time that the current capsuleID is being requested. A data structure is used to hold specific information required by the threads including capsuleID, camera position, quadrant, master flag, the threadID, the cyclic buffer index, local and remote capsule IDs, and the capsule count. A basic flow diagram of the *inspect* application is shown in Figure 6.3.

The image acquisition thread and image inspection threads are like sub-processes within the main inspect process. The acquisition and inspection thread flow diagrams are shown below in Figure 6.4 and Figure 6.5 respectively. Note that there will be four of each of these thread running simultaneously, one for each camera.

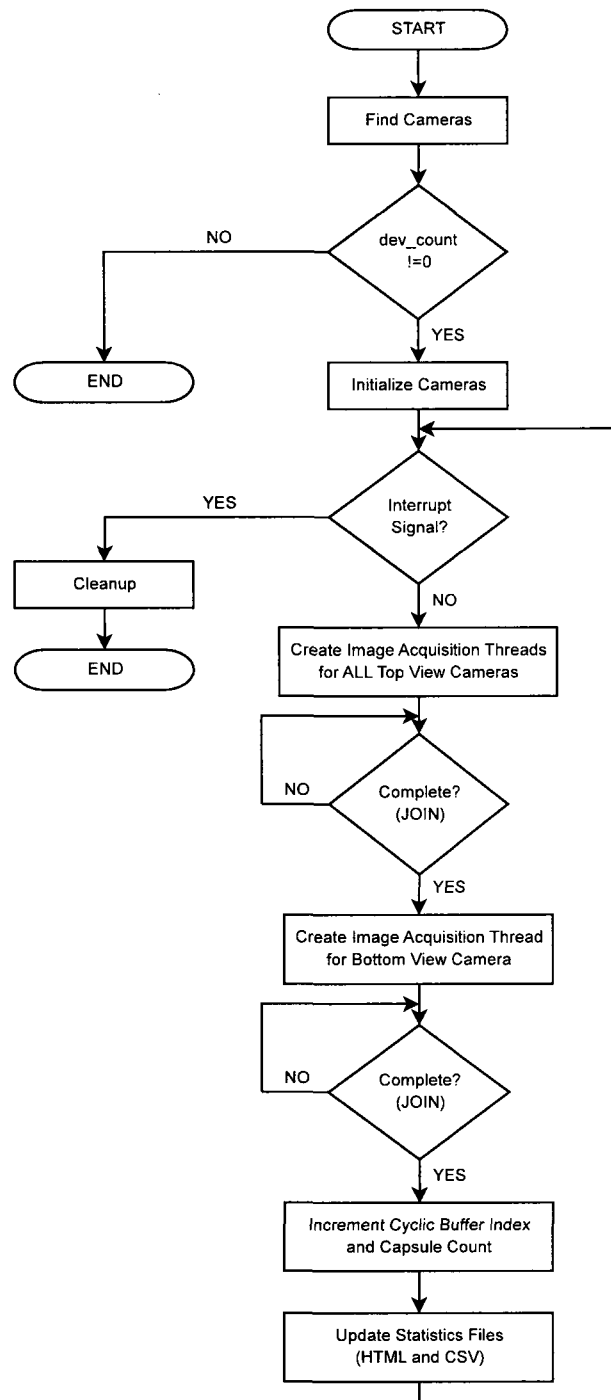


Figure 6.3: Inspect Main Flow Diagram

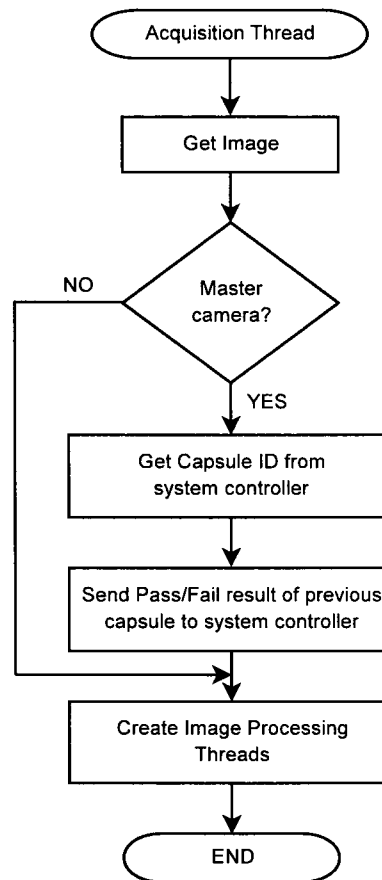


Figure 6.4: Inspect Image Acquisition Flow Diagram



*inspect* is intended to provide a system to control the inspection of capsules where some statistical data is generated. For every capsule being inspected, HTML and CSV statistics files are updated with capsule counts, misalignment error counter and the inspection rate. This file can be used for a larger data collection system if desired.

A screenshot of the output HTML file is shown in Figure 6.6.

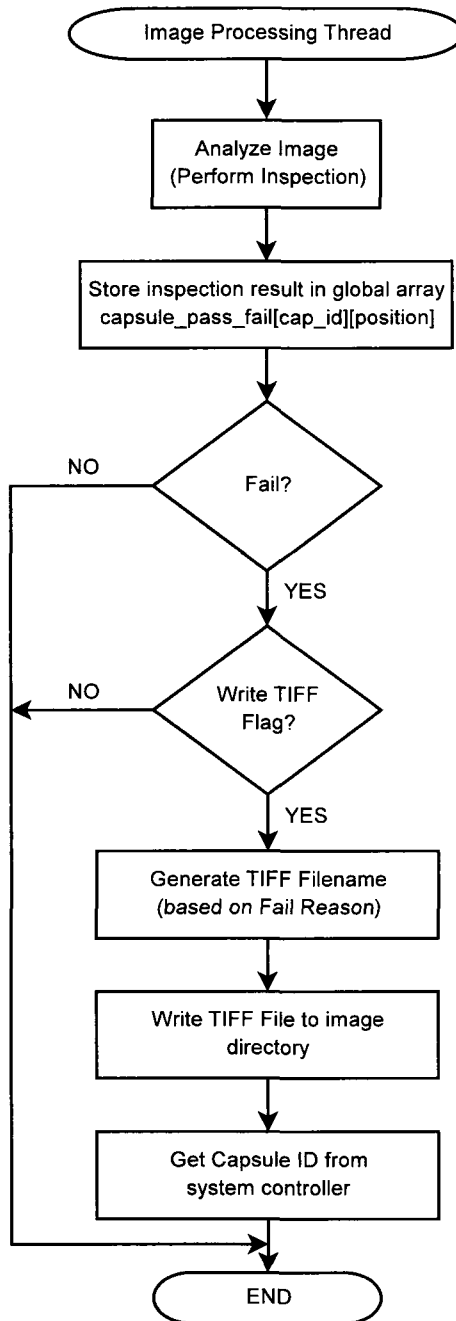
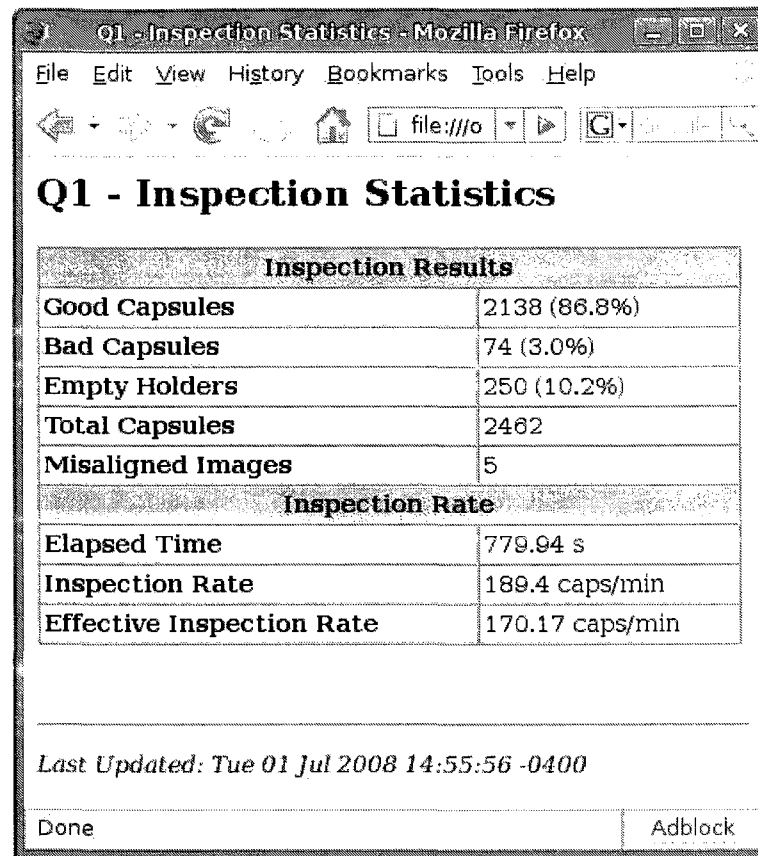


Figure 6.5: Inspect Image Processing Flow Diagram



The screenshot shows a Mozilla Firefox browser window with the title 'Q1 - Inspection Statistics - Mozilla Firefox'. The address bar shows 'file:///o'. The main content area displays 'Q1 - Inspection Statistics' followed by two tables. The first table, 'Inspection Results', shows counts and percentages for Good Capsules, Bad Capsules, Empty Holders, Total Capsules, and Misaligned Images. The second table, 'Inspection Rate', shows Elapsed Time, Inspection Rate, and Effective Inspection Rate. At the bottom, it says 'Last Updated: Tue 01 Jul 2008 14:55:56 -0400' and has 'Done' and 'Adblock' buttons.

Inspection Results	
Good Capsules	2138 (86.8%)
Bad Capsules	74 (3.0%)
Empty Holders	250 (10.2%)
Total Capsules	2462
Misaligned Images	5

Inspection Rate	
Elapsed Time	779.94 s
Inspection Rate	189.4 caps/min
Effective Inspection Rate	170.17 caps/min

Last Updated: Tue 01 Jul 2008 14:55:56 -0400

Done Adblock

Figure 6.6: Inspect HTML Output File

## 6.5 *fpga\_loader.ss*

The minimization of components on the USB2.0 camera circuit board led to a minimalist technique of loading firmware to the FPGA. In order to load the FPGA configuration bit-file, the FX2 firmware was programmed with a series of USB vendor requests to fulfill the loading using the slave-serial method as described in 4.3.6. This requires only five GPIO of the FX2 and is capable of programming a 250kB file in under 30 seconds. This is reasonable for the application since the FPGA is only loaded once, at startup.

Along with the firmware in the FX2, accompanying software must be developed. For this, an application titled *fpga\_loader.ss* was developed. This application reads in a FPGA configuration bit-file, and through a series of vendor requests to the USB2.0 camera instructs the FX2 to “bit bang” the bit-file data to the FPGA. The command line arguments required are a device address of the device to program and a BIT file to load. Optional arguments include a verbose option and FPGA powering options. The listing below is the help screen for the *fpga\_loader.ss* application.

---

```
Usage: fpga_load_ss [OPTIONS]... [BIT FILE]
Options:
  -h, --help            Display this help screen
  -v, --verbose          Increase verbosity
  -d, --device=BUS.DEV  Select the bus and device to use
  -l, --list            List the devices on the USB bus
  -p, --power <state>   Soft power control to FPGA [on / off]
  -r, --reset           Soft reset to FPGA (once configured)
  -V, --version          Display version information
```

[BIT FILE] is the path to the BIT file generated using ISE  
 'path/foo.bit'

## 6.6 *cam\_init*

During the initial setup of the PharmaSorter, each camera initially must be loaded with location information including the quadrant, position and a master flag. This information is stored in the EEPROM of each device and is used by *inspect* to coordinate inspection. The *cam\_init* utility is intended for loading this information. Along with the camera location information, the windowing parameters for the particular camera can also be loaded into the EEPROM following the memory map in Section 4.5. This however, requires manually setting memory values using the *cam\_init* tool by specifying the memory address and data to write. The help screen for this utility is listed below.

```
Usage: cam_init -d=[BUS.DEV] [OPTIONS]...
Camera first time setup.

  -d, --device            Specify device to target as BUS.DEV
  -q, --quadrant          Specify camera quadrant
  -p, --position          Specify camera position
  -m, --master            Specify camera as quadrant master (responsible for comm. with control board)
  -r, --read              Retrieve ALL EEPROM data to screen
      --quickread         Retrieve the camera quadrant, position and master flag
  -s, --save <FILENAME>  Read data from EEPROM and save to <FILENAME>
  -l, --list              List all devices on USB bus
      --blank             Clear EEPROM memory with 0xff
  -w, --write_eeprom [ADDR] [VAL] Write a value to a specific memory location of the EEPROM
  -rb, --read_eeprom [ADDR] Read a specific byte of EEPROM memory
  -rp, --reload_params    Reload window parameters from EEPROM
                          *ADDR and VAL are decimal numbers

cam_init v0.01
```

Configuring a camera for a particular location requires the values specified for the *quadrant*, *position* and *master* at the command line. Each of these command line arguments must be followed by a value. The quadrant value ranges from 1-4, the position value ranges from 1-4 and the master value is 0 or 1. The camera position identifiers are listed in Table 6.2.

Value	Location
0x01	Center
0x02	Left
0x03	Right
0x04	Bottom
0xff	Undefined

Table 6.2: USB2.0 Camera Position Identifiers

## 6.7 *pyWindowConfig*

To provide an easier way of configuring the camera window parameters, a GUI was created using Python as a front-end for *cam\_init*. This application uses the pyGTK libraries to create a graphical interface to read and set window parameters. A screenshot of the GUI is shown in Figure 6.7. The device selection window is also shown in Figure 6.8 where the user can select the specific camera to configure.

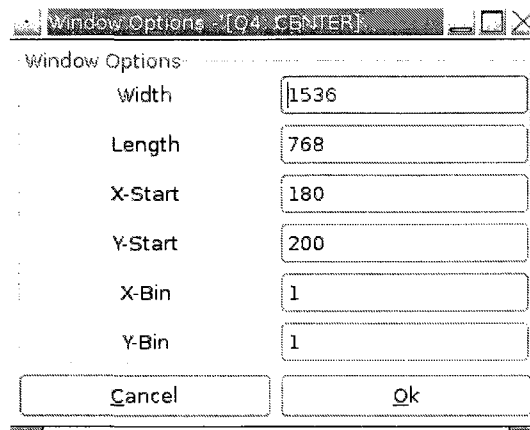


Figure 6.7: pyWindowConfig Screen Shot

## 6.8 *pyCamCal*

A calibration tool was created to simplify the calibration process when installing cameras. Each camera must be aligned such that the capsule is in the correct orientation in the image, and the focus is set to the surface being inspected. This is a time consuming process, however with *pyCamCal*, this process is more user-friendly. Using a version of inspect that allows command line window parameters, and a single-shot mode option, *pyCamCal* can be used as a front-end for displaying images according the the desired parameters set in the software. *pyCamCal* was developing in Python and uses the pyGTK libraries. The following screenshots illustrate this application in operation, as shown in Figure 6.8 and Figure 6.9).

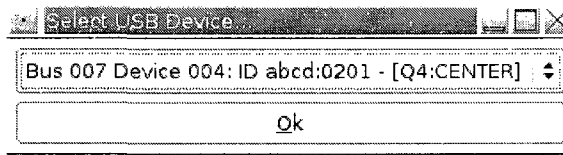


Figure 6.8: pyCamCal USB Device Selection Screen Shot

## 6.9 W32 Control Panel Application

A control panel application was developed as an example front-end for advanced machine control. This application was written in Microsoft Visual Basic 6.0 and uses the MSCOMM activeX control to communicate over RS232 with the system control board. The application uses the commands listed

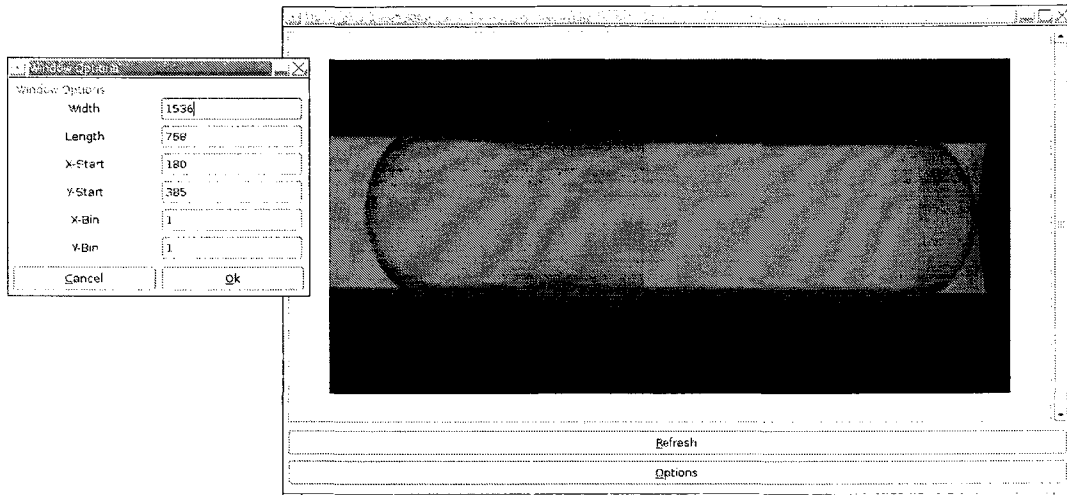


Figure 6.9: pyCamCal Main Window Screen Shot

in Section 5.4.12 to control the machine and acquire status information. Although not intended to be used in a finished commercial product, the application demonstrates how such an application can be developed in a rapid application development environment that can interface with the machine. A screenshot of the demonstration application is shown in Figure 6.10.

## 6.10 Data Collection

As mentioned previously, *inspect* outputs both a CSV and HTML file with real-time statistics regarding inspection. With this data available, a data collection system can poll each host PC to gather this information. In order to facilitate this, an Apache webserver was setup for each inspection PC to easily share data over Ethernet. Each host PC is connected to a network switch, where individual systems can be accessed by their static IP address. This provides means to access each system for maintenance or data collection remotely. The following table outlines the static IP addressing scheme (see Table 6.3).

Quadrant	IP Address
Q1	192.168.1.101
Q2	192.168.1.102
Q3	192.168.1.103
Q4	192.168.1.104

Table 6.3: Host PC IP Addresses

The Apache webserver uses the `/var/www` directory to store HTTP accessible information. To

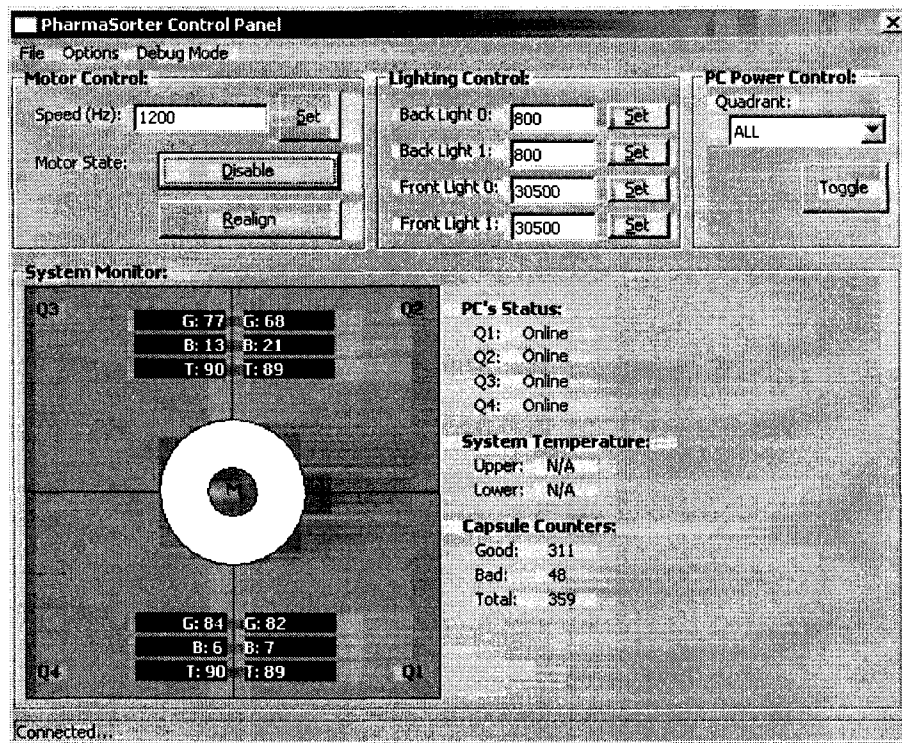


Figure 6.10: W32 Control Panel Application Screen Shot

make the system data available over a simple web interface, the following files were linked (symbolically) to this directory.

- **statistics.html** - Link to HTML output of *inspect* statistics
- **statistics.csv** - Link to CSV output of *inspect* statistics
- **inspect.log** - Link to output log of inspect
- **./images/** - Link to images directory containing inspection images

## 6.11 File Organization

All executable files pertaining to the operation of the system are stored in */opt/pill\_machine* of each system. This includes all Linux applications listed above as well as the output statistical information (*statistics.html* and *statistics.csv*). The image files are stored in */images* and the log file is stored in */var/log/pill\_machine/inspect.log*. This file organization deemed suitable for the prototype and can be easily modified.



## 6.12 Start Up

### 6.12.1 inittab

The start up sequence of the system is important for ensuring the system can operate autonomously. This involved creating a simple start up script to load firmware to the hardware and start the *inspect* application. The start up script is stored in */opt/pill\_machine/startup*. In order to have the start up script to run on start up, it set as a process in the Linux */etc/inittab* file. The entry in the *inittab* file appears as follow.

```
6:23:respawn:/opt/pill_machine/startup
```

The code listing for the startup script is shown below, where *program\_cam\_revB\_ALL* is the firmware loader script.

```
#!/bin/bash

#PROGRAM CAMERAS
sleep 2
/opt/pill_machine/fpga_loader/program_cam_revB_ALL

sleep 4

#START INSPECTION SOFTWARE
/opt/pill_machine/inspect_v6/src/inspect -le > /var/log/pill_machine/inspect
```

The output from the inspect application is stored in a log file in */var/log/pill\_machine/inspect*. This can be used to diagnose system errors and is linked to the */var/www* directory for access over HTTP.

### 6.12.2 Firmware Loading Script

The firmware loading script is used to load firmware to the camera FX2 microcontroller and the FPGA. The FX2 firmware is loaded using an open-source tool called *cycfx2load* and the FPGA firmware is loaded using a custom application called *fpga\_loader\_ss* (see Section 6.5). The firmware loading script (*program\_cam\_revB\_ALL*) is listed below.

```
#!/bin/bash

if [ ! -n "$1" ]
then
    for pm_bus_addr in `lsusb | egrep 'abcd:0201' | tr -d ':' | awk '{print $2"."$4}'`
    do
        echo Programming FX2 on $pm_bus_addr.
        cycfx2prog -d=$pm_bus_addr prg:fx2cam_firmware_revB.ihx
        cycfx2prog -d=$pm_bus_addr run
    done
fi
```

```
done

sleep 2

for pm_bus_addr in `lsusb | egrep 'abcd:0201' | tr -d ':' | awk '{print $2"."$4}'`
do
    echo Hard Power Reset of FPGA on $pm_bus_addr.
    ./fpga_load_ss -d=$pm_bus_addr -p off
    sleep 0.5
    ./fpga_load_ss -d=$pm_bus_addr -p on
done

sleep 2

for pm_bus_addr in `lsusb | egrep 'abcd:0201' | tr -d ':' | awk '{print $2"."$4}'`
do
    echo Loading FPGA bit-file on $pm_bus_addr.
    ./fpga_load_ss -d=$pm_bus_addr frame_grabber_v1.bin &
done

wait

else
    cycfx2prog $1 prg:fx2cam_firmware_revB.ihx ; cycfx2prog $1 run
fi
```

This script uses *lsusb* to identify the device address of matching devices by productID and vendorID. Using the USB device address, each specific device is programmed with the camera FX2 firmware. Following the loading of the FX2 firmware, the device will re-numerate [4] itself with the camera program. This will cause the USB device address to increase. Using the new address, the FPGA bit-file is loaded using *fpga.load\_ss*.

---

## Chapter 7

### *Assembling the Prototype*

---

So far, each component of the system has been described in some detail. This chapter describes each component's role in the prototype system along with the interconnection of components and the setup requirements of each component. The PharmaSorter follows the paradigm of a standard machine vision system. It is comprised of an image acquisition component, a data processing component and a machine control component. For the PharmaSorter, these components are the USB2.0 cameras, host PCs and system control board respectively. For specifics on each component, see the corresponding sections of this thesis and supporting documents in the appendix.

The assembly of the prototype starts with the removal of antiquated electronics from the Opti-Sorter. This includes camera circuit boards, acquisition and processing boards, controller boards, etc. Once stripped of all of these components, many unconnected wires will remain. It is recommended that these wires are traced and labelled accordingly as assembly of the prototype involved connecting existing hardware to new circuit boards. The assembly of the prototype also involved finding areas to mount circuit boards and modifications to existing circuit board holders to house new ones.

#### 7.1 Wiring

For the prototype, electrical standard codes were not followed. The goal was to obtain a proof-of-concept prototype, not a working commercial unit. The circuit board layout was designed to fit all required circuit boards in the panel of the system. The following figures illustrate the circuit board layout of the fixture in the system, see Figure 7.1 and Figure 7.2).

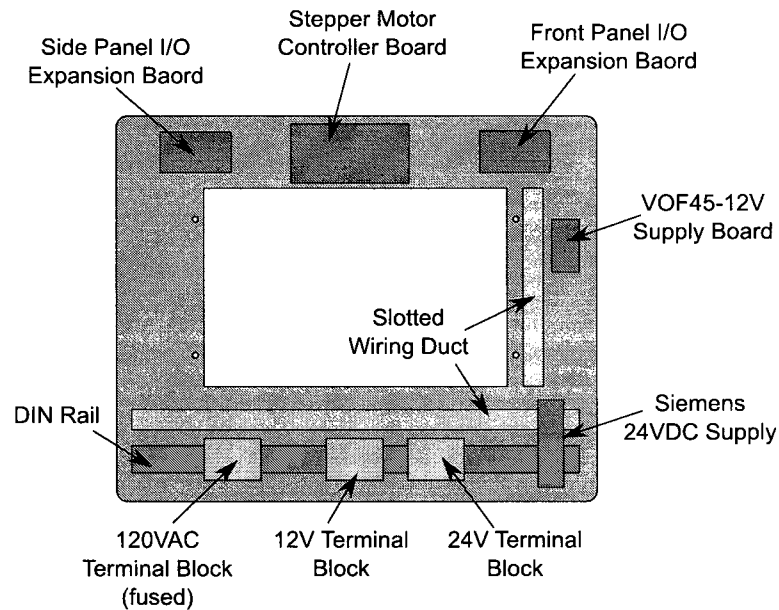


Figure 7.1: Panel Layout (Front)

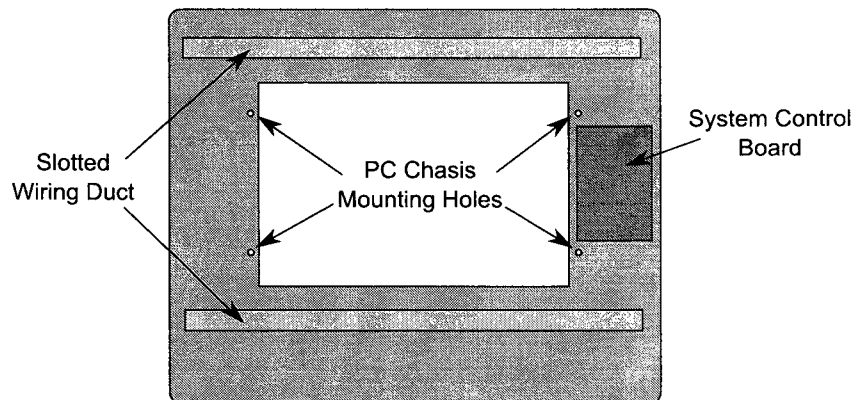


Figure 7.2: Panel Layout (Rear)

The host PC chassis is not shown in the panel layout, however must be designed specific to the motherboards used in the system. Section 6.2 describes the host PC chassis used in the prototype. The existing wiring was recycled and used for making all electrical connections, with the exception of some standard cables such as USB and power supply cables. This required custom connectors to be made for the wiring in some cases where connectors were used. The wiring diagram of the system is shown in Figure 7.3. This is a high-level wiring diagram and for more specific wiring information, refer to the appropriate sections of this document and additional documentation provided in the appendices.

### 7.1.1 USB2.0 Camera

The cameras are designed to fit in the existing housings and must be aligned such that the optical center of the sensor is at the center of the lens. This requires modification to the housing. The wiring for the USB2.0 cameras is rather straightforward. All cameras must be connected to their respective quadrant host PC using standard USB Type-A to Type-B mini cables. For the prototype, 2 meter cables were used. A trigger cable must be connected to all cameras as well. The top cameras use trigger 0 and the bottom camera uses trigger 1. The camera trigger connector has a trigger-IN and a trigger-OUT. Thus, for the three top cameras, the trigger signal can be piggy-backed. The delegated master camera of each quadrant also requires a connector to the I<sup>2</sup>C slave channel of the system control board for the respective quadrant.

## 7.2 Firmware

The firmware for the system control board must be loaded using the Microchip ICD2 programmer and the MPLAB IDE software. The firmware files are stored in the project directory in the *fw.dsPIC/* directory.

The firmware for the USB2.0 cameras are soft-loaded during machine initialization through software.

## 7.3 Host PCs

The host PC hardware must be installed in the system using a modified motherboard chassis. This is briefly described in Section 6.2. Formal drawings were not created for this due to the fact that the chassis dimensions are dependent on the motherboard selected. The host PC chassis must mount motherboards and hard disk drives. For connection information regarding the specific host PC hardware selected, refer to the corresponding user manuals.

### 7.3.1 Operating System

The OS used in the prototype systems was Debian 4.0 rev 3 with minimal packages installed. A base X server with fluxbox window manager was installed to help debug system issues and aid in calibration. Since the host PCs do not have input devices or monitors, remote access is required. The SSH daemon was installed along with a VNC server. This allows graphical access to the remote systems. An Apache web server was installed and setup to allow access to real-time statistical information from *inspect*.



Once a working system was established on a single system, the hard drives were mirrored using the *dd* tool. This allowed direct copies of the source drive to the additional three drives.

Each individual host PC requires specific configuration for network access. The */etc/network/interfaces* file was modified to set a static IP address for the system. The IP addressing is set to 192.168.1.10 $x$  where  $x$  represents the quadrant (ie, 1-4). This is used to remotely access each system. Also, the hostname of each individual system was set in the */etc/hostname* file that contained the machine and quadrant number, ie (PM0Q1). Note that after the drive is cloned, the network adapter interface number will change on the cloned systems. *ifconfig* can be used to determine the interface number to use when configuring the */etc/network/interfaces* file.

The */etc/inittab* file is modified to include loading the camera firmware and starting the inspection software as specified in Section 6.12.

### 7.3.2 Software

The software required for machine operation is stored on each machine in */opt/pill.machine/*. These files can be obtained from the project folder *deploy* directory where the README file contains more specific information. There are no install scripts and thus each system must be configured manually.

---

## Chapter 8

---

### *Recommendations and Conclusions*

---

In order for any machine vision system to be introduced into the quality control process of the manufacturing of any product, it is essential that some means of system verification is performed. For a system to meet the standards required by the manufacturer, it must pass absolutely no flawed product, and the amount of rejected good product should be minimized. Pharmaphil has developed a document to quantify the competency of a system before introducing it into their process known as the Factory Acceptance Test (F.A.T) [30]. Because the developed system is merely a prototype, a thorough F.A.T has not been completed as software and hardware are continually evolving. The F.A.T must be passed before the system can be introduced into the facility.

As a measure to ensure the prototype system would meet expectations, a testing and verification stage was essential. This test was performed earlier in the development of the system and was not formalized. The system was configured with two cameras per quadrant (one centre and one bottom). For this stage of testing, the image processing routines for left and right images were incomplete and a lack of cameras prohibited a full test. The majority of information resides in these angles anyway. After various system tests, it was determined that the system was capable of providing reasonable inspection at a rate of approximately 850 capsules per minute. The inspection results were not formally recorded, however the following results were achieved:

- Absolutely no foreign capsules passed (ie. incorrect size, incorrect colour)
- Absolutely no incorrect size capsules passed
- Absolutely no dented capsules passed
- Absolutely no double capped capsules passed



- Most bubbles were failed, although some did pass. This is likely due to the orientation of the capsule, resulting in the bubble not being visible.

From the initial tests with two cameras per quadrant, the system was meeting its expectations, however it proved that four cameras per quadrant were required for a thorough inspection. Without modifying the camera angles in the inspection stage, the left and right angles provide little information on the top surface of the capsule. An example of a capsule with a bubble defect is shown in Figure 8.1. The orientation of the capsule is such that the defect is not clearly visible from the left or right camera angles, proving that with the current configuration, a center camera is required to provide a thorough inspection.

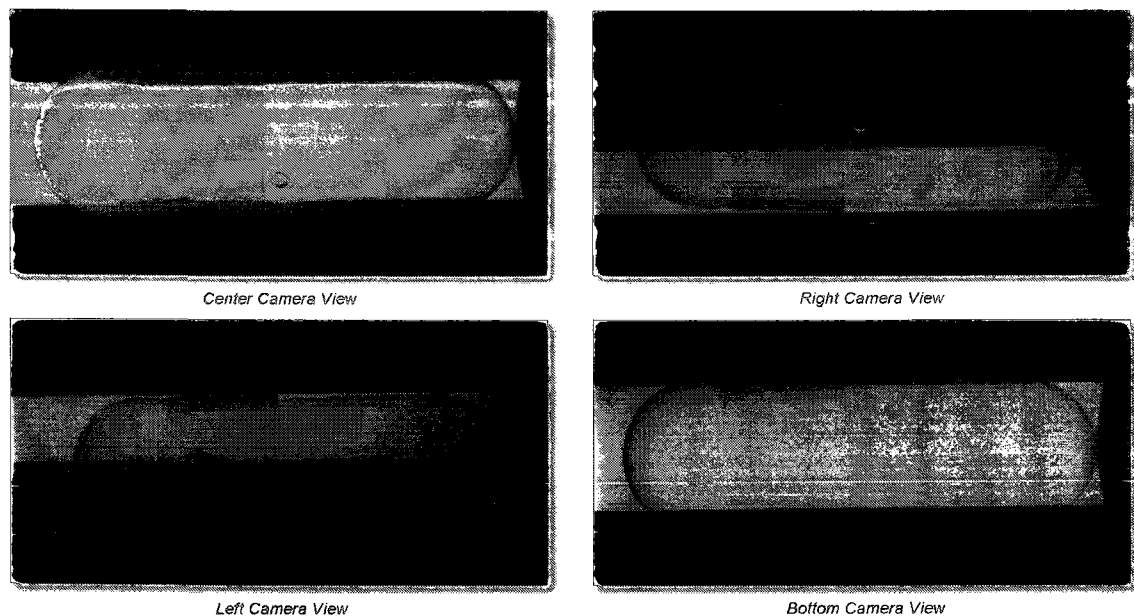


Figure 8.1: Camera Views of Bubble Defect

Preliminary tests with a four-camera inspection quadrant prove successful in meeting the inspection criterion. This preliminary testing should be followed by a formal and structured verification process.

The contributions to the project involve machine control and image acquisition. The capability of inspection is primarily reliant on the image processing portion. The overall machine control and image acquisition must be tested and verified to ensure proper images are obtained and that the ejection of capsule is in the correct slot (accept/reject).

The testing of the machine control portion of the system primarily verified that the system can operate at the desired inspection rate. Testing the system with a forced pass on all capsules

demonstrated that the maximum operational capsule inspection rate exceeds 1200 capsules per minute without skipping any capsules. Another verification was to ensure that the images arriving at the inspection PC match the order required to perform inspection. Measures to ensure proper alignment were necessary to ensure that the capsule images match the inspected capsule. Verification of image acquisition was also required to ensure the camera drivers and associated software function properly. This was accomplished by ensuring that the images match the capsule being inspected.

The overall goal of determining whether or not a system could be developed to perform individual inspection of two-part gelatin capsules was successful. It is evident from the research and development of this project that it is indeed possible to create a low-cost system that is capable of inspecting capsules with a great deal of accuracy. Although the developed prototype system does suffer from flaws, they are not significant enough to deter future development with the current design.

Some design errors, due to inexperience and time constraints, were witnessed in the project. A design error in the layout of the USB2.0 camera PCB resulted in the modification of the board. This modification involved attaching patch wires to various regions of the board to the Cypress FX2 USB2.0 MCU. The heat directly on the pins of the FX2 MCU damaged the device such that it experienced intermittent resets. This problem seemed to be accelerated by the addition of light to the image sensor (which is directly connected to the Cypress FX2). As a result, the modified USB2.0 cameras experience failure after a short running time. There is no simple remedy for this problem since the FX2 MCU is permanently damaged. Although replacing the device may successfully remedy the problem, the time and complexity required for this would be overwhelming. The simplest solution for the intermittent reset of the FX2 MCU is the submission of a corrected layout design for fabrication.

USB2.0 seems to adequately meet the transfer rate requirements to meet timing requirements. The maximum resolution of 3.1 megapixels is far more than what is required for the application. By reducing the image resolution to about 0.3 megapixels, not only is the transfer time reduced but the processing time is also reduced without compromising inspection. By reducing the image size from 2048x1536 to a windowed region with pixel skip enabled for both the horizontal and vertical directions [22], with a resulting image size of 768x384, the transfer and inspection times are effectively decreased by a factor of about 10. This greatly improves system performance and allows for better system throughput while meeting inspection detail requirements.

The addition of the fourth camera to the quadrants was a necessary step in ensuring a thorough inspection. Although redundant information appears, this information does not negatively affect inspection in any way.

From the testing stage, it seems that the software is quite stable and can run for extended periods.

*inspect* was tested overnight to verify its stability. Using two cameras with the sensors in test data mode (to prevent failure from the instability problem mentioned earlier), the system was tested for a period of about 15 hours with an external trigger firing every 500 ms. This test resulted in a total of 200,000 successful image acquisition with a total of 12 misaligned images. Misalignment can be caused by the OS shifting resources to other processes and is inevitable. This affected only 0.0123% of the “inspections” and is quite negligible. Any inspection affected by a misaligned image will be rejected.

The system control board seems to be quite stable when operated for extended periods. To improve robustness of the system, the UART handler firmware should be improved. If UART requests are performed incorrectly, the UART module will not work as expected and cannot recover on its own. Thus, error handlers must be created so the UART module is not in an unexpected state. On occasion, the system control board seems to report incorrect counts. This source of this problem has not been determined but could be the way the variables are stored, or due to errors occurring during the transmission to the PC over the UART interface.

An oversight in the design of the system control board requires that jumpers are placed across the input interrupt pin and the 3.3V pin of the buffered I<sup>2</sup>C connectors for the interrupt to be properly triggered. This will only affect the system if the interrupt is required on the buffered I<sup>2</sup>C slave devices. This is because the interrupt lines are active low and are in an open drain configuration. The simplest solution would be to include the pull-up resistors for the interrupt lines directly on the system control board rather than on the peripheral board.

An additional oversight on the design of the system control board involves the soft PC powering circuit. The motherboard tested for the design had different powering requirements than the actual motherboard used in the final prototype. This required a minor circuit modification to properly function. The circuit was designed such that the power switch on the motherboard caused the input to be pulled high, however in the ASUS motherboards used, the power pin must be pulled to ground. The circuit modification for the prototype was made by incorporating the changes in the cable following the schematic shown in Figure 8.2. The two required modifications are outlined and include the addition of a pull up resistor on the PWR\_SW\_+<sub>n</sub> line to 3.3V, and connecting the PWR\_SW\_-<sub>n</sub> line to ground.

The USB2.0 camera FPGA firmware does not support colour interpolation and is limited to outputting RAW data or grayscale interpolated image data. To support colour images, more output data is required from the FPGA than is inputted to it. This would require the system to support more buffering room, or the PXCLK of the image sensor would need to be divided accordingly. When considering computational time versus transfer time, it must also be determined at which point it is more feasible to perform interpolation in software rather than hardware. With the current system

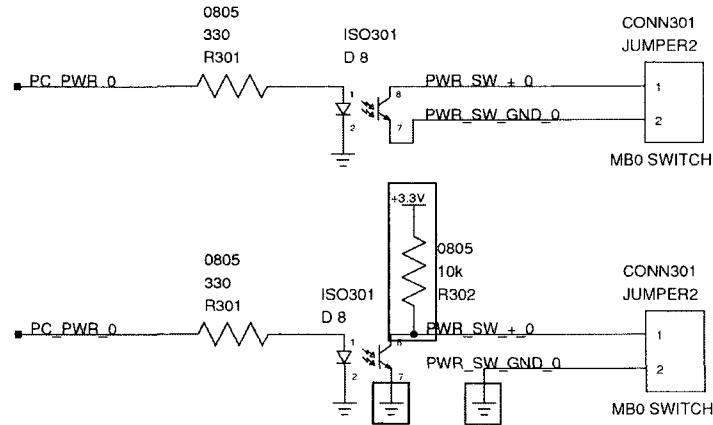


Figure 8.2: System Control Board Soft PC Power Fix

configuration, the transfer time is roughly 15 to 20 ms. Thus, if full interpolation was performed to output Y, U and V channels, the transfer time would be effectively three times the current transfer rate, around 45 to 60 ms. Other data methods may include using colour channel data whereby interpolation is not performed at all. In this technique, from the Bayer pattern image, the red, green and blue channels can be extracted without interpolation. From this, a YUV image data can be obtained with an effective image size of 1/4 of the interpolated version. It must be verified that the data loss by skipping interpolation does not significantly impact inspection. This would seemingly provide a reasonable image for the purposes of inspection. From preliminary tests using this technique, the time required to extract the RGB colour channels from a 0.3 megapixel raw bayer image is about 1.5 ms, with an extra 4.8 ms required to convert it to YUV. This is a total of about 7ms compared to 45 to 60 ms required to transfer the full YUV channels if converted in the camera. It may be beneficial to explore colour channel extraction techniques further.

The finished prototype did have flaws and limited operating time, but was sufficient to prove that such a system is realizable. With the development to date, the areas of weakness can be assessed and with a proper development strategy, the system can be taken from the prototype stage to the first generation commercial unit. Some optimization is required to reach the target inspection rate of 1000 capsules per minute from the current approximately 800 capsules per minute. This should be realizable in software and is an attainable target. The proof-of-concept prototype system was developed throughout the course of the project from which Pharmaphil can begin designing the first generation commercial units.

# References

- [1] A. Karloff, N. Scott and R. Muscedere. A Flexible Design for a Cost Effective, High Throughput Inspection System for Pharmaceutical Capsules. In *11th International Conference on Information Technology, (ICIT 2008)*, April 2008.
- [2] Analog Devices, Inc. CMOS 4-/8- Channel Analog Multiplexers ADG508A/ADG509A. [http://www.analog.com/static/imported-files/data\\_sheets/ADG508A\\_509A.pdf](http://www.analog.com/static/imported-files/data_sheets/ADG508A_509A.pdf), 2007.
- [3] Analog Devices, Inc. Triple/Quad SPDT CMOS Switches ADG1433/ADG1434. [http://www.analog.com/static/imported-files/data\\_sheets/ADG1433.1434.pdf](http://www.analog.com/static/imported-files/data_sheets/ADG1433.1434.pdf), 2008.
- [4] Cypress Semiconductor Corporation. EZ-USB Technical Reference Manual. [http://download.cypress.com.edgesuite.net/design\\_resources/technical\\_reference\\_manuals/contents/ez\\_usb\\_r\\_\\_technical\\_reference\\_manual\\_trm\\_14.pdf](http://download.cypress.com.edgesuite.net/design_resources/technical_reference_manuals/contents/ez_usb_r__technical_reference_manual_trm_14.pdf), 2005.
- [5] Daiichi Jitsugyo Viswill Co. Ltd. Capsule Visual Inspection System - CVIS-SXX-E. [http://www.viswill.jp/English/CVIS\\_E/cvis.index.e.html](http://www.viswill.jp/English/CVIS_E/cvis.index.e.html), 2005.
- [6] Edmund Optics Inc. Comparison of Camera Interfaces, April 2007.
- [7] Eisai Machinery U.S.A. Inc. Printing and Inspection for Tablets and Capsules. <http://www.eisaiusa.com/printingandinspection.htm>, 2007.
- [8] Fairchild Semiconductor. 74AC139, 74ACT139 Dual 1-of-4 Decoder/Demultiplexer. <http://www.fairchildsemi.com/ds/74/74AC139.pdf>, November 1999.
- [9] Fairchild Semiconductor Corporation. MOCD207M, MOCD208M Dual Channel Phototransistor Small Outline Surface Mount Optocouplers. <http://www.fairchildsemi.com/ds/MO/MOCD208-M.pdf>, November 2006.
- [10] Hitachi Semiconductor. HD44780 - Dot Matrix Liquid Crystal Display Controller / Driver. <http://www.sparkfun.com/datasheets/LCD/HD44780.pdf>.
- [11] International Rectifier. IRF7103Q HEXFET Power MOSFET. <http://www.irf.com/product-info/datasheets/data/irf7103q.pdf>, March 2002.
- [12] International Rectifier. IRF7306 HEXFET Power MOSFET. <http://www.irf.com/product-info/datasheets/data/irf7306.pdf>, September 1997.
- [13] Keil(TM). C51 Development Tools - Product Overview. <http://www.keil.com/c51/>, 2008.
- [14] Lambda Photometrics. Illumination. <http://www.lambdaphoto.co.uk/products/120.105>, 2008.

- 
- [15] Lawrence Livermore National Laboratory. POSIX Threads Programming. <https://computing.llnl.gov/tutorials/pthreads/>, 2008.
  - [16] Linear Technology Corporation. LTC1386 3.3V Low Power EIA/TIA562 Transceiver. <http://www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1007,C1016,P1044,D1590>, 1994.
  - [17] Linear Technology Corporation. LT1763 Series 500mA, Low Noise, LDO Micropower Regulators. <http://www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1003,C1040,C1055,P1778,D3903>, 1999.
  - [18] Microchip Technology Inc. dsPIC33FJXXXGPX06/X08/X10 Data Sheet High-Performance, 16-Bit Digital Signal Controllers. <http://ww1.microchip.com/downloads/en/DeviceDoc/70286A.pdf>, 2007.
  - [19] Microchip Technology Inc. dsPIC33F Family Reference Manul High-Performance Digital Signal Controllers. <http://ww1.microchip.com/downloads/en/DeviceDoc/70046E.pdf>, February 2006.
  - [20] Microchip Technology Inc. Explorer 16 Development Board User's Guide. <http://ww1.microchip.com/downloads/en/DeviceDoc/Explorer%2016%20User%20Guide%2051589a.pdf>, July 2006.
  - [21] Microchip Technology Inc. dsPIC30F/33F Programmer's Reference Manul High-Performance Digital Signal Controllers. <http://ww1.microchip.com/downloads/en/DeviceDoc/70157C.pdf>, March 2008.
  - [22] Micron Technology Inc. MT9T001 1/2-Inch 3.1-Megapixel Digital Image Sensor Data Sheet. <http://www.micron.com>, 2004.
  - [23] Micron Technology Inc. Micron Imaging FX2 Firmware Overview. Evaluation Board Companion CD [CONFIDENTIAL], October 2004.
  - [24] Neil E. Scott. USB2.0 Camera Technical Reference Manual. Internal Document, August 2008.
  - [25] Neil E. Scott. System Control Board Technical Reference Manual. Internal Document, July 2007.
  - [26] Neil E. Scott. I/O Expansion Board Technical Reference Manual. Internal Document, July 2008.
  - [27] NXP Semiconductors. BSS84 P-Channel Enhancement Mode Vertical D-MOS Transistor Product Data Sheet. [http://www.nxp.com/acrobat\\_download/datasheets/BSS84\\_4.pdf](http://www.nxp.com/acrobat_download/datasheets/BSS84_4.pdf), July 2007.
  - [28] NXP Semiconductors. Remote 16-bit I/O expander for I2C-Bus with Interrupt. [http://www.nxp.com/acrobat/datasheets/PCA8575\\_2.pdf](http://www.nxp.com/acrobat/datasheets/PCA8575_2.pdf), March 2007.
  - [29] NXP Semiconductors. P82B715 I2C-Bus Extender Product Data Sheet. [http://www.nxp.com/acrobat\\_download/datasheets/P82B715\\_7.pdf](http://www.nxp.com/acrobat_download/datasheets/P82B715_7.pdf), May 2008.
  - [30] Pharmaphil Inc. Factory Acceptance Test. Internal Document, January 2005.
  - [31] STMicroelectronics. L78xxAB L78xxAC Precision 1A Regulators. <http://www.st.com/stonline/books/pdf/docs/2144.pdf>, July 2008.
  - [32] Texas Instruments. Digital Temperature Sensor with Two-Wire Interface (Rev. J). <http://focus.ti.com/lit/ds/symlink/tmp175.pdf>, December 2007.
-

- [33] Texas Instruments Inc. TPS750003 Triple-Supply Power Managment IC for Powering FPGAs and DSPs Data Sheet. <http://focus.ti.com/lit/ds/symlink/tps75003.pdf>, 2007.
- [34] Texas Instruments Incorporated. CD4068B Types CMOS 8-Input NAND/AND Gate. <http://focus.ti.com/lit/ds/symlink/cd4068b.pdf>, September 2003.
- [35] Texas Instruments Incorporated. CD74HCT126. <http://focus.ti.com/lit/ds/symlink/cd74hc126.pdf>, September 2003.
- [36] Universal Serial Bus. Universal Serial Bus 2.0 Specification. <http://www.usb.org/>, April 2000.
- [37] Unknown. GNU Radio - The GNU Software Radio. <http://gnuradio.org/trac>, 2008.
- [38] Unknown. SDCC - Small Device C Compiler. <http://sdcc.sourceforge.net/>, 2008.
- [39] Wikipedia.org. Universal Serial Bus. <http://en.wikipedia.org/wiki/USB>, July 2008.
- [40] WingmanTeam(R). USB Snoopy. <http://www.wingmanteam.com/usbsnoopy/>, 2002.
- [41] Xilinx Inc. Spartan-3E FPGA Family: Complete Data Sheet. [http://www.xilinx.com/support/documentation/data\\_sheets/ds312.pdf](http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf), April 2008.

---

## Appendix A

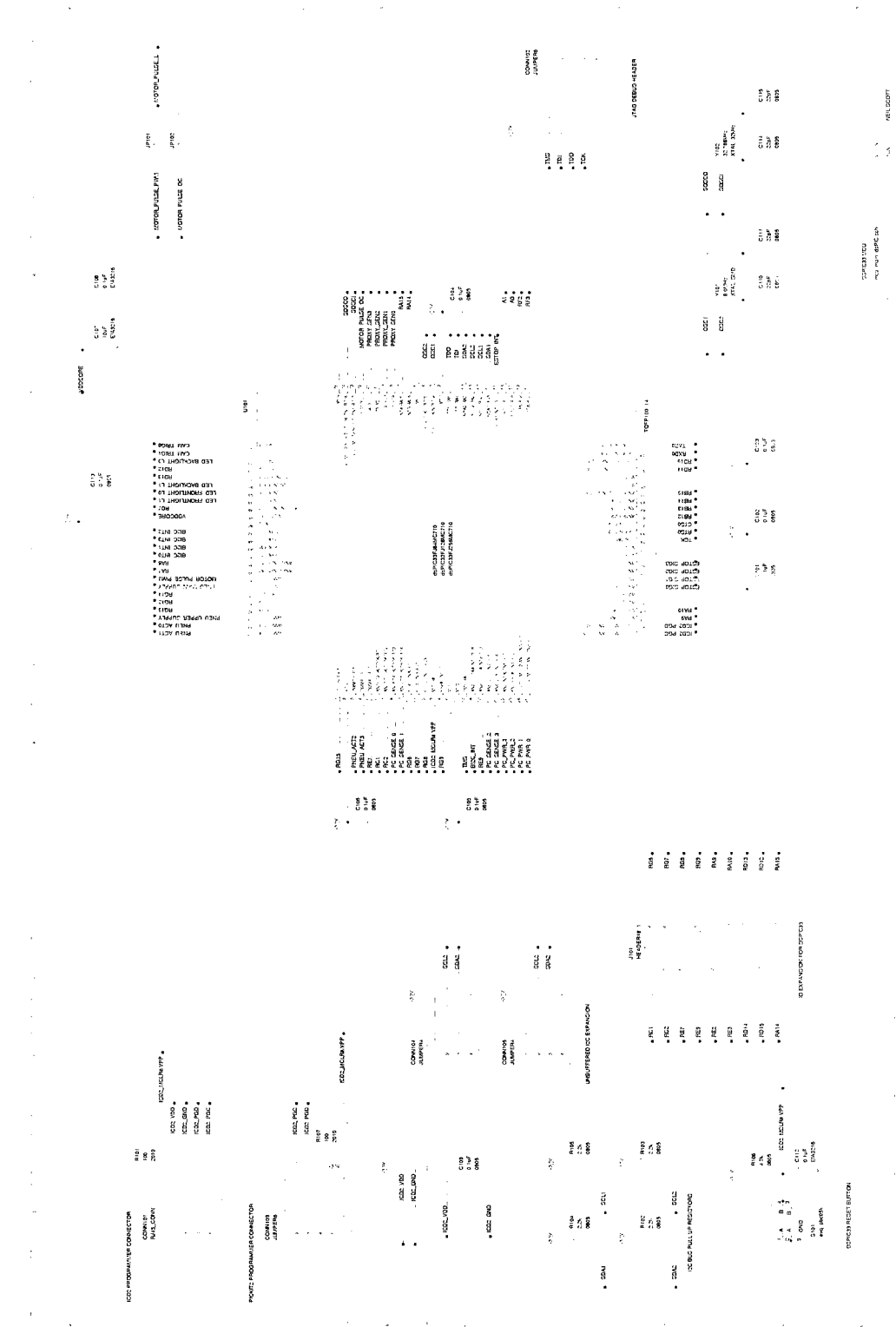
### *Control Board Design Reference*

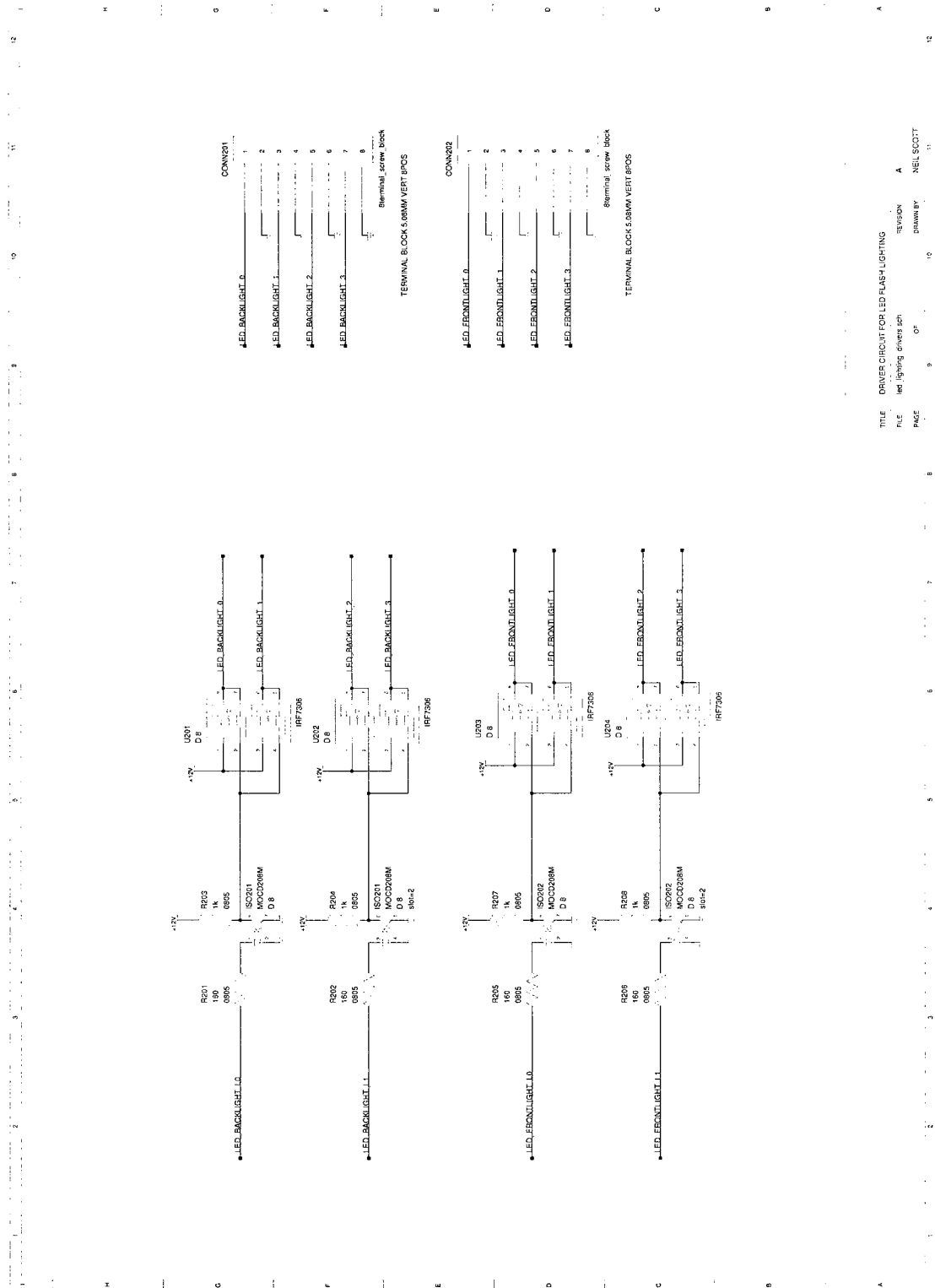
---

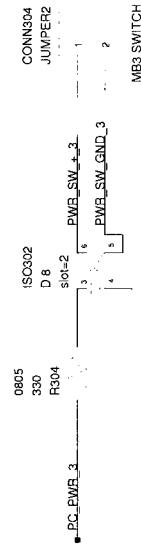
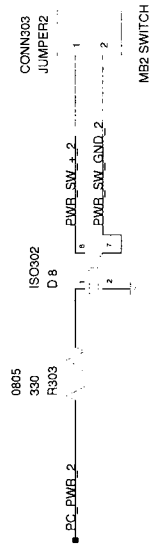
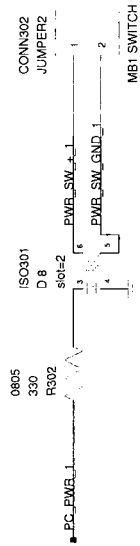
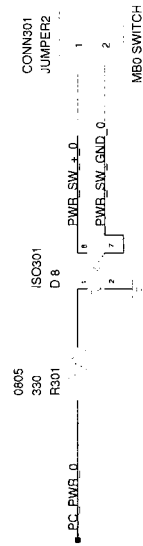
#### A.1 Control Board Schematics

The design schematics for the system control board.

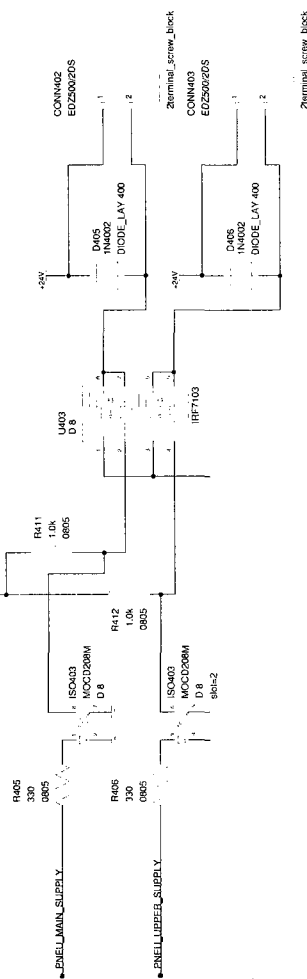






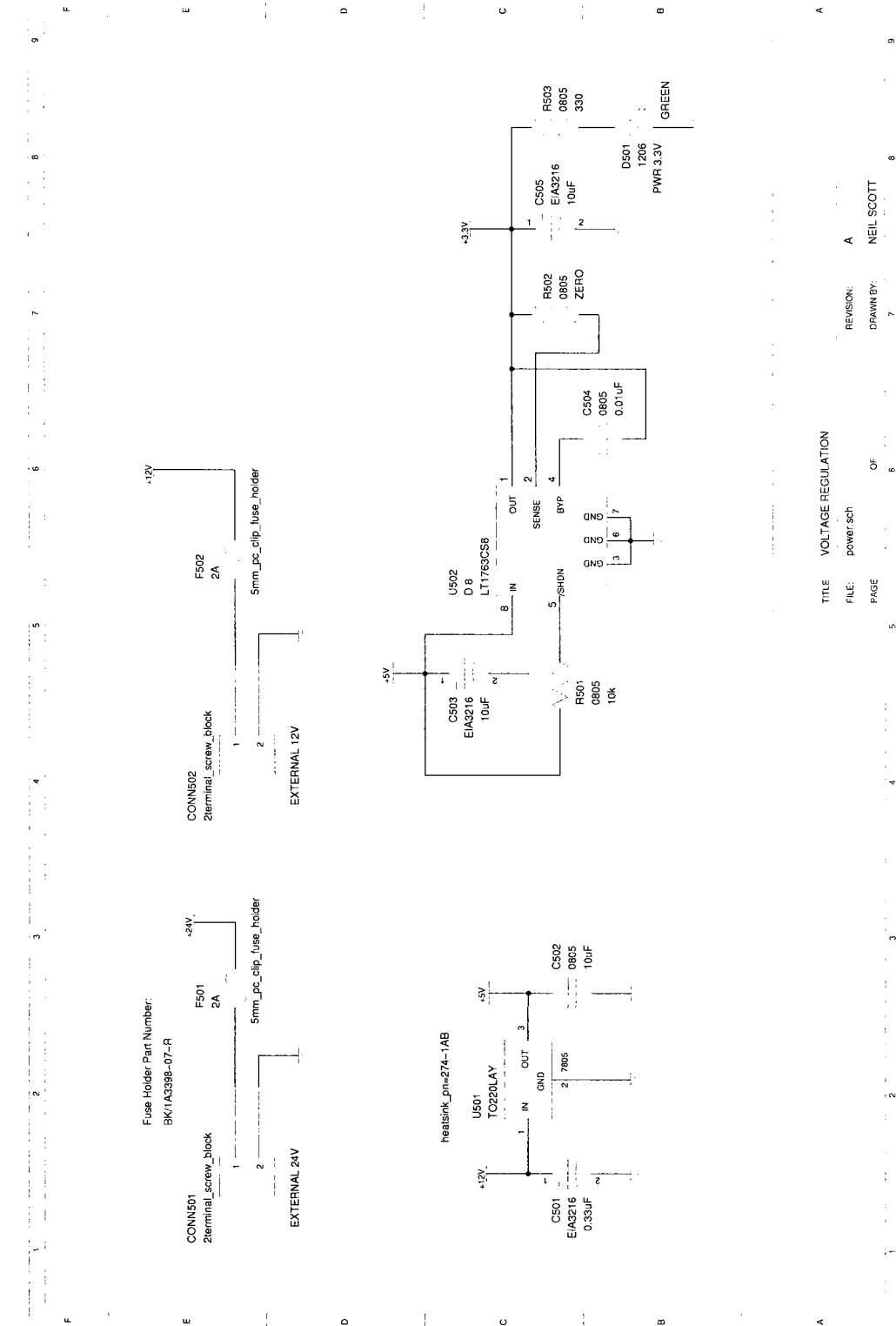


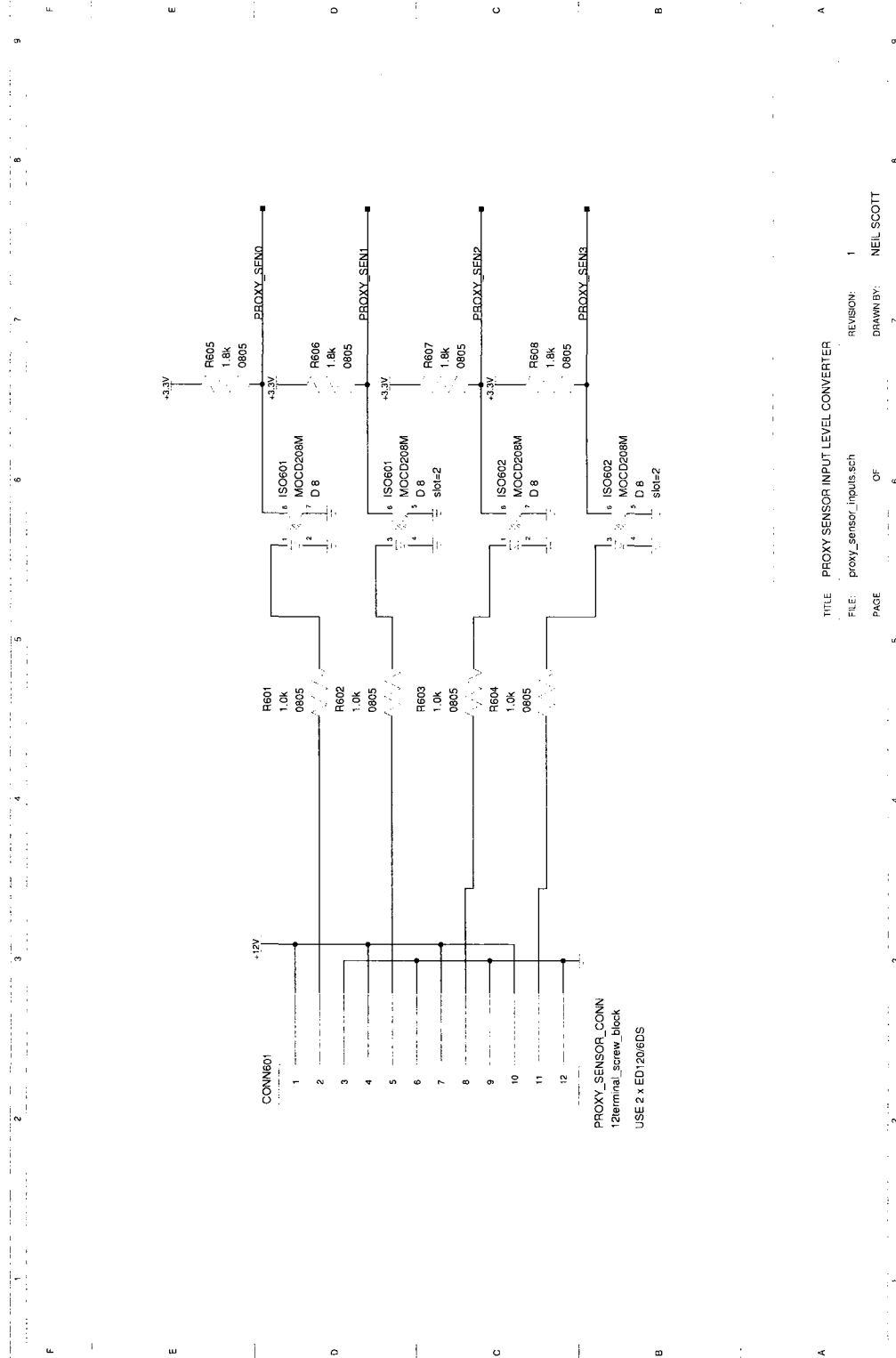
TITLE SOFT PC POWER SWITCH  
 FILE: PC\_power.sch  
 PAGE 1 OF 1  
 REVISION: 1  
 DRAWN BY: NEIL SCOTT

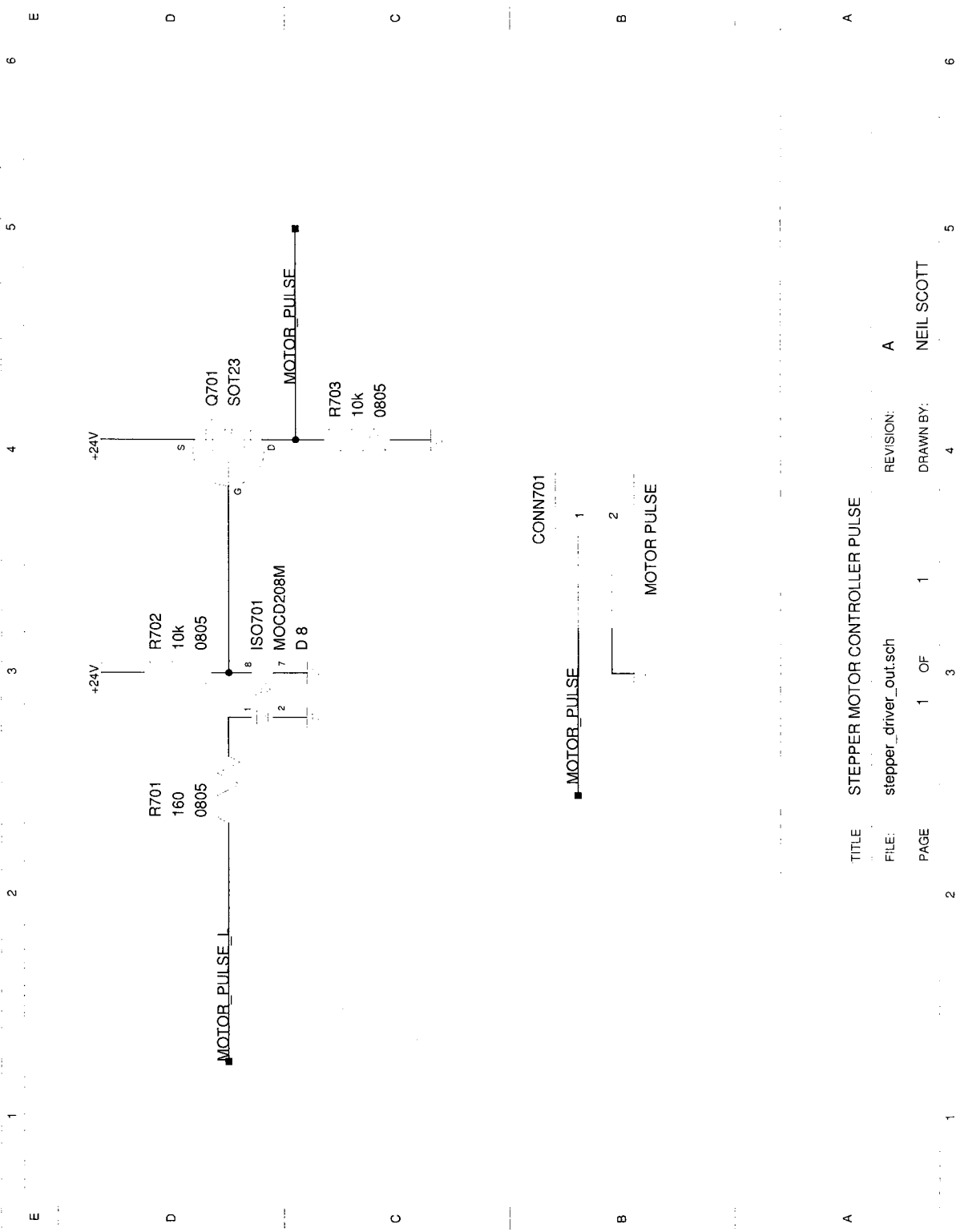


**TITLE** Driver Circuit for Pneumatic Actuators  
**FILE** pneu\_actuator\_driver.sch

NEVSON 1  
DRAWN BY NEIL SCOTT  
10







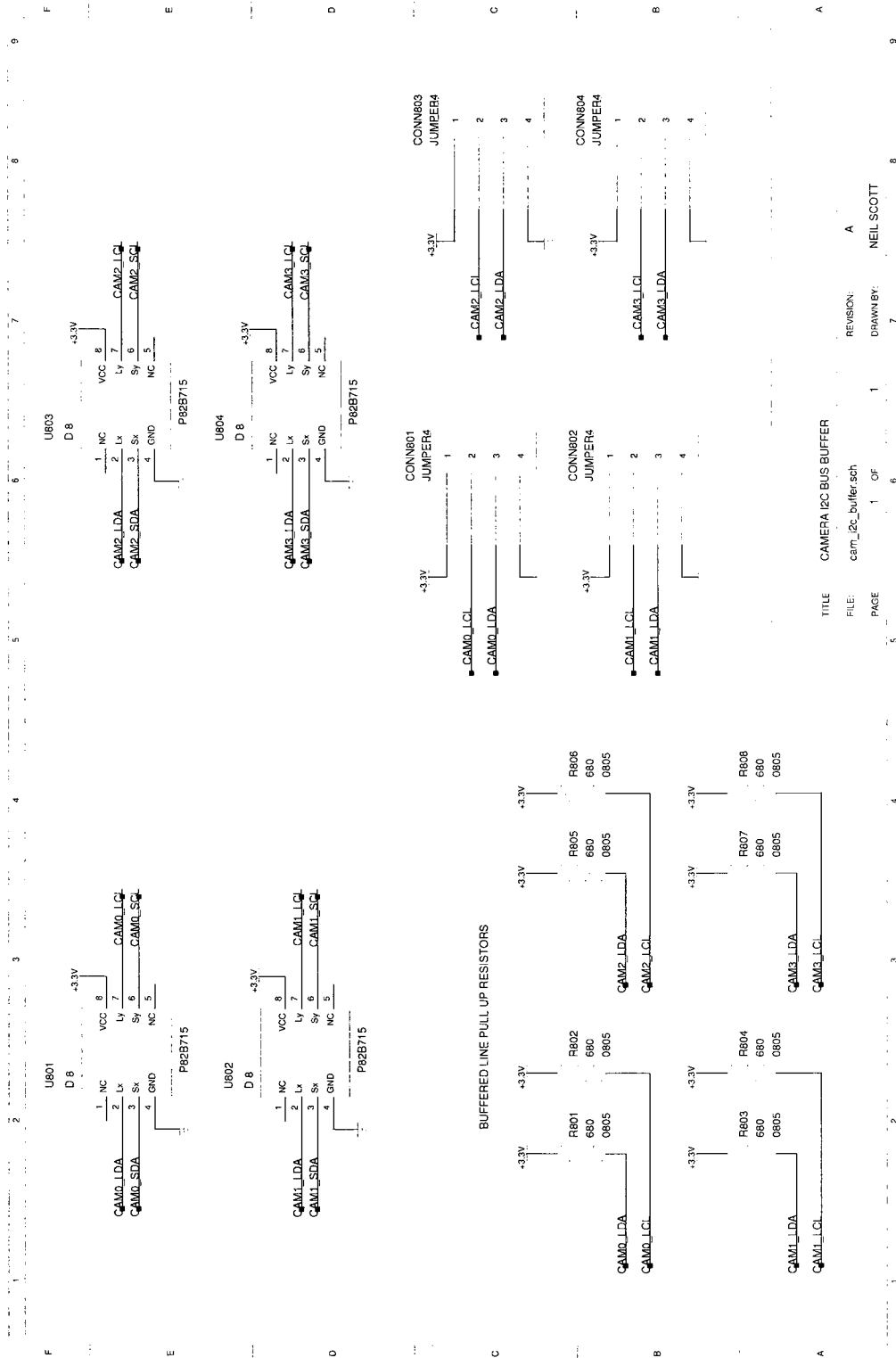
TITLE STEPPER MOTOR CONTROLLER PULSE

FILE: stepper\_driver\_out.sch

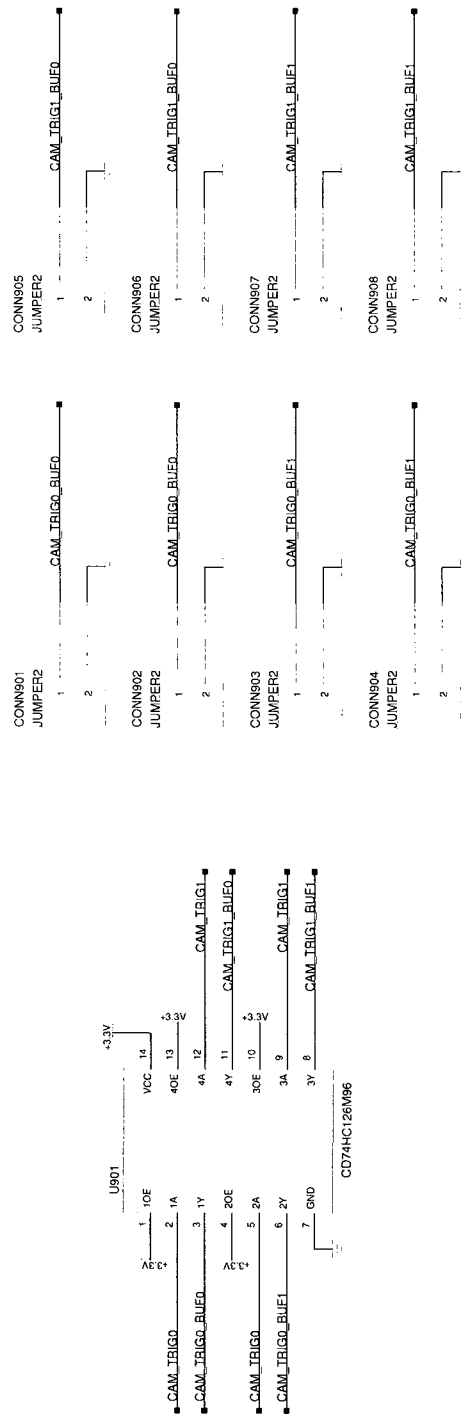
PAGE 1 OF 3

REVISION: A

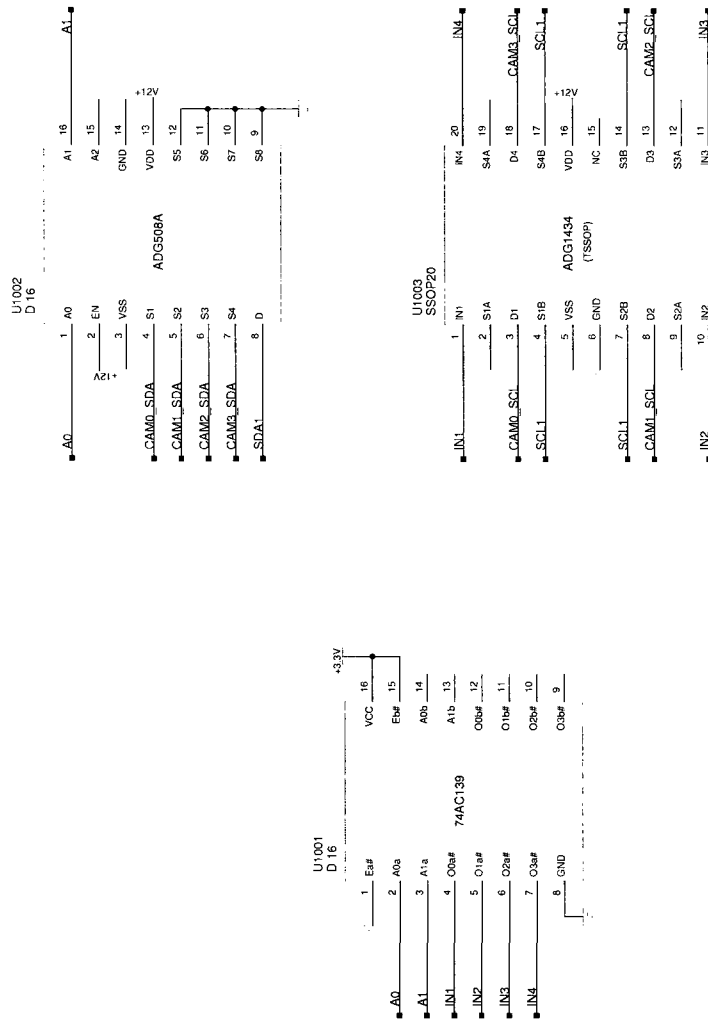
DRAWN BY: NEIL SCOTT



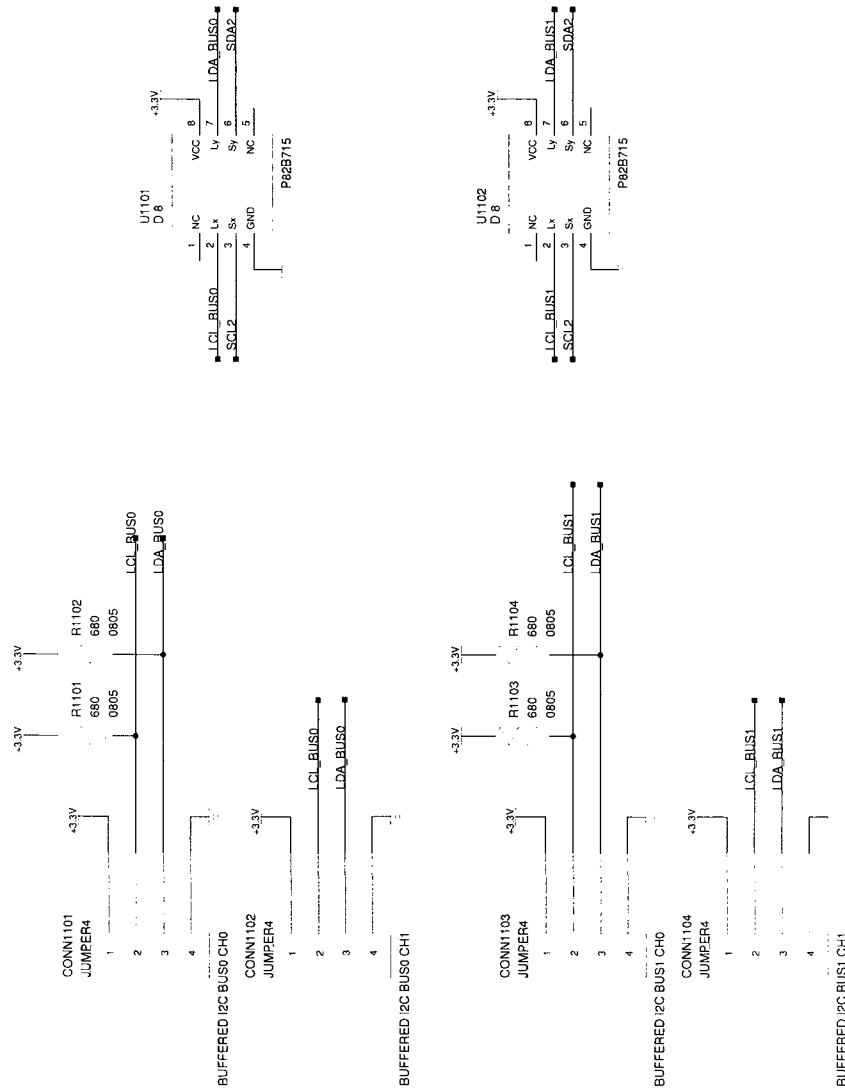




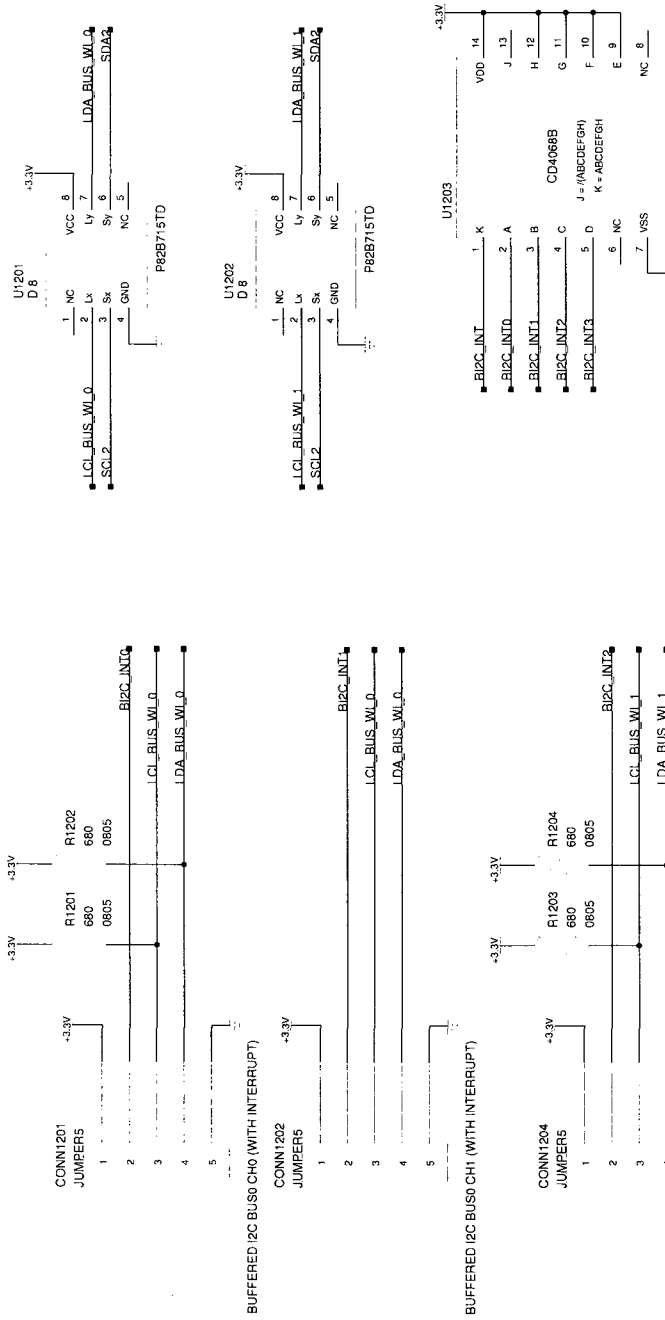
TITLE	CAMERA TRIGGER LINE BUFFER	REVISION	A
FILE	cam_triggers.sch	DRAWN BY:	NEIL SCOTT
PAGE	OF		



TITLE I2C BUS SWITCH  
FILE: i2c\_bus\_switch.sch  
PAGE 1 OF 1  
REVISION: A  
DRAWN BY: NEIL SCOTT



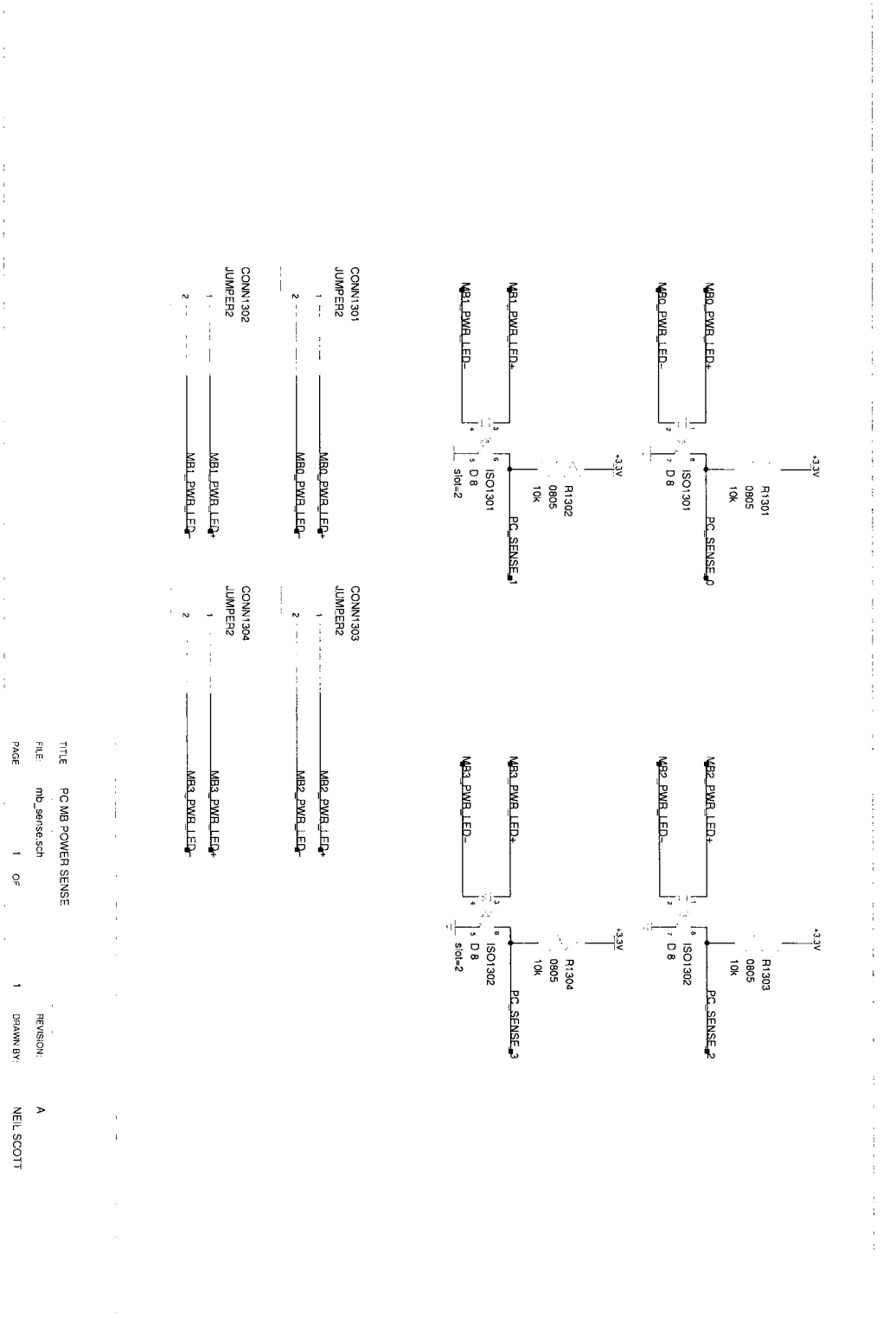
NOTE: ONLY SLAVE DEVICES SHOULD BE CONNECTED TO THIS BUFFERED BUS.	* CURRENT DRAW SHOULD BE LIMITED TO A MAXIMUM OF 200mA FOR ALL EXPANSION CONNECTORS 3.3V SUPPLY			
	TITLE	I2C EXPANSION	REVISION	A
	FILE:	i2c_expansion.sch	DRAWN BY:	NEIL SCOTT
	PAGE	OF		



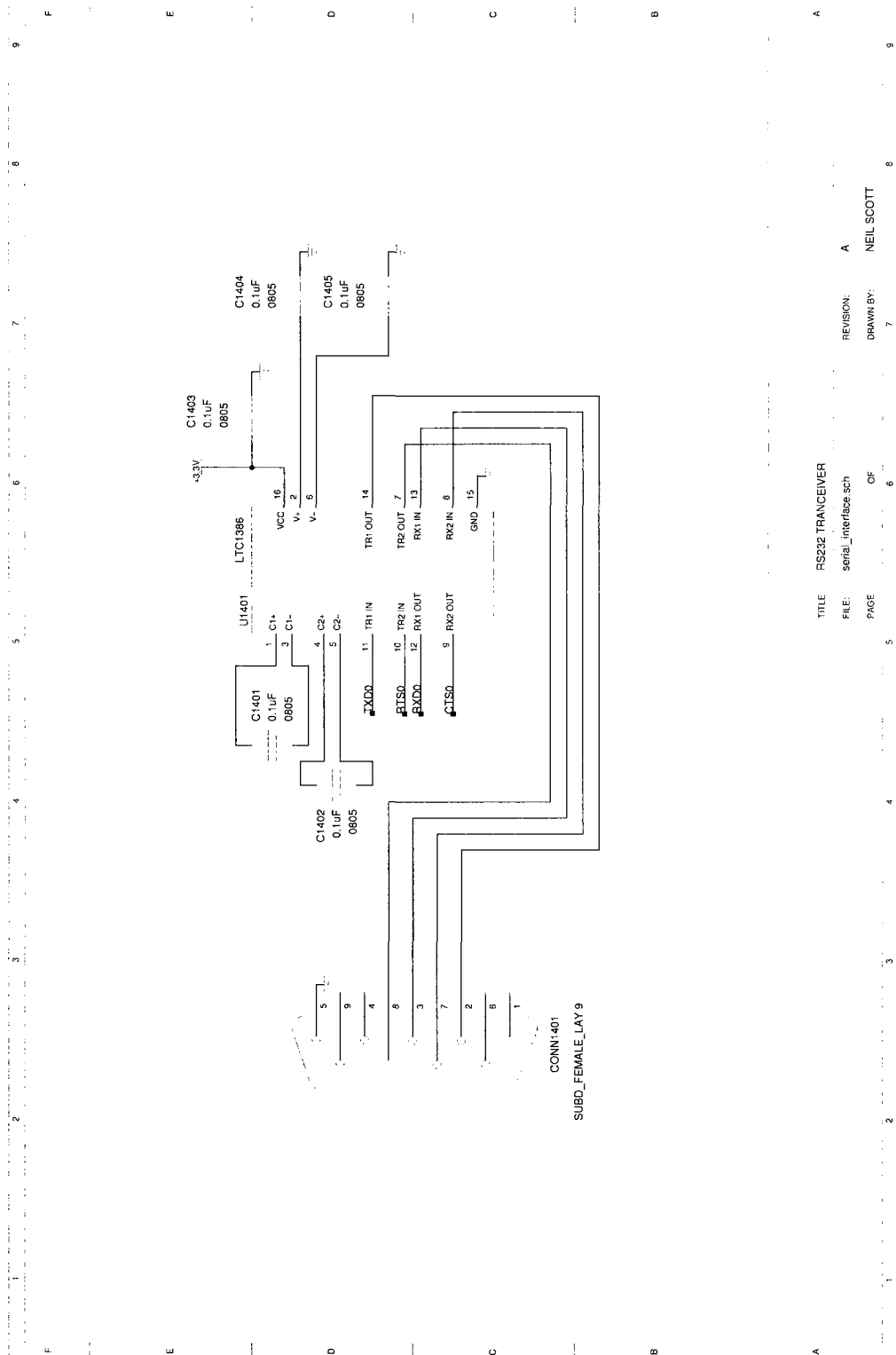
NOTE: ALL B12C\_INTx LINES ARE ACTIVE LOW

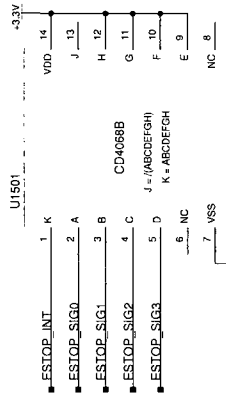
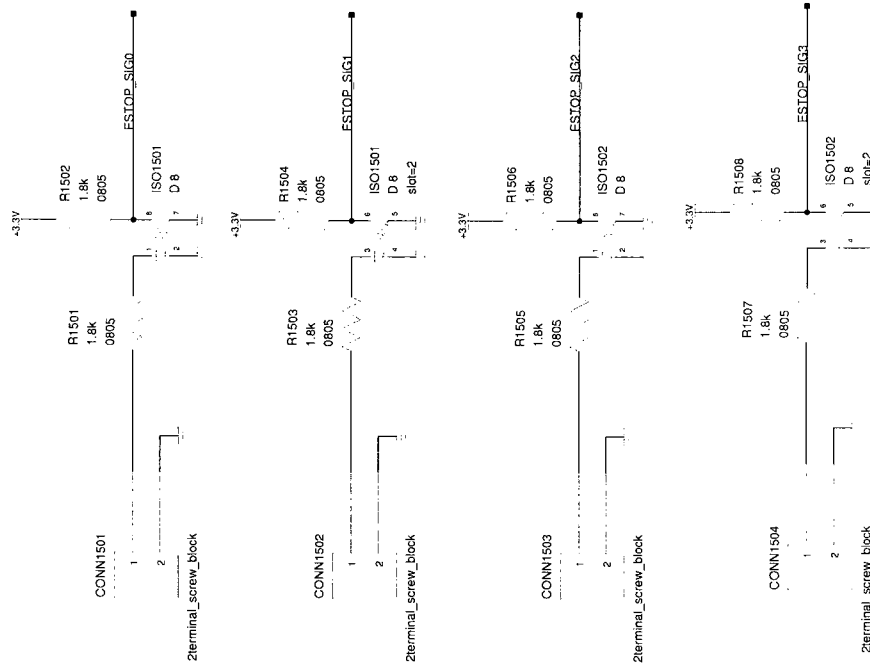
NOTE: MAXIMUM CURRENT DRAW SHOULD BE 200mA FOR ALL BUFFERED I2C EXPANSION CONNECTORS

TITLE: I2C EXPANSION  
 FILE: i2c\_expansion.sch  
 PAGE: OF  
 REVISION: A  
 DRAWN BY: NEIL SCOTT



TITLE PC MB POWER SENSE  
FILE mb\_sense.sch  
PAGE 1 OF 1  
REVISION: A  
DRAWN BY: NEIL SCOTT





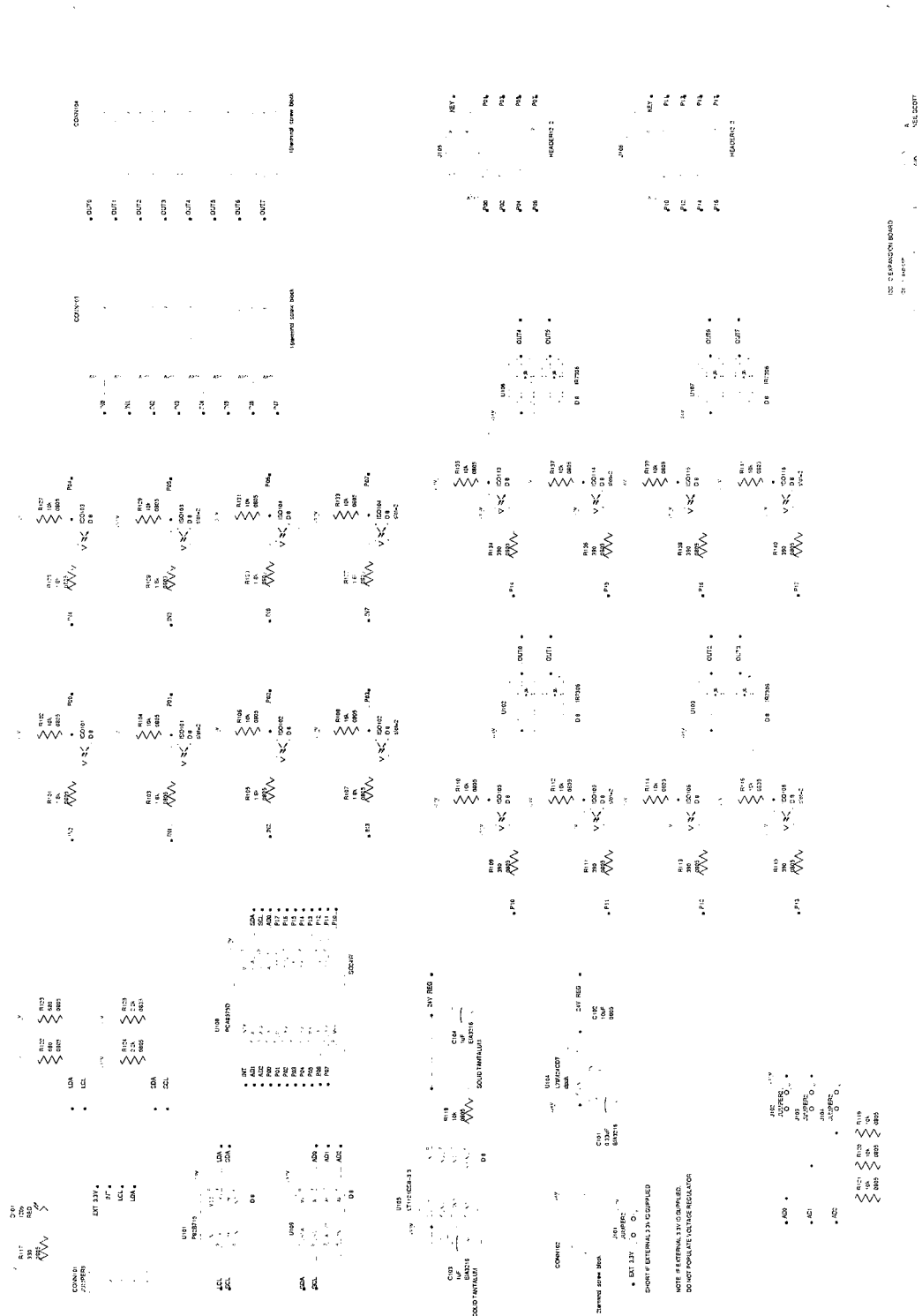
E-STOP SIGNALS ARE EXPECTED TO BE 12-24V  
FOR DIFFERENT VOLTAGES, ADJUST RESISTANCE VALUES

TITLE E-STOP INTERFACE FOR MULTIPLE E-STOP INPUTS  
FILE estop\_crt/sch  
PAGE OF  
REVISION: A  
DRAWN BY: NEIL SCOTT

## A.2 I<sup>2</sup>CI/O Expansion Board Schematics

The design schematics for the I<sup>2</sup>CI/O expansion board.





---

## Appendix B

### *USB2.0 Camera FX2 Firmware*

---

This section contains the firmware source code developed for the Cypress FX2 USB microcontroller of the USB2.0 camera.

#### B.1 Cypress EZ-USB FX2 Vendor Requests

A summary of the USB vendor requests of the USB2.0 camera.

```
/* Vendor Specific Requests - Note: 0xA0 to 0xAF are reserved */
#define VRQ_I2C_READ                0x81    /* wValueL: i2c addr; wLength: data length */
#define VRQ_I2C_WRITE               0x08    /* wValueL: i2c addr; wLength: data length */

#define VRQ_READ_EEPROM_SM          0xE3
#define VRQ_WRITE_EEPROM_SM         0xE4    /* wValueL: i2c addr; wIndex: eeprom_addr;
                                           wLengthL: data length */

#define VRQ_READ_EEPROM_LG          0xE6    /* wValueL: i2c addr; wIndex: eeprom_addr;
                                           wLengthL: data length */
#define VRQ_WRITE_EEPROM_LG         0xE7

/* Set I2C Bus Speed */
#define VRQ_SET_I2C_SPEED           0xE5

/* Camera Specific */
#define VRQ_GET_CAM_STATUS           0xE0
#define VRQ_READ_CAM_POSITION        0xE1

/* Pill Accept/Reject */
#define VRQ_PILL_REJECT_ACCEPT       0xF0    /* wValueL: accept/reject {1,0}; */
```

```
/* Get Camera Quadrant and Position (read on startup from EEPROM) */
#define VRQ_GET_QUAD_POS                0xFA

/* FPGA Commands */
#define VRQ_FPGA_FLUSH                  0xF1
#define VRQ_FPGA_RESET                  0xF2    /* wValueL: enable/disable {1,0} */
#define VRQ_FPGA_POWER                  0xF3    /* wValueL: enable/disable {1,0} */

/* FPGA Loader Commands */
#define VRQ_FPGA_LOAD_SS                0xF5
/* Sub Commands */
#define FPGA_LOAD_START                 0x01
#define FPGA_LOAD_DATA                 0x02
#define FPGA_LOAD_CHECK_DONE           0x03

/* Frame Synchronization Commands */
#define VRQ_GET_FRAME_DROP_COUNT        0x30
#define VRQ_RESET_FRAME_DROP_COUNT      0x31

/* Turn on or off Frame Drop Interrupt */
#define VRQ_FRAME_DROP_INTERRUPT        0x32    /* wValueL: enable/disable {1,0} */

/* Window Set Commands */
#define VRQ_GET_WINDOW_PARAM            0x35    /* wIndexL: sub command */
/* sub commands */
#define VRQ_GET_WINDOW_WIDTH            0x01
#define VRQ_GET_WINDOW_LENGTH          0x02
#define VRQ_GET_WINDOW_COL_START       0x03
#define VRQ_GET_WINDOW_ROW_START       0x04
#define VRQ_GET_WINDOW_COL_SKIP        0x05
#define VRQ_GET_WINDOW_ROW_SKIP        0x06

#define VRQ_UPDATE_PARAMS               0xaa
```

## B.2 Micron Image Sensor Register Definitions

### B.2.1 mi\_regs.h

```
/* Filename:
 *   mi_regs.h
 *
 * Description:
 *   Micron MT9T001 CCD Register Definitions
 *
 * Author:
 *   Neil Scott
 *
 * Date:
 *   December 14, 2006
 */

#ifndef _MI_REGS_H
#define _MI_REGS_H 1

#define MI_REG_CHIP_VERSION            0x00
```

```
#define MI_REG_ROW_START          0x01
#define MI_REG_COLUMN_START      0x02
#define MI_REG_ROW_SIZE          0x03
#define MI_REG_COL_SIZE          0x04
#define MI_REG_HORIZ_BLANKING    0x05
#define MI_REG_VERT_BLANKING     0x06
#define MI_REG_OUTPUT_CONTROL    0x07
#define MI_REG_SHUTTER_WID_UPPER 0x08
#define MI_REG_SHUTTER_WIDTH     0x09
#define MI_REG_PX_CLK_CTRL       0x0A
#define MI_REG_RESTART           0x0B
#define MI_REG_SHUTTER_DELAY     0x0C
#define MI_REG_RESET             0x0D
#define MI_REG_READ_MODE_1       0x1E
#define MI_REG_READ_MODE_2       0x20
#define MI_REG_READ_MODE_3       0x21
#define MI_REG_ROW_ADDR_MODE     0x22
#define MI_REG_COL_ADDR_MODE     0x23
#define MI_REG_GREEN1_GAIN       0x2B
#define MI_REG_BLUE_GAIN        0x2C
#define MI_REG_RED_GAIN          0x2D
#define MI_REG_GREEN2_GAIN       0x2E
#define MI_REG_TEST_DATA         0x32
#define MI_REG_GLOBAL_GAIN       0x35
#define MI_REG_BLACK_LEVEL       0x49
#define MI_REG_ROW_BLK_DEFAULT_OFFSET 0x4B
#define MI_REG_BLC_DELTA_THRESH  0x5D
#define MI_REG_CAL_THRESH        0x5F
#define MI_REG_GREEN1_OFFSET     0x60
#define MI_REG_GREEN2_OFFSET     0x61
#define MI_REG_BLK_LEVEL_CAL     0x62
#define MI_REG_RED_OFFSET        0x63
#define MI_REG_BLUE_OFFSET       0x64
#define MI_REG_CHIP_EN_SYNC      0xF8
#define MI_REG_CHIP_VERSION2     0xFF

/* Skip and Bin Modes */
#define MI_COL_SKIP_NONE 0
#define MI_COL_SKIP_2X 1
#define MI_COL_SKIP_3X 2
#define MI_COL_SKIP_4X 3
#define MI_COL_SKIP_8X 4

#define MI_ROW_SKIP_NONE 0
#define MI_ROW_SKIP_2X 1
#define MI_ROW_SKIP_3X 2
#define MI_ROW_SKIP_4X 3
#define MI_ROW_SKIP_8X 4

#endif /* _MI_REGS_H */
```

## B.3 FX2 Firmware

### B.3.1 Makefile

```
srcdir = .
top_srcdir = ../..

XCC = sdcc -mmcs51 --no-xinit-opt
XAS = asx8051 -plogff
DEFINES = -DHAVE_USRP1
DEFS = -DHAVE_CONFIG_H

INCLUDES = -I$(top_srcdir)/firmware/include -I$(top_srcdir)/firmware/src -I../common

MEMOPTS = --code-loc 0x0000 --code-size 0x1800 --xram-loc 0x1800 --xram-size 0x0800 \
-Wl '-b USBDESCSEG = 0xE000'

LIBOPTS = -L ../lib libfx2.lib
```

```
LIBDEP = ../lib/libfx2.lib
LINKOPTS = $(MEMOPTS) $(LIBOPTS)
HEXFILES = \
    fx2cam_firmware.ihx

STARTUP = _startup.rel
FX2CAM_OBJS = \
    vectors.rel \
    fx2cam_main.rel \
    fx2cam_common.rel \
    usb_descriptors.rel \
    $(STARTUP)

CLEANFILES = \
    *.ihx *.lnk *.lst *.map *.mem *.rel *.rst *.sym *.asm *.lib

all: $(HEXFILES)

clean:
    rm -rf $(CLEANFILES)

%.rel : %.c
    $(XCC) $(INCLUDES) $(DEFINES) \
        -c -o $@ `test -f '$<' || echo '$(srcdir)'/`'$<

%.rel : %.a51
    test -f `basename '$<'` || ln -s '$<' .
    test -f ../common/`basename '$<'` -o \
        \! -f `dirname '$<'`/../common/`basename '$<'` \
        || ln -s `dirname '$<'`/../common/`basename '$<'` ../common/.
    $(XAS) `basename '$<'`

fx2cam_firmware.ihx: $(FX2CAM_OBJS) $(LIBDEP)
    $(XCC) $(LINKOPTS) -o $@ $(FX2CAM_OBJS)

#fx2cam_blink_leds.ihx: $(FX2CAM_OBJS) $(LIBDEP)
# $(XCC) $(LINKOPTS) -o $@ $(FX2CAM_OBJS)

fx2cam_main.rel: fx2cam_common.h ../include/fx2cam_commands.h ../include/fx2regs.h ../include/
    usb_common.h
fx2cam_common.rel: fx2cam_common.h ../include/fx2cam_commands.h ../include/fx2regs.h
```

### B.3.2 fx2cam\_common.h

```
/* Filename:
 *   fx2cam_common.h
 *
 * Description:
 *   Camera firmware constants.
 *
 * Author:
 *   Neil Scott
 *
 * Date:
 *   September 13, 2007
 *
 * Notes:
 *   February 25, 2008:
 *   Revised for REV.B of camera board. New routing.
 */

#ifndef _FX2CAM_COMMON_H
#define _FX2CAM_COMMON_H 1

#include "fx2regs.h"
#include <syncdelay.h>

/* USB Setup Packet */
#define bRequestType      SETUPDAT[0]
#define bRequest          SETUPDAT[1]
#define wValueL           SETUPDAT[2]
```

```

#define wValueH      SETUPDAT[3]
#define wIndexL      SETUPDAT[4]
#define wIndexH      SETUPDAT[5]
#define wLengthL     SETUPDAT[6]
#define wLengthH     SETUPDAT[7]

/* Camera Constants */
#define EP0BUFF_SIZE 0x40 /* 64-bytes */
#define I2C_EEPROM_ADDR 0x51 /* EEPROM I2C Address */

/* ===== FPGA RESET CONTROL ===== */
#define FPGA_RESET_PORT IOA
#define FPGA_RESET_BIT  bmBIT7

/* ===== FPGA POWER CHIP CS ===== */
/* FPGA Power Port */
#define FPGA_POWER_PORT IOE
/* FPGA Power Inputs */
#define FPGA_POWER_VCC_AUX  bmBIT3
#define FPGA_POWER_VCC_O  bmBIT2
#define FPGA_POWER_VCC_INT  bmBIT1

/* ===== FPGA LOADER ===== */
/* FPGA Loader Port */
#define FPGA_LOAD_PORT IOC
/* FPGA Loader Outputs */
#define FPGA_LOAD_CLK_BIT  bmBIT3
#define FPGA_LOAD_DATA_BIT  bmBIT2
#define FPGA_LOAD_PROG_BIT  bmBIT4
/* FPGA Loader Inputs */
#define FPGA_LOAD_INIT_BIT  bmBIT0
#define FPGA_LOAD_DONE_BIT  bmBIT1

/* TODO: FPGA Interrupt Lines
 * - Start of frame
 * - End of frame
 * - Frame error
 */
#define FPGA_FS_PORT IOA
#define FPGA_FS0_BIT  bmBIT2
#define FPGA_FS1_BIT  bmBIT3

/* TODO: I2C SCL Loop back
 * - Used to check bus state before starting transmission
 */
#define SCL_LB_PORT IOC
#define SCL_LB_BIT  bmBIT5

/* TODO: ADD ERROR CONDITIONS -- Maybe put in separate header */
/* System Error Codes */
#define ERR_FPGA_LOAD 0xA1
#define ERR_FPGA_FIFO_FULL 0xB1

/* Function Prototypes */
void init_fx2cam (void);

#endif /* _FX2CAM_COMMON_H */

```

### B.3.3 fx2cam\_ids.h

```

/* Filename:
 *   fx2cam_ids.h
 *
 * Description:
 *   FX2 Camera USB IDs. Must be consistent with usb_descriptors.a51
 *
 * Author:
 *   Neil Scott
 *
 * Date:
 *   December 08, 2006

```

```
*/

#ifndef _FX2CAM_IDS_H_
#define _FX2CAM_IDS_H_ 1

/* Vendor ID and Product ID */
#define USB_FX2CAM_VID      0xabcd
#define USB_FX2CAM_PID      0x0201

/* Unconfigured Device ID */
#define USB_FX2CAM_DID_0    0x0000

/* Configured Device ID */
#define USB_FX2CAM_DID_1    0x0001

#endif /* _FX2CAM_IDS_H_ */
```

### B.3.4 fx2cam\_i2c\_addr.h

```
/* Filename:
 *   fx2cam_i2c_addr.h
 *
 * Description:
 *   Defines for all i2c device addresses on bus
 *
 * Author:
 *   Neil Scott
 *
 * Date:
 *   December 08, 2006
 */

#ifndef FX2CAM_I2C_ADDR_H
#define FX2CAM_I2C_ADDR_H 1

/* i2c addresses */
#define I2C_EEPROM          0x50    /* Microchip 24LC??? */
#define I2C_IO_EXP_7SEG      0x21    /* Philips I/O Expander for Seven Segment Display */
#define I2C_IO_EXP_FKEYSi    0x20    /* Philips I/O Expander for PBs F1 to F4 (dev board) */
#define MI_I2C_ADDR         0x5D    /* Micron Sensor I2C slave address */
#define FPGA_I2C_ADDR        0x55    /* FPGA I2C slave address */
#define CB_I2C_ADDR          0x44    /* Control Board I2C slave address */
#define TMP175_I2C_ADDR      0x48    /* TMP175 Temperature Sensor I2C slave address */
#endif /* FX2CAM_I2C_ADDR_H */
```

### B.3.5 fx2cam\_usb.h

```
#include <usb.h>
#include "fx2cam_commands.h"

#define FX2CAM_VENDOR_ID    0xABCD
#define FX2CAM_PRODUCT_ID   0x0201

/* USB Specific */
#define VENDOR_REQUEST_OUT   0x40
#define VENDOR_REQUEST_IN    0xC0
#define BULK_EP2_OUT_ADDR    0x02
#define BULK_EP6_IN_ADDR     0x86

#define BULK_EP2_SIZE        512
#define BULK_EP6_SIZE        512

/* Vendor Specific */
#define VRQ_I2C_WRITE         0x08
#define VRQ_I2C_READ          0x81
#define VRQ_NEIL_GET_CAMERA_POS 0xFE

#define VRQ_WRITE_EEPROM_SM   0xFA
#define VRQ_READ_EEPROM_SM    0xFB
#define VRQ_WRITE_EEPROM_LG   0xFC
#define VRQ_READ_EEPROM_LG    0xFD
```

```
#define VRQ_SET_I2C_SPEED      0xE5

/* 7 Segment Display I2C Addr */
#define LED7SEG_I2C_ADDR      0x21
#define PBFKEYS_I2C_ADDR      0x20

#define FX2CAM_USB_TIMEOUT     2000
```

### B.3.6 \_startup.a51

```
;;; -*- asm -*-
;;;
;;; Copyright 2003,2004 Free Software Foundation, Inc.
;;;
;;; This file is part of GNU Radio
;;;
;;; GNU Radio is free software; you can redistribute it and/or modify
;;; it under the terms of the GNU General Public License as published by
;;; the Free Software Foundation; either version 2, or (at your option)
;;; any later version.
;;;
;;; GNU Radio is distributed in the hope that it will be useful,
;;; but WITHOUT ANY WARRANTY; without even the implied warranty of
;;; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
;;; GNU General Public License for more details.
;;;
;;; You should have received a copy of the GNU General Public License
;;; along with GNU Radio; see the file COPYING. If not, write to
;;; the Free Software Foundation, Inc., 59 Temple Place - Suite 330,
;;; Boston, MA 02111-1307, USA.

;;; The default external memory initialization provided by sdcc is not
;;; appropriate to the FX2. This is derived from the sdcc code, but uses
;;; the FX2 specific _MPAGE sfr.

    .area XISEG    (XDATA) ; the initialized external data area
    .area XINIT    (CODE) ; the code space consts to init XISEG
    .area XSEG      (XDATA) ; zero initialized xdata
    .area USBDESCSEG (XDATA) ; usb descriptors

    .area CSEG      (CODE)

    ; sfr that sets upper address byte of MOVX using @r0 or @r1
    _MPAGE = 0x0092

__sdcc_external_startup::
    ; This system is now compiled with the --no-xinit-opt
    ; which means that any initialized XDATA is handled
    ; inline by code in the GSINIT segs emitted for each file.
    ;
    ; We zero XSEG and all of the internal ram to ensure
    ; a known good state for uninitialized variables.

; _mcs51_genRAMCLEAR() start
mov r0,#1_XSEG
mov a,r0
orl a,#(1_XSEG >> 8)
jz 00002$
mov rl,#((1_XSEG + 255) >> 8)
mov dptr,#s_XSEG
clr a

00001$: movx @dptr,a
inc dptr
djnz r0,00001$
djnz rl,00001$

    ; We're about to clear internal memory. This will overwrite
```



```
;; the stack which contains our return address.
;; Pop our return address into DPH, DPL
00002$: pop dph
pop dpl

;; R0 and A contain 0. This loop will execute 256 times.
;;
;; FWIW the first iteration writes direct address 0x00,
;; which is the location of r0. We get lucky, we're
;; writing the correct value (0)

00003$: mov @r0,a
dijnz r0,00003$

push dpl ; restore our return address
push dph

mov dpl,#0 ; indicate that data init is still required
ret
```

### B.3.7 usb\_descriptors.a51

```
;;; Filename:
;;; usb_descriptors.a51
;;;
;;; Description:
;;; USB Descriptor table for the fx2 usb camera
;;;
;;; Author:
;;; Neil Scott
;;;
;;; Date:
;;; December 08, 2006
;;; April 17, 2007 -- Added EP1 Interrupt IN endpoint
;;;

.module usb_descriptors

;TODO -- Set VID and PID once obtained, if obtained
VID_FX2CAM = 0xABCD ; Made Up VID
PID_FX2CAM = 0x0201 ; Made Up PID

;DID used to indicate if loaded with firmware
DID_FX2CAM = 0x0001 ; Device ID (bcd)

DSCR_DEVICE = 1 ; Descriptor type: Device
DSCR_CONFIG = 2 ; Descriptor type: Configuration
DSCR_STRING = 3 ; Descriptor type: String
DSCR_INTRFC = 4 ; Descriptor type: Interface
DSCR_ENDPNT = 5 ; Descriptor type: Endpoint
DSCR_DEVQUAL = 6 ; Descriptor type: Device Qualifier

DSCR_DEVICE_LEN = 18
DSCR_CONFIG_LEN = 9
DSCR_INTRFC_LEN = 9
DSCR_ENDPNT_LEN = 7
DSCR_DEVQUAL_LEN = 10

ET_CONTROL = 0 ; Endpoint type: Control
ET_ISO = 1 ; Endpoint type: Isochronous
ET_BULK = 2 ; Endpoint type: Bulk
ET_INT = 3 ; Endpoint type: Interrupt

;; configuration attributes
bmSELF_POWERED = 1 << 6

;;; -----
;;; external ram data
;;; -----
```

```

.area USBDESCSEG      (XDATA)

;;; -----
;;; descriptors used when operating at high speed (480Mb/sec)
;;; -----

.even ; descriptors must be 2-byte aligned for SUDPTR{H,L} to work

;; The .even directive isn't really honored by the linker.  Bummer!
;; (There's no way to specify an alignment requirement for a given area,
;; hence when they're concatenated together, even doesn't work.)
;;
;; We work around this by telling the linker to put USBDESCSEG
;; at 0xE000 absolute.  This means that the maximum length of this
;; segment is 480 bytes, leaving room for the two hash slots
;; at 0xE1E0 to 0xE1FF.
;;
;; As of July 7, 2004, this segment is 326 bytes long

_high_speed_device_descr::
.db DSCR_DEVICE_LEN
.db DSCR_DEVICE
.db <0x0200 ; Specification version (LSB)
.db >0x0200 ; Specification version (MSB)
.db 0xff ; device class (vendor specific)
.db 0xff ; device subclass (vendor specific)
.db 0xff ; device protocol (vendor specific)
.db 64 ; bMaxPacketSize0 for endpoint 0
.db <VID_FX2CAM ; idVendor
.db >VID_FX2CAM ; idVendor
.db <PID_FX2CAM ; idProduct
.db >PID_FX2CAM ; idProduct
.db <DID_FX2CAM ; bcdDevice
.db >DID_FX2CAM ; bcdDevice
.db SI_VENDOR ; iManufacturer (string index)
.db SI_PRODUCT ; iProduct (string index)
.db SI_SERIAL ; iSerial number (string index)
.db 1 ; bNumConfigurations

;;; describes the other speed (12Mb/sec)
.even
_high_speed_devqual_descr::
.db DSCR_DEVQUAL_LEN
.db DSCR_DEVQUAL
.db <0x0200 ; bcdUSB (LSB)
.db >0x0200 ; bcdUSB (MSB)
.db 0xff ; bDeviceClass
.db 0xff ; bDeviceSubClass
.db 0xff ; bDeviceProtocol
.db 64 ; bMaxPacketSize0
.db 1 ; bNumConfigurations (one config at 12Mb/sec)
.db 0 ; bReserved

.even
_high_speed_config_descr::
.db DSCR_CONFIG_LEN
.db DSCR_CONFIG
.db <(_high_speed_config_descr_end - _high_speed_config_descr) ; LSB
.db >(_high_speed_config_descr_end - _high_speed_config_descr) ; MSB
.db 1 ; bNumInterfaces
.db 1 ; bConfigurationValue
.db 0 ; iConfiguration
.db 0x80 ; bmAttributes
.db 100 ; bMaxPower

;; interface descriptor 0

.db DSCR_INTRFC_LEN
.db DSCR_INTRFC
.db 0 ; bInterfaceNumber (zero based)
.db 0 ; bAlternateSetting
.db 1 ; bNumEndpoints
.db 0xff ; bInterfaceClass (vendor specific)
.db 0xff ; bInterfaceSubClass (vendor specific)

```

```

.db 0xff      ; bInterfaceProtocol (vendor specific)
.db 0         ; iInterface (description)

;; endpoint descriptor EP2IN
.db DSCR_ENDPNT_LEN
.db DSCR_ENDPNT
.db 0x82      ; bEndpointAddress (ep 2 IN)
.db ET_BULK   ; bmAttributes
.db <512     ; wMaxPacketSize (LSB)
.db >512     ; wMaxPacketSize (MSB)
.db 0        ; bInterval (iso only)

_high_speed_config_descr_end:

;;; -----
;;; descriptors used when operating at full speed (12Mb/sec)
;;; -----

.even
_full_speed_device_descr::
.db DSCR_DEVICE_LEN
.db DSCR_DEVICE
.db <0x0200   ; Specification version (LSB)
.db >0x0200   ; Specification version (MSB)
.db 0xff     ; device class (vendor specific)
.db 0xff     ; device subclass (vendor specific)
.db 0xff     ; device protocol (vendor specific)
.db 64       ; bMaxPacketSize0 for endpoint 0
.db <VID_FX2CAM ; idVendor
.db >VID_FX2CAM ; idVendor
.db <PID_FX2CAM ; idProduct
.db >PID_FX2CAM ; idProduct
.db <DID_FX2CAM ; bcdDevice
.db >DID_FX2CAM ; bcdDevice
.db SI_VENDOR ; iManufacturer (string index)
.db SI_PRODUCT ; iProduct (string index)
.db SI_NONE   ; iSerial number (None)
.db 1        ; bNumConfigurations

;;; describes the other speed (480Mb/sec)
.even
_full_speed_devqual_descr::
.db DSCR_DEVQUAL_LEN
.db DSCR_DEVQUAL
.db <0x0200   ; bcdUSB
.db >0x0200   ; bcdUSB
.db 0xff     ; bDeviceClass
.db 0xff     ; bDeviceSubClass
.db 0xff     ; bDeviceProtocol
.db 64       ; bMaxPacketSize0
.db 1        ; bNumConfigurations (one config at 480Mb/sec)
.db 0        ; bReserved

.even
_full_speed_config_descr::
.db DSCR_CONFIG_LEN
.db DSCR_CONFIG
.db <(_full_speed_config_descr_end - _full_speed_config_descr) ; LSB
.db >(_full_speed_config_descr_end - _full_speed_config_descr) ; MSB
.db 1        ; bNumInterfaces
.db 1        ; bConfigurationValue
.db 0        ; iConfiguration
.db 0x80     ; bmAttributes
.db 100     ; bMaxPower

;; interface descriptor 0 (command & status, ep0 COMMAND)

.db DSCR_INTRFC_LEN
.db DSCR_INTRFC
.db 0        ; bInterfaceNumber (zero based)
.db 0        ; bAlternateSetting
.db 0        ; bNumEndpoints
.db 0xff     ; bInterfaceClass (vendor specific)

```

---

```

.db 0xff      ; bInterfaceSubClass (vendor specific)
.db 0xff      ; bInterfaceProtocol (vendor specific)
.db 0         ; iInterface (description)

_full_speed_config_descr_end:

;;; -----
;;;      string descriptors
;;; -----

_nstring_descriptors::
.db (_string_descriptors_end - _string_descriptors) / 2

_string_descriptors::
.db <str0, >str0
.db <str1, >str1
.db <str2, >str2
.db <str3, >str3
_string_descriptors_end:

    SI_NONE = 0
    ;; str0 contains the language ID's.
    .even
str0: .db str0_end - str0
     .db DSCR_STRING
     .db 0
     .db 0
     .db <0x0409    ; magic code for US English (LSB)
     .db >0x0409    ; magic code for US English (MSB)
str0_end:

    SI_VENDOR = 1
    .even
str1: .db str1_end - str1
     .db DSCR_STRING
     .db 'U, 0      ; 16-bit unicode
     .db 'n, 0
     .db 'i, 0
     .db 'v, 0
     .db 'e, 0
     .db 'r, 0
     .db 's, 0
     .db 'i, 0
     .db 't, 0
     .db 'y, 0
     .db ' , 0
     .db 'o, 0
     .db 'f, 0
     .db ' , 0
     .db 'W, 0
     .db 'i, 0
     .db 'n, 0
     .db 'd, 0
     .db 's, 0
     .db 'o, 0
     .db 'r, 0
str1_end:

    SI_PRODUCT = 2
    .even
str2: .db str2_end - str2
     .db DSCR_STRING
     .db 'U, 0
     .db 'S, 0
     .db 'B, 0
     .db ' , 0
     .db 'C, 0
     .db 'a, 0
     .db 'm, 0
     .db 'e, 0
     .db 'r, 0
     .db 'a, 0
str2_end:

```

---

```
SI_SERIAL = 3
.even
str3: .db str3_end - str3
.db DSCR_STRING
.db '0, 0
.db '1, 0
.db '2, 0
.db '3, 0
.db '4, 0
.db '5, 0
.db '6, 0
.db '7, 0
str3_end:
```

### B.3.8 vectors.a51

```
.include "../common/vectors.a51"
```

### B.3.9 eeprom\_regs.h

```
/* eeprom_regs.h
 * =====
 * Definition of EEPROM registers
 *
 * Author: Neil Scott
 */

#ifndef EEPROM_REGS_H
#define EEPROM_REGS_H

#define PM_EEPROM_WIDTH_HIGH_ADDR 0x20
#define PM_EEPROM_WIDTH_LOW_ADDR 0x21
#define PM_EEPROM_LENGTH_HIGH_ADDR 0x22
#define PM_EEPROM_LENGTH_LOW_ADDR 0x23
#define PM_EEPROM_COL_START_HIGH_ADDR 0x24
#define PM_EEPROM_COL_START_LOW_ADDR 0x25
#define PM_EEPROM_ROW_START_HIGH_ADDR 0x26
#define PM_EEPROM_ROW_START_LOW_ADDR 0x27
#define PM_EEPROM_COL_SKIP_ADDR 0x28
#define PM_EEPROM_ROW_SKIP_ADDR 0x29

#endif /* EEPROM_REGS_H */
```

### B.3.10 fx2cam\_common.c

```
/* Filename;
 * fx2cam_common.c
 *
 * Description:
 * Initialization for the Cypress FX2 MCU.
 *
 * Author:
 * Neil Scott
 *
 * Date:
 * December 08, 2006
 *
 * August 10, 2007 - Changed IFCONFIG so IFCLK runs at 30MHz instead of 48MHz,
 * This seems to have corrected the data loss problem when
 * the slave FIFO was filling.
 */

#include "fx2cam_common.h"

void
init_fx2cam (void)
{
```

```

/* Set CPU Clock to 48MHz */
CPUCS = bmCLKSPD1 | bmCLKOE; //48MHz
//CPUCS = bmCLKSPD0 | bmCLKOE; //24MHz
SYNCDELAY;
/* Set MOVX instruction to take 2 cycles (default is 3) */

/* Set IFCONFIG Register
 * Set 3048MHZ to 48MHz FIFO Clock
 * Modified - Set 3048MHZ to 30MHz FIFO Clock
 * -- attempt to fix data loss problems
 * Set IFCLKSRC to external clock on IFCLK pin
 * Set ASYNC to 0 for synchronous FIFO operation
 */
IFCONFIG = bmIFCLKSRC | bm3048MHZ | bmIFCLKOE | bmIFCFGMASK; SYNCDELAY;

/* Set Inputs SLOE, FIFOADDR0, FIFOADDR1, PKTEND */

/* Configure I/O Ports for 100pin FX2
 * by default, All set to input
 */

/* Port A Initial State */
IOA = 0x00;
/* Set Port A Direction */
OEA = 0;

/* Port C Initial State */
IOC = 0;
/* Set Port C Direction */
OEC = 0; /* Leave Floating initially */

/* Port E Initial State */
IOE = FPGA_RESET_BIT;
/* Set Port E Direction */
OEE = FPGA_POWER_VCC_AUX | FPGA_POWER_VCC_O | FPGA_POWER_VCC_INT | FPGA_RESET_BIT;

/* Disable Auto Arming of AUTOOUT.
 * Set Enhanced Packet Handling
 *
 * Recommended by Cypress to set both bits high
 * (P15.24 TRM)
 */
REVCTL = bmDYN_OUT | bmENH_PKT; SYNCDELAY;

/* Configure USB Endpoints */

/* Disable EP1 */
EP1OUTCFG = 0; SYNCDELAY;
EP1INCFG = 0; SYNCDELAY;

/* Camera Data Endpoint is Quad-Buff Bulk-In EP2 */
EP2CFG = bmVALID | bmIN | bmBULK | bmQUADBUF;
SYNCDELAY;

/* Disable other EPs */
EP4CFG = 0; SYNCDELAY;
EP6CFG = 0; SYNCDELAY;
EP8CFG = 0; SYNCDELAY;

/* NAK All transfers from host */
FIFORESET = bmNAKALL; SYNCDELAY;

/* Reset EP2 FIFO */
FIFORESET = 0x02; SYNCDELAY;
FIFORESET = 0x00; SYNCDELAY;

/* EP2 AUTOOUT = 0, AUTOIN = 1, ZEROLEN = 1, WORDWIDE = 1 */
//EP2FIFOCFG = 0x0D; SYNCDELAY;

EP2FIFOCFG = bmAUTOIN | bmZEROLENIN | bmWORDWIDE; SYNCDELAY;

/* Set FLAGA to EP2 Full Flag */
FINFLAGSAB = 0x0C; SYNCDELAY;

```

```
/* Auto Commit 512 byte packets */
EP2AUTOINLENH = 0x02;    SYNCDELAY;
EP2AUTOINLENL = 0x00;    SYNCDELAY;

FIFOPINPOLAR = 0x3F;
SYNCDELAY;

/* Set Polarity of EP2 Full Flag */
EP2FIFOPFH = 0x80;
SYNCDELAY;
EP2F1FOPFL = 0x00;
SYNCDELAY;

/* Must reset EP0BCH because power-on-reset state
 * is undefined (P8.8 TRM)
 */
EP0BCH = 0;    SYNCDELAY;

/* Set I2C serial clock to 400KHz (default is 100kHz)*/
I2CTL |= bm400KHZ;
}
```

### B.3.11 fx2cam\_main.c

```
/* Filename:
 *   fx2cam_main.c
 *
 * Description:
 *   USB Firmware for FX2 Camera.
 *
 * Author:
 *   Neil Scott
 *
 * Date:
 *   December 10, 2006
 */

#include "usb_common.h"
#include "fx2cam_common.h"
#include "fx2cam_commands.h"
#include "fx2utils.h"
#include "i2c.h"
#include "isr.h"
#include "delay.h"
#include "fx2cam_i2c_addr.h"
#include "mi_regs.h"
#include "eeprom_regs.h"

/* Camera Position Registers */
volatile unsigned char camera_quadrant;
volatile unsigned char camera_position;
volatile unsigned char camera_master;
volatile unsigned int frame_drop_count;

/* Sensor Window Registers */
volatile unsigned int window_width;
volatile unsigned int window_length;
volatile unsigned int x_TEMP;
volatile unsigned int window_col_start;
volatile unsigned int window_row_start;
volatile unsigned char window_col_skip;
volatile unsigned char window_row_skip;

/* Error Code Register */
volatile unsigned int err_reg;

/* Function Prototypes */
void load_camera_config (void);

void
get_ep0_data (void)
```

```
{
    /* Arm EPO */
    EPOBCL = 0;

    /* Wait until busy flag is clear */
    while (EPOCS & bmEPBUSY);
}

/* Handle Vendor Requests to endpoint 0
 * If handled non zero is returned
 */
unsigned char
app_vendor_cmd (void)
{
    unsigned short    addr;
    unsigned short    len;
    unsigned short    bc;
    unsigned char     xdata ee_str[3];
    unsigned char     i, j, k;
    unsigned long      timeout;

    /* In Requests */
    if (bRequestType == VENDOR_REQUEST_IN) {
        switch (bRequest) {
            /* Read I2C bus, data is returned through EPOBUF */
            case VRQ_I2C_READ:
                /* Wait of SCL pin (must be high) */
                while (!(SCL_LB_PORT & SCL_LB_BIT));

                if (!i2c_read (wValueL, EPOBUF, wLengthL))
                    return 0;

                EPOBCH = 0;
                EPOBCL = wLengthL;
                break;

            case VRQ_SET_I2C_SPEED:
                /* wValueL = 0 - 100KHz
                 * wValueL = 1 - 400KHz
                 */
                /*if (!wValueL)
                    I2CTL &= ~bm400KHZ;
                else
                    I2CTL |= bm400KHZ;
                */
                I2CTL = wValueL;

                EPOBUF[0] = 0x08;          /* ACK */
                EPOBCH = 0;
                EPOBCL = 1;

                break;

            case VRQ_GET_CAM_STATUS:

                break;

            case VRQ_READ_CAM_POSITION:

                break;

            case VRQ_READ_EEPROM_SM:

                break;

            case VRQ_READ_EEPROM_LG:
                addr = wValueL;
                addr |= wValueH << 8;

                len = wLengthL;
                len |= wLengthH << 8;

                while (len) {
```



---

```

    /* One Packet at a time */
    while (EP0CS & bmEPBUSY);

    if (len < EP0BUF_SIZE)
        bc = len;
    else
        bc = EP0BUF_SIZE;

    i = 0;
                                //wValueH;
    ee_str[i++] = addr >> 8;
                                //wValueL;
    ee_str[i++] = addr & 0x00ff;

    /* Write EEPROM address to device */
    if (!i2c_write (I2C_EEPROM_ADDR, ee_str, i))
        return 0;

    /* Read EEPROM data to buffer */
    if (!i2c_read (I2C_EEPROM_ADDR, EP0BUF, bc))
        return 0;

    EP0BCH = 0;
    EP0BCL = bc;

    addr += bc;
    len -= bc;
}

break;

case VRQ_GET_QUAD_POS:
    /* Respond with Camera position / master info */
    EP0BUF[0] = camera_quadrant;
    EP0BUF[1] = camera_position;
    EP0BUF[2] = camera_master;
    EP0BCH = 0;
    EP0BCL = 3;
    break;

case VRQ_GET_WINDOW_PARAM:
    switch (wIndexL) {
        case VRQ_UPDATE_PARAMS:
            /* Re-read from EEPROM */
            load_camera_config();
            EP0BUF[0] = 0x08;
            EP0BCH = 0;
            EP0BCL = 1;
            break;

        case VRQ_GET_WINDOW_WIDTH:
            /* Respond with window width */
            EP0BUF[0] = window_width >> 8;
            EP0BUF[1] = window_width & 0xff;
            EP0BCH = 0;
            EP0BCL = 2;
            break;

        case VRQ_GET_WINDOW_LENGTH:
            /* Respond with window length */
            EP0BUF[0] = window_length >> 8;
            EP0BUF[1] = window_length & 0xff;
            EP0BCH = 0;
            EP0BCL = 2;
            break;

        case VRQ_GET_WINDOW_COL_START:
            /* Respond with window column start */
            EP0BUF[0] = window_col_start >> 8;
            EP0BUF[1] = window_col_start & 0xff;
            EP0BCH = 0;
            EP0BCL = 2;
            break;
    }

```

---

```

case VRQ_GET_WINDOW_ROW_START:
    /* Respond with window row start */
    EP0BUF[0] = window_row_start >> 8;
    EP0BUF[1] = window_row_start & 0xff;
    EP0BCH = 0;
    EP0BCL = 2;
    break;

case VRQ_GET_WINDOW_COL_SKIP:
    /* Respond with window column skip (x - binning) */
    EP0BUF[0] = window_col_skip;
    EP0BCH = 0;
    EP0BCL = 1;
    break;

case VRQ_GET_WINDOW_ROW_SKIP:
    /* Respond with window row skip (y - binning) */
    EP0BUF[0] = window_row_skip;
    EP0BCH = 0;
    EP0BCL = 1;
    break;
}

case VRQ_FPGA_LOAD_SS:
    switch (wIndexL) {
        case FPGA_LOAD_START:
            /* Send the FPGA the Start signal */
            /* Pulse PROG_B (active low) */

            /* Set FPGA LOAD port I/O directions */
            IOC |= FPGA_LOAD_DATA_BIT;
            OEC = FPGA_LOAD_CLK_BIT | FPGA_LOAD_DATA_BIT | FPGA_LOAD_PROG_BIT;

            /* Tiny Delay */
            mdelay (10);

            /* timeout - just in case */
            timeout = 0x05ff;

            /* Pulse PROG_B low */

            /* Wait for INIT_B line to go high, or timeout to expire */
            while ( !(FPGA_LOAD_PORT & FPGA_LOAD_INIT_BIT) && timeout ) {
                FPGA_LOAD_PORT &= ~FPGA_LOAD_PROG_BIT;
                udelay(500);
                timeout--;
            }

            /* Send response to host */
            if (FPGA_LOAD_PORT & FPGA_LOAD_INIT_BIT)
                EP0BUF[0] = 0x01; /* Success */
            else
                EP0BUF[0] = 0x00; /* Failure */

            /* Acknowledge */
            EP0BUF[1] = 0x08;

            EP0BCH = 0;
            EP0BCL = 2;
            break;

        case FPGA_LOAD_CHECK_DONE:
            /* Check the DONE bit */

            /* timeout */
            timeout = 0x2ffff;

            /* Set DIN low and supply CLKS */
            FPGA_LOAD_PORT &= ~FPGA_LOAD_DATA_BIT;

            /* Supply CLK until DONE_B goes high */
            while ( !(FPGA_LOAD_PORT & FPGA_LOAD_DONE_BIT) && timeout ) {
                FPGA_LOAD_PORT |= FPGA_LOAD_CLK_BIT;
                FPGA_LOAD_PORT &= ~FPGA_LOAD_CLK_BIT;
            }
        }
    }

```

---

```

        timeout--;
    }

    /* Send response to host */
    if (FPGA_LOAD_PORT & FPGA_LOAD_DONE_BIT)
        EP0BUF[0] = 0x01;
    else
        EP0BUF[0] = 0x00;

    /* Acknowledge */
    EP0BUF[1] = 0x08;

    EP0BCH = 0;
    EP0BCL = 2;
    break;

default:
    break;
}
break;

case VRQ_FPGA_POWER:
    /* Disable FPGA Power Chip */
    if (!wValueL) {
        FPGA_POWER_PORT &= ~(FPGA_POWER_VCC_AUX
            | FPGA_POWER_VCC_O
            | FPGA_POWER_VCC_INT);

        /* Set FPGA Loader port to all inputs with initial state of DIN high */
        IOC = FPGA_LOAD_DATA_BIT;
        OEC = 0;

        /* Respond with ACK to host */
        EP0BUF[0] = 0x08;
    }
    /* Enable FPGA Power Chip */
    else if (wValueL == 1) {
        FPGA_POWER_PORT |= FPGA_POWER_VCC_AUX
            | FPGA_POWER_VCC_O
            | FPGA_POWER_VCC_INT;

        /* Respond with ACK to host */
        EP0BUF[0] = 0x08;
    }
    else {
        /* Respond with NACK to host */
        EP0BUF[0] = 0x00;
    }

    EP0BCH = 0;
    EP0BCL = 1;
    break;

case VRQ_FPGA_RESET:
    /* Set FPGA Reset Mode: 0-disable, 1-enable */
    if (!wValueL) {
        /* Take FPGA out of reset */
        FPGA_RESET_PORT |= FPGA_RESET_BIT;
        /* Respond with ACK to host */
        EP0BUF[0] = 0x08;
    }
    else if (wValueL == 1) {
        /* Put FPGA in reset */
        FPGA_RESET_PORT &= ~FPGA_RESET_BIT;
        /* Respond with ACK to host */
        EP0BUF[0] = 0x08;
    }
    else {
        /* Respond with NACK to host */
        EP0BUF[0] = 0x00;
    }

    EP0BCH = 0;
    EP0BCL = 1;

```

---

```
        break;

    case VRQ_FRAME_DROP_INTERRUPT:
        /* Enable / Disable Frame Drop Interrupt: 0-disable, 1-enable */
        if (!wValueL) {
            /* Respond with ACK to host */
            EPOBUF[0] = 0x08;
        }
        else if (wValueL == 1) {
            /* Respond with ACK to host */
            EPOBUF[0] = 0x08;
        }
        else {
            /* Respond with NACK to host */
            EPOBUF[0] = 0x00;
        }

        EPOBCH = 0;
        EPOBCL = 1;
        break;

    case VRQ_GET_FRAME_DROP_COUNT:
        /* Fill EP0 buffer with frame drop count */
        EPOBUF[1] = (frame_drop_count >> 8) & 0x00ff;
        EPOBUF[0] = frame_drop_count & 0x00ff;

        EPOBCH = 0;
        EPOBCL = 2;

        break;

    case VRQ_RESET_FRAME_DROP_COUNT:
        /* Reset frame drop counter */
        frame_drop_count = 0;

        /* Respond ACK to host */
        EPOBUF[0] = 0x08;

        EPOBCH = 0;
        EPOBCL = 1;
        break;

    default:
        return 0;
}

/* Out Requests */
else if (bRequestType == VENDOR_REQUEST_OUT) {
    switch (bRequest) {
        case VRQ_I2C_WRITE:
            get_ep0_data ();

            /* Wait of SCL pin (must be high) */
            while (!(SCL_LB_PORT & SCL_LB_BIT));

            if (!i2c_write(wValueL, EPOBUF, EPOBCL))
                return 0;
            break;

        case VRQ_PILL_REJECT_ACCEPT:
            get_ep0_data ();
            break;

        /* Write to large size microchip EEPROM (address more than 8 bits) */
        case VRQ_WRITE_EEPROM_LG:
            addr = wValueL;
            addr |= wValueH << 8;

            len = wLengthL;
            len |= wLengthH << 8;

            while (len) {
                //get_ep0_data ();
            }
        }
    }
}
```

---

---

```

EP0BCH = 0;
EP0BCL = 0;
while (EP0CS & bmEPBUSY);

bc = EP0BCL;

for (i = 0; i < bc; i++) {
    ee_str[0] = addr >> 8;
    ee_str[1] = addr & 0x00ff;
    ee_str[2] = EP0BUF[i];
    if (!i2c_write(I2C_EEPROM_ADDR, ee_str, 3))
        return 0;
    mdelay(5);

    addr++;
}

len -= bc;
}

break;

/* TODO: Is this necessary? */
case VRQ_FPGA_FLUSH:
    if (!wValueL) {
        /* Skip Comitting Out Packets */
        OUTPKTEND = 0x82; SYNCDELAY;
        OUTPKTEND = 0x82; SYNCDELAY;

        EP2FIFOBCH = 0x00; SYNCDELAY;
        EP2FIFOBCL = 0x00; SYNCDELAY;
        FIFORESET = 0x80; SYNCDELAY;
        FIFORESET = 0x02; SYNCDELAY;
        FIFORESET = 0x00; SYNCDELAY;
        OUTPKTEND = 0x82; SYNCDELAY;
        OUTPKTEND = 0x82; SYNCDELAY;
    }
    else {
        /* Skip Comitting Out Packets */
        OUTPKTEND = 0x82; SYNCDELAY;
        OUTPKTEND = 0x82; SYNCDELAY;

        EP2FIFOBCH = 0x00; SYNCDELAY;
        EP2FIFOBCL = 0x00; SYNCDELAY;
        FIFORESET = 0x80; SYNCDELAY;
        FIFORESET = 0x02; SYNCDELAY;
        FIFORESET = 0x00; SYNCDELAY;
        OUTPKTEND = 0x82; SYNCDELAY;
        OUTPKTEND = 0x82; SYNCDELAY;
        OUTPKTEND = 0x82; SYNCDELAY;
    }
    break;

/* FPGA load routine using slave-serial mode */
case VRQ_FPGA_LOAD_SS:
    switch (wIndexL) {
        /* Write data */
        case FPGA_LOAD_DATA:
            get_ep0_data();

            /* Get Byte Count */
            bc = EP0BCL;

            /* Bit-Bang data to FPGA */
            for (j = 0; j < bc; j++) {
                k = EP0BUF[j];

                /* Loop through each byte */
                /* MSB first */
                for (i = 0; i < 8; i++) {
                    /* Set Data bit */
                    if ( (k & 0x80) )
                        FPGA_LOAD_PORT |= FPGA_LOAD_DATA_BIT;

```

---

```

        else
            FPGA_LOAD_PORT &= ~FPGA_LOAD_DATA_BIT;

            /* Shift data one left */
            k = k << 1;

            /* Pulse Clock */
            FPGA_LOAD_PORT |= FPGA_LOAD_CLK_BIT;
            FPGA_LOAD_PORT &= ~FPGA_LOAD_CLK_BIT;
        }
    }

    /* Check INIT_B - goes LOW on error */
    if ( !(FPGA_LOAD_PORT & FPGA_LOAD_INIT_BIT) ) {
        /* Set Error Flag */
        err_reg |= ERR_FPGA_LOAD;
        return 0;          /* Will cause broken pipe error */
    }
    break;

default:
    return 0;
}

break;

default:
    return 0;
}
}
else {
    /* Invalid Request Type */
    return 0;
}

return 1;
}

/* Read from EEPROM camera configuration (position / calibrated sensor values) */
/* TODO: Right now only reading Camera Quadrant and Position
 *      Assuming this is stored at 0x10
 */
void
load_camera_config (void)
{
    unsigned char i;
    unsigned char xdata ee_str[3];
    unsigned char xdata tmp;

    i = 0;
    ee_str[i++] = 0;
    ee_str[i++] = 0x10;          //wValueL;

    /* Write EEPROM address to device */
    if (i2c_write (I2C_EEPROM_ADDR, ee_str, i)) {
        /* Read EEPROM data to buffer */
        i2c_read (I2C_EEPROM_ADDR, EP0BUF, 3);
        camera_quadrant = EP0BUF[0];
        camera_position = EP0BUF[1];
        camera_master = EP0BUF[2];
    }

    /* Write EEPROM Address to device */
    i = 0;
    ee_str[i++] = 0;
    ee_str[i++] = PM_EEPROM_WIDTH_HIGH_ADDR;
    if (i2c_write (I2C_EEPROM_ADDR, ee_str, i)) {
        /* Read EEPROM byte */
        i2c_read (I2C_EEPROM_ADDR, &tmp, 1);
        window_width = (tmp << 8);
    }

    /* Write EEPROM Address to device */

```

```

i = 0;
ee_str[i++] = 0;
ee_str[i++] = PM_EEPROM_WIDTH_LOW_ADDR;
if (i2c_write (I2C_EEPROM_ADDR, ee_str, i)) {
    /* Read EEPROM byte */
    i2c_read (I2C_EEPROM_ADDR, &tmp, 1);
    window_width |= tmp;
}

/* Write EEPROM Address to device */
i = 0;
ee_str[i++] = 0;
ee_str[i++] = PM_EEPROM_LENGTH_HIGH_ADDR;
if (i2c_write (I2C_EEPROM_ADDR, ee_str, i)) {
    /* Read EEPROM byte */
    i2c_read (I2C_EEPROM_ADDR, &tmp, 1);
    window_length = (tmp << 8);
}

/* Write EEPROM Address to device */
i = 0;
ee_str[i++] = 0;
ee_str[i++] = PM_EEPROM_LENGTH_LOW_ADDR;
if (i2c_write (I2C_EEPROM_ADDR, ee_str, i)) {
    /* Read EEPROM byte */
    i2c_read (I2C_EEPROM_ADDR, &tmp, 1);
    window_length |= tmp;
}

/* Write EEPROM Address to device */
i = 0;
ee_str[i++] = 0;
ee_str[i++] = PM_EEPROM_COL_START_HIGH_ADDR;
if (i2c_write (I2C_EEPROM_ADDR, ee_str, i)) {
    /* Read EEPROM byte */
    i2c_read (I2C_EEPROM_ADDR, &tmp, 1);
    window_col_start = (tmp << 8);
}

/* Write EEPROM Address to device */
i = 0;
ee_str[i++] = 0;
ee_str[i++] = PM_EEPROM_COL_START_LOW_ADDR;
if (i2c_write (I2C_EEPROM_ADDR, ee_str, i)) {
    /* Read EEPROM byte */
    i2c_read (I2C_EEPROM_ADDR, &tmp, 1);
    window_col_start |= tmp;
}

/* Write EEPROM Address to device */
i = 0;
ee_str[i++] = 0;
ee_str[i++] = PM_EEPROM_ROW_START_HIGH_ADDR;
if (i2c_write (I2C_EEPROM_ADDR, ee_str, i)) {
    /* Read EEPROM byte */
    i2c_read (I2C_EEPROM_ADDR, &tmp, 1);
    window_row_start = (tmp << 8);
}

/* Write EEPROM Address to device */
i = 0;
ee_str[i++] = 0;
ee_str[i++] = PM_EEPROM_ROW_START_LOW_ADDR;
if (i2c_write (I2C_EEPROM_ADDR, ee_str, i)) {
    /* Read EEPROM byte */
    i2c_read (I2C_EEPROM_ADDR, &tmp, 1);
    window_row_start |= tmp;
}

/* Write EEPROM Address to device */
i = 0;
ee_str[i++] = 0;
ee_str[i++] = PM_EEPROM_COL_SKIP_ADDR;
if (i2c_write (I2C_EEPROM_ADDR, ee_str, i)) {

```

```
/* Read EEPROM byte */
i2c_read (I2C_EEPROM_ADDR, &tmp, 1);
window_col_skip = tmp;
}

/* Write EEPROM Address to device */
i = 0;
ee_str[i++] = 0;
ee_str[i++] = PM_EEPROM_ROW_SKIP_ADDR;
if (i2c_write (I2C_EEPROM_ADDR, ee_str, i)) {
    /* Read EEPROM byte */
    i2c_read (I2C_EEPROM_ADDR, &tmp, 1);
    window_row_skip = tmp;
}
}

/* Main Program Loop for handling USB requests */
void
main_loop (void)
{
    /* Task Dispatcher */
    while(1) {
        /* Currently Set to Auto Commit BULK IN EP2 packets */
        /* Check for Incoming Setup Packet */
        if (usb_setup_packet_avail ())
            usb_handle_setup_packet ();
    }
}

void
main (void)
{
    /* Initialization Routine for fx2cam */
    /* Initialize FX2 registers */
    init_fx2cam();

    /* Configure PA0 as external interrupt INT0# */
    /* TODO: DON'T FORGET TO UPDATE INTERRUPT VECTORS */
    // PORTACFG = bmINT0;

    /* Configure External INT0 on falling edge - disabled on startup */
    // EX0 = 0; /* Disable INT0 */
    // IE0 = 0; /* INT0 Edge-Sensitive */
    // IT0 = 1; /* INT0 detected on falling edge */

    /* Disable all interrupts */
    EA = 0;

    setup_autovectors ();
    usb_install_handlers ();

    /* Enable all interrupts */
    EA = 1;

    /* Simulate a reconnect */
    fx2_renumerate ();

    /* Ensure FPGA Power chip (TI) is disabled */
    FPGA_POWER_PORT &= ~FPGA_POWER_VCC_AUX;
    FPGA_POWER_PORT &= ~FPGA_POWER_VCC_O;
    FPGA_POWER_PORT &= ~FPGA_POWER_VCC_INT;

    /* Set FPGA Loader port to all inputs with initial state of DIN high */
    IOC = FPGA_LOAD_DATA_BIT;
    OEC = 0;

    /* Ensure FPGA is held in reset (although off) by default */
    FPGA_RESET_PORT &= ~FPGA_RESET_BIT;

    /* On Startup -- Read Camera Quadrant and Position */
    camera_quadrant = 0xaa;
    camera_position = 0xaa;
}
```



```
camera_master = 0xff;

load_camera_config ();

/* TODO:
 * Set preliminary register values for MI Sensor*/

/* Initialize frame drop counter */
frame_drop_count = 0;

/* Go to main program loop */
main_loop ();
}
```

---

## Appendix C

### *USB2.0 Camera Linux Driver*

---

#### C.1 IMGUSB Fast USB Class

##### C.1.1 imgusb.h

```
/* Filename:
 *   imgusb.h
 *
 * Description:
 *   Header file for imgusb class for fast USB bulk transfer.
 *
 * Author:
 *   Neil Scott, Roberto Muscedere
 *
 * Date:
 *   November 15th, 2007
 */

#include <linux/usbdevice_fs.h>

#ifndef __IMGUSB_H__
#define __IMGUSB_H__

class imgusb
{
private:
    struct usb_dev_handle    *d_udh;
    usbdevfs_urb             **d_urbs;

    int                      d_ep;
    int                      d_block_size;
    int                      d_n_blocks;
    int                      d_image_size;

protected:

public:
    imgusb (struct usb_dev_handle *dev_hdl, int ep, int block_size);
    ~imgusb ();
};
```

```

    bool allocate_urbs(int image_size);
    int get_image_size(void) {return d_image_size;};
    bool get_image (char *buf);
};
#endif /* _IMGUSB_H_ */

```

## C.1.2 imgusb.cc

```

/*
 * Filename:
 *   imgusb.cc
 *
 * Description:
 *   Fast USB class adapted from the SSRP project.
 *
 * Author:
 *   Neil Scott, Roberto Muscedere
 *
 * Date:
 *   November 15th, 2007
 *
 * Notes:
 *   Adapted from the SSRP Project, Simple Software Radio Project
 *
 * Reference:
 *   http://oscar.dcarr.org/ssrp/
 */

#include <stdio.h>
#include <stdlib.h>
#include <usb.h>           /* LibUSB support */
#include <stdexcept>
#include <errno.h>
#include <linux/usbdevice_fs.h>
#include <linux/compiler.h>
#include <sys/ioctl.h>
#include <assert.h>

#include "imgusb.h"

static const int MAX_BLOCK_SIZE = 16 * 1024;          // hard limit
static const int DEFAULT_BLOCK_SIZE = MAX_BLOCK_SIZE;
static const int DEFAULT_BUFFER_SIZE = 16 * (1L << 20); // 16 MB / endpoint

// Totally evil and fragile extraction of file descriptor from
// guts of libusb. They don't install usbi.h, which is what we'd need
// to do this nicely.
//
// FIXME if everything breaks someday in the future, look here...
static int
fd_from_usb_dev_handle (usb_dev_handle *udh)
{
    return *((int *) udh);
}

imgusb::imgusb (struct usb_dev_handle *dev_hdl, int ep, int block_size)
{
    d_udh = dev_hdl;
    d_ep = ep;
    d_block_size = block_size;

    /* Must Ensure Block Size are legitimate */
    if (d_block_size < 0 || d_block_size > MAX_BLOCK_SIZE)
        throw std::out_of_range ("imgusb: block_size");
}

imgusb::~imgusb ()
{
}

```

---

```

// TODO: Make sure any outstanding urbs are removed (generally handled by reap)

/* Allocate URBs */
for (int i=0; i < d_n_blocks; i++) {
    delete d_urbs[i];
}

delete d_urbs;
}

bool
imgusb::allocate_urbs(int image_size)
{
    d_image_size = image_size;

    d_n_blocks = d_image_size / d_block_size;

    if (d_n_blocks * d_block_size != d_image_size )
        throw std::out_of_range ("imgusb: image_size must be a multiple of block_size");

    /* Allocate URBs */
    d_urbs = new usbdevfs_urb*[d_n_blocks];

    for (int i=0; i < d_n_blocks; i++) {
        d_urbs[i] = new usbdevfs_urb;

        memset(d_urbs[i], 0, sizeof (struct usbdevfs_urb) );

        d_urbs[i]->type = USBDEVFS_URB_TYPE_BULK;
        /* for IN endpoint */
        d_urbs[i]->endpoint = (d_ep & 0x7f) | 0x80;
        d_urbs[i]->signr = 0;
    }

    return true;
}

bool
imgusb::get_image (char *buf)
{
    int          ret;
    usbdevfs_urb *urb;      // = 0;
    int          fd;

    fd = fd_from_usb_dev_handle (d_udh);

    for (int i=0; i < d_n_blocks; i++) {
        d_urbs[i]->buffer_length = d_block_size;
        d_urbs[i]->actual_length = 0;

        /* TODO: Some redundancy */
        d_urbs[i]->type = USBDEVFS_URB_TYPE_BULK;
        d_urbs[i]->endpoint = (d_ep & 0x7f) | 0x80;
        d_urbs[i]->flags = 0;
        d_urbs[i]->buffer = ((char *) (buf + (i * d_block_size)));
        d_urbs[i]->buffer_length = d_block_size;
        d_urbs[i]->signr = 0;
        d_urbs[i]->actual_length = 0;
        d_urbs[i]->number_of_packets = 0;
        //;NULL;
        d_urbs[i]->usercontext = (void *) i;
    }

    for (int i=0; i < d_n_blocks; i++) {
        /* submit urbs */
        ret = ioctl (fd, USBDEVFS_SUBMITURB, d_urbs[i]);

        if (ret < 0)
            fprintf (stdout, "imgusb: Error on SUBMITURB - %s\n", strerror(errno) );

        if (ret < 0) {
            for (i;i>=0;i--) {

```

---

```
        ret = ioctl(fd, USBDEVFS_DISCARDURB, d_urbs[i]);
    }
    return false;
}

urb = NULL;

while ((ret = ioctl (fd, USBDEVFS_REAPURB, &urb)) == 0) {
    if (urb->status != 0 && urb->status != -ENOENT) {
        fprintf (stderr, "imgusb{fd=%d}: REAPURB: urb->status = %d, actual_length = %5d\n",
            fd, urb->status, urb->actual_length);

        /* discard urb - unlink */
        ret = ioctl (fd, USBDEVFS_DISCARDURB, &urb);
        if (ret < 0)
            fprintf (stderr, "error discarding URB: %s\n", strerror(errno));

        /* must also reap unlinked urb */
        ioctl(fd, USBDEVFS_REAPURB, &urb);
    }

    if ((int)(urb->usercontext)==d_n_blocks-1) break;
}

if (ret) return false;

return true;
}
```

## C.2 PM\_CAM USB Primitives

### C.2.1 pm\_prims.h

```
/*
 * Filename:
 *   pm_prims.cc
 *
 * Description:
 *   Header file for pm_prims.cc - contains USB functions to find the device,
 *   verify the device and handle control requests.
 *
 * Author:
 *   Neil Scott
 *
 * Date:
 *   January 17, 2007
 *
 * Notes:
 *   Adapted from the SSRP Project, Simple Software Radio Project
 *
 * Reference:
 *   http://oscar.dcarr.org/ssrp/
 */

#ifndef _PM_PRIMS_H
#define _PM_PRIMS_H

/* Initalization for libusb */
void pm_init_usb (void);

/* Find PM Cameras on the bus and return the count */
int pm_get_device_count (void);

/* Find PM Camera on bus */
struct usb_device * pm_find_camera (int n_th);

/* Returns true if device is loaded with firmware, false if not or if DID is unknown */
bool pm_camera_configured (struct usb_device *d);
```

```

/* Returns true if device is NOT loaded with firmware, false if it is or if DID is unknown */
bool pm_camera_unconfigured (struct usb_device *d);

/* Claims the interface and sets the alt interface */
struct usb_dev_handle * pm_open_interface (struct usb_device *d, int if_num, int alt_if_num);

/* Closes the device interface - returns true if success */
bool pm_close (struct usb_dev_handle *udh);

#endif /* _PM_PRIMS_H */

```

## C.2.2 pm\_prims.cc

```

/*
 * Filename:
 *   pm_prims.cc
 *
 * Description:
 *   Header file for pm_prims.cc - contains USB functions to find the device,
 *   verify the device and handle control requests.
 *
 * Author:
 *   Neil Scott
 *
 * Date:
 *   January 17, 2007
 *
 * Notes:
 *   Adapted from the SSRP Project, Simple Software Radio Project
 *
 * Reference:
 *   http://oscar.dcarr.org/ssrp/
 */

#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <usb.h>
#include "pm_prims.h"
#include "pm_ids.h"

/*****
 * Function:      pm_init_usb
 * Description:   Perform initialization for libusb to initialize the USB bus.
 * Parameters:    none
 * Returns:       void
 *****/
void
pm_init_usb (void)
{
    static bool first = true;

    if (first) {
        first = false;
        /* Initialize libusb */
        usb_init ();
        usb_find_busses ();
        usb_find_devices ();
    }
}

/*****
 * Function:      pm_get_device_count
 * Description:   Search the USB bus for the corresponding product id and vendor id
 *               and count the number of instance found.
 * Parameters:    none
 * Returns:       number of matching devices found
 *
 *****/
int
pm_get_device_count (void)

```

```

{
    struct usb_bus      *b;
    struct usb_device    *d;
    int                  dev_count = 0;

    for (b = usb_busses; b != NULL; b = b->next) {
        for (d = b->devices; d != NULL; d = d->next) {
            /* Check VID and PID */
            if (d->descriptor.idVendor == USB_PM_VID_CAM &&
                d->descriptor.idProduct == USB_PM_PID_CAM) {
                dev_count++;
            }
        }
    }

    return dev_count;
}

/*****
 * Function:      pm_find_camera
 * Description:   Searches the USB bus for the product id and vendor id that
 *               matches the camera.
 * Parameters:    n_th - device instance on the bus
 * Returns:       pointer to usb device
 *****/
struct usb_device *
pm_find_camera (int n_th)
{
    struct usb_bus      *b;
    struct usb_device    *d;
    int                  dev_count = 0;

    for (b = usb_busses; b != NULL; b = b->next) {
        for (d = b->devices; d != NULL; d = d->next) {
            /* Check VID and PID */
            if (d->descriptor.idVendor == USB_PM_VID_CAM &&
                d->descriptor.idProduct == USB_PM_PID_CAM) {
                if (n_th == dev_count++)
                    return d;
            }
        }
    }

    /* if not found */
    return 0;
}

/*****
 * Function:      pm_camera_configured
 * Description:   Reads the Device ID to determine if the pm_camera has been loaded
 *               with firmware. Returns true if configured.
 * Parameters:    *d, pointer to a USB device
 * Returns:       true, if configured
 *               false, if unconfigured or unknown DID
 *****/
bool
pm_camera_configured (struct usb_device *d)
{
    return (d->descriptor.bcdDevice == USB_PM_DID_CAM_CONFIGURED);
}

/*****
 * Function:      pm_camera_unconfigured
 * Description:   Reads the Device ID to determine if the pm_camera has NOT been
 *               loaded with firmware. Returns true if unconfigured.
 * Parameters:    *d, pointer to a USB device
 * Returns:       true, if unconfigured
 *               false, if configured or unknown DID
 *****/
bool
pm_camera_unconfigured (struct usb_device *d)

```

```

{
    return (d->descriptor.bcdDevice == USB_PM_DID_CAM_UNCONFIGURED);
}

/*****
 * Function:      pm_open_interface
 * Description:   Claims the interface defined by if_num and sets te alternative
 *               interface. Returns a pointer to a device handle.
 * Parameters:    *d, pointer to a USB device
 *               if_num, interface number to claim
 *               alt_if_num, alternative interface to select
 * Returns:       pointer to usb device handle
 *****/
struct usb_dev_handle *
pm_open_interface (struct usb_device *d, int if_num, int alt_if_num)
{
    struct usb_dev_handle      *udh = usb_open (d);

    if (d == 0) {
        fprintf (stderr, "pm_open_interface: Error on usb_open - %s\n", usb_strerror ());
        return 0;
    }

    /* Claim device interface */
    if (usb_claim_interface (udh, if_num) < 0) {
        fprintf (stderr, "pm_open_interface: Error on usb_claim_device - %s\n", usb_strerror ());
        return 0;
    }

    /* Set Alt Interface */
    if (usb_set_altinterface (udh, alt_if_num) < 0) {
        fprintf (stderr, "pm_open_interface: Error on usb_set_altinterface - %s\n", usb_strerror ());
        return 0;
    }

    return udh;
}

/*****
 * Function:      pm_close
 * Description:   Closes the USB interface with a given device handle
 * Parameters:    *udh, pointer to a USB device handle
 * Returns:       true, on success
 *               false, on failure
 *****/
bool
pm_close (struct usb_dev_handle *udh)
{
    if (usb_close (udh) < 0)
        return false;

    return true;
}

```

## C.3 PM\_CAM USB2.0 Camera Driver Class

### C.3.1 pm\_cam.h

```

/*
 * Filename:
 *   pm_cam.h
 *
 * Description:
 *   Header file for camera class.
 *
 * Author:
 *   Neil Scott
 */

```



```

*   Date:
*   January 25, 2007
*
*/

#ifndef _PM_CAM_H
#define _PM_CAM_H

#include <stdio.h>
#include <stdlib.h>

// #include <fusb.h>
#include "imgusb.h"

#define PM_CAM_BULK_EPIN_ADDR      0x82
#define PM_CAM_BLOCK_SIZE          8*1024
                                   /* milli-seconds */
#define PM_CAM_USB_TIMEOUT         450

#define PM_IMAGE_FORMAT_TIFF_COL    0
#define PM_IMAGE_FORMAT_TIFF_BW     1

#define PM_TIFF_RED_OFFSET          0
#define PM_TIFF_GREEN_OFFSET        1
#define PM_TIFF_BLUE_OFFSET         2

/* Camera Positions */
#define PM_CAM_CENTER               1
#define PM_CAM_LEFT                 2
#define PM_CAM_RIGHT                3
#define PM_CAM_BOTTOM               4

/* Output Messaging */
#define OUT_MSG                      stdout
#define OUT_ERR_MSG                  stderr

class pm_cam
{
private:
    int      window_width, window_height;
    int      image_width, image_height;
    int      eeprom_window_width, eeprom_window_height;
    int      eeprom_window_col_start, eeprom_window_row_start;
    int      eeprom_window_col_skip, eeprom_window_row_skip;
    int      col_skip, row_skip;
    int      cam_position;
    int      cam_quadrant;
    bool     cam_master;

protected:
    /* LibUSB device device handle pointer */
    struct usb_dev_handle *d_udh;

    /* FastUSB device handle pointer and endpoint handle pointer */
    /* Endpoint handle is for EP2IN, for bulk IN transfers */
    // fusb_ephandle *d_feph;

    /* ImgUSB pointer for fast USB transfer */
    imgusb *d_imgusb;

public:
    /* Constructor -- take a libusb device handle pointer */
    pm_cam (struct usb_dev_handle *udh, int block_size, int n_blocks);

    /* Destructor */
    ~pm_cam ();

    /* For verbose messaging */
    bool verbose_p;

    /* Internal Functions */
    int pm_cam_rx (unsigned char *buf, long buf_size);

    /* imgUSB URB allocation */

```

```

bool imgusb_allocate_urbs() {d_imgusb->allocate_urbs(image_width*image_height);}

/* Grab Frame */
long grab_frame (unsigned char *buf);

/* libUSB function abstraction */
int write_cmd (int requesttype, int request, int value, int index, char *data, int len);
int bulk_read (int ep, char *data, int size);
int bulk_read (int ep, char *data, int size, int timeout);

/* Get camera position, quadrant and master flag */
int get_cam_location (void);
int read_window_params (void);
int get_cam_position (void) {return cam_position;}
int get_cam_quadrant (void) {return cam_quadrant;}
bool get_cam_master (void) {return cam_master;}
int get_eeeprom_window_width (void) {return eeeprom_window_width;}
int get_eeeprom_window_height (void) {return eeeprom_window_height;}
int get_eeeprom_window_col_start (void) {return eeeprom_window_col_start;}
int get_eeeprom_window_row_start (void) {return eeeprom_window_row_start;}
int get_eeeprom_window_col_skip (void) {return eeeprom_window_col_skip;}
int get_eeeprom_window_row_skip (void) {return eeeprom_window_row_skip;}
double get_cam_temp(void);

/* Device Controls */
bool cam_fpga_reset (int state);
bool cam_fpga_power (int state);

/* FPGA Register Write */
bool fpga_write_reg (unsigned char reg, short value);

/* MI Sensor Register Read or Write */
int read_reg (unsigned char reg, short *dat);
int write_reg (unsigned char reg, short value);

/* Image Conversion */
int bayer2rgb (unsigned char *bayer, unsigned long **rgb, int width, int height);
int bayer2gray (unsigned char *bayer, unsigned char *buf, int width, int height);
int bayer2tiff (unsigned char *buf_in, char *filename, int width, int height);
int write_tiff (unsigned char *buf_in, char *filename, int width, int height);
int inspect (unsigned long **rgb);

/* Camera Registry Settings File */
bool import_reg_data_file (const char *filename);
bool write_reg_data_file (const char *filename);

int get_window_width (void) {return window_width;}
int get_window_height (void) {return window_height;}
int get_image_width (void) {return image_width;}
int get_image_height (void) {return image_height;}
void set_window_width (int _width);
void set_window_height (int _height);
void set_window_width_skip (int _width, int _skip);
void set_window_height_skip (int _height, int _skip);
void set_window_col_start (int _col_start);
void set_window_row_start (int _row_start);
void set_binning (int _width, int _height, int row_skip, int col_skip);

/* Misc Image Processing Algorithms */
unsigned char **convert_grayscale (unsigned long **rgb);
};
#endif /* _PM_CAM_H */

```

### C.3.2 pm\_cam.cc

```

/*
 * Filename:
 *   pm_cam.cc
 *
 * Description:
 *   PM Camera USB2.0 Driver Class.
 */

```

```

* Author:
*   Neil Scott
*
* Date:
*   January 25, 2007
*
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <list>
#include <linux/usbdevice_fs.h>
#include <unistd.h>
#include <tiffio.h>
#include <time.h>
#include <stdexcept>
#include <usb.h>                /* LibUSB header */
#include "imgusb.h"

#include "pm_cam.h"
#include "fx2cam_ids.h"
#include "fx2cam_commands.h"    /* USB requests, Shared with Firmware */
#include "fx2cam_i2c_addr.h"    /* I2C Bus addresses, Shared with Firmware */
#include "mi_regs.h"            /* Registers of Micron Sensor */
#include "eeprom_regs.h"

#define DEBUG                    1

#define RED_OFFSET               0
#define GREEN_OFFSET            1
#define BLUE_OFFSET             2

#define PM_CAM_INTERRUPT_EP 0x81

/* Constructor - Create an instance of the device.  Create a FastUSB
 * device handle and FastUSB endpoint handle.
 */
pm_cam::pm_cam (struct usb_dev_handle *udh, int block_size, int n_blocks)
{
    /* Set internal device handle pointer */
    d_udh = udh;

    /* Create FUSB endpoint handle */
    d_imgusb = new imgusb (d_udh, PM_CAM_BULK_EPIN_ADDR, block_size);

    /* Set defaults for camera location info */
    cam_position = 0xef;
    cam_quadrant = 0xef;
    cam_master = false;

    /* Initially low-verbosity level */
    verbose_p = false;
}

/* Destructor - Free up memory. Delete FastUSB device handle and
 * endpoint handle
 */
pm_cam::~pm_cam ()
{
    /* Delete IMG_USB objects */
    delete d_imgusb;
}

/* using PM_USB */
long
pm_cam::grab_frame (unsigned char *buf)
{
    int ret;
    long bc = 0;

    if (!d_imgusb->get_image( (char *) buf)) {

```

```

        fprintf (stderr, "Error on imgusb->get_image()\n");
        return -1;
    }

    bc = d_imgusb->get_image_size();
    return bc;
}

int pm_cam::write_cmd (int requesttype, int request, int value, int index, char *data, int len)
{
    int ret;
    ret = usb_control_msg (    d_udh,
        requesttype,
        request,
        value,
        index,
        data,
        len,
        PM_CAM_USB_TIMEOUT);

    if (ret < 0)
        fprintf (stderr, "pm_cam::write_cmd - Error: %s\n", strerror(ret) );

    return (ret);
}

int
pm_cam::read_reg (unsigned char reg, short *dat)
{
    int ret;
    char data[2];

    data [0] = reg;

    /* Write Register Address to MI Sensor */
    ret = write_cmd (VENDOR_REQUEST_OUT, VRQ_I2C_WRITE, MI_I2C_ADDR, 0, data, 1);

    if (ret < 0)
        return ret;

    /* Read Data from MI Sensor */
    ret = write_cmd (VENDOR_REQUEST_IN, VRQ_I2C_READ, MI_I2C_ADDR, 0, data, 2);

    dat[0] = (0x00FF & data[1]) + (0xFF00 & (data[0] << 8));

    if (verbose_p)
        fprintf (OUT_MSG, "Register 0x%04x read with a value of 0x%04x\n", reg, dat[0]);

    return ret;
}

int
pm_cam::write_reg (unsigned char reg, short value)
{
    int ret;
    char data[3];
    short verify = 0;

    data [0] = reg;
    data [1] = (( 0xFF00 & value) >> 8);
    data [2] = (0x00FF & value);

    /* Write Register Address to MI Sensor */
    ret = write_cmd (VENDOR_REQUEST_OUT, VRQ_I2C_WRITE, MI_I2C_ADDR, 0, data, 3);

    if (ret < 0)
        fprintf (stderr, "pm_cam::write_reg: Error writing MI Register - %s\n", usb_strerror ());
    else
        if (verbose_p) fprintf (OUT_MSG, "Register 0x%04x written with 0x%04x\n", reg, value);

    /* Verify Correct value was written */

```

```
    ret = read_reg (reg, &verify);

    if (verify != value)
        fprintf (stderr, "Unexpected Register Value Read Back: Register %04x\n", reg);

    return ret;
}

int
pm_cam::bulk_read (int ep, char *data, int size)
{
    return (usb_bulk_read (d_udh, ep, data, size, PM_CAM_USB_TIMEOUT));
}

int
pm_cam::bulk_read (int ep, char *data, int size, int timeout)
{
    return (usb_bulk_read (d_udh, ep, data, size, timeout));
}

int
pm_cam::get_cam_location (void)
{
    unsigned char data[3];
    int ret;
    /* Control Transfer request for position, quadrant and master flag */
    ret = write_cmd (VENDOR_REQUEST_IN,
                    VRQ_GET_QUAD_POS,
                    0,
                    0,
                    (char *) data,
                    3);

    if (ret < 0)
        return ret;

    cam_quadrant = data[0];
    cam_position = data[1];
    cam_master = ((data[2]) ? true : false);

    return ret;
}

/* Read in window parameters from EEPROM */
int
pm_cam::read_window_params (void)
{
    unsigned char data[2];
    int ret;

    /* Control transfer request for window width */
    ret = write_cmd (VENDOR_REQUEST_IN,
                    VRQ_GET_WINDOW_PARAM,
                    0,
                    VRQ_GET_WINDOW_WIDTH,
                    (char *) data,
                    2);

    if (ret < 0)
        return ret;

    eeprom_window_width = (data[0] << 8) | data[1];

    /* Control transfer request for window height */
    ret = write_cmd (VENDOR_REQUEST_IN,
                    VRQ_GET_WINDOW_PARAM,
                    0,
                    VRQ_GET_WINDOW_LENGTH,
                    (char *) data,
                    2);
}
```

```
if (ret < 0)
    return ret;

eeprom_window_height = (data[0] << 8) | data[1];

/* TODO: Problem with COL_START when reading from FX2
 * Unknown problem, hack to read directly from EEPROM in
 * the meantime
 */

ret = write_cmd ( VENDOR_REQUEST_IN,
                  VRQ_READ_EEPROM_LG,
                  PM_EEPROM_COL_START_HIGH_ADDR,
                  0,
                  (char *) data,
                  1);

if (ret < 0)
    return ret;

eeprom_window_col_start = (data[0] << 8);

ret = write_cmd ( VENDOR_REQUEST_IN,
                  VRQ_READ_EEPROM_LG,
                  PM_EEPROM_COL_START_LOW_ADDR,
                  0,
                  (char *) data,
                  1);

if (ret < 0)
    return ret;

eeprom_window_col_start |= data[0];

/* Control transfer request for window row start */
ret = write_cmd ( VENDOR_REQUEST_IN,
                  VRQ_GET_WINDOW_PARAM,
                  0,
                  VRQ_GET_WINDOW_ROW_START,
                  (char *) data,
                  2);

if (ret < 0)
    return ret;

eeprom_window_row_start = (data[0] << 8) | data[1];

/* Control transfer request for window col skip */
ret = write_cmd ( VENDOR_REQUEST_IN,
                  VRQ_GET_WINDOW_PARAM,
                  0,
                  VRQ_GET_WINDOW_COL_SKIP,
                  (char *) data,
                  1);

if (ret < 0)
    return ret;

eeprom_window_col_skip = data[0];

/* Control transfer request for window row skip */
ret = write_cmd ( VENDOR_REQUEST_IN,
                  VRQ_GET_WINDOW_PARAM,
                  0,
                  VRQ_GET_WINDOW_ROW_SKIP,
                  (char *) data,
                  2);

if (ret < 0)
    return ret;

eeprom_window_row_skip = data[0];
```

---

```

    return 0;
}

/* Return temperature reading from TMP175 sensor */
double
pm_cam::get_cam_temp (void)
{
    int temp;
    int ret;
    double rtemp;
    char data[3];
    static bool set_ts_config = false;

    /* If first read, set configuration register for 12-bit readout */
    if (set_ts_config) {
        /* Set sensor resolution to 12 bits */
        data [0] = 1;
        data [1] = 0x60;

        write_cmd ( VENDOR_REQUEST_OUT,
                    VRQ_I2C_WRITE,
                    TMP175_I2C_ADDR,
                    0,
                    data,
                    2);

        /* read configuration register */
        data [0] = 1;
        write_cmd ( VENDOR_REQUEST_OUT,
                    VRQ_I2C_WRITE,
                    TMP175_I2C_ADDR,
                    0,
                    data,
                    1);

        write_cmd ( VENDOR_REQUEST_IN,
                    VRQ_I2C_READ,
                    TMP175_I2C_ADDR,
                    0,
                    data,
                    1);

        if (verbose_p)
            fprintf (OUT_MSG, "Configuration Register: 0x%02x\n", data[0]);
    }

    /* read temp readout */
    data[0] = 0;
    ret = write_cmd ( VENDOR_REQUEST_OUT,
                    VRQ_I2C_WRITE,
                    TMP175_I2C_ADDR,
                    0,
                    data,
                    1);

    if (ret < 0)
        fprintf (OUT_ERR_MSG, "Error writing to I2C device...\n");

    ret = write_cmd ( VENDOR_REQUEST_IN,
                    VRQ_I2C_READ,
                    TMP175_I2C_ADDR,
                    0,
                    data,
                    2);

    if (ret < 0)
        fprintf (OUT_ERR_MSG, "Error reading from I2C device...\n");

    temp = ((unsigned char) data[1] | ((unsigned char) data[0] << 8));
    temp >>= 4;

    rtemp = (double) temp / 16.0;

    return (rtemp);
}

```

---

```
}

/* Set FPGA reset state
 * 0 - out of reset
 * 1 - in reset
 */
bool
pm_cam::cam_fpga_reset (int state)
{
    int ret;
    char data;

    ret = write_cmd ( VENDOR_REQUEST_IN,
                      VRQ_FPGA_RESET,
                      state,
                      0,
                      &data,
                      1);

    if (ret < 0) {
        return false;
    }
    else {
        if (data != 0x08) {
            fprintf (stderr, "Error setting FPGA reset mode: NACK received!\n");
            return false;
        }
    }

    return true;
}

/* Set FPGA power state
 * 0 - power off
 * 1 - power on
 */
bool
pm_cam::cam_fpga_power (int state)
{
    int ret;
    char data;

    ret = write_cmd ( VENDOR_REQUEST_IN,
                      VRQ_FPGA_POWER,
                      state,
                      0,
                      &data,
                      1);

    if (ret < 0) {
        return false;
    }
    else {
        if (data != 0x08) {
            fprintf (stderr, "Error setting FPGA power mode: NACK received!\n");
            return false;
        }
    }

    return true;
}

/* Write to FPGA registers over I2C */
bool
pm_cam::fpga_write_reg (unsigned char reg, short value)
{
    int ret;
    char data[3];
    short verify = 0;
```



---

```

data [0] = reg;
data [1] = (( 0xFF00 & value) >> 8);
data [2] = (0x00FF & value);

/* Write Register Address to MI Sensor */
ret = write_cmd (VENDOR_REQUEST_OUT, VRQ_I2C_WRITE, FPGA_I2C_ADDR, 0, data, 3);

if (ret < 0)
    fprintf (stderr, "pm_cam::fpga_write_reg: Error writing FPGA Register - %s\n", usb_strerror ());
else
    if (verbose_p) fprintf (OUT_MSG, "FPGA Register 0x%04x written with 0x%04x\n", reg, value);

    return ret;
}

/* Function to set window width */
void
pm_cam::set_window_width (int _width)
{
    window_width = _width;
    write_reg (MI_REG_COL_SIZE, _width-1);
}

/* Function to set window height */
void
pm_cam::set_window_height (int _height)
{
    window_height = _height;
    write_reg (MI_REG_ROW_SIZE, _height-1);
}

/* Function to set window width with col skip */
void
pm_cam::set_window_width_skip (int _width, int _skip)
{
    short val;
    int factor;

    window_width = _width;
    col_skip = _skip;

    /* verify width is an even number */
    if ( (_width%2) )
        throw std::out_of_range ("Window width must be an even value!\n");

    if ( (_skip < 0) || (_skip > 4) )
        throw std::out_of_range ("Column Skip must be between 0 and 4!\n");

    switch (_skip) {
        case MI_COL_SKIP_NONE:
            factor = 1;
            break;
        case MI_COL_SKIP_2X:
            factor = 2;
            break;
        case MI_COL_SKIP_3X:
            factor = 3;
            break;
        case MI_COL_SKIP_4X:
            factor = 4;
            break;
        case MI_COL_SKIP_8X:
            factor = 8;
            break;
        default:
            factor = 1;
    }

    if ( (_width%factor) )
        throw std::out_of_range ("Specified width is not evenly divisible by the specified skip factor!\n");
}

```

---

---

```

    image_width = _width / factor;

    write_reg (MI_REG_COL_SIZE, _width-1);

    /* set col skip */
    read_reg (MI_REG_COL_ADDR_MODE, &val);
    val |= _skip;

    write_reg(MI_REG_COL_ADDR_MODE, val);
}

/* Function to set window width with col skip */
void
pm_cam::set_window_height_skip (int _height, int _skip)
{
    short val;
    int factor;

    window_height = _height;
    row_skip = _skip;

    /* verify width is an even number */
    if ( (_height%2) )
        throw std::out_of_range ("Window width must be an even value!\n");

    if ( (_skip < 0) || (_skip > 4) )
        throw std::out_of_range ("Column Skip must be between 0 and 4!\n");

    switch (_skip) {
        case MI_ROW_SKIP_NONE:
            factor = 1;
            break;
        case MI_ROW_SKIP_2X:
            factor = 2;
            break;
        case MI_ROW_SKIP_3X:
            factor = 3;
            break;
        case MI_ROW_SKIP_4X:
            factor = 4;
            break;
        case MI_ROW_SKIP_8X:
            factor = 8;
            break;
        default:
            factor = 1;
    }

    if ( (_height%factor) )
        throw std::out_of_range ("Specified height is not evenly divisible by the specified skip factor!\n"
        );

    image_height = _height / factor;

    write_reg (MI_REG_ROW_SIZE, _height-1);

    /* set row skip */
    read_reg (MI_REG_ROW_ADDR_MODE, &val);
    val |= _skip;

    write_reg(MI_REG_ROW_ADDR_MODE, val);
}

/* Set the start column for image readout
 */
void
pm_cam::set_window_col_start (int _col_start)
{
    /* Ensure col_skip specified is within the boundaries of the sensor */
    if ( _col_start > 2047)
        throw std::out_of_range ("Specified column start out of range!\n");
}

```

---

---

```

    write_reg (MI_REG_COLUMN_START, _col_start);
}

/* Set the start row for image readout
 */
void
pm_cam::set_window_row_start (int _row_start)
{
    /* Ensure col_skip specified is within the boundaries of the sensor */
    if ( _row_start > 1535)
        throw std::out_of_range ("Specified column start out of range!\n");

    write_reg (MI_REG_ROW_START, _row_start);
}

/* Function to enable binning
 * Width and Height are the window widths and height,
 * row_skip and col_skip are the respective number of rows or cols to skip
 * window_width and window_height are calculated for the image size
 */
void
pm_cam::set_binning (int _width, int _height, int row_skip, int col_skip)
{
    short val;
    write_reg (MI_REG_COL_SIZE, _width-1);
    write_reg (MI_REG_ROW_SIZE, _height-1);

    /* Set binning */
    read_reg (MI_REG_ROW_ADDR_MODE, &val);
    val |= row_skip-1;
    write_reg (MI_REG_ROW_ADDR_MODE, val);

    read_reg (MI_REG_COL_ADDR_MODE, &val);
    val |= col_skip-1;
    write_reg (MI_REG_COL_ADDR_MODE, val);
}

int
pm_cam::bayer2gray (unsigned char *bayer, unsigned char *buf, int width, int height)
{
    for (int y = 0; y < height - 1; y++) {
        for (int x = 0; x < width; x++) {
            buf[width * y + x] = (unsigned char) ((bayer[width * y + x] + bayer[width * y + x + 1] + bayer[
                width * (y+1) + x] + bayer[width * (y+1) + x + 1]) / 4);
        }
    }
}

int
pm_cam::bayer2rgb (unsigned char *bayer, unsigned long **rgb, int width, int height)
{
    char *raster;

    raster = new char [3 * width * height];

    /* Covert Bayer 8 data to RGB */
    /* Using Nearest neighbor */

    /* Data Comes in G R G R G R G R ... */
    /*           B G B G B G B G ... */
    for (long j = 0; j < height; j++) {
        for (long w = 0; w < width; w++) {
            if (!(j % 2)) {
                if (!(w % 2)) {
                    raster [j*width*3 + w*3 + GREEN_OFFSET] = bayer [j*width + w];
                    raster [j*width*3 + w*3 + RED_OFFSET] = bayer [j*width + w + 1];
                    raster [j*width*3 + w*3 + BLUE_OFFSET] = bayer [(j+1)*(width) + w];
                }
            }
        }
    }
}

```

---

```

    else {
        raster[j*width*3 + w*3 + GREEN_OFFSET] = bayer[j*width + w - 1];
        raster[j*width*3 + w*3 + RED_OFFSET] = bayer[j*width + w];
        raster[j*width*3 + w*3 + BLUE_OFFSET] = bayer[(j+1)*(width) + w - 1];
    }
}
else {
    if (!(w%2)) {
        raster[j*width*3 + w*3 + GREEN_OFFSET] = bayer[(j-1)*width + w];
        raster[j*width*3 + w*3 + RED_OFFSET] = bayer[(j-1)*width + w + 1];
        raster[j*width*3 + w*3 + BLUE_OFFSET] = bayer[j*width + w];
    }
    else {
        raster[j*width*3 + w*3 + GREEN_OFFSET] = bayer[(j-1)*width + w - 1];
        raster[j*width*3 + w*3 + RED_OFFSET] = bayer[(j-1)*width + w];
        raster[j*width*3 + w*3 + BLUE_OFFSET] = bayer[j*width + w - 1];
    }
}
}
}

for (int x = 0; x < width; x++)
    for (int y = 0; y < height; y++)
        rgb[x][y] = raster[x*width*3 + y*3 + RED_OFFSET] +
            (raster[x*width*3 + y*3 + GREEN_OFFSET] << 8) +
            (raster[x*width*3 + y*3 + BLUE_OFFSET] << 16);

delete raster;
}

/* Perform interpolation and write TIFF file from raw bayer data */
int
pm_cam::bayer2tiff (unsigned char *buf_in, char *filename, int width, int height)
{
    TIFF                *tiff_fp;
    char                *raster;
    char                *time_stamp;
    int                 ret;

    if ((tiff_fp = TIFFOpen (filename, "w")) == NULL) {
        fprintf (stderr, "Error opening file...\n");
        return -1;
    }

    /* Allocate Memory for Image */
    if ((raster = (char *) malloc (sizeof (char) * width * height * 3)) == NULL) {
        fprintf (stderr, "Unable to allocate memory\n");
        return -1;
    }

    /* Covert RAW data to TIFF image */
    /* Using Nearest neighbor */

    /* Data Comes in G R G R G R G R ... */
    /*           B G B G B G B G ... */
    for (long j = 0; j < height; j++) {
        for (long w = 0; w < width; w++) {
            if (!(j % 2)) {
                if (!(w % 2)) {
                    raster[j*width*3 + w*3 + RED_OFFSET] = buf_in[j*width + w];
                    raster[j*width*3 + w*3 + GREEN_OFFSET] = buf_in[j*width + w + 1];
                    raster[j*width*3 + w*3 + BLUE_OFFSET] = buf_in[(j+1)*(width) + w + 1];
                }
                else {
                    raster[j*width*3 + w*3 + GREEN_OFFSET] = buf_in[j*width + w];
                    raster[j*width*3 + w*3 + RED_OFFSET] = buf_in[j*width + w - 1];
                    raster[j*width*3 + w*3 + BLUE_OFFSET] = buf_in[(j+1)*(width) + w];
                }
            }
            else {
                if (!(w%2)) {
                    raster[j*width*3 + w*3 + GREEN_OFFSET] = buf_in[j*width + w];
                }
            }
        }
    }
}

```

```

        raster [j*width*3 + w*3 + RED_OFFSET] = buf_in [(j-1)*width + w];
        raster [j*width*3 + w*3 + BLUE_OFFSET] = buf_in [j*width + w + 1];
    }
    else {
        raster [j*width*3 + w*3 + GREEN_OFFSET] = buf_in [(j-1)*width + w];
        raster [j*width*3 + w*3 + RED_OFFSET] = buf_in [(j-1)*width + w - 1];
        raster [j*width*3 + w*3 + BLUE_OFFSET] = buf_in [j*width + w];
    }
}
}

/* Set Image Values */
TIFFSetField (tiff_fp, TIFFTAG_IMAGEWIDTH, width);
TIFFSetField (tiff_fp, TIFFTAG_IMAGELENGTH, height);
TIFFSetField (tiff_fp, TIFFTAG_BITSPERSAMPLE, 8);
TIFFSetField (tiff_fp, TIFFTAG_SAMPLESPERPIXEL, 3);

/* Set Compression */
/* No Compression */
TIFFSetField (tiff_fp, TIFFTAG_COMPRESSION, COMPRESSION_NONE);
TIFFSetField (tiff_fp, TIFFTAG_PHOTOMETRIC, PHOTOMETRIC_RGB);
TIFFSetField (tiff_fp, TIFFTAG_PLANARCONFIG, PLANARCONFIG_CONTIG);

/* Write Image Information to TIFF file */
if (TIFFWriteEncodedStrip (tiff_fp, 0, raster, width * height * 3) == 0) {
    fprintf (stderr, "Unable to write to file\n");
    return -1;
}

/* Deallocate memory used */
free (raster);

/* Close TIFF file */
TIFFClose (tiff_fp);

return 0;
}

int
pm_cam::write_tiff (unsigned char *buf_in, char *filename, int width, int height)
{
    TIFF          *tiff_fp;
    char          *raster;
    int           ret;

    if ( (tiff_fp = TIFFOpen (filename, "w")) == NULL ) {
        fprintf (stderr, "Error opening file...\n");
        return -1;
    }

    /* Set Image Values */
    TIFFSetField (tiff_fp, TIFFTAG_IMAGEWIDTH, width);
    TIFFSetField (tiff_fp, TIFFTAG_IMAGELENGTH, height);
    TIFFSetField (tiff_fp, TIFFTAG_BITSPERSAMPLE, 8);
    TIFFSetField (tiff_fp, TIFFTAG_SAMPLESPERPIXEL, 1);

    /* Set Compression */
    /* No Compression */
    TIFFSetField (tiff_fp, TIFFTAG_COMPRESSION, COMPRESSION_NONE);
    TIFFSetField (tiff_fp, TIFFTAG_PHOTOMETRIC, PHOTOMETRIC_MINISBLACK);
    TIFFSetField (tiff_fp, TIFFTAG_PLANARCONFIG, PLANARCONFIG_CONTIG);

    /* Write Image Information to TIFF file */
    if (TIFFWriteEncodedStrip (tiff_fp, 0, buf_in, width * height * 1) == 0) {
        fprintf (stderr, "Unable to write to file\n");
        return -1;
    }

    /* Close TIFF file */
    TIFFClose (tiff_fp);

    return 0;
}

```

---

```

/* Convert the RGB to Grayscale */
unsigned char **
pm_cam::convert_grayscale (unsigned long **rgb)
{
    unsigned char **grayscale;
    int W, H;
    int R, G, B;

    /* Get Image Size */
    W = image_width;
    H = image_height;

    /* Threshold the Image */
    /* allocate memory */
    grayscale = new unsigned char *[W];

    for (int i = 0; i < W; i++)
        grayscale[i] = new unsigned char [H];

    /* Extract Grayscale from RGB */
    float **Y;
    float max = 0.0;
    float min = (float) (1<<20);

    /* Allocate memory for Y */
    Y = new float *[W];
    for (int x = 0; x < W; x++)
        Y[x] = new float [H];

    /* Convert active Image form image to Grayscale */
    for (int x = 0; x < W; x++) {
        for (int y = 0; y < H; y++) {
            /* Extract RGB Components */
            R = (int) rgb[x][y] & 0x000000FF;
            G = (int) rgb[x][y] & 0x0000FF00 >> 8;
            B = (int) rgb[x][y] & 0x00FF0000 >> 16;

            /* Calculate Luminance */
            Y[x][y] = (float) R + (4.5907 * (float) G) + (0.0601 * (float) B);

            if (Y[x][y] > max)
                max = Y[x][y];

            if (Y[x][y] < min)
                min = Y[x][y];
        }
    }

    /* Scale the Y values between 0 and 255 */
    /* Draw a new picture with Grayscale Colours (All set to Y) */
    for (int x = 0; x < W; x++) {
        for (int y = 0; y < H; y++) {
            Y[x][y] = Y[x][y] * (255.0 / (max - min));
            G = (int) Y[x][y];
            grayscale[x][y] = (unsigned char) G;
        }
    }

    return grayscale;
}

```

---

---

## Appendix D

### *System Control Board Firmware*

---

This chapter contains all firmware source code developed for the dsPIC33 microcontroller of the system control board.

#### D.1 common.h

```
/* common.h
 * =====
 * Contains global definitions for system
 *
 * Author: Neil Scott
 * Date: August 01, 2007
 */

#ifndef COMMON_H
#define COMMON_H

#include <p33FJ256GP710.h>
#include "delay.h"
#include "uart2.h"
#include "i2c_2.h"
#include "lcd_i2c.h"
#include "i2c_slave.h"
#include "job_ids.h"
#include "i2c_commands.h"
#include "uart_commands.h"
#include "err.h"
#include "i2c_io_exp.h"

/* Delay for Shutdown Timer */
#define SHUTDOWN_TIMER_DEFAULT 20

/* I2C Slave Device Addresses */
#define I2C_ADDR_FRONT_PANEL 0x27
#define I2C_ADDR_SIDE_PANEL 0x23

/*FOR DEBUGGING */
#define ERR_LED LATCbits.LATC1
#define ERR_LED_TRIS TRISCbits.TRISC1
```

```

#define Fcy 4000000

/* System Globals */
#define ASSERTED 1
#define DEASSERTED 0
#define TRUE 1
#define FALSE 0

#define MTR_PULSE_WIDTH 40
#define CAM_TRIG0_PULSE_WIDTH 232 /*1800*/ /* Global Shutter Control */
#define CAM_TRIG1_PULSE_WIDTH 232 /*1800*/ /* Global Shutter Control */
#define BLO_PULSE_WIDTH 800
#define BL1_PULSE_WIDTH 800
#define FLO_PULSE_WIDTH 30500
#define FL1_PULSE_WIDTH 30500
#define MTR_DEFAULT_FREQ 1600 /* Hz */
#define PULSE_COUNTER_MAX 650

#define CALIBRATION_START_POS 450

/* Pulse Count Constants for given positions */
/* Optisorter Original Holder Configuration */
#if 1
#define CAM0_PULSE_POSITION_DEFAULT 1
#define CAM1_PULSE_POSITION_DEFAULT 320
#define ACCEPT_ON_PULSE_POSITION_DEFAULT 450
#define ACCEPT_OFF_PULSE_POSITION_DEFAULT 175
#endif

/* PC Definitions */
#define PC0 0x01
#define PC1 0x02
#define PC2 0x03
#define PC3 0x04

/* I/O Definitions */
/* Pneumatic Controls */
#define PNEU_MAIN LATEbits.LATE2
#define PNEU_UPPER LATEbits.LATE1
#define PNEU_ACCEPT0 LATEbits.LATE3
#define PNEU_ACCEPT1 LATEbits.LATE4
#define PNEU_ACCEPT2 LATEbits.LATE5
#define PNEU_ACCEPT3 LATEbits.LATE6

#define PNEU_MAIN_TRIS TRISEbits.TRISE1
#define PNEU_UPPER_TRIS TRISEbits.TRISE2
#define PNEU_ACCEPT0_TRIS TRISEbits.TRISE3
#define PNEU_ACCEPT1_TRIS TRISEbits.TRISE4
#define PNEU_ACCEPT2_TRIS TRISEbits.TRISE5
#define PNEU_ACCEPT3_TRIS TRISEbits.TRISE6

/* LED Lighting Controls */
#define IO_LED_BLO LATDbits.LATD3
#define IO_LED_BL1 LATDbits.LATD4
#define IO_LED_FLO LATDbits.LATD5
#define IO_LED_FL1 LATDbits.LATD6

#define IO_LED_BLO_TRIS TRISDbits.TRISD3
#define IO_LED_BL1_TRIS TRISDbits.TRISD4
#define IO_LED_FLO_TRIS TRISDbits.TRISD5
#define IO_LED_FL1_TRIS TRISDbits.TRISD6

/* Camera Trigger / Global Shutter Control */
#define IO_CAM_TRIG0 LATDbits.LATD1
#define IO_CAM_TRIG1 LATDbits.LATD2

#define IO_CAM_TRIG0_TRIS TRISDbits.TRISD2
#define IO_CAM_TRIG1_TRIS TRISDbits.TRISD1

/* PC Power Sense */
#define PC_SENSE0 PORTCbits.RC3
#define PC_SENSE1 PORTCbits.RC4
#define PC_SENSE2 PORTBbits.RB5

```



```

#define PC_SENSE3                                PORTBbits.RB4

/* PC Power Control */
#define PC_PWR0                                LATBbits.LATB0
#define PC_PWR1                                LATBbits.LATB1
#define PC_PWR2                                LATBbits.LATB2
#define PC_PWR3                                LATBbits.LATB3

/* Tristate for PC Sense inputs */
#define PC_SENSE0_TRIS                          TRISCbits.TRISC3
#define PC_SENSE1_TRIS                          TRISCbits.TRISC4
#define PC_SENSE2_TRIS                          TRISBbits.TRISB5
#define PC_SENSE3_TRIS                          TRISBbits.TRISB4

/* Tristate Control of PC PWR outputs */
#define PC_PWR0_TRIS                            TRISBbits.TRISB0
#define PC_PWR1_TRIS                            TRISBbits.TRISB1
#define PC_PWR2_TRIS                            TRISBbits.TRISB2
#define PC_PWR3_TRIS                            TRISBbits.TRISB3

/* E-Stop Inputs */
#define ESTOP_SIG0                              PORTBbits.RB8
#define ESTOP_SIG1                              PORTBbits.RB9
#define ESTOP_SIG2                              PORTBbits.RB10
#define ESTOP_SIG3                              PORTBbits.RB11

/* E-Stop Tristates */
#define ESTOP_SIG0_TRIS                          TRISBbits.TRISB8
#define ESTOP_SIG1_TRIS                          TRISBbits.TRISB9
#define ESTOP_SIG2_TRIS                          TRISBbits.TRISB10
#define ESTOP_SIG3_TRIS                          TRISBbits.TRISB11

#define IO_ESTOP_INT_TRIS                       TRISFbits.TRISF6

/* Interrupt Register Definitions */
/* E-Stop Interrupt Control Registers */
#define INT_ESTOP_CONbits                       INTCON2bits

/* E-Stop Interrupt Enable Register <BIT> */
#define INT_ESTOP_IE                           IEC0bits.INT0IE

/* E-Stop Interrupt Status Register <BIT> */
#define INT_ESTOP_IF                           IFS0bits.INT0IF

/* BI2C Inputs */
#define BI2C_INT0                              PORTGbits.RG0
#define BI2C_INT1                              PORTGbits.RG1
#define BI2C_INT2                              PORTFbits.RF0
#define BI2C_INT3                              PORTFbits.RF1

#define BI2C_INT0_TRIS                          TRISGbits.TRISG0
#define BI2C_INT1_TRIS                          TRISGbits.TRISG1
#define BI2C_INT2_TRIS                          TRISFbits.TRISF0
#define BI2C_INT3_TRIS                          TRISFbits.TRISF1

#define IO_BI2C_INT_TRIS                        TRISAbits.TRISA12
#define INT_BI2C_CONbits                       INTCON2bits

#define INT_BI2C_IE                             IEC1bits.INT1IE
#define INT_BI2C_IF                             IFS1bits.INT1IF

/* I2C Bus Switch line control */
#define I2C_BUS_SW_A0_TRIS                      TRISFbits.TRISF8
#define I2C_BUS_SW_A1_TRIS                      TRISFbits.TRISF7

#define I2C_BUS_SW_A0                          LATFbits.LATF8
#define I2C_BUS_SW_A1                          LATFbits.LATF7

/* OC Definitions */
/* Output Captuer Control Registers */
#define OC_MTR_CTRL_CONbits                     OC1CONbits
#define OC_CAM_TRIG0_CONbits                    OC3CONbits
#define OC_CAM_TRIG1_CONbits                    OC2CONbits
#define OC_LED_BL0_CONbits                      OC4CONbits

```

```

#define OC_LED_BL1_CONbits      OC5CONbits
#define OC_LED_FL0_CONbits      OC6CONbits
#define OC_LED_FL1_CONbits      OC7CONbits

/* Start / Stop Registers */
#define OC_MTR_CTRL_R           OC1R    /* Motor Control */
#define OC_MTR_CTRL_RS          OC1RS

#define OC_CAM_TRIG0_R          OC2R    /* Camera Trigger 0 */
#define OC_CAM_TRIG0_RS         OC2RS
#define OC_CAM_TRIG1_R          OC3R    /* Camera Trigger 1 */
#define OC_CAM_TRIG1_RS         OC3RS

#define OC_LED_BL0_R            OC4R    /* LED Backlight 0 */
#define OC_LED_BL0_RS           OC4R
#define OC_LED_BL1_R            OC5R    /* LED Backlight 1 */
#define OC_LED_BL1_RS           OC5RS

#define OC_LED_FL0_R            OC6R    /* LED Frontlight 0 */
#define OC_LED_FL0_RS           OC6RS
#define OC_LED_FL1_R            OC7R    /* LED Frontlight 1 */
#define OC_LED_FL1_RS           OC7RS

/* Output Compare Interrupt Status Register <BIT> */
#define OC_MTR_CTRL_IF          IFS0bits.OC1IF
#define OC_CAM_TRIG0_IF         IFS0bits.OC2IF
#define OC_CAM_TRIG1_IF         IFS1bits.OC3IF
#define OC_LED_BL0_IF           IFS1bits.OC4IF
#define OC_LED_BL1_IF           IFS2bits.OC5IF
#define OC_LED_FL0_IF           IFS2bits.OC6IF
#define OC_LED_FL1_IF           IFS2bits.OC7IF

/* Output Compare Interrupt Enable Register <BIT> */
#define OC_MTR_CTRL_IE          IEC0bits.OC1IE
#define OC_CAM_TRIG0_IE         IEC0bits.OC2IE
#define OC_CAM_TRIG1_IE         IEC1bits.OC3IE
#define OC_LED_BL0_IE           IEC1bits.OC4IE
#define OC_LED_BL1_IE           IEC2bits.OC5IE
#define OC_LED_FL0_IE           IEC2bits.OC6IE
#define OC_LED_FL1_IE           IEC2bits.OC7IE

/* Input Capture Definitions */
/* Input Capture Control Registers */
#define IC_PS0_CONbits          IC1CONbits
#define IC_PS1_CONbits          IC2CONbits
#define IC_PS2_CONbits          IC3CONbits
#define IC_PS3_CONbits          IC4CONbits

/* Input Capture Interrupt Enable Register <BIT> */
#define IC_PS0_IE               IEC0bits.IC1IE
#define IC_PS1_IE               IEC0bits.IC2IE
#define IC_PS2_IE               IEC2bits.IC3IE
#define IC_PS3_IE               IEC2bits.IC4IE

/* Input Capture Interrupt Status Register <BIT> */
#define IC_PS0_IF               IFS0bits.IC1IF
#define IC_PS1_IF               IFS0bits.IC2IF
#define IC_PS2_IF               IFS2bits.IC3IF
#define IC_PS3_IF               IFS2bits.IC4IF

/* Timer Definitions */
/* Timer Control Registers */
#define TMR_BUS_SWITCH_CONbits  T4CONbits
#define TMR_BUS_SWITCH_PR       PR4
#define TMR_BUS_SWITCH_TMR      TMR4

/* Timer Interrupt Enable Register <BIT> */
#define TMR_BUS_SWITCH_IE       IEC1bits.T4IE

```

---

```

/* Timer Interrupt Status Register <BIT> */
#define TMR_BUS_SWITCH_IF          IFS1bits.T4IF

/* Interrupt Service Routine Definitions */
#define isr_MTR_CTRL                _OC1Interrupt
#define isr_CAM_TRIG0               _OC2Interrupt
#define isr_CAM_TRIG1               _OC3Interrupt
#define isr_LED_BL0                 _OC4Interrupt
#define isr_LED_BL1                 _OC5Interrupt
#define isr_LED_FL0                 _OC6Interrupt
#define isr_LED_FL1                 _OC7Interrupt

#define isr_PS0                     _IC1Interrupt
#define isr_PS1                     _IC2Interrupt
#define isr_PS2                     _IC3Interrupt
#define isr_PS3                     _IC4Interrupt

#define isr_BUS_SWITCH              _T4Interrupt

#define isr_ESTOP                   _INT0Interrupt
#define isr_BI2C                    _INT1Interrupt

/* Some dsPIC33 Constants */
#define OCM_DISABLED                0x00
#define OCM_FORCE_HIGH              0x01
#define OCM_FORCE_LOW               0x02
#define OCM_TOGGLE                  0x03
#define OCM_SINGLE_PULSE            0x04
#define OCM_CONT_PULSE              0x05
#define OCM_PWM_NOFAULT             0x06
#define OCM_PWM_FAULT               0x07

#define OCTSEL_TIMER2               0x00
#define OCTSEL_TIMER3               0x01

#define ICM_DISABLED                0x00
#define ICM_RISE_FALL_EDGE          0x01
#define ICM_FALL_EDGE               0x02
#define ICM_RISE_EDGE               0x03
#define ICM_4TH_RISE_EDGE           0x04
#define ICM_16TH_RISE_EDGE          0x05
#define ICM_INTERRUPT_ONLY          0x07

#define ICTMR_TIMER2                0x01
#define ICTMR_TIMER3                0x00

/* Machine Specifics */
#define MACHINE_STOPPED              0x00
#define MACHINE_RUNNING              0x01
#define MACHINE_FAULTED              0x02
#define MACHINE_DEBUG                0x03

/* Defines for PC Power State bitmask */
#define PC_POWER_ON                  0x01    /* Motherboard is not powered */
#define PC_POWER_READY               0x02    /* Inspect software is running */

#define RAMP_NONE                    0
#define RAMP_UP                      1
#define RAMP_DOWN                    2

#define MOTOR_MIN_FREQ               400
#define I2C_BUS_SW_PR                 350    /* (350) Timer Period */

#define PULSES_PER HOLDER            495    /* Number of pulses between holders (approx) */

#define T2TCKPS                      0x03    /* Timer 2 Prescale factor register */
#define T2PF                         256     /* Timer 2 Prescale factor */

#define MAX_JOBS                     128

#define I2C_SLAVE_ADDRESS             0x44    /* 7-bit I2C Slave Address */

```

---

```
#define BI2C_CHANNEL_IO_SIDE_PANEL      3
#define BI2C_CHANNEL_IO_FRONT_PANEL    2

/* Front Panel Defines */
/* INPUTS */
#define FRONT_PANEL_PB1_bit             0x02
#define FRONT_PANEL_PB2_bit             0x04
#define FRONT_PANEL_SW_bit              0x01

/* Side Panel Defines */
/* INPUTS */
#define SIDE_PANEL_PB_bit                0x01

/* Global Structure Definitions */
struct SYSTEM_STATUS {
    unsigned state:2;
};

/* Power State of PCs */
struct PC_PWR_STATUS {
    unsigned state:2;
};

/* Job Structure */
struct JOB {
    unsigned char job_id;
    unsigned char data[32];
    unsigned char bc;
};

#endif /* COMMON_H */
```

## D.2 job\_ids.h

```
/* job_ids.h
 * =====
 * Contains job IDs for various system jobs
 *
 * Author: Neil Scott
 * Date: September 17, 2007
 */

#ifndef JOB_IDS_H
#define JOB_IDS_H

#define I2C1_REQUEST      0x02
#define MOTOR_PULSE       0x03
#define UART_REQUEST      0x04
#define I2C_BUS_SWITCH    0x05
#define I2C2_REQUEST      0x06
#define LCD_WRITE         0x07
#define LCD_WRITE_LOC     0x08
#define LCD_CLEAR         0x09
#define LCD_LED           0x0a
#define BI2C_EVENT        0x0b
#define PC_POWER_SET      0x0c
#define PC_POWER_RELEASE  0x0d
#define HALT_SYSTEM       0xff

#endif /* JOB_IDS_H */
```

## D.3 main.c

```
/* main.c
 * =====
 * Control Board Firmware initial release
 * - Controls motor speed
```

```
* - Controls lighting and provides camera triggers
* - Controls pneumatic valves to eject capsules
* - Monitors and reports system health
* - Tracks Capsules and provides a communication interface
*   over I2C to each quadrant
* - RS-232 communication for reading statistical data
*
* Author: Neil Scott
* Date: May 28, 2007
*/

#include "common.h"
#include "delay.h"
#include "uart2.h"
#include "uart_commands.h"

#define LCD_LOCK_TIMEOUT 12

extern struct I2C_IO_EXP temp_sense;
extern struct I2C_IO_EXP io_side_panel;
extern struct I2C_IO_EXP io_front_panel;

/* Side Panel Bitmaps */
/* Outputs */
#define SP_LED_bit      0x01
#define SP_BUZZ_bit     0x02
#define SP_AUX_AIR_bit  0x04
/* Inputs */
#define SP_SW_bit       0x01

/* Front Panel Bitmaps */
/* Outputs */
#define FP_LED_bit      0x08

volatile struct JOB job_list[MAX_JOBS];
volatile unsigned char curr_job;
volatile unsigned char last_job;
volatile char *PM_TITLE1 = {"-PharmaSorter-"};
volatile char *PM_TITLE2 = {"I-START II-STOP"};

volatile unsigned char hb;
volatile unsigned char first_write_p = TRUE;

volatile unsigned int trigger_comp = 0;
volatile unsigned int extra_comp = 0;

volatile unsigned char lcd_lock = 0;
volatile unsigned char bi2c_lock = 0;

volatile unsigned char one_step = 0;

volatile unsigned char disable_count = 0;
volatile unsigned char enable_count = 0;

/* Initially in calibration mode to align holder */
volatile unsigned char calibration_mode = 1;
volatile unsigned char calibration_found_zero = 0;

volatile unsigned char pcs_ready = 0;
volatile unsigned char pc_ready_done_flag = 0;

volatile unsigned char shutdown_start_timer_flag = FALSE;
volatile unsigned char shutdown_timer = SHUTDOWN_TIMER_DEFAULT;
volatile unsigned char shutdown_in_progress_flag = FALSE;
volatile unsigned char wait_to_halt_flag = FALSE;

volatile unsigned char prev_fp_sw_state;
volatile unsigned char fp_sw_state;

volatile unsigned int mtr_pulse_width;
volatile unsigned int mtr_running_freq;
volatile unsigned int cam_trig0_pulse_width;
volatile unsigned int cam_trig1_pulse_width;
volatile unsigned int bl0_pulse_width;
```

```
volatile unsigned int bll_pulse_width;
volatile unsigned int fl0_pulse_width;
volatile unsigned int fl1_pulse_width;

volatile unsigned int cam0_pulse_position;
volatile unsigned int cam1_pulse_position;
volatile unsigned int accept_on_pulse_position;
volatile unsigned int accept_off_pulse_position;

extern char i2c_io_outdata;

/* Job Queue Get Next Available Job */
unsigned char get_next_job (void)
{
    unsigned char j;

    j = last_job + 1;

    if (j > (MAX_JOBS - 1)) {
        j = 0;
    }

    return j;
}

/* Job Queue Get Next Unserved Job */
unsigned char get_next_curr_job (void)
{
    unsigned char j;

    j = curr_job + 1;

    if (j > (MAX_JOBS - 1)) {
        j = 0;
    }

    return j;
}

/* Job Queue Add New Job */
void add_job (unsigned char job_id)
{
    last_job++;

    if (last_job > (MAX_JOBS - 1)) {
        last_job = 0;
    }

    job_list[last_job].job_id = job_id;
}

/* Job Queue Clear last job */
void complete_job (void)
{
    job_list[curr_job].bc = 0;

    curr_job++;

    if (curr_job > (MAX_JOBS - 1)) {
        curr_job = 0;
    }
}

/* Configure PLL and WDT */
_FOSCSEL(FNOSC_PRIPLL);
_FOSC(FCKSM_CSDCMD & OSCIOFNC_OFF & POSCMD_XT);
_FWDT (FWDTEN_OFF);

/* Function Prototypes */
```

---

---

```

void init_io(void);
void init_check(void);
void init_sys(void);
void calibrate_sys(void);
void uart2_hdlr (void);
static inline void motor_step_hdlr (void);
void i2c1_request_hdlr (unsigned char i_job);
void i2c2_request_hdlr (unsigned char i_job);
void disable_motor (void);
void enable_motor (void);
void debug_mode (int set);
void set_pc_pwr (unsigned char pc);

/* Global Variables */
volatile unsigned char ramp_mode;
volatile unsigned int motor_speed;          /* in Hz */
volatile unsigned int motor_speed_target;  /* in Hz */
volatile unsigned char motor_stop_flag;
volatile unsigned char i2c_bus_lock;
volatile unsigned char i2c_bus_lock_count;
volatile unsigned char i2c_bs_active_bus;
volatile unsigned int pulse_counter;        /* Track arm position in terms of motor step pulses */
volatile unsigned int cap_count;
volatile unsigned char capsule_passfail[4][16]; /* Store Pass/Fail result from inspections */
volatile unsigned char pc_pwr_release_flag = FALSE;
volatile unsigned char refresh_title_flag = FALSE;

/* System Fault and Status Structure */
volatile struct SYSTEM_STATUS system_status;

/* PC Power State Structure */
volatile struct PC_PWR_STATUS pc_pwr_status[4];

/* UART Buffers and Pointers */
extern struct UART_Rx  uart_rx;
extern struct UART_Tx  uart_tx;
unsigned char uart_rx_buf[MAX_UART_RX_BUF];
unsigned char uart_tx_buf[MAX_UART_TX_BUF];
unsigned char uart_cmd_flag;

/* Capsule Counters */
volatile unsigned long capsule_good_count[4];
volatile unsigned long capsule_bad_count[4];
volatile unsigned long capsule_total_count[4];

/* Output Compare Interrupt Service Routine for Motor PWM Control */
void
__attribute__((interrupt, no_auto_psv)) isr_MTR_CTRL (void)
{
    unsigned char tjob;

    /* Clear Interrupt Flag */
    OC_MTR_CTRL_IF = DEASSERTED;

    if (calibration_mode) {
        if (!calibration_found_zero) {
            if (pulse_counter == 0) {
                calibration_found_zero = TRUE;
                OC_MTR_CTRL_CONbits.OCM = OCM_DISABLED;
                Delay (Delay_1S_Cnt);
                OC_MTR_CTRL_CONbits.OCM = OCM_CONT_PULSE;
            }
        }
        else {
            /* Find Start Pos */
            if (pulse_counter == CALIBRATION_START_POS) {
                /* Stop Motor */
                OC_MTR_CTRL_CONbits.OCM = OCM_DISABLED;
                calibration_mode = 0;
            }
        }
    }

    pulse_counter++;

```

---

```
if ( (pulse_counter > PULSE_COUNTER_MAX) && (system_status.state == MACHINE_RUNNING)) {
    disable_motor();
    OC_MTR_CTRL_CONbits.OCM = OCM_DISABLED;
    system_status.state = MACHINE_FAULTED;
}

motor_step_hdlr();
}

/* Output Compare Interrupt Service Routine for LED Backlight 0 */
void
__attribute__((interrupt, no_auto_psv)) isr_LED_BL0 (void)
{
    /* Clear Interrupt Flag */
    OC_LED_BL0_IF = DEASSERTED;

    /* Turn Off Output Compare Module */
    OC_LED_BL0_CONbits.OCM = OCM_DISABLED;
}

/* Output Compare Interrupt Service Routine for LED Backlight 0 */
void
__attribute__((interrupt, no_auto_psv)) isr_LED_BL1 (void)
{
    /* Clear Interrupt Flag */
    OC_LED_BL1_IF = DEASSERTED;

    /* Turn Off Output Compare Module */
    OC_LED_BL1_CONbits.OCM = OCM_DISABLED;
}

/* Output Compare Interrupt Service Routine for Camera Trigger 0 */
void
__attribute__((interrupt, no_auto_psv)) isr_CAM_TRIG0 (void)
{
    /* Clear Interrupt Flag */
    OC_CAM_TRIG0_IF = DEASSERTED;

    /* Turn Off Output Compare Module */
    OC_CAM_TRIG0_CONbits.OCM = OCM_DISABLED;
}

/* Output Compare Interrupt Service Routine for Camera Trigger 0 */
void
__attribute__((interrupt, no_auto_psv)) isr_CAM_TRIG1 (void)
{
    /* Clear Interrupt Flag */
    OC_CAM_TRIG1_IF = DEASSERTED;

    /* Turn Off Output Compare Module */
    OC_CAM_TRIG1_CONbits.OCM = OCM_DISABLED;
}

/* Input Capture Interrupt Service Routine for Proximity Sensor PS0 */
void
__attribute__((interrupt, no_auto_psv)) isr_PS0 (void)
{
    /* Clear Interrupt Flag */
    IC_PS0_IF = DEASSERTED;
}

/* Input Capture Interrupt Service Routine for Proximity Sensor PS1 */
void
__attribute__((interrupt, no_auto_psv)) isr_PS1 (void)
{
    /* Clear Interrupt Flag */
    IC_PS1_IF = DEASSERTED;
}
```



```
}

/* Input Capture Interrupt Service Routine for Proximity Sensor PS2 */
void
__attribute__((interrupt, no_auto_psv)) isr_PS2 (void)
{
    /* Clear Interrupt Flag */
    IC_PS2_IF = DEASSERTED;
}

/* Input Capture Interrupt Service Routine for Proximity Sensor PS3 */
void
__attribute__((interrupt, no_auto_psv)) isr_PS3 (void)
{
    unsigned char tjob;
    unsigned char i;
    /* Clear Interrupt Flag */
    IC_PS3_IF = DEASSERTED;

    /* Reset pulse counter */
    pulse_counter = 0;

    /* Increment capsule counter */
    cap_count++;
    if (cap_count > 15) {
        cap_count = 0;
    }

    for (i = 0; i < 4; i++) {
        capsule_passfail[i][cap_count] = 0;
        capsule_total_count[i]++;
    }
}

/* Timer 2 Interrupt Service Routine */
void
__attribute__((interrupt, no_auto_psv)) _T2Interrupt (void)
{
    IFS0bits.T2IF = DEASSERTED;
}

/* Timer 4 Interrupt Service Routine */
void
__attribute__((interrupt, no_auto_psv)) isr_BUS_SWITCH (void)
{
    /* Clear Interrupt Flag */
    TMR_BUS_SWITCH_IF = DEASSERTED;

    LATCbits.LATC1 = DEASSERTED;
    LATGbits.LATG6 = DEASSERTED;

    /* Verify not between START and STOP */
    if (I2C1STATbits.S) {
        LATCbits.LATC1 = ASSERTED;
        LATGbits.LATG6 = DEASSERTED;
        return;
    }

    /* Cycle active bus */
    switch (i2c_bs_active_bus) {
        case 0:
            I2C_BUS_SW_A0 = ASSERTED;
            I2C_BUS_SW_A1 = DEASSERTED;
            i2c_bs_active_bus = 1;
            break;

        case 1:
            I2C_BUS_SW_A0 = DEASSERTED;
            I2C_BUS_SW_A1 = ASSERTED;
            i2c_bs_active_bus = 2;
    }
}
```

```

        break;

    case 2:
        I2C_BUS_SW_A0 = ASSERTED;
        I2C_BUS_SW_A1 = ASSERTED;
        i2c_bs_active_bus = 3;
        break;

    case 3:
        I2C_BUS_SW_A0 = DEASSERTED;
        I2C_BUS_SW_A1 = DEASSERTED;
        i2c_bs_active_bus = 0;
        break;

    default:
        I2C_BUS_SW_A0 = DEASSERTED;
        I2C_BUS_SW_A1 = DEASSERTED;
        i2c_bs_active_bus = 0;
        break;
    }
}

/* Timer 5 Interrupt Service Routine */
void
__attribute__((interrupt, no_auto_psv)) _T5Interrupt (void)
{
    unsigned char tjob;

    /* Clear Interrupt Flag */
    IFS1bits.T5IF = DEASSERTED;

    /* Calculate Trigger Compensation */
    if (motor_speed > 300) {
        trigger_comp = (int) (0.09 * (double) motor_speed) - 34;
        extra_comp = (int) 0.05 * (double) motor_speed;
    }
    else {
        trigger_comp = 0;
    }

    /* If Ramp Up */
    if (ramp_mode == RAMP_UP) {
        motor_speed++;
        PR2 = (unsigned int) (((((double) Fcy) / (double) motor_speed) - 1.0) / (double) T2PF);
        PNEU_MAIN = DEASSERTED;
    }

    /* If Ramp Down */
    if (ramp_mode == RAMP_DOWN) {
        motor_speed--;
        PR2 = (unsigned int) (((((double) Fcy) / (double) motor_speed) - 1.0) / (double) T2PF);
    }

    /* If desired speed reached - stop this timer */
    if (motor_speed == motor_speed_target) {
        T5CONbits.TON = FALSE;
        enable_count = 0;
        if (motor_stop_flag) {
            disable_count = 0;

            OC_MTR_CTRL_CONbits.OCM = OCM_DISABLED;
            PNEU_MAIN = ASSERTED;
            motor_stop_flag = FALSE;
            refresh_title_flag = TRUE;
        }
    }
}

/* Timer 7 LCD Timer */
void
__attribute__((interrupt, no_auto_psv)) _T7Interrupt (void)
{

```

```
/* Clear Interrupt Flag */
IFS3bits.T7IF = DEASSERTED;

/* Disable Timer */
T7CONbits.TON = FALSE;

refresh_title_flag = TRUE;
}

/* Timer 8 Interrupt Service Routine */
void
__attribute__((interrupt, no_auto_psv)) _T8Interrupt (void)
{
    unsigned char tjob;
    unsigned char temp;

    /* Clear Interrupt Flag */
    IFS3bits.T8IF = DEASSERTED;

    /* Release Soft Power SW */
    if (pc_pwr_release_flag) {
        pc_pwr_release_flag = FALSE;
        PC_PWR0 = FALSE;
        PC_PWR1 = FALSE;
        PC_PWR2 = FALSE;
        PC_PWR3 = FALSE;
    }

    /* Look for shutdown request */
    if (shutdown_start_timer_flag) {
        shutdown_timer--;
    }

    if (!shutdown_timer) {
        shutdown_start_timer_flag = FALSE;
        shutdown_timer = SHUTDOWN_TIMER_DEFAULT;
        shutdown_in_progress_flag = TRUE;
        wait_to_halt_flag = TRUE;

        add_job (LCD_CLEAR);

        tjob = get_next_job();
        job_list[tjob].data[16] = 1;
        job_list[tjob].data[17] = 0;
        job_list[tjob].bc = 16;
        sprintf (job_list[tjob].data, "Shutting Down...");
        add_job (LCD_WRITE);

        /* Initiate Shutdown */
        PC_PWR0 = TRUE;
        PC_PWR1 = TRUE;
        PC_PWR2 = TRUE;
        PC_PWR3 = TRUE;
        pc_pwr_release_flag = TRUE;
    }

    /* Heartbeat LED */
    if (hb > 5)
        hb = 0;
    else
        hb++;

    /* Add Job */
    tjob = get_next_job();
    job_list[tjob].bc = 1;

    /* Blink LEDs according to machine state */
    if (system_status.state == MACHINE_RUNNING) {
        job_list[tjob].data[0] = (hb > 3);
        add_job (LCD_LED);
    }
    else if (system_status.state == MACHINE_FAULTED) {
        job_list[tjob].data[0] = (hb % 2);
    }
}
```

---

```

    add_job (LCD_LED);
}
else if (!pcs_ready) {
    job_list[tjob].data[0] = (hb % 3);
    add_job (LCD_LED);
    pc_ready_done_flag = TRUE;
}
else if (pc_ready_done_flag) {
    pc_ready_done_flag = FALSE;
    refresh_title_flag = TRUE;
}
else if (shutdown_in_progress_flag) {
    job_list[tjob].data[0] = (hb % 2);
    add_job (LCD_LED);
}
else {
    job_list[tjob].data[0] = 1;
    add_job (LCD_LED);

    if (refresh_title_flag) {
        refresh_title_flag = FALSE;
        /* Display title message on LCD */
        add_job (LCD_CLEAR);

        tjob = get_next_job();
        job_list[tjob].data[16] = 1;
        job_list[tjob].data[17] = 0;
        job_list[tjob].bc = 16;
        sprintf (job_list[tjob].data, "%s", PM_TITLE1);
        add_job (LCD_WRITE_LOC);

        tjob = get_next_job();
        job_list[tjob].data[16] = 2;
        job_list[tjob].data[17] = 0;
        job_list[tjob].bc = 15;
        sprintf (job_list[tjob].data, "%s", PM_TITLE2);

        add_job (LCD_WRITE_LOC);
    }
}

if (lcd_lock)
    lcd_lock--;

/* Update PC Power state */
if (!PC_SENSE0)
    pc_pwr_status[0].state |= PC_POWER_ON;
else {
    pc_pwr_status[0].state &= ~PC_POWER_ON;
    pc_pwr_status[0].state &= ~PC_POWER_READY;
}

if (!PC_SENSE1)
    pc_pwr_status[1].state |= PC_POWER_ON;
else {
    pc_pwr_status[1].state &= ~PC_POWER_ON;
    pc_pwr_status[1].state &= ~PC_POWER_READY;
}

if (!PC_SENSE2)
    pc_pwr_status[2].state |= PC_POWER_ON;
else {
    pc_pwr_status[2].state &= ~PC_POWER_ON;
    pc_pwr_status[2].state &= ~PC_POWER_READY;
}

if (!PC_SENSE3)
    pc_pwr_status[3].state |= PC_POWER_ON;
else {
    pc_pwr_status[3].state &= ~PC_POWER_ON;
    pc_pwr_status[3].state &= ~PC_POWER_READY;
}

/* Halt Flag */

```

---

```
if (wait_to_halt_flag) {
    if (PC_SENSE0 & PC_SENSE1 & PC_SENSE2 & PC_SENSE3) {
        add_job(HALT_SYSTEM);
    }
}

/* Update PC Ready Signal */
pcs_ready = (pc_pwr_status[0].state & PC_POWER_READY) & (pc_pwr_status[1].state & PC_POWER_READY)
    & (pc_pwr_status[2].state & PC_POWER_READY) & (pc_pwr_status[3].state & PC_POWER_READY);

/* BI2C debounce Lock */
bi2c_lock = FALSE;
}

/* Timer 9 Interrupt Service Routine */
void
__attribute__((interrupt, no_auto_psv)) _T9Interrupt (void)
{
    static unsigned char data[2];
    static unsigned char alternate;
    unsigned char tjob;

    /* Reset Timer 3 for Backlight control and Camera Trigger Control */
    T3CONbits.TON = FALSE;
    TMR3 = 0;

    /* Enable Single Pulse Mode for LED Backlight Output Compare */
    if (!alternate) {
        OC_LED_BL0_CONbits.OCM = OCM_SINGLE_PULSE;
        OC_LED_BL1_CONbits.OCM = OCM_SINGLE_PULSE;
        OC_CAM_TRIG0_CONbits.OCM = OCM_SINGLE_PULSE;
        alternate = 1;
    }
    else {
        OC_LED_BL1_CONbits.OCM = OCM_SINGLE_PULSE;
        OC_LED_BL0_CONbits.OCM = OCM_SINGLE_PULSE;
        OC_CAM_TRIG1_CONbits.OCM = OCM_SINGLE_PULSE;
        alternate = 0;
    }

    T3CONbits.TON = TRUE;

    /* Clear Interrupt Flag */
    IFS3bits.T9IF = DEASSERTED;
}

/* ESTOP Interrupt Service Routine */
void
__attribute__((interrupt, no_auto_psv)) isr_ESTOP (void)
{
    unsigned char tjob;
    unsigned char estop_source = 0xff;

    IFS0bits.INT0IF = DEASSERTED;

    /* Determine Source of EStop */
    if (!ESTOP_SIG0)
        estop_source = 0;
    if (!ESTOP_SIG1)
        estop_source += 1;
    if (!ESTOP_SIG2)
        estop_source += 2;
    if (!ESTOP_SIG3)
        estop_source += 3;

    IO_LED_FL0 = TRUE;
    IO_LED_FL1 = TRUE;

    /* Disable Motor */
    disable_motor();

    /* Turn OFF All PCs */
}
```

---

```

if (!PC_SENSE0) {
    tjob = get_next_job();
    job_list[tjob].data[0] = 1;
    add_job(PC_POWER_SET);
    tjob = get_next_job();
    job_list[tjob].data[0] = 1;
    add_job(PC_POWER_SET);
}
if (!PC_SENSE1) {
    tjob = get_next_job();
    job_list[tjob].data[0] = 2;
    add_job(PC_POWER_SET);
    tjob = get_next_job();
    job_list[tjob].data[0] = 2;
    add_job(PC_POWER_SET);
}
if (!PC_SENSE2) {
    tjob = get_next_job();
    job_list[tjob].data[0] = 4;
    add_job(PC_POWER_SET);
    tjob = get_next_job();
    job_list[tjob].data[0] = 4;
    add_job(PC_POWER_SET);
}
if (!PC_SENSE3) {
    tjob = get_next_job();
    job_list[tjob].data[0] = 8;
    add_job(PC_POWER_SET);
    tjob = get_next_job();
    job_list[tjob].data[0] = 8;
    add_job(PC_POWER_SET);
}

/* Clear LCD */
add_job (LCD_CLEAR);

/* Display E-Stop Message */
tjob = get_next_job();
job_list[tjob].data[16] = 1;
job_list[tjob].data[17] = 0;
job_list[tjob].bc = 12;
sprintf (job_list[tjob].data, "E-Stop Fault");
add_job (LCD_WRITE);

system_status.state = MACHINE_FAULTED;
}

void
__attribute__((interrupt, no_auto_psv)) isr_BI2C (void)
{
    unsigned char tjob;

    INT_BI2C_IF = DEASSERTED;
    if (!bi2c_lock) {
        bi2c_lock = TRUE;
        tjob = get_next_job();
        job_list[tjob].bc = 1;

        job_list[tjob].data[0] = 0xff;
        if (!BI2C_INT0)
            job_list[tjob].data[0] = 0;
        if (!BI2C_INT1)
            job_list[tjob].data[0] = 1;
        if (!BI2C_INT2)
            job_list[tjob].data[0] = 2;
        if (!BI2C_INT3)
            job_list[tjob].data[0] = 3;

        add_job (BI2C_EVENT);
    }
}

```

---

```

int
main (void)
{
    unsigned int i, j, c;
    unsigned char data[2] = {0x01, 0x60};
    unsigned char ret;
    unsigned char tjob;
    char lcd_msg[16];
    int rval;

    /* Configure Oscillator to run at 40MHz */
    /* Fosc = Fin*M/(N1*N2), Fcy = Fosc/2 */
    /* Fosc = 8M*40(2*2) = 80MHz */
    PLLFBD = 38;          /* M = 40 */
    CLKDIVbits.PLLPOST = 0; /* N1 = 2 */
    CLKDIVbits.PLLPRE = 0; /* N2 = 2 */
    OSCTUN = 0;           /* Tune FRC oscillator if FRC is used */

    /* Wait for PLL to lock */
    while (OSCCONbits.LOCK != 1)
        ;

    /* Initialize I2C2 Module as Master */
    init_i2c2 ();

    /* Initialize System I/O, Timers, OCs, etc */
    init_sys();

    /* Initialize Panel I/O Expanders */
    io_side_panel.i2c_addr = I2C_ADDR_SIDE_PANEL;
    io_front_panel.i2c_addr = I2C_ADDR_FRONT_PANEL;

    io_side_panel.bi2c_channel = BI2C_CHANNEL_IO_SIDE_PANEL;
    io_front_panel.bi2c_channel = BI2C_CHANNEL_IO_FRONT_PANEL;

    /* Side Panel */
    io_side_panel.outp = 0xff;
    io_side_panel.outp &= ~SP_LED_bit;
    i2c_io_exp_write (&io_side_panel);

    /* Front Panel */
    io_front_panel.outp = 0xff;
    io_front_panel.outp &= ~FP_LED_bit;
    i2c_io_exp_write (&io_front_panel);

    /* Blink HMI LEDs */
    for (c = 0; c < 6; c++) {
        if (io_side_panel.outp & SP_LED_bit)
            io_side_panel.outp &= ~SP_LED_bit;
        else
            io_side_panel.outp |= SP_LED_bit;

        if (io_front_panel.outp & FP_LED_bit)
            io_front_panel.outp &= ~FP_LED_bit;
        else
            io_front_panel.outp |= FP_LED_bit;

        i2c_io_exp_write (&io_side_panel);
        i2c_io_exp_write (&io_front_panel);
        Delay (Delay_1S_Cnt/3);
    }

    /* Turn OFF LEDs */
    io_side_panel.outp |= SP_LED_bit;
    io_front_panel.outp |= FP_LED_bit;

    i2c_io_exp_write (&io_side_panel);
    i2c_io_exp_write (&io_front_panel);

    /* Set PC Power defaults */
    pc_pwr_status[0].state = 0;
    pc_pwr_status[1].state = 0;
    pc_pwr_status[2].state = 0;
    pc_pwr_status[3].state = 0;

```

```
/* Initialize LCD */
lcd_init ();
lcd_print ("Initializing...", 15);

/* Initialize UART2 Module */
init_uart2 ();

/* Initialize I2C Slave Module */
init_i2c_slave ();

/* Initialize GPIO - Direction and Initial State */
init_io();

/* Turn ON main air supply - for calibration*/
PNEU_MAIN = DEASSERTED;

/* Calibrate Arm Position */
calibrate_sys();

/* Turn OFF Main Air */
PNEU_MAIN = ASSERTED;

/* Re-initialize */
init_sys();

/* Enable Power on all PCs if not already on */
pcs_ready = FALSE;
pc_ready_done_flag = FALSE;
if (PC_SENSE0)
    set_pc_pwr (0);
if (PC_SENSE1)
    set_pc_pwr (1);
if (PC_SENSE2)
    set_pc_pwr (2);
if (PC_SENSE3)
    set_pc_pwr (3);

/* Initialize Job Dispatcher */
curr_job = 0;
last_job = 0;

/* Reset Pass / Fail Array */
cap_count = 0;
for (j = 0; j < 4; j++)
    for (i = 0; i < 16; i++)
        capsule_passfail[j][i] = 0;

/* Set bus switch to quadrant 1 (default for now) */
I2C_BUS_SW_A0 = DEASSERTED;
I2C_BUS_SW_A1 = DEASSERTED;

/* System Check */
// init_check();

/* Turn ON LEDs */
io_side_panel.outp &= ~SP_LED_bit;
io_front_panel.outp &= ~FP_LED_bit;

i2c_io_exp_write (&io_side_panel);
i2c_io_exp_write (&io_front_panel);

/* Read data from I2C I/O Boards */
i2c_io_exp_read (&io_side_panel);
i2c_io_exp_read (&io_front_panel);

/* Display message waiting on PCs */
lcd_clear();
lcd_print("Waiting for PC", 14);
lcd_cursor_to(2, 0);
lcd_print("ready signal...", 15);

/* Current state of front panel switch */
i2c_io_exp_read (&io_front_panel);
```



```
prev_fp_sw_state = (~io_front_panel.inp) & FRONT_PANEL_SW_bit;

/* The Main Loop */
while (1) {
    if (curr_job != last_job) {
        switch (job_list[get_next_curr_job()].job_id) {
            case MOTOR_PULSE:
                motor_step_hdlr ();
                complete_job ();
                break;

            case LCD_WRITE:
                lcd_cursor_to (job_list[get_next_curr_job()].data[16],0);
                lcd_print (job_list[get_next_curr_job()].data, job_list[get_next_curr_job()].bc);
                complete_job();
                break;

            case LCD_WRITE_LOC:
                lcd_cursor_to (job_list[get_next_curr_job()].data[16], job_list[get_next_curr_job()].data[17]);
                ;
                lcd_print (job_list[get_next_curr_job()].data, job_list[get_next_curr_job()].bc);
                complete_job();
                break;

            case LCD_CLEAR:
                lcd_clear ();
                complete_job();
                break;

            case LCD_LED:
                lcd_set_bl (job_list[get_next_curr_job()].data[0]);
                complete_job();
                break;

            case BI2C_EVENT:
                /* Determine which BI2C line was triggered*/
                switch (job_list[get_next_curr_job()].data[0]) {
                    case 0:
                        break;

                    case 1:
                        /* Front Panel */
                        /* Read I/O Inputs */
                        i2c_io_exp_read (&io_front_panel);

                        fp_sw_state = (~io_front_panel.inp) & FRONT_PANEL_SW_bit;

                        /* Check I/O Inputs */
                        /* Check PB1 Pressed */
                        if ( (~io_front_panel.inp) & FRONT_PANEL_PB1_bit) {
                            if (system_status.state != MACHINE_RUNNING)
                                enable_motor();
                        }
                        /* Check PB2 Pressed */
                        else if ( (~io_front_panel.inp) & FRONT_PANEL_PB2_bit) {
                            if (system_status.state == MACHINE_RUNNING)
                                disable_motor();
                        }
                        /* Check SW1 change of state */
                        if (fp_sw_state != prev_fp_sw_state) {
                            prev_fp_sw_state = fp_sw_state;
                            if ( (~io_front_panel.inp) & FRONT_PANEL_SW_bit) {
                                /* Enable Debug Mode */
                                if (!shutdown_in_progress_flag) {
                                    add_job (LCD_CLEAR);
                                    tjob = get_next_job();
                                    job_list[tjob].data[16] = 1;
                                    job_list[tjob].data[17] = 0;
                                    job_list[tjob].bc = 16;
                                    sprintf (job_list[tjob].data, "HOLD TO SHUTDOWN");
                                    add_job (LCD_WRITE);
                                    tjob = get_next_job();
                                    job_list[tjob].data[16] = 2;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```

        job_list[tjob].data[17] = 0;
        job_list[tjob].bc = 15;
        sprintf (job_list[tjob].data, " *****");
        add_job (LCD_WRITE);
        shutdown_start_timer_flag = TRUE;
    }
}
else {
    if (!shutdown_in_progress_flag) {
        add_job (LCD_CLEAR);
        tjob = get_next_job();
        job_list[tjob].data[16] = 1;
        job_list[tjob].data[17] = 0;
        job_list[tjob].bc = 13;
        sprintf (job_list[tjob].data, "      SHUTDOWN");
        add_job (LCD_WRITE);
        tjob = get_next_job();
        job_list[tjob].data[16] = 2;
        job_list[tjob].data[17] = 0;
        job_list[tjob].bc = 14;
        sprintf (job_list[tjob].data, "    CANCELLED!");
        add_job (LCD_WRITE);
        /* Refresh LCD Title Timer */
        T7CONbits.TON = TRUE;
        shutdown_start_timer_flag = FALSE;
        shutdown_timer = SHUTDOWN_TIMER_DEFAULT;
    }
}
break;

case 2:
    /* Side Panel */
    /* Read I/O Inputs */
    i2c_io_exp_read (&io_side_panel);

    /* Check I/O Inputs */
    /* Check PB Pressed */
    if ( (~io_side_panel.inp) & SP_SW_bit) {
        /* Turn ON or OFF AUX air supply */
        io_side_panel.outp &= ~SP_AUX_AIR_bit;
        //io_side_panel.outp &= ~SP_BUZZ_bit;
    }
    else {
        io_side_panel.outp |= SP_AUX_AIR_bit;
        //io_side_panel.outp |= SP_BUZZ_bit;
    }

    break;

case 3:

    break;

default:

    break;
}
complete_job();

break;

case PC_POWER_SET:
    /* Toggle power SW to specified motherboards for 400ms */
    if (job_list[get_next_curr_job()].data[0] & 0x01)
        PC_PWR0 = TRUE;
    if (job_list[get_next_curr_job()].data[0] & 0x02)
        PC_PWR1 = TRUE;
    if (job_list[get_next_curr_job()].data[0] & 0x04)
        PC_PWR2 = TRUE;
    if (job_list[get_next_curr_job()].data[0] & 0x08)
        PC_PWR3 = TRUE;

    pc_pwr_release_flag = TRUE;

```

```

        complete_job();
        break;

    case PC_POWER_RELEASE:
        PC_PWR0 = FALSE;
        PC_PWR1 = FALSE;
        PC_PWR2 = FALSE;
        PC_PWR3 = FALSE;

        complete_job();
        break;

    case HALT_SYSTEM:
        lcd_cursor_to (1, 0);
        lcd_print (" Safe to Power", 14);
        lcd_cursor_to (2, 0);
        lcd_print ("   Down Now!", 12);
        while(1);
        break;

    default:
        break;
}
}
}

return 0;
}

```

```

/* System Startup Check Routine (Debug)
 * Flash some lights, switch air, etc.
 * Used to verify system connections
 */

```

```

void
init_check (void)
{
    unsigned char i;
    Delay (Delay_1S_Cnt);

    Delay (Delay_1S_Cnt/8);
    PNEU_MAIN = DEASSERTED;
    Delay (Delay_1S_Cnt/8);
    PNEU_MAIN = ASSERTED;

    Delay (Delay_1S_Cnt/8);
    PNEU_UPPER = DEASSERTED;
    Delay (Delay_1S_Cnt/8);
    PNEU_UPPER = ASSERTED;

    Delay (Delay_1S_Cnt/8);
    PNEU_ACCEPT0 = DEASSERTED;
    Delay (Delay_1S_Cnt/8);
    PNEU_ACCEPT0 = ASSERTED;

    Delay (Delay_1S_Cnt/8);
    PNEU_ACCEPT1 = DEASSERTED;
    Delay (Delay_1S_Cnt/8);
    PNEU_ACCEPT1 = ASSERTED;

    Delay (Delay_1S_Cnt/8);
    PNEU_ACCEPT2 = DEASSERTED;
    Delay (Delay_1S_Cnt/8);
    PNEU_ACCEPT2 = ASSERTED;

    Delay (Delay_1S_Cnt/8);
    PNEU_ACCEPT3 = DEASSERTED;
    Delay (Delay_1S_Cnt/8);
    PNEU_ACCEPT3 = ASSERTED;

    Delay (Delay_1S_Cnt);
    for (i = 0; i < 3; i++) {
        PNEU_MAIN = DEASSERTED;
    }
}

```

```
PNEU_UPPER = DEASSERTED;
PNEU_ACCEPT0 = DEASSERTED;
PNEU_ACCEPT1 = DEASSERTED;
PNEU_ACCEPT2 = DEASSERTED;
PNEU_ACCEPT3 = DEASSERTED;

Delay (Delay_1S_Cnt/8);

PNEU_MAIN = ASSERTED;
PNEU_UPPER = ASSERTED;
PNEU_ACCEPT0 = ASSERTED;
PNEU_ACCEPT1 = ASSERTED;
PNEU_ACCEPT2 = ASSERTED;
PNEU_ACCEPT3 = ASSERTED;

Delay (Delay_1S_Cnt/8);
}

/* Cycle Lighting */
for (i = 0; i < 4; i++) {
    IO_LED_FL1 = DEASSERTED;
    IO_LED_BL0 = ASSERTED;
    Delay (Delay_1S_Cnt/8);

    IO_LED_BL0 = DEASSERTED;
    IO_LED_BL1 = ASSERTED;
    Delay (Delay_1S_Cnt/8);

    IO_LED_BL1 = DEASSERTED;
    IO_LED_FL0 = ASSERTED;
    Delay (Delay_1S_Cnt/8);

    IO_LED_FL0 = DEASSERTED;
    IO_LED_FL1 = ASSERTED;
    Delay (Delay_1S_Cnt/8);
}

#if 1
for (i = 0; i < 5; i++) {
    IO_CAM_TRIG0 = ASSERTED;
    IO_CAM_TRIG1 = ASSERTED;
    Delay (Delay_1S_Cnt);
    IO_CAM_TRIG0 = DEASSERTED;
    IO_CAM_TRIG1 = DEASSERTED;
    Delay (Delay_1S_Cnt);
}
#endif
}

void
init_io (void)
{
    /* Turn OFF all pneumatic valves */
    PNEU_MAIN = ASSERTED;
    PNEU_UPPER = ASSERTED;
    PNEU_ACCEPT0 = ASSERTED;
    PNEU_ACCEPT1 = ASSERTED;
    PNEU_ACCEPT2 = ASSERTED;
    PNEU_ACCEPT3 = ASSERTED;

    /* Set Tristate Mode of Pneumatics I/O */
    PNEU_MAIN_TRIS = FALSE;
    PNEU_UPPER_TRIS = FALSE;
    PNEU_ACCEPT0_TRIS = FALSE;
    PNEU_ACCEPT1_TRIS = FALSE;
    PNEU_ACCEPT2_TRIS = FALSE;
    PNEU_ACCEPT3_TRIS = FALSE;

    /* Turn OFF all LED back/front light controls */
    IO_LED_BL0 = DEASSERTED;
    IO_LED_BL1 = DEASSERTED;
    IO_LED_FL0 = DEASSERTED;
    IO_LED_FL1 = DEASSERTED;
}
```

```
/* Set Tristate Mode of LED I/O */
IO_LED_BL0_TRIS = FALSE;
IO_LED_BL1_TRIS = FALSE;
IO_LED_FL0_TRIS = FALSE;
IO_LED_FL1_TRIS = FALSE;

/* Turn OFF all Camera Trigger Signals */
IO_CAM_TRIG0 = DEASSERTED;
IO_CAM_TRIG1 = DEASSERTED;

/* Set Tristate mode of Camera Triggers */
IO_CAM_TRIG0_TRIS = FALSE;
IO_CAM_TRIG1_TRIS = FALSE;

/* Set Tristate mode for I2C Bus Switch */
I2C_BUS_SW_A0_TRIS = FALSE;
I2C_BUS_SW_A1_TRIS = FALSE;

/* Set Tristate mode for ESTOP Signals */
ESTOP_SIG0_TRIS = TRUE;
ESTOP_SIG1_TRIS = TRUE;
ESTOP_SIG2_TRIS = TRUE;
ESTOP_SIG3_TRIS = TRUE;
/* Set Tristate mode for ESTOP Interrupt (INT0) */
IO_ESTOP_INT_TRIS = TRUE;

/* Set Tristate mode for BI2C INT Signals */
BI2C_INT0_TRIS = TRUE;
BI2C_INT1_TRIS = TRUE;
BI2C_INT2_TRIS = TRUE;
BI2C_INT3_TRIS = TRUE;

/* Set Initial Value for PC Soft Power Control Outputs */
/* Set Tristate mode for PC Soft Power Control */
PC_PWR0 = FALSE;
PC_PWR1 = FALSE;
PC_PWR2 = FALSE;
PC_PWR3 = FALSE;

PC_PWR0_TRIS = FALSE;
PC_PWR1_TRIS = FALSE;
PC_PWR2_TRIS = FALSE;
PC_PWR3_TRIS = FALSE;

/* Set Tristate mode for PC Soft Power Sense (inputs) */
PC_SENSE0_TRIS = TRUE;
PC_SENSE1_TRIS = TRUE;
PC_SENSE2_TRIS = TRUE;
PC_SENSE3_TRIS = TRUE;

PC_SENSE0 = TRUE;
PC_SENSE1 = TRUE;
PC_SENSE2 = TRUE;
PC_SENSE3 = TRUE;
}

/* Perform System Initialization
 * - Read current system parameters
 * - Set Initial Outputs for system Parameters
 * - Initial System Fault Check
 */
void
init_sys (void)
{
    char i;

    /* Setup some initial system parameters */
    mtr_pulse_width = MTR_PULSE_WIDTH;
    cam_trig0_pulse_width = CAM_TRIG0_PULSE_WIDTH;
    cam_trig1_pulse_width = CAM_TRIG1_PULSE_WIDTH;
    bl0_pulse_width = BL0_PULSE_WIDTH;
    bl1_pulse_width = BL1_PULSE_WIDTH;
```

---

```

fl0_pulse_width = FL0_PULSE_WIDTH;
fl1_pulse_width = FL1_PULSE_WIDTH;

cam0_pulse_position = CAM0_PULSE_POSITION_DEFAULT;
cam1_pulse_position = CAM1_PULSE_POSITION_DEFAULT;
accept_on_pulse_position = ACCEPT_ON_PULSE_POSITION_DEFAULT;
accept_off_pulse_position = ACCEPT_OFF_PULSE_POSITION_DEFAULT;

mtr_running_freq = MTR_DEFAULT_FREQ;

/* Set Analog Pins to Digital */
AD1PCFGL = 0xffff;
AD1PCFGH = 0xffff;
AD2PCFGL = 0xffff;

/* Trigger Compensation */
if (motor_speed_target > 300) {
    trigger_comp = (int) (0.09 * (double) motor_speed_target) - 34;
}
else {
    trigger_comp = 0;
}

/* Reset counters */
for (i = 0; i < 4; i++) {
    capsule_good_count[i] = 0;
    capsule_bad_count[i] = 0;
    capsule_total_count[i] = 0;
}

/* Configure Timer2 for Output Capture */
T2CONbits.T32 = 0;
PR2 = (unsigned int) (((double) Fcy) / (double) motor_speed) - 1.0;
PR2 = PR2 / T2PF;
T2CONbits.TON = 1;
T2CONbits.TCKPS = T2TCKPS;

/* Configure Timer3 for Output Compare */
PR3 = 0xffff;
T3CONbits.TCKPS = 2;
T3CONbits.TON = TRUE;

/* Configure Timer4 for I2C Bus Switch at about 12 kHz*/
TMR_BUS_SWITCH_CONbits.TON = TRUE;
TMR_BUS_SWITCH_PR = I2C_BUS_SW_PR;
/* Div by 8 PS */
TMR_BUS_SWITCH_CONbits.TCKPS = 3;
TMR_BUS_SWITCH_IF = FALSE; /* Clear Interrupt Flag */
TMR_BUS_SWITCH_IE = TRUE; /* Enable Interrupt */

i2c_bus_lock = FALSE;
i2c_bus_lock_count = 0;
i2c_bs_active_bus = 0;

/* Enable All Input Captures for Proximity Sensors */
IC_PS0_CONbits.ICM = ICM_FALL_EDGE;
IC_PS0_CONbits.ICTMR = ICTMR_TIMER2;
IC_PS0_IE = TRUE;

IC_PS1_CONbits.ICM = ICM_FALL_EDGE;
IC_PS1_CONbits.ICTMR = ICTMR_TIMER2;
IC_PS1_IE = TRUE;

IC_PS2_CONbits.ICM = ICM_FALL_EDGE;
IC_PS2_CONbits.ICTMR = ICTMR_TIMER2;
IC_PS2_IE = TRUE;

IC_PS3_CONbits.ICM = ICM_FALL_EDGE;
IC_PS3_CONbits.ICTMR = ICTMR_TIMER2;
IC_PS3_IE = TRUE;

/* Enable Output Compare to generate LED Backlight Pulse */
OC_LED_BL0_CONbits.OCM = OCM_DISABLED;
OC_LED_BL0_CONbits.OCTSEL = OCTSEL_TIMER3;

```

---

---

```

OC_LED_BL0_R = 0;
OC_LED_BL0_RS = BL0_PULSE_WIDTH;

OC_LED_BL1_CONbits.OCM = OCM_DISABLED;
OC_LED_BL1_CONbits.OCTSEL = OCTSEL_TIMER3;
OC_LED_BL1_R = 0;
OC_LED_BL1_RS = BL1_PULSE_WIDTH;

OC_CAM_TRIG0_CONbits.OCM = OCM_DISABLED;
OC_CAM_TRIG0_CONbits.OCTSEL = OCTSEL_TIMER3;
OC_CAM_TRIG0_R = 0;
OC_CAM_TRIG0_RS = CAM_TRIG0_PULSE_WIDTH;

OC_CAM_TRIG1_CONbits.OCM = OCM_DISABLED;
OC_CAM_TRIG1_CONbits.OCTSEL = OCTSEL_TIMER3;
OC_CAM_TRIG1_R = 0;
OC_CAM_TRIG1_RS = CAM_TRIG1_PULSE_WIDTH;

/* Enable Output Compare to generate PWM for motor control */
/* Initially OFF */
OC_MTR_CTRL_CONbits.OCM = OCM_DISABLED;
/* Use Timer2 */
OC_MTR_CTRL_CONbits.OCTSEL = OCTSEL_TIMER2;

OC_MTR_CTRL_R = 0;
OC_MTR_CTRL_RS = MTR_PULSE_WIDTH;
OC_MTR_CTRL_IF = DEASSERTED;
OC_MTR_CTRL_IE = TRUE;

/* Configure Timer5 for Ramp Up/Down */
PR5 = 200;
T5CONbits.TON = FALSE;
T5CONbits.TCKPS = 3;
IFS1bits.T5IF = FALSE;
IEC1bits.T5IE = TRUE;

ramp_mode = RAMP_NONE;
motor_speed = MOTOR_MIN_FREQ;
motor_speed_target = MOTOR_MIN_FREQ;

/* Start Heartbeat Timer */
PR8 = 32535;
T8CONbits.TCKPS = 3;
T8CONbits.TON = TRUE;
IEC3bits.T8IE = TRUE;

/* LCD Timer */
PR7 = 65535;
T7CONbits.TCKPS = 3;
T7CONbits.TON = FALSE;
IFS3bits.T7IF = FALSE;
IEC3bits.T7IE = TRUE;

/* Start the Still Image Capture Timer */
PR9 = 30000;
T9CONbits.TCKPS = 3;
IEC3bits.T9IE = TRUE;

/* Setup Interrupt0 for E-Stop Interrupt */
INT_ESTOP_CONbits.INT0EP = 1; /* Interrupt on negative edge */
INT_ESTOP_IF = DEASSERTED; /* Reset interrupt flag */
INT_ESTOP_IE = TRUE; /* Enable ESTOP interrupt */

/* Setup Interrupt1 for BI2C Interrupt */
INT_BI2C_CONbits.INT1EP = 1; /* Interrupt on positive edge */
INT_BI2C_IF = DEASSERTED;
INT_BI2C_IE = TRUE;
}

/* Check Motor Step Position. For given position trigger
 * system functions
 */
static inline void motor_step_hdlr (void)

```

---

```
{
    unsigned char i;
    unsigned char tjob;
    unsigned int pc_test;

    pc_test = pulse_counter + trigger_comp;

    if (pc_test >= PULSES_PER HOLDER) {
        pc_test = (pulse_counter + trigger_comp) - PULSES_PER HOLDER;
    }

    if (pulse_counter == accept_on_pulse_position) {
        /* Enable Pneu Act - Station 0*/
        if (cap_count < 1) {
            if (capsule_passfail[0][cap_count + 15] == 2) {
                PNEU_ACCEPT0 = DEASSERTED;
                capsule_good_count[0]++;
            }
            else if (capsule_passfail[0][cap_count + 15] == 1) {
                capsule_bad_count[0]++;
            }
        }
        else {
            if (capsule_passfail[0][cap_count - 1] == 2) {
                PNEU_ACCEPT0 = DEASSERTED;
                capsule_good_count[0]++;
            }
            else if (capsule_passfail[0][cap_count - 1] == 1) {
                capsule_bad_count[0]++;
            }
        }
    }

    /* Enable Pneu Act - Station 1*/
    if (cap_count < 1) {
        if (capsule_passfail[1][cap_count + 15] == 2) {
            PNEU_ACCEPT1 = DEASSERTED;
            capsule_good_count[1]++;
        }
        else if (capsule_passfail[1][cap_count + 15] == 1) {
            capsule_bad_count[1]++;
        }
    }
    else {
        if (capsule_passfail[1][cap_count - 1] == 2) {
            PNEU_ACCEPT1 = DEASSERTED;
            capsule_good_count[1]++;
        }
        else if (capsule_passfail[1][cap_count - 1] == 1) {
            capsule_bad_count[1]++;
        }
    }
}

/* Enable Pneu Act - Station 2*/
if (cap_count < 1) {
    if (capsule_passfail[2][cap_count + 15] == 2) {
        PNEU_ACCEPT2 = DEASSERTED;
        capsule_good_count[2]++;
    }
    else if (capsule_passfail[2][cap_count + 15] == 1) {
        capsule_bad_count[2]++;
    }
}
else {
    if (capsule_passfail[2][cap_count - 1] == 2) {
        PNEU_ACCEPT2 = DEASSERTED;
        capsule_good_count[2]++;
    }
    else if (capsule_passfail[2][cap_count - 1] == 1) {
        capsule_bad_count[2]++;
    }
}

/* Enable Pneu Act - Station 3*/
if (cap_count < 1) {
```



```
    if (capsule_passfail[3][cap_count + 15] == 2) {
        PNEU_ACCEPT3 = DEASSERTED;
        capsule_good_count[3]++;
    }
    else if (capsule_passfail[3][cap_count + 15] == 1) {
        capsule_bad_count[3]++;
    }
}
else {
    if (capsule_passfail[3][cap_count - 1] == 2) {
        PNEU_ACCEPT3 = DEASSERTED;
        capsule_good_count[3]++;
    }
    else if (capsule_passfail[3][cap_count - 1] == 1) {
        capsule_bad_count[3]++;
    }
}
}

if (pulse_counter == accept_off_pulse_position) {
    PNEU_ACCEPT0 = ASSERTED;
    PNEU_ACCEPT1 = ASSERTED;
    PNEU_ACCEPT2 = ASSERTED;
    PNEU_ACCEPT3 = ASSERTED;
}

if (pc_test == cam0_pulse_position) {
    /* Enable Trigger0 and BL0 strobe */
    /* Reset Timer 3 for Backlight control and Camera Trigger Control */
    T3CONbits.TON = FALSE;
    TMR3 = 0;

    /* Enable Single Pulse Mode for LED Backlight Output Compare */
    OC_LED_BL0_CONbits.OCM = OCM_SINGLE_PULSE;
    OC_CAM_TRIG0_CONbits.OCM = OCM_SINGLE_PULSE;

    /* Enable Timer */
    T3CONbits.TON = TRUE;
}

pc_test = pulse_counter + trigger_comp + extra_comp;

if (pc_test >= PULSES_PER HOLDER) {
    pc_test = (pulse_counter + trigger_comp + extra_comp) - PULSES_PER HOLDER;
}

if (pc_test == cam1_pulse_position) {
    /* Enable Trigger1 and BL1 strobe */
    /* Reset Timer 3 for Backlight control and Camera Trigger Control */
    T3CONbits.TON = FALSE;
    TMR3 = 0;

    /* Enable Single Pulse Mode for LED Backlight Output Compare */
    OC_LED_BL1_CONbits.OCM = OCM_SINGLE_PULSE;
    OC_CAM_TRIG1_CONbits.OCM = OCM_SINGLE_PULSE;

    /* Enable Timer */
    T3CONbits.TON = TRUE;
}

if (pc_test == 470) {
    if (one_step) {
        /* Disable Motor */
        OC_MTR_CTRL_CONbits.OCM = OCM_DISABLED;
        /* Flash BL0 and send trigger */
        T3CONbits.TON = FALSE;
        TMR3 = 0;

        /* Enable Single Pulse Mode for LED Backlight Output Compare */
        OC_LED_BL0_CONbits.OCM = OCM_SINGLE_PULSE;
        OC_CAM_TRIG0_CONbits.OCM = OCM_SINGLE_PULSE;

        T3CONbits.TON = TRUE;
        PNEU_MAIN = ASSERTED;
    }
}
```

```
        one_step = 0;
    }
}

void
i2c2_request_hdlr (unsigned char i_job)
{
    unsigned char ret;
    /* Data formatted as follows
       data[0]      - I2C Slave Address
       data[1]      - Read/Write Direction
       data[2-15]   - I2C transfer buffer
       bc           - holds data size (including byte 0 and 1)
    */

    /* Check for Read / Write */
    if (!job_list[i_job].data[1]) {
        ret = i2c2_write (job_list[i_job].data[0], *(job_list[i_job].data + 2), job_list[i_job].bc - 2);
    }
}

/* Handle incoming command from the UART2 module
*/
void
uart2_hdlr (void)
{
    unsigned char i = 0, err = 0, c;
    unsigned char cmd;
    unsigned char dat[2];
    unsigned char tjob;
    unsigned char tmp;
    unsigned long ltmp;

    /* Perform Error Check on data */
    for (c = uart_rx.rd; c < uart_rx.rd + 3; c++) {
        if ((uart_rx_buf[c] ^ uart_rx_buf[c+3]) != 0xff)
            err++;
    }

    /* Respond accordingly */
    if (err) {
        U2TXREG = 0x15;          /* NACK */

        /* Reset buffer pointers equal */
        uart_rx.rd = uart_rx.wr = 0;
        uart_cmd_flag = 0;
        return;
    }
    else {
        U2TXREG = 0x06;          /* ACK */
        /* Wait for ACK msg to be sent */
        while (!U2STAbits.TRMT) ;
    }

    cmd = uart_rx_buf[uart_rx.rd];

    uart_rx.rd++;
    if (uart_rx.rd >= MAX_UART_RX_BUF)
        uart_rx.rd = 0;

    for (; i < 2; i++) {
        dat[i] = uart_rx_buf[uart_rx.rd];

        uart_rx.rd++;
        if (uart_rx.rd >= MAX_UART_RX_BUF)
            uart_rx.rd = 0;
    }

    switch (cmd) {
        /* Set Commands */
        case UARTCMD_SET_MOTOR_STAT:
```

---

```
switch (dat[0]) {
    case 0xff:
        /* Enable Motor */
        enable_motor();
        break;

    default:
        /* Disable Motor */
        disable_motor ();
        break;
}
break;

case UARTCMD_SET_MOTOR_FREQ:
    motor_speed_target = dat[1];
    motor_speed_target |= dat[0] << 8;

    if (motor_speed_target < MOTOR_MIN_FREQ)
        motor_speed_target = MOTOR_MIN_FREQ;

    mtr_running_freq = motor_speed_target;

    /* If Running, ramp appropriately */
    if (OC_MTR_CTRL_CONbits.OCM) {
        if (motor_speed_target > motor_speed) {
            ramp_mode = RAMP_UP;
            T5CONbits.TON = TRUE;
        }
        else if (motor_speed_target == motor_speed) {
            /* Do Nothing */
        }
        else {
            ramp_mode = RAMP_DOWN;
            T5CONbits.TON = TRUE;
        }
    }

    break;

case UARTCMD_SET_BL0_WIDTH:
    /* Set BL0 Pulse Width */
    bl0_pulse_width = dat[1];
    bl0_pulse_width |= (dat[0] << 8);
    OC_LED_BL0_RS = bl0_pulse_width;
    break;

case UARTCMD_SET_BL1_WIDTH:
    /* Set BL1 Pulse Width */
    bl1_pulse_width = dat[1];
    bl1_pulse_width |= (dat[0] << 8);
    OC_LED_BL1_RS = bl1_pulse_width;
    break;

case UARTCMD_SET_FL0_WIDTH:
    /* Set FL0 Pulse Width */
    fl0_pulse_width = dat[1];
    fl0_pulse_width |= (dat[0] << 8);
    OC_LED_FL0_RS = fl0_pulse_width;
    break;

case UARTCMD_SET_FL1_WIDTH:
    /* Set FL1 Pulse Width */
    fl1_pulse_width = dat[1];
    fl1_pulse_width |= (dat[0] << 8);
    OC_LED_FL1_RS = fl1_pulse_width;
    break;

case UARTCMD_RESET_COUNTERS:
    /* Reset capsule counters */
    for (i = 0; i < 4; i++) {
        capsule_good_count[i] = 0;
        capsule_bad_count[i] = 0;
    }
    break;
```

```
case UARTCMD_DEBUG_MODE:
    /* Set in debug mode */
    switch (dat[0]) {
        case 0xff:
            debug_mode (1);
            break;

        default:
            debug_mode (0);
            break;
    }

    break;

case UARTCMD_ONE_STEP:
    /* Set in debug mode */
    if (dat[0] == 0xff) {
        /* Display Message on LCD */
        add_job (LCD_CLEAR);
        tjob = get_next_job();
        job_list[tjob].data[16] = 1;
        job_list[tjob].data[17] = 0;
        job_list[tjob].bc = 14;
        sprintf (job_list[tjob].data, " One-Step Mode");
        add_job (LCD_WRITE);

        one_step = 1;
        OC_MTR_CTRL_CONbits.OCM = OCM_CONT_PULSE;

        PNEU_MAIN = DEASSERTED;
    }

    break;

case UARTCMD_POWER_ON_PCS:
    tjob = get_next_job();
    job_list[tjob].data[0] = dat[0];
    add_job(PC_POWER_SET);
    break;

case UARTCMD_SET_CAM0_PULSE_POS:
    cam0_pulse_position = dat[1];
    cam0_pulse_position |= (dat[0] << 8);
    break;

case UARTCMD_SET_CAM1_PULSE_POS:
    cam1_pulse_position = dat[1];
    cam1_pulse_position |= (dat[0] << 8);
    break;

case UARTCMD_SET_ACCEPT_ON_PULSE_POS:
    accept_on_pulse_position = dat[1];
    accept_on_pulse_position |= (dat[0] << 8);
    break;

case UARTCMD_SET_ACCEPT_OFF_PULSE_POS:
    accept_off_pulse_position = dat[1];
    accept_off_pulse_position |= (dat[0] << 8);
    break;

    /* Get Commands */
case UARTCMD_GET_MOTOR_STAT:
    /* Respond with Motor Status - on / off */
    U2TXREG = (OC_MTR_CTRL_CONbits.OCM) ? 0xff : 0x00;
    break;

case UARTCMD_GET_MOTOR_FREQ:
    /* Respond with motor pulse freq */
    U2TXREG = (mtr_running_freq >> 8) & 0x00ff;
    U2TXREG = mtr_running_freq & 0x00ff;
    break;

case UARTCMD_GET_BL0_WIDTH:
```

```
/* Respond with BL0 pulse width */
U2TXREG = (bl0_pulse_width >> 8) & 0xff;
U2TXREG = bl0_pulse_width & 0xff;
break;

case UARTCMD_GET_BL1_WIDTH:
/* Respond with BL1 pulse width */
U2TXREG = (bl1_pulse_width >> 8) & 0xff;
U2TXREG = bl1_pulse_width & 0xff;
break;

case UARTCMD_GET_FL0_WIDTH:
/* Respond with FL0 pulse width */
U2TXREG = (fl0_pulse_width >> 8) & 0xff;
U2TXREG = fl0_pulse_width & 0xff;
break;

case UARTCMD_GET_FL1_WIDTH:
/* Respond with FL1 pulse width */
U2TXREG = (fl1_pulse_width >> 8) & 0xff;
U2TXREG = fl1_pulse_width & 0xff;
break;

case UARTCMD_GOOD_COUNT:
/* Respond with Good capsule counter value - 4 bytes */
switch (dat[0]) {
case 1:
ltmp = capsule_good_count[0];
U2TXREG = (ltmp >> 24) & 0xff;
while (U2STAbits.UTXBF);
U2TXREG = (ltmp >> 16) & 0xff;
while (U2STAbits.UTXBF);
U2TXREG = (ltmp >> 8) & 0xff;
while (U2STAbits.UTXBF);
U2TXREG = ltmp & 0xff;
break;
case 2:
ltmp = capsule_good_count[1];
U2TXREG = (ltmp >> 24) & 0xff;
while (U2STAbits.UTXBF);
U2TXREG = (ltmp >> 16) & 0xff;
while (U2STAbits.UTXBF);
U2TXREG = (ltmp >> 8) & 0xff;
while (U2STAbits.UTXBF);
U2TXREG = ltmp & 0xff;
break;
case 3:
ltmp = capsule_good_count[2];
U2TXREG = (ltmp >> 24) & 0xff;
while (U2STAbits.UTXBF);
U2TXREG = (ltmp >> 16) & 0xff;
while (U2STAbits.UTXBF);
U2TXREG = (ltmp >> 8) & 0xff;
while (U2STAbits.UTXBF);
U2TXREG = ltmp & 0xff;
break;
case 4:
ltmp = capsule_good_count[3];
U2TXREG = (ltmp >> 24) & 0xff;
while (U2STAbits.UTXBF);
U2TXREG = (ltmp >> 16) & 0xff;
while (U2STAbits.UTXBF);
U2TXREG = (ltmp >> 8) & 0xff;
while (U2STAbits.UTXBF);
U2TXREG = ltmp & 0xff;
break;

default:
break;
}
break;

case UARTCMD_BAD_COUNT:
/* Respond with Bad capsule counter value - 4 bytes */
```

---

```

switch (dat[0]) {
case 1:
    ltmp = capsule_bad_count[0];
    U2TXREG = (ltmp >> 24) & 0xff;
    while (U2STAbits.UTXBF);
    U2TXREG = (ltmp >> 16) & 0xff;
    while (U2STAbits.UTXBF);
    U2TXREG = (ltmp >> 8) & 0xff;
    while (U2STAbits.UTXBF);
    U2TXREG = ltmp & 0xff;
    break;
case 2:
    ltmp = capsule_bad_count[1];
    U2TXREG = (ltmp >> 24) & 0xff;
    while (U2STAbits.UTXBF);
    U2TXREG = (ltmp >> 16) & 0xff;
    while (U2STAbits.UTXBF);
    U2TXREG = (ltmp >> 8) & 0xff;
    while (U2STAbits.UTXBF);
    U2TXREG = ltmp & 0xff;
    break;
case 3:
    ltmp = capsule_bad_count[2];
    U2TXREG = (ltmp >> 24) & 0xff;
    while (U2STAbits.UTXBF);
    U2TXREG = (ltmp >> 16) & 0xff;
    while (U2STAbits.UTXBF);
    U2TXREG = (ltmp >> 8) & 0xff;
    while (U2STAbits.UTXBF);
    U2TXREG = ltmp & 0xff;
    break;
case 4:
    ltmp = capsule_bad_count[3];
    U2TXREG = (ltmp >> 24) & 0xff;
    while (U2STAbits.UTXBF);
    U2TXREG = (ltmp >> 16) & 0xff;
    while (U2STAbits.UTXBF);
    U2TXREG = (ltmp >> 8) & 0xff;
    while (U2STAbits.UTXBF);
    U2TXREG = ltmp & 0xff;
    break;

default:
    break;
}
break;

case UARTCMD_TOTAL_COUNT:
    /* Respond with Bad capsule counter value - 4 bytes */
    switch (dat[0]) {
case 1:
        ltmp = capsule_total_count[0];
        U2TXREG = (ltmp >> 24) & 0xff;
        while (U2STAbits.UTXBF);
        U2TXREG = (ltmp >> 16) & 0xff;
        while (U2STAbits.UTXBF);
        U2TXREG = (ltmp >> 8) & 0xff;
        while (U2STAbits.UTXBF);
        U2TXREG = ltmp & 0xff;
        break;
case 2:
        ltmp = capsule_total_count[1];
        U2TXREG = (ltmp >> 24) & 0xff;
        while (U2STAbits.UTXBF);
        U2TXREG = (ltmp >> 16) & 0xff;
        while (U2STAbits.UTXBF);
        U2TXREG = (ltmp >> 8) & 0xff;
        while (U2STAbits.UTXBF);
        U2TXREG = ltmp & 0xff;
        break;
case 3:
        ltmp = capsule_total_count[2];
        U2TXREG = (ltmp >> 24) & 0xff;
        while (U2STAbits.UTXBF);

```

```

        U2TXREG = (ltmp >> 16) & 0xff;
        while (U2STAbits.UTXBF);
        U2TXREG = (ltmp >> 8) & 0xff;
        while (U2STAbits.UTXBF);
        U2TXREG = ltmp & 0xff;
        break;
    case 4:
        ltmp = capsule_total_count[3];
        U2TXREG = (ltmp >> 24) & 0xff;
        while (U2STAbits.UTXBF);
        U2TXREG = (ltmp >> 16) & 0xff;
        while (U2STAbits.UTXBF);
        U2TXREG = (ltmp >> 8) & 0xff;
        while (U2STAbits.UTXBF);
        U2TXREG = ltmp & 0xff;
        break;

    default:
        break;
}

case UARTCMD_GET_CAM0_PULSE_POS:
    /* Respond with motor pulse freq */
    U2TXREG = (cam0_pulse_position >> 8) & 0x00ff;
    U2TXREG = cam0_pulse_position & 0x00ff;
    break;

case UARTCMD_GET_CAM1_PULSE_POS:
    /* Respond with motor pulse freq */
    U2TXREG = (cam1_pulse_position >> 8) & 0x00ff;
    U2TXREG = cam1_pulse_position & 0x00ff;
    break;

case UARTCMD_GET_ACCEPT_ON_PULSE_POS:
    /* Respond with motor pulse freq */
    U2TXREG = (accept_on_pulse_position >> 8) & 0x00ff;
    U2TXREG = accept_on_pulse_position & 0x00ff;
    break;

case UARTCMD_GET_ACCEPT_OFF_PULSE_POS:
    /* Respond with motor pulse freq */
    U2TXREG = (accept_off_pulse_position >> 8) & 0x00ff;
    U2TXREG = accept_off_pulse_position & 0x00ff;
    break;

case UARTCMD_GET_PC_PWR_STATE:
    /* Respond with power state of PCs */
    tmp = 0;
    tmp |= pc_pwr_status[0].state & 0x03;
    tmp |= ((pc_pwr_status[1].state << 2) & 0x0c);
    tmp |= ((pc_pwr_status[2].state << 4) & 0x30);
    tmp |= ((pc_pwr_status[3].state << 6) & 0xc0);
    U2TXREG = tmp;

    break;

default:
    /* Do nothing */
    break;
}

/* Set buffer pointers equal */
uart_rx.rd = uart_rx.wr = 0;
uart_cmd_flag = 0;

//complete_job();
}

void
enable_motor (void)
{
    unsigned char tjob;
    char msg[16];

```

```
system_status.state = MACHINE_RUNNING;
/* Enable Motor */
motor_speed_target = mtr_running_freq;

add_job (LCD_CLEAR);

/* Display Message on LCD */
tjob = get_next_job();
job_list[tjob].data[16] = 1;
job_list[tjob].data[17] = 0;
job_list[tjob].bc = 13;
sprintf (job_list[tjob].data, "Motor Enabled");
add_job (LCD_WRITE);

tjob = get_next_job();
job_list[tjob].data[16] = 2;
job_list[tjob].data[17] = 0;
sprintf (msg, "Speed: %d Hz", motor_speed_target);
job_list[tjob].bc = strlen (msg);
sprintf (job_list[tjob].data, msg);
add_job (LCD_WRITE);
lcd_lock = LCD_LOCK_TIMEOUT;

if (motor_speed < motor_speed_target) {
    OC_MTR_CTRL_CONbits.OCM = OCM_CONT_PULSE;
    ramp_mode = RAMP_UP;
    motor_stop_flag = FALSE;
    T5CONbits.TON = TRUE;
}
}

void
disable_motor (void)
{
    unsigned char tjob;

    if (system_status.state == MACHINE_RUNNING)
        system_status.state = MACHINE_STOPPED;

    add_job (LCD_CLEAR);

    /* Display Message on LCD */
    tjob = get_next_job();
    job_list[tjob].data[16] = 1;
    job_list[tjob].data[17] = 0;
    job_list[tjob].bc = 14;
    sprintf (job_list[tjob].data, "Motor Disabled");
    add_job (LCD_WRITE);

    lcd_lock = LCD_LOCK_TIMEOUT;

    motor_speed_target = MOTOR_MIN_FREQ;
    if (motor_speed > motor_speed_target) {
        ramp_mode = RAMP_DOWN;
        motor_stop_flag = TRUE;
        T5CONbits.TON = TRUE;
    }
}

void
debug_mode (int set)
{
    int tjob;

    /* Display Message on LCD */
    add_job (LCD_CLEAR);
    tjob = get_next_job();
    job_list[tjob].data[16] = 1;
    job_list[tjob].data[17] = 0;
    job_list[tjob].bc = 13;
    sprintf (job_list[tjob].data, "    Debug Mode");
```



```
add_job (LCD_WRITE);

if (set) {
    if (system_status.state == MACHINE_RUNNING)
        disable_motor();

    system_status.state = MACHINE_DEBUG;

    /* Display Message on LCD */
    tjob = get_next_job();
    job_list[tjob].data[16] = 2;
    job_list[tjob].data[17] = 0;
    job_list[tjob].bc = 12;
    sprintf (job_list[tjob].data, "    Enabled");
    add_job (LCD_WRITE);

    /* Enable Timer 9 */
    T9CONbits.TON = TRUE;
}
else {
    system_status.state = MACHINE_STOPPED;

    /* Display Message on LCD */
    tjob = get_next_job();
    job_list[tjob].data[16] = 2;
    job_list[tjob].data[17] = 0;
    job_list[tjob].bc = 13;
    sprintf (job_list[tjob].data, "    Disabled");
    add_job (LCD_WRITE);

    /* Disable Timer 9 */
    T9CONbits.TON = FALSE;
}
}

void
calibrate_sys (void)
{
    /* initially set to 1 (arbitrary) */
    pulse_counter = 1;

    motor_stop_flag = FALSE;

    /* Write to LCD */
    lcd_clear ();
    lcd_print ("Calibrating...", 14);

    /* Align to Zero Position */
    /* Set Motor Speed */
    PR2 = 0x03ff;
    T2CONbits.TON = 1;

    /* Enable Calibration Mode */
    calibration_mode = 1;

    /* Start Motor */
    /* Initially OFF */
    OC_MTR_CTRL_CONbits.OCM = OCM_DISABLED;
    /* Use Timer2 */
    OC_MTR_CTRL_CONbits.OCTSEL = OCTSEL_TIMER2;
    OC_MTR_CTRL_CONbits.OCM = OCM_CONT_PULSE;

    /* Wait for calibration to complete */
    while (calibration_mode);
}

void
set_pc_pwr (unsigned char pc)
{
    /* Check pc number, if 0xff, power all */
    if (pc == 0xff) {
```

```
/* Switch PWR SW signal to ALL MBs to ON */
PC_PWR0 = TRUE;
PC_PWR1 = TRUE;
PC_PWR2 = TRUE;
PC_PWR3 = TRUE;

/* Delay for 400 ms */
Delay (Delay_1mS_Cnt * 400);

/* Switch PWR SW signal to ALL MBs to OFF */
PC_PWR0 = FALSE;
PC_PWR1 = FALSE;
PC_PWR2 = FALSE;
PC_PWR3 = FALSE;
}
else if (pc == 0) {
/* Switch PWR SW signal to MB 0 to ON */
PC_PWR0 = TRUE;

/* Delay for 400 ms */
Delay (Delay_1mS_Cnt * 400);

/* Switch PWR SW signal to MB 0 to OFF */
PC_PWR0 = FALSE;
}
else if (pc == 1) {
/* Switch PWR SW signal to MB 1 to ON */
PC_PWR1 = TRUE;

/* Delay for 400 ms */
Delay (Delay_1mS_Cnt * 400);

/* Switch PWR SW signal to MB 1 to OFF */
PC_PWR1 = FALSE;
}
else if (pc == 2) {
/* Switch PWR SW signal to MB 2 to ON */
PC_PWR2 = TRUE;

/* Delay for 400 ms */
Delay (Delay_1mS_Cnt * 400);

/* Switch PWR SW signal to MB 2 to OFF */
PC_PWR2 = FALSE;
}
else if (pc == 3) {
/* Switch PWR SW signal to MB 3 to ON */
PC_PWR3 = TRUE;

/* Delay for 400 ms */
Delay (Delay_1mS_Cnt * 400);

/* Switch PWR SW signal to MB 3 to OFF */
PC_PWR3 = FALSE;
}
}
```

## D.4 i2c\_slave.h

```
/* i2c_slave.h
 * =====
 * Routines to handle I2C Slave module. Uses I2C_1 module for
 * slave, and I2C_2 for master.
 *
 *
 * Author: Neil Scott
 * Date: September 12, 2007
 */

#ifndef I2C_SLAVE_H
#define I2C_SLAVE_H
```

---

```
#include <p33FJ256GP710.h>
#include "common.h"

void init_i2c_slave(void);

#endif /* I2C_SLAVE_H */
```

## D.5 i2c\_slave.c

```
/* i2c_slave.c
 * =====
 * Routines to handle I2C Slave module.  Uses I2C_1 module for
 * slave, and I2C_2 for master.
 *
 *
 * Author: Neil Scott
 * Date: September 12, 2007
 */

#include "common.h"

#define I2C1_FREQ 100000 /* Hz */
#define I2C1_BRG 363 /* ((Fcy / I2C2_FREQ) - (Fcy / 1111111) ) - 1 */

extern struct JOB job_list[MAX_JOBS];
extern int cap_id;
extern unsigned char cap_count;
extern unsigned char capsule_passfail[4][16];
extern volatile unsigned char i2c_bs_active_bus;
extern volatile struct PC_PWR_STATUS pc_pwr_status[4];

volatile unsigned char i2c_cmd;
volatile unsigned char i2c_data;

volatile unsigned char i2c1_data_flag;
volatile unsigned char i2c1_addr_flag;
volatile unsigned char i2c1_cmd_flag;
volatile unsigned char i2c1_last_cmd;

/* Interrupt Service routine for I2C Slave module */
void
__attribute__((interrupt, no_auto_psv)) _SI2C1Interrupt (void)
{
    unsigned char tjob;
    unsigned char tmp;
    static unsigned char i_job;
    unsigned char cap_id;
    static unsigned char cmd;
    unsigned char data;
    unsigned char buf;
    unsigned char bc;

    /* Reset I2C Bus Switch Timer */
    TMR_BUS_SWITCH_TMR = TMR_BUS_SWITCH_PR/4;

    /* Create a Job */
    if (!I2C1STATbits.R_W) {
        buf = I2C1RCV;
        bc = 1;
    }
    else {
        tmp = I2C1RCV;
        bc = 0;
    }

    /* Clear Interrupt Flag */
    IFS1bits.SI2C1IF = 0;

    if (bc) {
        /* Unload Address from Receive Buffer */

```

```
    if (!i2c1_addr_flag) {
        i2c1_addr_flag = TRUE;
        return;
    }

    /* Get Command */
    if (!i2c1_data_flag) {
        cmd = buf;
        i2c1_data_flag = TRUE;
        return;
    }
}
else {
    I2C1TRN = cap_count;
    I2C1CONbits.SCLREL = TRUE;
}

switch (cmd) {
    case I2CCMD_GET_CAPID:
        /* Check for read */
        if (!bc) {
            I2C1TRN = cap_count;
            I2C1CONbits.SCLREL = TRUE;

            i2c1_data_flag = FALSE;
            i2c1_addr_flag = FALSE;
        }
        break;

    case I2CCMD_SET_PF:
        data = buf;
        cap_id = ((0xf0 & data) >> 4) & 0x0f;
        capsule_passfail[i2c_bs_active_bus][cap_id] = data & 0x0f;

        i2c1_data_flag = FALSE;
        i2c1_addr_flag = FALSE;

        break;

    case I2CCMD_SET_PC_READY:
        data = buf;
        if (data)
            pc_pwr_status[i2c_bs_active_bus].state |= PC_POWER_READY;

        i2c1_data_flag = FALSE;
        i2c1_addr_flag = FALSE;

        break;

    default:
        break;
}
}

void
init_i2c_slave (void)
{
    /* Enable I2C1 module as Slave */
    I2C1CONbits.I2CEN = TRUE;

    /* Set for 7-bit Address mode */
    I2C1CONbits.A10M = FALSE;

    /* Disable Master Interrupt */
    IEC1bits.MI2C1IE = FALSE;

    /* Enable Slave Interrupt */
    IEC1bits.SI2C1IE = TRUE;

    /* Set interrupt priority of SI2C1IE */
    IPC4bits.SI2C1IP = 0x06;

    /* Clear Interrupt Flag */
}
```

```
IFS1bits.SI2C1IF = DEASSERTED;

/* Set SCL1 and SDA1 as open drain */
ODCGbits.ODCG2 = TRUE;      /* SCL1 */
ODCGbits.ODCG3 = TRUE;      /* SDA1 */

/* Set I2C Clock Rate */
I2C1BRG = I2C1_BRG;

/* Set I2C1 Slave Address */
I2C1ADD = I2C_SLAVE_ADDRESS;

/* Clear Last Start and Set Last Stop Flag */
I2C1STATbits.P = 1;
I2C1STATbits.S = 0;

i2c1_data_flag = FALSE;
i2c1_addr_flag = FALSE;
i2c1_cmd_flag = FALSE;
}
```

## D.6 i2c\_commands.h

```
/* i2c_commands.h
 * =====
 * Definitions for I2C slave module commands.
 *
 * Author: Neil Scott
 * Date: September 17, 2007
 */

#ifndef I2C_COMMANDS_H
#define I2C_COMMANDS_H

#define I2CCMD_GET_CAPID 0xA0 /* Get Capsule ID */
#define I2CCMD_SET_PF 0x80 /* Set Pass/Fail */
#define I2CCMD_SET_PC_READY 0x81 /* Set Ready Flag for Inspect Software */

#endif /* I2C_COMMANDS_H */
```

## D.7 i2c\_2.h

```
/* i2c_2.h
 * =====
 * Routines to handle I2C Master module using polling. The
 * I2C_2 module is used as the master, and I2C_1 module is
 * used as the slave.
 *
 * Author: Neil Scott
 * Date: August 06, 2007
 */

#ifndef I2C_2_H
#define I2C_2_H

#include <p33FJ256GP710.h>

void init_i2c2 (void);
char i2c2_write (unsigned char addr, const unsigned char *buf, unsigned char len);
char i2c2_read (unsigned char addr, unsigned char *buf, unsigned char len);

#endif /* I2C_2_H */
```

## D.8 i2c\_2.c

---

```

/* i2c_2.c
 * =====
 * Routines to handle I2C Master module using polling. The
 * I2C_2 module is used as the master, and I2C_1 module is
 * used as the slave.
 *
 * Author: Neil Scott
 * Date: August 06, 2007
 */

#include "i2c_2.h"
#include "common.h"

#define ACK_TIMEOUT 40
#define I2C2_FREQ 100000 /* Hz */
#define I2C2_BRG 363 /* ((Fcy / I2C2_FREQ) - (Fcy / 1111111) ) - 1 */

#define TIMEOUT_SEN 2500 /* Start Enable Response Timeout */
#define TIMEOUT_TRN 800
#define TIMEOUT_PEN 250
#define TIMEOUT_ACKSTAT 350
#define TIMEOUT_TBF 250
#define TIMEOUT_RCEN 250
#define TIMEOUT_JDONE 250

volatile unsigned char jDone;

/* I2C2 Master Module Interrupt Service Routine */
void
__attribute__((interrupt, no_auto_psv)) _MI2C2Interrupt (void)
{
    jDone = 1;

    /* Clear Interrupt Flag */
    IFS3bits.MI2C2IF = DEASSERTED;
}

void
init_i2c2 (void)
{
    /* Set SCL2 and SDA2 as open drain */
    ODCAbits.ODCA2 = TRUE; /* SCL2 */
    ODCAbits.ODCA3 = TRUE; /* SDA2 */

    /* Set I2C Clock Rate */
    I2C2BRG = I2C2_BRG;

    /* Enable the I2C Module as Master */
    I2C2CONbits.I2CEN = TRUE;
    /* Enable Master Interrupt */
    IEC3bits.MI2C2IE = TRUE;
    /* Clear Interrupt Flag */
    IFS3bits.MI2C2IF = DEASSERTED;

    /* Reset Done Flag */
    jDone = 0;
}

/* Returns zero on success */
char
i2c2_write (unsigned char addr, const unsigned char *buf, unsigned char len)
{
    unsigned char count = 0;
    unsigned int timeout = 0;

    jDone = 0;
    I2C2CONbits.SEN = TRUE;
    Nop();
    timeout = TIMEOUT_SEN;

    while (!jDone && timeout) {
        timeout--;
    }
}

```

---

```
}

if (!timeout) {
    return ERR_I2C_MASTER_START;
}

jDone = 0;
I2C2TRN = (addr << 1);
Nop();
timeout = TIMEOUT_TRN;

while (!jDone && timeout) {
    timeout--;
}

if (!timeout) {
    return ERR_I2C_MASTER_TRN;
}

/* Check for ACK */
if (I2C2STATbits.ACKSTAT) {
    return ERR_I2C_MASTER_NACK;
}

Nop ();

for (; count < len; count++) {
    I2C2TRN = buf[count];
    timeout = TIMEOUT_TBF;
    Nop();
    jDone = 0;
    timeout = 2000;          //TIMEOUT_JDONE;

    while (!jDone && timeout) {
        timeout--;
    }

    if (I2C2STATbits.ACKSTAT) {
        return 1;
    }

    if (!timeout) {
        return 1;
    }

    Nop();
}

jDone = 0;
I2C2CONbits.PEN = TRUE;
Nop();
Nop();

while (!jDone);

return 0;
}

/* Returns non-zero on success */
char
i2c2_read (unsigned char addr, unsigned char *buf, unsigned char len)
{
    unsigned char count = 0;
    unsigned int timeout = 0;

    jDone = 0;
    I2C2CONbits.SEN = TRUE;
    Nop();
    Nop();
    timeout = TIMEOUT_SEN;

    while (!jDone && timeout) {
        timeout--;
    }
}
```

```
}

if (!timeout) {
    return -1;
}

/* For Read, bit 0 must be high */
jDone = 0;
I2C2TRN = (addr << 1) | 0x01;
Nop();
timeout = TIMEOUT_TRN;

while (!jDone && timeout) {
    timeout--;
}

if (!timeout) {
    return ERR_I2C_MASTER_TRN;
}

/* Check for ACK */
if (I2C2STATbits.ACKSTAT) {
    return 1;
}

Nop();

for (; count < len; count++) {

    jDone = 0;

    /* Enable Receive Mode */
    I2C2CONbits.RCEN = TRUE;

    timeout = TIMEOUT_RCEN;

    while (!jDone && timeout) {
        timeout--;
    }

    buf[count] = I2C2RCV;

    /* If last byte, set to NACK, otherwise set to ACK */
    if (count == (len - 1)) {
        /* Set to send NACK */
        I2C2CONbits.ACKDT = ASSERTED;
    }
    else {
        /* Set to send ACK */
        I2C2CONbits.ACKDT = DEASSERTED;
    }

    jDone = 0;

    /* Generate ACK */
    I2C2CONbits.ACKEN = TRUE;

    while (!jDone);
}

jDone = 0;
I2C2CONbits.PEN = TRUE;
Nop();
timeout = TIMEOUT_PEN;

while (!jDone && timeout) {
    timeout--;
}

return 0;
}
```



## D.9 i2c\_io\_exp.h

```
/* i2c_io_exp.h
 * =====
 * Contains definitions for I2C I/O Expansion board
 *
 * Author: Neil Scott
 * Date: January 28, 2008
 */

#ifndef I2C_IO_EXP_H
#define I2C_IO_EXP_H

/* I2C I/O Expansion Board abstraction */
struct I2C_IO_EXP
{
    unsigned char i2c_addr;
    unsigned char bi2c_channel;
    unsigned char inp;
    unsigned char outp;
    unsigned char ts_i2c_addr;
    unsigned char ts_conf_reg;
    unsigned char ts_temp_reg[2];
};

/* Function Prototypes */
int i2c_io_exp_write (struct I2C_IO_EXP *io_exp);
int i2c_io_exp_read (struct I2C_IO_EXP *io_exp);
int i2c_io_exp_set_ts_conf (struct I2C_IO_EXP *io_exp);
int i2c_io_exp_get_ts_conf (struct I2C_IO_EXP *io_exp);
int i2c_io_exp_get_ts_temp (struct I2C_IO_EXP *io_exp);

#endif /* I2C_IO_EXP_H */
```

## D.10 i2c\_io\_exp.c

```
/* i2c_io_exp.c
 * =====
 * Routines to communicate with I/O expansion boards.
 *
 * Author: Neil Scott
 * Date: January 28, 2008
 */

#include "i2c_io_exp.h"

struct I2C_IO_EXP temp_sense;
struct I2C_IO_EXP io_side_panel;
struct I2C_IO_EXP io_front_panel;

/* Perform I2C request to write output data to the specified I2C I/O Expansion board */
int i2c_io_exp_write (struct I2C_IO_EXP *io_exp)
{
    unsigned char data[2];

    /* Must write 0xFF to input port of NXP PCA8575 */
    data[0] = 0xff;

    /* Copy output data from structure */
    data[1] = io_exp->outp;

    /* Send data to device */
    return (i2c2_write (io_exp->i2c_addr, data, 2));
}

/* Perform I2C request to read input data from specified I2C I/O Expansion Board */
int i2c_io_exp_read (struct I2C_IO_EXP *io_exp)
{
    unsigned char data[2];
```

```
int ret;

ret = i2c2_read (io_exp->i2c_addr, data, 2);

/* Copy data read from device to structure */
io_exp->inp = data[0];
io_exp->outp = data[1];

return ret;
}

/* Set the configuration register of the I2C temperature sensor */
int i2c_io_exp_set_ts_conf (struct I2C_IO_EXP *io_exp)
{
    unsigned char data[2];

    /* To select Configuration register, first byte is 0x01 */
    data[0] = 0x01;
    data[1] = io_exp->ts_conf_reg;

    return (i2c2_write (io_exp->ts_i2c_addr, data, 2));
}

/* Read the configuration register of the I2C temperature sensor */
int i2c_io_exp_get_ts_conf (struct I2C_IO_EXP *io_exp)
{
    unsigned char data[2];
    int ret;

    /* To select Configuration register, must write 0x01 */
    data[0] = 0x01;

    if ( ret = i2c2_write (io_exp->ts_i2c_addr, data, 1) ) {
        return ret;
    }

    ret = i2c2_read (io_exp->ts_i2c_addr, data, 1);
    io_exp->ts_conf_reg = data[0];

    return ret;
}

/* Read the temperature register of the I2C temperature sensor */
int i2c_io_exp_get_ts_temp (struct I2C_IO_EXP *io_exp)
{
    unsigned char data[2];
    int ret;

    /* To select Temperature register, must write 0x00 */
    data[0] = 0x00;

    if ( ret = i2c2_write (io_exp->ts_i2c_addr, data, 1) ) {
        return ret;
    }

    ret = i2c2_read (io_exp->ts_i2c_addr, &io_exp->ts_temp_reg, 2);

    return ret;
}
```

## D.11 lcd\_i2c.h

```
/* lcd_i2c.h
 * =====
 * Header file for LCD module controlled over I2C using
 * I/O Extender.
 *
 * Author: Neil Scott
```

```

* Date:    August 14, 2007
*
* Standard HITACHI LCD (HD44780) Pinout:
*
* 1  - GND
* 2  - 5V
* 3  - LCD Driver Vee
* 4  - RS
* 5  - R/W
* 6  - Enable
* 7-14 - DATA0 to DATA7
* 15 - LED+
* 16 - LED-
*/

#ifndef LCD_I2C_H
#define LCD_I2C_H

#define I2C_LCD_SLAVE_ADDR    0x27 /* 7-bit addr */

/* Bitmasks */
#define bmLCD_RS                0x01
// #define bmLCD_RW            0x02 /* Shorted to GND for write */
#define bmLCD_PWR                0x02
#define bmLCD_EN                0x04
#define bmLCD_LED              0x08

#define bmLCD_DATA0             0x00
#define bmLCD_DATA1             0x00
#define bmLCD_DATA2             0x00
#define bmLCD_DATA3             0x00

#define bmLCD_DATA4             0x10
#define bmLCD_DATA5             0x20
#define bmLCD_DATA6             0x40
#define bmLCD_DATA7             0x80

/* Function Prototypes */
char lcd_init (void);
char lcd_push_nibble (char nibble);
char lcd_write_cmd (char cmd);
char lcd_write_data (char data);
char lcd_set_bl (char state);
char lcd_cursor_to (char line, char x);
char lcd_print (char *msg, char len);
char lcd_clear (void);
char lcd_print_loc (char *msg, char line, char x);

#endif /* LCD_I2C_H */

```

## D.12 lcd\_i2c.c

```

/* lcd_i2c.c
 * =====
 * Routines to display text on LCD through I2C I/O expander.
 * Requires the I2C routine function i2c2_write().
 *
 *
 * Author:  Neil Scott
 * Date:    August 14, 2007
 */

#include "common.h"
#include "lcd_i2c.h"

#define DELAY_LCD_EN    (Delay200uS_count*3)

extern struct I2C_IO_EXP io_front_panel; /* Front Panel I2C Exp controls LCD */

char
push_nibble (char nibble)

```

```
{
    char val;
    char ret;

    val = 0xff;

    /* Clear DATA bits */
    io_front_panel.outp |= (bmLCD_DATA4 | bmLCD_DATA5 | bmLCD_DATA6 | bmLCD_DATA7);

    if (nibble & 0x01)
        io_front_panel.outp &= ~bmLCD_DATA4;
    if (nibble & 0x02)
        io_front_panel.outp &= ~bmLCD_DATA5;
    if (nibble & 0x04)
        io_front_panel.outp &= ~bmLCD_DATA6;
    if (nibble & 0x08)
        io_front_panel.outp &= ~bmLCD_DATA7;

    io_front_panel.outp |= bmLCD_EN;

    /* Set DATA line */
    ret = i2c_io_exp_write (&io_front_panel);

    /* Error during I2C write */
    if (ret) {
        return ret;
    }

    Delay_Us (DELAY_LCD_EN);

    /* Set EN */
    io_front_panel.outp &= ~bmLCD_EN;
    ret = i2c_io_exp_write (&io_front_panel);

    Delay_Us (2*DELAY_LCD_EN);

    /* Release EN */
    io_front_panel.outp |= bmLCD_EN;
    ret = i2c_io_exp_write (&io_front_panel);

    Delay_Us (DELAY_LCD_EN);

    return 0;
}

char
lcd_write_cmd (char cmd)
{
    char val;
    char ret;

    /* Set RS to LOW */
    io_front_panel.outp |= bmLCD_RS;

    /* Push High Nibble then Low Nibble */
    val = (cmd >> 4) & 0x0f;
    ret = push_nibble (val);

    if (ret)
        return ret;

    Delay_Us (DELAY_LCD_EN);

    val = cmd & 0x0f;
    push_nibble (val);

    Delay_Us (DELAY_LCD_EN);

    return 0;
}

char
```

---

```
lcd_write_data (char data)
{
    char val;
    char ret;

    /* Set RS HIGH */
    io_front_panel.outp &= ~bmLCD_RS;

    /* Push High Nibble then Low Nibble */
    val = (data >> 4) & 0x0f;
    ret = push_nibble (val);

    if (ret)
        return ret;

    Delay_Us (DELAY_LCD_EN);

    val = data & 0x0f;
    push_nibble (val);

    Delay_Us (DELAY_LCD_EN);

    return 0;
}

char
lcd_init (void)
{
    char ret;

    /* Set Default Output Data */
    /* Inputs all high */
    io_front_panel.outp = bmLCD_LED | bmLCD_RS | bmLCD_PWR | bmLCD_EN | bmLCD_DATA4 | bmLCD_DATA5 |
        bmLCD_DATA6 | bmLCD_DATA7;
    ret = i2c_io_exp_write (&io_front_panel);

    ret = push_nibble (0x02);

    if (ret) {
        return ret;
    }

    /* Initialize LCD in 4-bit mode */
    Delay(20 * Delay_1mS_Cnt);
    push_nibble (0x02);
    Delay(20 * Delay_1mS_Cnt);
    push_nibble (0x08);
    Delay(20 * Delay_1mS_Cnt);
    push_nibble (0x00);
    Delay(20 * Delay_1mS_Cnt);
    push_nibble (0x0c); /* Enable Display - No Blinking Cursor */
    Delay(20 * Delay_1mS_Cnt);
    push_nibble (0x00);
    Delay(20 * Delay_1mS_Cnt);
    push_nibble (0x01);
    Delay(20 * Delay_1mS_Cnt);
    push_nibble (0x00);
    Delay(20 * Delay_1mS_Cnt);
    push_nibble (0x06);
    Delay(20 * Delay_1mS_Cnt);
    push_nibble (0x00);
    Delay(20 * Delay_1mS_Cnt);
    push_nibble (0x01);
    Delay(20 * Delay_1mS_Cnt);

    return 0;
}

char
lcd_set_bl (char state)
{
    char ret;
```

---

```
/* Turn off back-light */
if (!state)
    io_front_panel.outp |= bmLCD_LED;
else
    io_front_panel.outp &= ~bmLCD_LED;

ret = i2c_io_exp_write (&io_front_panel);

if (ret) {
    return ret;
}

Delay (DELAY_LCD_EN);

return 0;
}

void
lcd_cursor_to (char line, char x)
{
    char i;

    lcd_write_cmd (0x02);

    x += (line - 1) * 40;

    for (i = 0; i < x; i++)
        lcd_write_cmd (0x14);
}

char
lcd_print (char *msg, char len)
{
    char i;
    char ret;

    for (i = 0; i < len; i++) {
        ret = lcd_write_data (msg[i]);

        if (ret)
            return ret;
    }

    Delay (DELAY_LCD_EN);

    return 0;
}

char
lcd_clear (void)
{
    char ret;

    ret = lcd_write_cmd (0x01);

    Delay (DELAY_LCD_EN*5);

    if (ret)
        return ret;

    return 0;
}

char
lcd_print_loc (char *msg, char line, char x)
{
    char i;
    char ret;
```

---

```
/* Go Home */
ret = lcd_write_cmd (0x02);

if (ret) {
    return ret;
}

Delay (DELAY_LCD_EN*5);

/* Goto location */
x += (line - 1) * 40;

for (i = 0; i < x; i++)
    lcd_write_cmd (0x14);

for (i = 0; i < strlen(msg); i++) {
    lcd_write_data (msg[i]);
}

return 0;
}
```

## D.13 uart\_commands.h

```
/* uart_commands.h
 * =====
 * Definitions for UART commands.
 *
 * Author: Neil Scott
 * Date: August 10, 2007
 */

#ifndef UART_COMMANDS_H
#define UART_COMMANDS_H

/* System Control - Set Commands */
/* Enable or Disable Motor */
#define UARTCMD_SET_MOTOR_STAT 0x90
/* Set Motor Speed */
#define UARTCMD_SET_MOTOR_FREQ 0x91
/* Set pulse width for BL0 */
#define UARTCMD_SET_BL0_WIDTH 0x92
/* Set pulse width for BL1 */
#define UARTCMD_SET_BL1_WIDTH 0x93
/* Set pulse width for FL0 */
#define UARTCMD_SET_FL0_WIDTH 0x94
/* Set pulse width for FL1 */
#define UARTCMD_SET_FL1_WIDTH 0x95

/* Set motor pulse position for CAM0 */
#define UARTCMD_SET_CAM0_PULSE_POS 0x9A
/* Set motor pulse position for CAM1 */
#define UARTCMD_SET_CAM1_PULSE_POS 0x9B
/* Set motor pulse position for ACCEPT ON */
#define UARTCMD_SET_ACCEPT_ON_PULSE_POS 0x9C
/* Set motor pulse position for ACCEPT OFF */
#define UARTCMD_SET_ACCEPT_OFF_PULSE_POS 0x9D

/* Toggle power SW for 400ms to PC MBs */
#define UARTCMD_POWER_ON_PCS 0x9E

/* System Control - Get Commands */
/* Enable or Disable Motor */
#define UARTCMD_GET_MOTOR_STAT 0x10
/* Get Motor Speed */
#define UARTCMD_GET_MOTOR_FREQ 0x11
/* Get pulse width for BL0 */
#define UARTCMD_GET_BL0_WIDTH 0x12
/* Get pulse width for BL1 */
#define UARTCMD_GET_BL1_WIDTH 0x13
/* Get pulse width for FL0 */
```

```

#define UARTCMD_GET_FL0_WIDTH 0x14
/* Get pulse width for FL1 */
#define UARTCMD_GET_FL1_WIDTH 0x15

/* Get motor pulse position for CAM0 */
#define UARTCMD_GET_CAM0_PULSE_POS 0x1A
/* Get motor pulse position for CAM1 */
#define UARTCMD_GET_CAM1_PULSE_POS 0x1B
/* Get motor pulse position for ACCEPT ON */
#define UARTCMD_GET_ACCEPT_ON_PULSE_POS 0x1C
/* Get motor pulse position for ACCEPT OFF */
#define UARTCMD_GET_ACCEPT_OFF_PULSE_POS 0x1D

/* Retrieve the power state of all PCs */
#define UARTCMD_GET_PC_PWR_STATE 0x1E

/* Inspection Status */
/* Get good capsule count for specified quadrant */
#define UARTCMD_GOOD_COUNT 0x21
/* Get reject capsule count for specified quadrant */
#define UARTCMD_BAD_COUNT 0x22
/* Get total capsule count from specified quadrant */
#define UARTCMD_TOTAL_COUNT 0x23
/* Reset the counters */
#define UARTCMD_RESET_COUNTERS 0xAF

/* Fault Registers */
/* Get fault count */
#define UARTCMD_FAULT_COUNT 0xF0
/* Get fault code of previous fault */
#define UARTCMD_FAULT_CODE 0xF1

/* Debug Modes */
/* Set in debug mode so images are acquired when motor is off */
#define UARTCMD_DEBUG_MODE 0xDD
/* Make system step one capsule, fires BL and trigger */
#define UARTCMD_ONE_STEP 0xDE

#endif /* UART_COMMANDS_H */

```

## D.14 uart2.h

```

/* UART Routines for UART2 Module */

#ifndef UART2_H
#define UART2_H

#define MAX_UART_RX_BUF 40
#define MAX_UART_TX_BUF 40

struct UART_Rx
{
    unsigned char wr;
    unsigned char rd;
};

struct UART_Tx
{
    unsigned char wr;
    unsigned char rd;
    unsigned char tx_complete_flag;
};

void init_uart2 (void);
void __attribute__((interrupt, no_auto_psv)) _U2RXInterrupt (void);
void __attribute__((interrupt, no_auto_psv)) _U2TXInterrupt (void);

```

## D.15 uart2.c



---

```

/* uart2.c
 * =====
 * Routines to handle the UART2 module
 *
 * Author: Neil Scott
 * Date: August 01, 2007
 */

#include "common.h"
#include "uart2.h"

#define FCY          40000000    /* 40MHz Clock */
#define BAUDRATE     57600      /* 38400 baud */
#define BRGVAL       ((FCY/BAUDRATE)/16)-1 /* Baud Rate Generator Register Value */

extern volatile struct SYSTEM_STATUS system_status;
extern unsigned char ramp_mode;
extern unsigned int motor_speed;
extern unsigned int motor_speed_target;
extern unsigned int temperature;

extern unsigned char uart_rx_buf[MAX_UART_RX_BUF];
extern unsigned char uart_tx_buf[MAX_UART_TX_BUF];
extern unsigned char uart_cmd_flag;

extern unsigned char curr_job;
extern unsigned char last_job;
extern struct JOB job_list[MAX_JOBS];

struct UART_Rx  uart_rx;
struct UART_Tx  uart_tx;

volatile unsigned char uart_byte_count;
volatile unsigned char uart_set_motor_speed;

/* Handle UART2 Interrupts */
void
__attribute__((interrupt, no_auto_psv)) _U2RXInterrupt(void)
{
    unsigned char next;

    /* Clear Interrupt Flag */
    IFS1bits.U2RXIF = DEASSERTED;

    /* Put data in the circular in-buffer */
    if (U2STAbits.URXDA) {
        uart_rx_buf[uart_rx.wr] = 0x00ff & U2RXREG;

        /* Increment buffer Write Address */
        uart_rx.wr++;
        if (uart_rx.wr >= MAX_UART_RX_BUF)
            uart_rx.wr = 0;
    }

    if (uart_rx.wr == (uart_rx.rd + 6)) {
        uart2_hdlr ();
    }
}

void
__attribute__((interrupt, no_auto_psv)) _U2TXInterrupt(void)
{
    /* Clear Interrupt Flag */
    IFS1bits.U2TXIF = DEASSERTED;
}

void
init_uart2(void)
{
    /* Configure UART2 Module */
    U2MODEbits.UARTEN = 0; /* Disabled for now */
    U2MODEbits.USIDL = 0; /* Continue in Idle */

```

---

```
U2MODEbits.IREN = 0;          /* No IR translation */
U2MODEbits.RTSMD = 0;        /* Simplex Mode */
U2MODEbits.UEN = 0;          /* TX, RX enabled; CTS, RTS disabled */
U2MODEbits.WAKE = 0;         /* Since always awake */
U2MODEbits.LPBACK = 0;       /* No loopback */
U2MODEbits.ABAUD = 0;        /* Disable Auto Baud Detect */
U2MODEbits.URXINV = 0;       /* Do not invert receive polarity bit */
U2MODEbits.BRGH = 0;         /* Not High Baud Rate (standard mode) */
U2MODEbits.PDSEL = 0;        /* No Parity */
U2MODEbits.STSEL = 0;        /* 1 - Stop Bit */

U2BRG = BRGVAL;              /* Set to 38400 baud */

U2STAbits.UTXISEL1 = 0;
U2STAbits.UTXINV = 0;
U2STAbits.UTXISEL0 = 0;
U2STAbits.UTXBRK = 0;
U2STAbits.UTXEN = 0;
U2STAbits.UTXBF = 0;
U2STAbits.TRMT = 0;
U2STAbits.URXISEL = 0;
U2STAbits.ADDEN = 0;
U2STAbits.RIDLE = 0;
U2STAbits.PERR = 0;
U2STAbits.FERR = 0;
U2STAbits.OERR = 0;
U2STAbits.URXDA = 0;

IFS1bits.U2TXIF = 0;         /* Clear TX Interrupt Flag */
IEC1bits.U2TXIE = 1;         /* Enable TX Interrupt */
IFS1bits.U2RXIF = 0;         /* Clear RX Interrupt Flag */
IEC1bits.U2RXIE = 1;         /* Enable RX Interrupt */

U2MODEbits.UARTEN = 1;
U2STAbits.UTXEN = 1;

uart_rx.rd = 0;
uart_rx.wr = 0;
uart_tx.rd = 0;
uart_tx.wr = 0;
}
```

---

## Appendix E

### *Host PC Software*

---

This chapter contains all source code developed for the host PC excluding the image processing library (libIP).

#### E.1 inspect

The main inspect software that facilitates inspection. Uses POSIX threads to parallelize image acquisition and inspection. Uses libIP image processing library object to perform image analysis.

##### E.1.1 Makefile

```
TOP_SRC    = ../../..
CC         = g++
#CFLAGS    = -O2 -fno-rtti -fno-exceptions
CFLAGS     = -g -O2
LDFLAGS    = -lusb -ltiff -lpthread
INCLUDE    = -I. -I../lib -I../libIP -I$(TOP_SRC)/firmware/fx2_revB/firmware/include
CLEANFILES = inspect test_ip
OBJS       = ../lib/pm_cam.o \
             ../lib/pm_prims.o \
             ../lib/imgusb.o \
             ../lib/time_calculations.o \
             ../libIP/ip.o \
             ../lib/img_conv.o \
             ../lib/bayer.o

all: inspect test_ip

inspect: inspect.cc ../libIP/ip.o
        $(CC) $(CFLAGS) $(LIB) $(LDFLAGS) $(INCLUDE) $(OBJS) $(DEFS) -o $@

test_ip: test_ip.cc ../libIP/ip.o
        $(CC) $(CFLAGS) $(LIB) $(LDFLAGS) $(INCLUDE) $(OBJS) $(DEFS) -o $@

clean:
        rm $(CLEANFILES)
```

# Dependencies

## E.1.2 inspect.h

```
/* Filename:
 *   inspect.h
 *
 * Description:
 *   Header file for inspect.cc
 *   Stores constants, structs and prototypes
 *
 * Author:
 *   Neil Scott
 *
 * Date
 *   May 5, 2008
 */

/* Camera Position Constants */
#define PM_CAM_POS_BOTTOM    0x04
#define PM_CAM_POS_CENTER    0x01
#define PM_CAM_POS_LEFT      0x02
#define PM_CAM_POS_RIGHT     0x03

/* FPGA Registers */
#define FPGA_REG_COLS        0x03
#define FPGA_REG_ROWS        0x02

#define FPGA_REG_PX_COUNT_HI  0x06
#define FPGA_REG_PX_COUNT_LOW 0x07

#define FPGA_REG_RESET        0x01

#define FPGA_REG_DATA_MODE    0x04

#define FPGA_DATA_MODE_NN     0x04
#define FPGA_DATA_MODE_RAW    0x02
#define FPGA_DATA_MODE_TEST   0x00

#define NUM_THREADS           5
#define MAX_DEV_COUNT         4
#define MAX_QUADRANT          4

/* Image defaults */
#define MI_WINDOW_HEIGHT_MAX  1536
#define MI_WINDOW_WIDTH_MAX   2048
#define MI_BYTES_PER_PX       1

/* Micron sensor defaults */
#define DEFAULT_WINDOW_HEIGHT 1024
#define DEFAULT_WINDOW_WIDTH  1024
#define DEFAULT_IMAGE_HEIGHT   512
#define DEFAULT_IMAGE_WIDTH    512
#define DEFAULT_COL_SKIP       1
#define DEFAULT_ROW_SKIP       1
#define DEFAULT_COL_START      28
#define DEFAULT_ROW_START      80
#define MI_BINNING_MODE        0x01 /* 2X binning */

/* FPGA power/reset */
#define FPGA_POWER_ON          1
#define FPGA_POWER_OFF         0
#define FPGA_RESET_ENABLE      1
#define FPGA_RESET_DISABLE     0

/* Output Messaging */
#define OUT_MSG                 stdout
#define OUT_ERR_MSG             stderr

/* Inspection Fail Return Codes */
#define IP_FAIL_GENERAL         -20
#define IP_FAIL_WRONG_COLOUR    -7
```

```
#define IP_FAIL_CAPSULE_LENGTH -6
#define IP_FAIL_SURFACE_FLAW -5
#define IP_FAIL_WRONG_BODY -4
#define IP_FAIL_CAP_RADIUS -3
#define IP_FAIL_MISSING_CAP -2
#define IP_FAIL HOLDER_ONLY -1
#define IP_FAIL_DIMENSION 1

/* Capsule P/F Array */
#define CAPSULE_BUFFER_SIZE 16

/* Thread Data structure */
struct thread_data {
    int cam_id; /* camera index */
    int cam_pos; /* camera position (angle) */
    int cam_quad; /* camera location (quadrant) */
    int cam_master; /* camera master flag */
    int thread_id; /* thread index */
    int local_cap_id; /* local capsule index */
    int remote_cap_id; /* remote capsule index */
    int capsule_count; /* capsule count */
    int buffer_id; /* buffer index */
    int insp_result; /* result of inspection */
};

void cleanup(void);
void print_cam_loc(int cam_id);
void cam_reg_setup(int cam_id);
```

### E.1.3 inspect.cc

```
/* Filename:
 * inspect.cc
 *
 * Description:
 * Capsule inspection application. This application retrieves images from
 * camera sensor, performs inspection using image processing and returns an
 * inspection result to the camera.
 *
 * Author:
 * Neil Scott
 *
 * Date:
 * April 16, 2007
 * Updated:
 * May 5, 2008
 * Aug 19, 2008 - Version 0.8 Using IP class instead of functions
 * (less memory allocation and deallocation).
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#include <usb.h>
#include <linux/usbdevice_fs.h>
#include <list>
#include <tiffio.h>
#include <pthread.h>
#include <time.h>

#include "img_conv.h"
#include "bayer.h"
#include "imgusb.h"
#include "pm_prims.h"
#include "pm_ids.h"
#include "pm_cam.h"
#include "time_calculations.h"
#include "mi_regs.h"
```

---

```

#include "fx2cam_ids.h"
#include "fx2cam_commands.h"
#include "fx2cam_i2c_addr.h"

#include "ip.h"
#include "inspect.h"

#define SHOW_TIME
#define DEBUG
// #define SW_INTERPOLATION
#define IP_DETAILS 1

#define STATS_UPDATE_INTERVAL 2.0/* s */
#define CONF_PARAMS_COUNT 7

const char APP_TITLE[] = "inspect";
const char APP_VERSION[] = "0.8";

char STAT_FILE_HTML[80] = "/opt/pill_machine/statistics.html";
char STAT_FILE_CSV[80] = "/opt/pill_machine/statistics.csv";
char IMAGE_DIR[80] = "/images/";
static char CONF_FILE[] = "inspect.conf";

/* Default Camera Gains */
short int CAM_GAIN_LEFT = 0x0a0a;
short int CAM_GAIN_RIGHT = 0x0a0a;
short int CAM_GAIN_CENTER = 0x0a0a;
short int CAM_GAIN_BOTTOM = 0x0a0a;
short int CAM_GAIN_DEFAULT = 0x0a0a;

/* Application Total Running Time */
double inspect_rate_start_time;
double inspect_rate_end_time;

/* Cyclic Buffers used for image data for various cameras and threads */
unsigned char *buf[NUM_THREADS*MAX_DEV_COUNT];

/* Globals used for pmcam class members */

int dev_count; /* number of matching USB devices */
pm_cam **pcam; /* pm_cam class for each device */
struct usb_device **pm_dev; /* libusb device pointer */
struct usb_dev_handle **pm_dev_hdl; /* libusb device handle */
int cam_master_index = 0xff; /* camera master index */

/* Global Image Processing Class Array */
pmIP **pm_IP; /* Image Processing Classes */

/* Application Flags defaults (global) */

bool inspect_p = true; /* Perform inspection */
bool verbose_p = false; /* Don't be verbose */
bool write_raw_p = false; /* Don't write RAW data to file */
bool write_tiff_p = false; /* Write image to TIFF file */

/* Interrupt Signal Event */
bool sig_int_p = false;
int sig_int_count = 0;

/* Capsule Counter */
long cap_count_good = 0;
long cap_count_bad = 0;
long capsule_count = 0;

/* Capsule pass/fail Array */
int **capsule_passfail;
int prev_remote_capsule_id[CAPSULE_BUFFER_SIZE];

/* Camera Position String Identifier */
char *pos_str[] = {"UNDEFINED", "CENTER", "LEFT", "RIGHT", "BOTTOM"};

/* Image Processing Thread Data array */
struct thread_data ip_data[NUM_THREADS * MAX_DEV_COUNT];
pthread_t ip_threads[MAX_DEV_COUNT * NUM_THREADS]; /* Image Processing Threads */

```

---

---

```

pthread_attr_t      ip_attr; /* POSIX Thread Attributes - IP threads */

/* Alignment Counter */
int      misalignment_counter = 0;
int      data_flush_counter = 0;

/* Function Prototypes */
int read_conf_var (char *conf_file, char *keyword, char *value);
void clear_stats_files (void);

/* Find USB Device on bus */
struct usb_device *
find_device (const char *bus, const char *dev)
{
    for (usb_bus *b = usb_busses; b; b = b->next) {
        if (!strcmp (b->dirname, bus)) {
            for (struct usb_device *d = b->devices; d; d = d->next) {
                if (!strcmp (d->filename, dev))
                    return (d);
            }
        }
    }

    return (NULL);
}

/* Interrupt signal handler */
void
handle_sig_int (int dummy)
{
    fprintf (OUT_MSG, "Interrupt signal caught...\n");

    sig_int_p = true;

    if (!sig_int_count) {
        /* Start Timer for capsule count */
        inspect_rate_end_time = get_elapsed_time();
    }

    sig_int_count++;

    if (sig_int_count > 2)
        cleanup();
}

/* Output Application usage */
void
usage (void)
{
    fprintf (stderr, "usage: %s [OPTION]\n", APP_TITLE);
    fprintf (stderr, "\nInspection Options:\n");
    fprintf (stderr, "  --disable_ip, -id      Disable inspection\n");
    fprintf (stderr, "  --write_raw, -raw      Write image buffer to data file\n");
    fprintf (stderr, "  --write_tiff, -wt      Write image data to TIFF file\n");
    fprintf (stderr, "\nApplication Options:\n");
    fprintf (stderr, "  --help, -h            Display this help screen\n");
    fprintf (stderr, "  --verbose, -v         Show verbose messaging\n");
    fprintf (stderr, "\nFPGA Options:\n");
    fprintf (stderr, "  --fpga_data_nn        Set FPGA output mode to Nearest Neighbour\n");
    fprintf (stderr, "  --fpga_data_raw       Set FPGA output mode to RAW sensor data\n");
    fprintf (stderr, "  --fpga_data_test      Set FPGA output mode to test data\n");
    fprintf (stderr, "\nSensor Options:\n");
    fprintf (stderr, "  --load_eeprom, -le    Load Window Settings from EEPROM\n");
    fprintf (stderr, "\n\n");
}

/* Image processing thread */
void *
tinspect (void *tdata)
{
    struct thread_data *targ;

```

---

---

```

int c_id;
int t_id;
int local_cap_id;
int cam_quad;
int cam_pos;
bool cam_master;
int buf_index;
int buf_count;
int ret;
int insp_retval;
char filename_raw[100];
char filename[100];
FILE *fp;
long cap_count;
unsigned char *gray;          /* buffer to hold grayscale data if interpolating in s/w */
unsigned char *rgb;           /* buffer to hold interpolated image data for s/w interpolation */
                               /* our camera uses this tile */
BayerTile bayer_tile = BAYER_TILE_GBRG;
TIFF *img;

/* Detach this thread */
ret = pthread_detach (pthread_self());

if (ret) {
    fprintf (stderr, "Error - pthread_detach: %s\n", strerror(ret) );
}

/* Get Thread ID */
targ = (struct thread_data *) tdata;
c_id = targ->cam_id;
t_id = targ->thread_id;
cam_quad = targ->cam_quad;
cam_pos = targ->cam_pos;
cam_master = targ->cam_master;
buf_index = targ->buffer_id;
local_cap_id = targ->local_cap_id;
cap_count = targ->capsule_count;

/* Software Interpolation */
#ifdef SW_INTERPOLATION
/* allocate grayscale buffer */
gray = new unsigned char [pcam[c_id]->get_image_width()*pcam[c_id]->get_image_height()];
/* allocate RGB buffer */
rgb = new unsigned char [pcam[c_id]->get_image_width()*pcam[c_id]->get_image_height()*3];

/* user bayer.h to conver to grayscale (error in bayer2gray) */
gp_bayer_decode (buf[buf_index],
    pcam[c_id]->get_image_width(),
    pcam[c_id]->get_image_height(),
    rgb,
    bayer_tile);

rgb2grayscale (rgb, gray, pcam[c_id]->get_image_width() * pcam[c_id]->get_image_height());
#endif
/* SW_INTERPOLATION */

/* Perform Inspection */
if (inspect_p) {
    if (verbose_p)
        fprintf (OUT_MSG, "Performing Inspection:\n===== \n\n");

    #ifdef SW_INTERPOLATION
    /* Inspect SW Interpolated Image */
    insp_retval = pm_IP[c_id]->inspect(gray, NULL, NULL);
    #else
    /* Inspect HW Interpolated Image */
    insp_retval = pm_IP[c_id]->inspect(buf[buf_index], NULL, NULL);
    fprintf (stdout, "IP: RET = %d\n", insp_retval);
    #endif
    /* SW_INTERPOLATION */

    targ->insp_result = insp_retval;
    capsule_passfail[c_id][local_cap_id] = insp_retval;
} /* inspect_p */

/* Make a filename */

```

---



---

```

sprintf (filename, "/images/IMG%04d_Q%d_%s", cap_count, cam_quad, pos_str[cam_pos]);
sprintf (filename_raw, "/images/raw/IMG%04d_Q%d_p%d.dat", cap_count, cam_quad, cam_pos);

//TODO -- ADD Failure Code Identifiers //
if (insp_retval == IP_FAIL_HOLDER_ONLY) {
    sprintf (filename, "%s_FAIL_HOLDER_ONLY.tiff", filename);
}
else if (insp_retval == IP_FAIL_DIMENSION) {
    sprintf (filename, "%s_FAIL_DIMENSION.tiff", filename);
}
else if (insp_retval < IP_FAIL_HOLDER_ONLY) {
    sprintf (filename, "%s_FAIL_GENERAL.tiff", filename);
}
else {
    sprintf (filename, "%s.tiff", filename);
}
//TODO//

fprintf (stdout, "TIFF FILENAME: %s\n", filename);

/* Write buffer directly to file - no interpolation performed */
if (write_tiff_p) {
    #ifdef SW_INTERPOLATION
        pcam[c_id]->write_tiff (gray, filename, pcam[c_id]->get_image_width(), pcam[c_id]->get_image_height
            ());
    #else
        pcam[c_id]->write_tiff (buf[buf_index], filename, pcam[c_id]->get_image_width(), pcam[c_id]->
            get_image_height());
    #endif
    if (verbose_p)
        fprintf (stdout, "TIFF File Written (%s)\n", filename);
}

if (write_raw_p) {
    /* Write raw data to disk */
    fp = fopen (filename_raw, "wb");
    fwrite (buf[buf_index], 1, (pcam[c_id]->get_image_width() * pcam[c_id]->get_image_height()), fp);
    if (verbose_p)
        fprintf (stdout, "\nRAW Image file created: %s\n", filename_raw);
    fclose (fp);
}

#ifdef SW_INTERPOLATION
delete gray;
delete rgb;
#endif

pthread_exit(NULL);
}

/* Grab frame thread */
void *
tgrab_frame (void *tdata)
{
    struct thread_data *targ;
    int c_id;
    int t_id;
    int p_id;
    int cam_quad;
    int cam_pos;
    bool cam_master;
    int buf_index;
    int buf_count;
    int cap_count;
    int local_cap_id;
    int remote_cap_id;
    int ret;
    char flush_buf[MI_WINDOW_HEIGHT_MAX*MI_WINDOW_WIDTH_MAX];

    /* Get Thread ID */
    targ = (struct thread_data *) tdata;
    c_id = targ->cam_id;
    t_id = targ->thread_id;

```

---

```
cam_quad = targ->cam_quad;
cam_pos = targ->cam_pos;
cam_master = targ->cam_master;
buf_index = targ->buffer_id;
cap_count = targ->capsule_count;
remote_cap_id = targ->remote_cap_id;
local_cap_id = targ->local_cap_id;

if (verbose_p)
    fprintf (OUT_MSG, "Grabbing Frame... (t_id = %d; c_id = %d)\n", t_id, c_id);

/* Grab Image Data - store in global buffer */
ret = pcam[c_id]->grab_frame (buf[buf_index]);

/* check return */
if (ret != (pcam[c_id]->get_image_height() * pcam[c_id]->get_image_width())) {
    fprintf (OUT_ERR_MSG, "grab_frame: ERROR!\n");
}

if (verbose_p)
    fprintf (OUT_MSG, "Frame Successfully Acquired: (t_id = %d), cap_count = %d\n", t_id, cap_count);

/* If this is the master camera, retrieve capsuleID */
/* Send Control message to control board to fetch capsule ID */
if (cam_master) {
    char data[3];
    int insp_result = 0;
    int no_cap_check = 0;

    data[0] = 0xA0;
    ret = pcam[cam_master_index]->write_cmd ( VENDOR_REQUEST_OUT,
        VRQ_I2C_WRITE,
        CB_I2C_ADDR,
        0,
        data,
        1);
    if (ret < 0)
        fprintf (OUT_ERR_MSG, "Error retrieving Capsule ID from system controller\n");

    ret = pcam[cam_master_index]->write_cmd ( VENDOR_REQUEST_IN,
        VRQ_I2C_READ,
        CB_I2C_ADDR,
        0,
        data,
        1);

    if (ret < 0)
        fprintf (OUT_ERR_MSG, "Error retrieving Capsule ID from system controller\n");

    remote_cap_id = data[0];
    targ->remote_cap_id = remote_cap_id;

    /* from previous inspection */
    int prev_local_cap_id;

    if (local_cap_id == 0)
        prev_local_cap_id = 15;
    else
        prev_local_cap_id = local_cap_id - 1;

    fprintf (stdout, "prev_local_cap_id = %d\n"
        "prev_remote_cap_id[prev_local_capsule_id] = %d\n",
        prev_local_cap_id,
        prev_remote_capsule_id[prev_local_cap_id]);

    /* process inspection results */
    insp_result = capsule_passfail[0][prev_local_cap_id] == 0;

    for (int i = 0; i < dev_count; i++) {
        fprintf (stdout, "capsule_passfail[%d][%d] = %d\n", i, prev_local_cap_id, capsule_passfail[i][
            prev_local_cap_id]);
    }

    for (int j = 1; j < dev_count; j++)
```

---

```

    insp_result += capsule_passfail[j][prev_local_cap_id] == 0;

no_cap_check = 0;

/* TODO */
/* pass all if not inspecting */
if (!inspect_p)
    insp_result = dev_count;

if (insp_result == dev_count)
    cap_count_good++;
else {
    for (int x = 0; x < dev_count; x++) {
        no_cap_check |= (capsule_passfail[x][prev_local_cap_id] == IP_FAIL_HOLDER_ONLY);
    }

    if (!no_cap_check)
        cap_count_bad++;
}

if (verbose_p)
    fprintf (OUT_MSG, "CapsuleID: %d\n", remote_cap_id);

/* respond to control board with result from previous capsule */
/* Build control message */
data[0] = 0x80;
data[1] = ((insp_result == dev_count) ? 2 : 1);
data[1] |= (prev_remote_capsule_id[prev_local_cap_id] << 4) & 0xf0;

ret = pcam[cam_master_index]->write_cmd(VENDOR_REQUEST_OUT,
    VRQ_I2C_WRITE,
    CB_I2C_ADDR,
    0,
    data,
    2);

if (ret < 0) {
    fprintf (OUT_ERR_MSG, "Error sending pass/fail result to system control board\n");
}
else {
    fprintf (OUT_MSG, "%04d: %s message successfully sent to system control board (CapsuleID = %d)\n"
        ,
        capsule_count, (insp_result == dev_count) ? "PASS" : "FAIL",
        prev_remote_capsule_id[prev_local_cap_id]);
}
}

bool p_align_marker = true;
int image_width = pcam[c_id]->get_image_width();
int image_height = pcam[c_id]->get_image_height();

#if 1
/* Check last six pixels for alignment marker */
for (int i = 0; i < 6; i+=2) {
    if ( (buf[buf_index][image_width*image_height - i - 2] != 0xaa) || (buf[buf_index][image_width*
        image_height - i - 1] != 0xaa) ) {
        p_align_marker = false;
        break;
    }
}

if (!p_align_marker) {
    misalignment_counter++;

    fprintf (stderr, "\n\n"
        "*****\n"
        "***** ALIGNMENT ERROR *****\n"
        "*****\n"
        "\n\n");

    /* Attempt to recover */
    /* Purge data using libusb bulk read */
    /* Stop the EP so libusb can use it */
    pcam[c_id]->ep_stop();

```

---

---

```

    for (int i = 0; i < 1; i++) {
        ret = usb_bulk_read (pm_dev_hdl[c_id], 0x82, (char *) buf[buf_index], pcam[c_id]->
            get_window_width() * pcam[c_id]->get_window_height(), 20);

        if (ret < 0)
            fprintf (stderr, "ERROR:::: %s\n", usb_strerror());

        if ( !(ret > 0) )
            break;
    }

    /* Restart the EP */
    pcam[c_id]->ep_start();

    /* Put FPGA into reset */
    if (!pcam[c_id]->cam_fpga_reset (FPGA_RESET_ENABLE))
        fprintf (stderr, "Error putting FPGA into reset [Device %d]\n", c_id);

    usleep (25000);

    if (!pcam[c_id]->cam_fpga_reset (FPGA_RESET_DISABLE))
        fprintf (stderr, "Error taking FPGA out of reset [Device %d]\n", c_id);

    ret = pcam[c_id]->write_cmd (VENDOR_REQUEST_OUT, VRQ_FPGA_FLUSH, 0, 0, NULL, 0);

    if (ret < 0)
        fprintf (stderr, "Error purging FPGA [Device %d]: %s\n", c_id, usb_strerror() );
    }
#endif

    /* create image processing thread */
    /* thread id */
    ip_data[buf_index].cam_id = c_id;
    ip_data[buf_index].cam_pos = pcam[c_id]->get_cam_position();
    ip_data[buf_index].cam_quad = pcam[c_id]->get_cam_quadrant();
    ip_data[buf_index].cam_master = pcam[c_id]->get_cam_master();
    ip_data[buf_index].thread_id = t_id;
    ip_data[buf_index].buffer_id = buf_index;
    ip_data[buf_index].local_cap_id = local_cap_id;
    ip_data[buf_index].capsule_count = capsule_count;
    ip_data[buf_index].insp_result = -1;

    /* create image processing thread */
    ret = pthread_create ( &ip_threads[buf_index],
        &ip_attr,
        tinspect,
        (void *) &ip_data[buf_index]);

    pthread_exit (NULL);
}

/* Supervisory Thread
 *
 * To provide fix for occasional startup hang, monitors capsule count
 * and if does not increment for 5 seconds, shutdown app. External
 * script required to start-up again.
 */
void *
tsupervisory (void *tid)
{
    long last_count;

    while (1) {
        last_count = capsule_count;
        usleep (5000000);

        if (last_count == capsule_count) {
            if (capsule_count < 20) {
                cleanup();
                exit(1);
            }
        }
    }
}

```

---

```

    }
}

/* Main Function
 *
 * Finds and configures all cameras and starts main loop
 */
int
main (int argc, char *argv[])
{
    // Data Acquisition Threads
    pthread_t      acq_threads[MAX_DEV_COUNT * NUM_THREADS];
    // POSIX Thread Attributes
    pthread_attr_t  acq_attr;
    // Supervisory Thread
    pthread_t      supervisor_thread;

    struct thread_data acq_data[NUM_THREADS * MAX_DEV_COUNT];

    int            ret;
    int            window_width = DEFAULT_WINDOW_WIDTH;
    int            window_height = DEFAULT_WINDOW_HEIGHT;
    int            image_width = DEFAULT_IMAGE_WIDTH;
    int            image_height = DEFAULT_IMAGE_HEIGHT;
    int            col_skip = DEFAULT_COL_SKIP;
    int            row_skip = DEFAULT_ROW_SKIP;
    int            col_start = DEFAULT_COL_START;
    int            row_start = DEFAULT_ROW_START;

    bool            fpga_data_nn_p = true;
    bool            fpga_data_raw_p = false;
    bool            fpga_data_test_p = false;

    bool            load_window_eeprom_p = false;

    /* Create String Start Time Stamp */
    char start_time_stamp[100];
    char update_time_stamp[100];
    time_t tm_stamp;
    struct tm *tmstmp;
    tm_stamp = time(NULL);
    tmstmp = localtime(&tm_stamp);
    strftime (start_time_stamp, sizeof(start_time_stamp), "%a %d %b %Y %H:%M:%S %z", tmstmp);

    /* parse argument list */
    for (int i = 1; i < argc; i++) {
        if (!strcmp(argv[i], "--help") || !strcmp(argv[i], "-h")) {
            usage();
            exit (1);
        }

        if (!strcmp(argv[i], "--verbose") || !strcmp(argv[i], "-v")) {
            verbose_p = true;
        }

        if (!strcmp(argv[i], "--disable_ip") || !strcmp(argv[i], "-id")) {
            inspect_p = false;
        }

        if (!strcmp(argv[i], "--write_raw") || !strcmp(argv[i], "-raw")) {
            write_raw_p = true;
        }

        if (!strcmp(argv[i], "--write_tiff") || !strcmp(argv[i], "-wt")) {
            write_tiff_p = true;
        }

        if (!strcmp(argv[i], "--fpga_data_nn"))
            fpga_data_nn_p = true;

        if (!strcmp(argv[i], "--fpga_data_raw")) {
            fpga_data_nn_p = false;
            fpga_data_raw_p = true;
        }
    }
}

```

---

```

    }

    if (!strcmp(argv[i], "--fpga_data_test")) {
        fpga_data_nn_p = false;
        fpga_data_test_p = true;
    }

    if (!strcmp(argv[i], "--load_eeeprom") || !strcmp(argv[i], "-le"))
        load_window_eeeprom_p = true;
}

/* print application title */
if (verbose_p) {
    fprintf (OUT_MSG, "%s v%s\n", APP_TITLE, APP_VERSION);
    for (int i = 0; i < (strlen(APP_TITLE)+strlen(APP_VERSION)+2); i++)
        fprintf (OUT_MSG, "=");

    fprintf (OUT_MSG, "\n");
}

char *conf_params[] = {"stats_html_file", "stats_csv_file", "image_dir", "left_gain", "right_gain", "
    center_gain", "bottom_gain"};
char conf_value[80];

/* Look for configuration file */
for (int i = 0; i < CONF_PARAMS_COUNT; i++) {
    ret = read_conf_var (CONF_FILE, conf_params[i], conf_value);
    if (!ret) {
        if (!strcmp(conf_params[i], "stats_html_file")) {
            sprintf (STAT_FILE_HTML, "%s", conf_value);
            fprintf (stdout, "STAT_FILE_HTML: %s\n", STAT_FILE_HTML);
        }

        if (!strcmp(conf_params[i], "stats_csv_file")) {
            sprintf (STAT_FILE_CSV, "%s", conf_value);
            fprintf (stdout, "STAT_FILE_CSV: %s\n", STAT_FILE_CSV);
        }

        if (!strcmp(conf_params[i], "image_dir")) {
            sprintf (IMAGE_DIR, "%s", conf_value);
            if ( (IMAGE_DIR[strlen(IMAGE_DIR)-1]) != '/')
                sprintf (IMAGE_DIR, "%s/", IMAGE_DIR);

            fprintf (stdout, "IMAGE_DIR: %s\n", IMAGE_DIR);
        }

        if (!strcmp(conf_params[i], "left_gain")) {
            sscanf (conf_value, "0x%04x", &CAM_GAIN_LEFT);
            fprintf (stdout, "CAM_GAIN_LEFT: 0x%04x\n", CAM_GAIN_LEFT);
        }

        if (!strcmp(conf_params[i], "right_gain")) {
            sscanf (conf_value, "0x%04x", &CAM_GAIN_RIGHT);
            fprintf (stdout, "CAM_GAIN_RIGHT: 0x%04x\n", CAM_GAIN_RIGHT);
        }

        if (!strcmp(conf_params[i], "center_gain")) {
            sscanf (conf_value, "0x%04x", &CAM_GAIN_CENTER);
            fprintf (stdout, "CAM_GAIN_CENTER: 0x%04x\n", CAM_GAIN_CENTER);
        }

        if (!strcmp(conf_params[i], "bottom_gain")) {
            sscanf (conf_value, "0x%04x", &CAM_GAIN_BOTTOM);
            fprintf (stdout, "CAM_GAIN_BOTTOM: 0x%04x\n", CAM_GAIN_BOTTOM);
        }
    }
    else {
        fprintf (stdout, "Error Processing Configuration File \"%s\"...\n", CONF_FILE);
        break;
    }
}

/* find camera(s) */
/* Search the USB bus for all cameras */
/* Search for the device on the USB bus */

```

---

---

```

/* Initialize USB - libusb library */
if (verbose_p)
    fprintf (OUT_MSG, "Initializing USB library...\n");

/* Initialize libusb */
pm_init_usb ();

/* Get the number of cameras found connected to the system */
dev_count = pm_get_device_count ();

/* Search for new Device use first instance found */
if (verbose_p)
    fprintf (OUT_MSG, "Searching for devices...\n");

if (!dev_count) {
    fprintf (OUT_ERR_MSG, "No devices connected...\n");
    return -1;
}

if (verbose_p)
    fprintf (OUT_MSG, "Found %d camera(s)\n", dev_count);

/* Allocate memory for pointers to device handles */
pm_dev = new struct usb_device *[dev_count];
pm_dev_hdl = new struct usb_dev_handle *[dev_count];
pcam = new pm_cam *[dev_count];

/* Allocate Memory for image buffer */
for (int i = 0; i < dev_count; i++)
    for (int j = 0; j < NUM_THREADS; j++)
        buf[ (i * NUM_THREADS) + j] = new unsigned char [MI_WINDOW_WIDTH_MAX*
            MI_WINDOW_HEIGHT_MAX*
            MI_BYTES_PER_PIX];

/* Allocate Memory for capsule pass/fail results */
capsule_passfail = new int *[dev_count];
for (int i = 0; i < dev_count; i++)
    capsule_passfail[i] = new int [CAPSULE_BUFFER_SIZE];

/* Fill Device Pointers */
for (int i = 0; i < dev_count; i++) {
    pm_dev[i] = pm_find_camera (i);

    if (pm_dev[i] == 0) {
        fprintf (stderr, "%s: Unable to find device...\n", APP_TITLE);
        return -1;
    }

    if (pm_camera_configured (pm_dev[i])) {
        if (verbose_p)
            fprintf (OUT_MSG, "Device %02d is configured...\n", i);
    }
    else if (pm_camera_unconfigured (pm_dev[i])) {
        if (verbose_p)
            fprintf (OUT_MSG, "Device %02d is unconfigured...\n", i);
    }
    else {
        fprintf (stderr, "%s: Unrecognized Device ID...\n", APP_TITLE);
        return -1;
    }
}

/* TODO: Handle Unconfigured Device by uploading firmware */

/* Create a device handle for the device found */
pm_dev_hdl[i] = pm_open_interface ( pm_dev[i],
    USB_PM_DEV_IF_DEFAULT,
    USB_PM_DEV_ALT_IF_DEFAULT);

if (!pm_dev_hdl[i]) {
    fprintf (stderr, "%s: Unable to claim device interface on device %02d...\n", APP_TITLE, i);
    exit (1);
}

if (verbose_p)

```

---

---

```

    fprintf (OUT_MSG, "Device %02d Interface Claimed...\n", i);

    /* Create an instance of the PM_CAM class for the device handle create */
    //PM_CAM_BLOCK_SIZE, 64);
    pcam[i] = new pm_cam(pm_dev_hdl[i], 8*1024, 64);

    /* Set verbose messaging */
    pcam[i]->verbose_p = verbose_p;
}

/* initialize camera(s) */

/* if using eeprom window settings, load them */
if (load_window_eeprom_p) {
    for (int i = 0; i < dev_count; i++) {
        /* Get Window width */
        ret = pcam[i]->read_window_params();

        if (ret < 0)
            fprintf (stderr, "Error retrieving window parameters [DEVICE %d]\n", i);

        /* TODO: Debug - Write to screen */
        #ifdef DEBUG
            fprintf (stdout, "Window Parameters for Camera %d\n", i);
            fprintf (stdout, "EEPROM WINDOW_WIDTH: 0x%04x\n", pcam[i]->get_eeprom_window_width());
            fprintf (stdout, "EEPROM WINDOW_LENGTH: 0x%04x\n", pcam[i]->get_eeprom_window_height());
            fprintf (stdout, "EEPROM WINDOW_COL_START: 0x%04x\n", pcam[i]->get_eeprom_window_col_start());
            fprintf (stdout, "EEPROM WINDOW_ROW_START: 0x%04x\n", pcam[i]->get_eeprom_window_row_start());
            fprintf (stdout, "EEPROM WINDOW_COL_SKIP: 0x%04x\n", pcam[i]->get_eeprom_window_col_skip());
            fprintf (stdout, "EEPROM WINDOW_ROW_SKIP: 0x%04x\n", pcam[i]->get_eeprom_window_row_skip());
        #endif
    }
}

/* get camera location (quadrant and position) */
for (int i = 0; i < dev_count; i++) {
    ret = pcam[i]->get_cam_location();

    if (ret < 0)
        fprintf (stderr, "Error retrieving camera location [DEVICE %d]\n", i);
    else if (verbose_p)
        print_cam_loc(i);

    /* Set master index, should only find one */
    if (pcam[i]->get_cam_master() )
        cam_master_index = i;
}

/* reset fpga */
for (int i = 0; i < dev_count; i++) {
    if (!pcam[i]->cam_fpga_reset (FPGA_RESET_ENABLE))
        fprintf (OUT_ERR_MSG, "Error putting FPGA into reset [Device %d]\n", i);
}

usleep (200000);

for (int i = 0; i < dev_count; i++) {
    if (!pcam[i]->cam_fpga_reset (FPGA_RESET_DISABLE))
        fprintf (OUT_ERR_MSG, "Error taking FPGA out of reset [Device %d]\n", i);
}

usleep (250000);

/* Reset MI Sensor */
for (int i = 0; i < dev_count; i++) {
    ret = pcam[i]->write_reg (MI_REG_RESET, 1);

    usleep (200000);
    ret |= pcam[i]->write_reg (MI_REG_RESET, 0);

    if (ret < 0) {
        fprintf (stderr, "Error resetting MI sensor [Device %d]\n", i);
    }
}

```

---



---

```

    usleep (5000);
}

/* set camera registers */
for (int i = 0; i < dev_count; i++) {

    if (load_window_eeeprom_p) {
        pcam[i]->set_window_width_skip (pcam[i]->get_eeeprom_window_width(), pcam[i]->
            get_eeeprom_window_col_skip());
        pcam[i]->set_window_height_skip (pcam[i]->get_eeeprom_window_height(), pcam[i]->
            get_eeeprom_window_row_skip());
        pcam[i]->set_window_col_start (pcam[i]->get_eeeprom_window_col_start());
        pcam[i]->set_window_row_start (pcam[i]->get_eeeprom_window_row_start());
    }
    else {
        pcam[i]->set_window_width_skip (window_width, col_skip);
        pcam[i]->set_window_height_skip (window_height, row_skip);
        pcam[i]->set_window_col_start (col_start);
        pcam[i]->set_window_row_start (row_start);
    }

    /* other MI sensor registers */
    cam_reg_setup(i);

    usleep(100000);
}

/* set fpga registers */
for (int i = 0; i < dev_count; i++) {
    /* Write FPGA registers */
    /* Set Window Size */
    ret = pcam[i]->fpga_write_reg (FPGA_REG_COLS, (pcam[i]->get_image_width() - 1));

    if (ret < 0) {
        fprintf (stderr, "Error setting FPGA register 0x02\n");
    }
    else {
        fprintf (stdout, "Successfully set FGPA window width register\n");
    }

    ret = pcam[i]->fpga_write_reg (FPGA_REG_ROWS, (pcam[i]->get_image_height() - 1));

    if (ret < 0) {
        fprintf (stderr, "Error setting FPGA register 0x03\n");
    }
    else {
        fprintf (stdout, "Successfully set FGPA window height register\n");
    }

    /* Total Pixel Count */
    ret = pcam[i]->fpga_write_reg (FPGA_REG_PX_COUNT_HI, (pcam[i]->get_image_width() * pcam[i]->
        get_image_height()) >> 16);
    ret = pcam[i]->fpga_write_reg (FPGA_REG_PX_COUNT_LOW, (pcam[i]->get_image_width() * pcam[i]->
        get_image_height()) & 0xffff);

    if (ret < 0) {
        fprintf (stderr, "Error setting FPGA pixel count register!\n");
    }
    else {
        fprintf (stdout, "Successfully set FGPA window height register\n");
    }

    /* Take FPGA out of reset */
    ret = pcam[i]->fpga_write_reg (FPGA_REG_RESET, 0x0000);

    if (ret < 0) {
        fprintf (stderr, "Error setting FPGA register 0x01 (RESET)\n");
    }
    else {
        fprintf (stdout, "Successfully reset FPGA\n");
    }

    /* Set FPGA Output Mode */
    if (fpga_data_nn_p) {

```

---

---

```

    ret = pcam[i]->fpga_write_reg (FPGA_REG_DATA_MODE, FPGA_DATA_MODE_NN);
    if (verbose_p)
        fprintf (OUT_MSG, "FPGA data mode set to nearest neighbor\n");
}

if (fpga_data_raw_p) {
    ret = pcam[i]->fpga_write_reg (FPGA_REG_DATA_MODE, FPGA_DATA_MODE_RAW);
    if (verbose_p)
        fprintf (OUT_MSG, "FPGA data mode set to raw data\n");
}

if (fpga_data_test_p) {
    ret = pcam[i]->fpga_write_reg (FPGA_REG_DATA_MODE, FPGA_DATA_MODE_TEST);
    if (verbose_p)
        fprintf (OUT_MSG, "FPGA data mode set to test sequence\n");
}
}

/* Set application signal handling */
signal (SIGINT, handle_sig_int);

int num_bytes;
unsigned char cap_id;

for (int i = 0; i < dev_count; i++) {
    /* Get Window Size */
    num_bytes = pcam[i]->get_image_width () * pcam[i]->get_image_height ();
}

/* Try to clear buffer */
if (verbose_p)
    fprintf (OUT_MSG, "Purging FPGA data...\n");

for (int i = 0; i < dev_count; i++) {
    ret = 1;

    for (int j = 0; j < NUM_THREADS; j++) {
        if (ret > 0) {
            ret = usb_bulk_read (pm_dev_hdl[i], 0x82, (char *) buf[i*NUM_THREADS], 1024*768, 150);

            if (verbose_p)
                fprintf (OUT_MSG, "Data received from purge [Device %d]: ret = %d\n", i, ret);

            if (ret < 0)
                fprintf (OUT_ERR_MSG, "usb_bulk_read() error - %s\n", usb_strerror());
            else
                pcam[i]->write_tiff (buf[i*NUM_THREADS], "PURGE.tiff", pcam[i]->get_image_width(), pcam[i]->
                    get_image_height());
        }
    }
}

for (int i = 0; i < dev_count; i++) {
    ret = pcam[i]->write_cmd (VENDOR_REQUEST_OUT, VRQ_FPGA_FLUSH, 0, 0, NULL, 0);
    if (ret < 0)
        fprintf (OUT_ERR_MSG, "Error purging FPGA [Device %d]: %s\n", i, usb_strerror());
}

/* Flush STDOUT */
fflush (OUT_MSG);

/* Allocate URBs for IMGUSB class */
for (int i = 0; i < dev_count; i++) {
    pcam[i]->imgusb_allocate_urbs();
}

for (int i = 0; i < dev_count; i++) {
    if (verbose_p)
        fprintf (OUT_MSG, "Starting bulk end-point [Device %d]\n", i);

    /* Start Bulk EP */
    pcam[i]->ep_start ();
}

```

---

```

}

if (cam_master_index != 0xff) {
    /* Send Ready Signal to System Controller */
    for (int i=0; i<4; i++) {
        char cb_ready = 0x81;
        pcam[cam_master_index]->write_cmd ( VENDOR_REQUEST_OUT,
            VRQ_I2C_WRITE,
            CB_I2C_ADDR,
            0,
            (char *) &cb_ready,
            1);
        usleep (10000*(pcam[cam_master_index]->get_cam_quadrant()+1));
    }
}

/* Set Thread detached attribute */
pthread_attr_init (&acq_attr);
pthread_attr_setdetachstate (&acq_attr, PTHREAD_CREATE_JOINABLE);

pthread_attr_init (&ip_attr);
pthread_attr_setdetachstate (&ip_attr, PTHREAD_CREATE_JOINABLE);

int capsule_id = 0;
int buffer_count = 0;
int no_cap_check = 0;
int local_cap_id = 0;
int remote_cap_id = 0;
int cam_bottom_index = 0xff;
char data[16];
int status;
int insp_result;
FILE *fp_stats_html = NULL;
FILE *fp_stats_csv = NULL;
double last_stat_update_time = get_elapsed_time();

/* Statistics Variables */
long empty_holder_count;
double good_cap_percentage;
double bad_cap_percentage;
double empty_holder_percentage;
double elapsed_time;
double inspection_rate;
double eff_inspection_rate;
short int val;

/* create image processing classes (one for each device) */
pm_IP = new pmIP*[dev_count];

for (int i=0; i < dev_count; i++) {
    pm_IP[i] = new pmIP(pcam[i]->get_image_width(),
        pcam[i]->get_image_height(),
        pcam[i]->get_cam_position(),
        IP_DETAILS);
}

/* reset inspection results */
for (int i = 0; i < dev_count; i++)
    capsule_passfail[i][15] = -1;

/* Write Blank stats files (HTML / CSV) */
clear_stats_files();

/* Create Supervisory Thread */
int t=1;
ret = pthread_create ( &supervisor_thread,
    NULL,
    tsupervisory,
    (void *) t);

/* program main loop */
while (!sig_int_p) {

    /* set thread arguments */

```

---

```

for (int i = 0; i < dev_count; i++) {
    /* acquire top camera images first (for synchronization) */
    if (pcam[i]->get_cam_position() == PM_CAM_POS_BOTTOM) {
        cam_bottom_index = i;
        continue;
    }

    /* thread id */
    acq_data[i*NUM_THREADS+buffer_count].cam_id = i;
    acq_data[i*NUM_THREADS+buffer_count].cam_pos = pcam[i]->get_cam_position();
    acq_data[i*NUM_THREADS+buffer_count].cam_quad = pcam[i]->get_cam_quadrant();
    acq_data[i*NUM_THREADS+buffer_count].cam_master = pcam[i]->get_cam_master();
    acq_data[i*NUM_THREADS+buffer_count].thread_id = i * NUM_THREADS + buffer_count;
    acq_data[i*NUM_THREADS+buffer_count].buffer_id = i * NUM_THREADS + buffer_count;
    acq_data[i*NUM_THREADS+buffer_count].local_cap_id = local_cap_id;
    acq_data[i*NUM_THREADS+buffer_count].remote_cap_id = remote_cap_id;
    acq_data[i*NUM_THREADS+buffer_count].capsule_count = capsule_count;

    /* create image acquisition threads */
    ret = pthread_create ( &acq_threads[i*NUM_THREADS + buffer_count],
        &acq_attr,
        tgrab_frame,
        (void *) &acq_data[i*NUM_THREADS + buffer_count]);

    if (ret) {
        fprintf (OUT_ERR_MSG, "Error - pthread_create(): %s\n", strerror(ret) );
    }
}

/* start inspection rate counter on first capsule */
if (capsule_count == 1) {
    /* Start Timer for capsule count */
    inspect_rate_start_time = get_elapsed_time();
}

/* join grab image threads */
for (int i = 0; i < dev_count; i++) {
    if (pcam[i]->get_cam_position() == PM_CAM_POS_BOTTOM)
        continue;

    /* Join Threads to Main */
    ret = pthread_join (acq_threads[i*NUM_THREADS + buffer_count], (void **) &status);
}

/* Top View Images Acquired Here */
if (verbose_p) fprintf (stdout, "\n\n*****TOP VIEW IMAGES ACQUIRED
    *****\n\n\n");

/* Now acquire bottom camera image (if it exists) */
if (cam_bottom_index != 0xff) {
    int i = cam_bottom_index;

    /* thread id */
    acq_data[i*NUM_THREADS+buffer_count].cam_id = i;
    acq_data[i*NUM_THREADS+buffer_count].cam_pos = pcam[i]->get_cam_position();
    acq_data[i*NUM_THREADS+buffer_count].cam_quad = pcam[i]->get_cam_quadrant();
    acq_data[i*NUM_THREADS+buffer_count].cam_master = pcam[i]->get_cam_master();
    acq_data[i*NUM_THREADS+buffer_count].thread_id = i * NUM_THREADS + buffer_count;
    acq_data[i*NUM_THREADS+buffer_count].buffer_id = i * NUM_THREADS + buffer_count;
    acq_data[i*NUM_THREADS+buffer_count].local_cap_id = local_cap_id;
    acq_data[i*NUM_THREADS+buffer_count].remote_cap_id = remote_cap_id;
    acq_data[i*NUM_THREADS+buffer_count].capsule_count = capsule_count;

    /* create image acquisition threads */
    ret = pthread_create ( &acq_threads[i*NUM_THREADS + buffer_count],
        &acq_attr,
        tgrab_frame,
        (void *) &acq_data[i*NUM_THREADS + buffer_count]);

    if (ret) {
        fprintf (OUT_ERR_MSG, "Error - pthread_create(): %s\n", strerror(ret) );
    }
}

/* Join Threads to Main */

```

---

---

```

    ret = pthread_join (acq_threads[i*NUM_THREADS + buffer_count], (void **) &status);
}

/* Bottom Camera Image Acquired */
if (cam_bottom_index != 0xff)
    if (verbose_p) fprintf (stdout, "\n\n*****BOTTOM VIEW IMAGES ACQUIRED
        *****\n\n\n");

/* get remote capsuleID from master cam */
if (cam_master_index != 0xff)
    remote_cap_id = acq_data[cam_master_index*NUM_THREADS+buffer_count].remote_cap_id;

/* remote capsuleID */
if (verbose_p) fprintf (stdout, "REMOTE CAPSULEID: %d\n", remote_cap_id);

/* increment capsule counter */
capsule_count++;

/* set previous remote to local capsuleID */
prev_remote_capsule_id[local_cap_id] = remote_cap_id;

/* increment local capsule_id counter */
local_cap_id++;
if (local_cap_id == CAPSULE_BUFFER_SIZE)
    local_cap_id = 0;

/* clear capsule pass/fail result for upcoming capsule */
for (int i = 0; i < dev_count; i++)
    capsule_passfail[i][local_cap_id] = -1;

/* increment buffer counter for cyclic buffer */
buffer_count++;
if (buffer_count == NUM_THREADS)
    buffer_count = 0;

/* Check stats update timer */
if ( (get_elapsed_time() - last_stat_update_time) < (STATS_UPDATE_INTERVAL))
    continue;
else
    last_stat_update_time = get_elapsed_time();

/* updata statistics file results */
fp_stats_html = fopen (STAT_FILE_HTML, "w");
fp_stats_csv = fopen (STAT_FILE_CSV, "w");

/* Make sure files were created */
if ( (fp_stats_html == NULL) || (fp_stats_csv == NULL)) {
    fprintf (stderr, "Error: Error creating stats files!\n");
    continue;
}

inspect_rate_end_time = get_elapsed_time();
tm_stamp = time(NULL);
tmstamp = localtime(&tm_stamp);
strftime (update_time_stamp, sizeof(update_time_stamp), "%a %d %b %Y %H:%M:%S %z", tmstamp);

/* calculate statistic parameters */
empty_holder_count = capsule_count - (cap_count_good + cap_count_bad);

good_cap_percentage = (double)cap_count_good/(cap_count_good+cap_count_bad)*100.0;
bad_cap_percentage = (double)cap_count_bad/(cap_count_good+cap_count_bad)*100.0;
empty_holder_percentage = (double)(capsule_count - (cap_count_good+cap_count_bad))/capsule_count
    *100.0;

/* seconds */
elapsed_time = (inspect_rate_end_time - inspect_rate_start_time);
/* caps/min */
inspection_rate = 60 * capsule_count / elapsed_time;
/* caps/min */
eff_inspection_rate = 60 * (cap_count_good+cap_count_bad) / elapsed_time;

/* HTML Output */
/* Header */
if (cam_master_index != 0xff)

```

---

---

```

    fprintf (fp_stats_html, "<html>\n\t<head>\n\t\t<title>Q%d - Inspection Statistics</title>\n\t</head>\n",
        pcam[cam_master_index]->get_cam_quadrant());
else
    fprintf (fp_stats_html, "<html>\n\t<head>\n\t\t<title>Q%d - Inspection Statistics</title>\n\t</head>\n",
        0xff);

/* Title */
fprintf (fp_stats_html, "\n\t<body>\n\t\t<h2>Q%d - Inspection Statistics</h2>\n");

/* results table */
fprintf (fp_stats_html, "\t\t<table width=420 border=1, cellpadding=2 cellspacing=0>\n");
fprintf (fp_stats_html, "\t\t\t<tr><td colspan=2 align=center bgcolor=\"#c0c0c0\"><b>Inspection Results</b></td></tr>\n");
fprintf (fp_stats_html, "\t\t\t<tr><td><b>Good Capsules</b></td><td>%d (%.3g%%)</td></tr>\n",
    cap_count_good, good_cap_percentage);

fprintf (fp_stats_html, "\t\t\t<tr><td><b>Bad Capsules</b></td><td>%d (%.3g%%)</td></tr>\n",
    cap_count_bad, bad_cap_percentage);

fprintf (fp_stats_html, "\t\t\t<tr><td><b>Empty Holders</b></td><td>%d (%.3g%%)</td></tr>\n",
    empty_holder_count, empty_holder_percentage);

fprintf (fp_stats_html, "\t\t\t<tr><td><b>Total Capsules</b></td><td>%d</td></tr>\n",
    capsule_count);

fprintf (fp_stats_html, "\t\t\t<tr><td><b>Misaligned Images</b></td><td>%d</td></tr>\n",
    misalignment_counter);

fprintf (fp_stats_html, "\t\t\t<tr><td colspan=2 align=center bgcolor=\"#c0c0c0\"><b>Inspection Rate</b></td></tr>\n");
fprintf (fp_stats_html, "\t\t\t\t<tr><td><b>Elapsed Time</b></td><td>%.5g s</td></tr>\n",
    ((inspect_rate_end_time - inspect_rate_start_time)));

fprintf (fp_stats_html, "\t\t\t\t<tr><td><b>Inspection Rate</b></td><td>%.5g caps/min</td></tr>\n",
    60 * capsule_count / (inspect_rate_end_time - inspect_rate_start_time));

fprintf (fp_stats_html, "\t\t\t\t<tr><td><b>Effective Inspection Rate</b></td><td>%.5g caps/min</td></tr>\n",
    60 * (cap_count_good+cap_count_bad) / (inspect_rate_end_time - inspect_rate_start_time));

fprintf (fp_stats_html, "\t\t</table>\n");

/* HTML Closing */
fprintf (fp_stats_html, "\t\t<p><br></p>\n\t\t<hr>\n\t\t<p><i>Last Updated: %s</i></p>\n",
    update_time_stamp);
fprintf (fp_stats_html, "\t</body>\n");
fprintf (fp_stats_html, "</html>\n");

fclose (fp_stats_html);

/* CSV File Output */
if (cam_master_index != 0xff)
    fprintf (fp_stats_csv, "Quadrant, %d\n", pcam[cam_master_index]->get_cam_quadrant());
else
    fprintf (fp_stats_csv, "Quadrant, UNKNOWN\n");

fprintf (fp_stats_csv, "Start Time, %s\n", start_time_stamp);
fprintf (fp_stats_csv, "Last Updated, %s\n", update_time_stamp);
fprintf (fp_stats_csv, "\nInspection Results, Count, Percentage\n");
fprintf (fp_stats_csv, "Good Capsules, %d, %.3g%%\n", cap_count_good, good_cap_percentage);
fprintf (fp_stats_csv, "Bad Capsules, %d, %.3g%%\n", cap_count_bad, bad_cap_percentage);
fprintf (fp_stats_csv, "Empty Holders, %d, %.3g%%\n", empty_holder_count, empty_holder_percentage);
fprintf (fp_stats_csv, "Total Capsules, %d\n", capsule_count);
fprintf (fp_stats_csv, "Misaligned Images, %d\n", misalignment_counter);
fprintf (fp_stats_csv, "\nInspection Rate\n");
fprintf (fp_stats_csv, "Elapsed Time, %.3g, seconds\n", elapsed_time);
fprintf (fp_stats_csv, "Inspection Rate, %.3g, caps/min\n", inspection_rate);
fprintf (fp_stats_csv, "Effective Inspection Rate, %.3g, caps/min\n", eff_inspection_rate);

fclose (fp_stats_csv);
}
/* while */

```

---

---

```

    /* cleanup */
    cleanup();

    /* success */
    return 0;
}

void
cleanup (void)
{
    /* stop bulk endpoint */
    for (int i = 0; i < dev_count; i++) {
        pcam[i]->ep_stop();
    }

    /* close USB device handles */
    for (int i = 0; i < dev_count; i++) {
        pm_close (pm_dev_hdl[i]);
    }

    delete pm_IP;
    delete pcam;
    delete pm_dev;
    delete pm_dev_hdl;

    exit(0);
}

void
print_cam_loc(int cam_id)
{
    /* Output this information */
    if (verbose_p) {
        fprintf (OUT_MSG, "CAMERA %d:\n=====\n", cam_id);
        fprintf (OUT_MSG, "\tQuadrant:  %d\n", pcam[cam_id]->get_cam_quadrant());
        fprintf (OUT_MSG, "\tPosition:  %d (%s)\n", pcam[cam_id]->get_cam_position(),
            pos_str[pcam[cam_id]->get_cam_position()]);
        fprintf (OUT_MSG, "\tMaster:    %s\n", ((pcam[cam_id]->get_cam_master()) ? "Yes" : "No"));
        fprintf (OUT_MSG, "\n");
    }
}

void
cam_reg_setup(int cam_id)
{
    short val;

    /* Set Camera Registers */
    /* Column and Row start */
    //pcam[cam_id]->write_reg(MI_REG_COLUMN_START, DEFAULT_COL_START);
    //pcam[cam_id]->write_reg(MI_REG_ROW_START, DEFAULT_ROW_START);

    /* Blanking Regions - Defaults (H=142, V=25) */
    //1210
    pcam[cam_id]->write_reg(MI_REG_HORIZ_BLANKING, 1023);
    //830
    pcam[cam_id]->write_reg(MI_REG_VERT_BLANKING, 5);

    /* Global Gain */
    /* Read in from conf file */
    if (pcam[cam_id]->get_cam_position() == PM_CAM_POS_BOTTOM) {
        pcam[cam_id]->write_reg(MI_REG_GLOBAL_GAIN, CAM_GAIN_BOTTOM);
    }
    else if (pcam[cam_id]->get_cam_position() == PM_CAM_POS_CENTER) {
        pcam[cam_id]->write_reg(MI_REG_GLOBAL_GAIN, CAM_GAIN_CENTER);
    }
    else if (pcam[cam_id]->get_cam_position() == PM_CAM_POS_LEFT) {
        pcam[cam_id]->write_reg(MI_REG_GLOBAL_GAIN, CAM_GAIN_LEFT);
    }
    else if (pcam[cam_id]->get_cam_position() == PM_CAM_POS_RIGHT) {
        pcam[cam_id]->write_reg(MI_REG_GLOBAL_GAIN, CAM_GAIN_RIGHT);
    }
}

```

---

---

```

    }
    else {
        pcam[cam_id]->write_reg(MI_REG_GLOBAL_GAIN, CAM_GAIN_DEFAULT);
    }

    /* For Left Camera, increase analog gain */
    if (pcam[cam_id]->get_cam_position() == PM_CAM_POS_LEFT) {
        pcam[cam_id]->write_reg(MI_REG_GLOBAL_GAIN, 0x0a0f);
    }

    #if 0
    /* Individual Channel Gains */
    pcam[cam_id]->write_reg(MI_REG_RED_GAIN, 0x000f);
    pcam[cam_id]->write_reg(MI_REG_GREEN1_GAIN, 0x002f);
    pcam[cam_id]->write_reg(MI_REG_GREEN2_GAIN, 0x002f);
    pcam[cam_id]->write_reg(MI_REG_BLUE_GAIN, 0x000f);

    /* Black Level */
    pcam[cam_id]->write_reg(MI_REG_BLACK_LEVEL, 0x00a8);
    #endif

    /* As per MT9T001 datasheet (p.13) */
    pcam[cam_id]->read_reg(0x4E, &val);
    val &= 0xffef;
    val |= 0x0020;
    pcam[cam_id]->write_reg(0x4E, val);

    #if 1
    /* Set Trigger Mode */
    pcam[cam_id]->read_reg(MI_REG_READ_MODE_1, &val);
    val &= 0xbfff;
    val |= 0x0100;
    pcam[cam_id]->write_reg(MI_REG_READ_MODE_1, val);
    #endif

    #if 0
    /* TODO: Putting in Test Data Mode */
    pcam[cam_id]->read_reg(MI_REG_OUTPUT_CONTROL, &val);
    val |= 0x0040; // bit 6 set for test data mode
    pcam[cam_id]->write_reg(MI_REG_OUTPUT_CONTROL, val);

    pcam[cam_id]->write_reg(MI_REG_TEST_DATA, 0x0000);
    /* */
    #endif

    /* Over-ride Black level calibration */
    #if 0
    pcam[cam_id]->read_reg(0x62, &val);
    val |= 0x0003;
    pcam[cam_id]->write_reg(0x62, val);
    #endif

    #if 1
    /* Set Read Mode 3 register for global shutter control (pg. 23 MT9T001 Datasheet) */
    pcam[cam_id]->read_reg(MI_REG_READ_MODE_3, &val);
    val |= 0x0003;
    pcam[cam_id]->write_reg(MI_REG_READ_MODE_3, val);
    pcam[cam_id]->write_reg(MI_REG_SHUTTER_WIDTH, 1);
    #endif
}

/* read_conf_var()
 *
 * reads variable from configuration file
 *
 * conf_file - configuration file string
 * keyword - variable keyword
 * value - value (if found)
 *
 * returns:
 * 0 - on success
 * -1 - on file error

```

---



---

```

*  -2 - on invalid keyword
*  -3 - on keyword not found
*/
int
read_conf_var (char *conf_file, char *keyword, char *value)
{
    FILE *fp=NULL;
    int len;
    char str[80];
    char *pch;
    int leq;

    if (keyword == NULL)
        return -2;

    len = strlen(keyword);

    if (len > 77)
        return -2;

    if (conf_file) {
        fp = fopen (conf_file, "r");
        if (fp == NULL)
            return -1;
    }
    else
        return -1;

    if ( fseek(fp, 0, SEEK_SET) ) {
        fclose (fp);
        return -1;
    }

    for (;;) {
        fgets (str, 80, fp);
        if (ferror(fp) || feof(fp))
            return -3;

        len = strlen(str);

        /* look for comment character */
        if (!strncmp ("#", str, 1))
            continue;

        if (strncmp (keyword, str, strlen(keyword)) == 0) {
            if (str[len-1] == '\n')
                str[--len] = 0;

            /* find equal sign */
            pch = strchr(str, '=');
            leq = pch-str+1;

            sprintf (value, "%s", &str[leq]);

            /* trim spaces */
            while (value[0] == ' ') strcpy(value, value+1);

            break;
        }
    }

    fclose (fp);

    /* success */
    return 0;
}

void
clear_stats_files (void)
{
    FILE *fp_stats_html;
    FILE *fp_stats_csv;

```

---

---

```

/* updata statistics file results */
fp_stats_html = fopen (STAT_FILE_HTML, "w");
fp_stats_csv = fopen (STAT_FILE_CSV, "w");

/* Make sure files were created */
if ( (fp_stats_html == NULL) || (fp_stats_csv == NULL)) {
    fprintf (stderr, "Error: Error creating stats files!\n");
}

/* calculate statistic parameters */
int empty_holder_count = 0; // capsule_count - (cap_count_good + cap_count_bad);

double good_cap_percentage = 0.0;
double bad_cap_percentage = 0.0;
double empty_holder_percentage = 0.0;

double elapsed_time = 0.0;
double inspection_rate = 0.0;
double eff_inspection_rate = 0.0;

/* HTML Output */
/* Header */
if (cam_master_index != 0xff)
    fprintf (fp_stats_html, "<html>\n<head>\n<title>Q%d - Inspection Statistics</title>\n</head>\n",
        0);
else
    fprintf (fp_stats_html, "<html>\n<head>\n<title>Q%d - Inspection Statistics</title>\n</head>\n",
        0);

/* Title */
fprintf (fp_stats_html, "\n<body>\n<h2>Q%d - Inspection Statistics</h2>\n");

/* results table */
fprintf (fp_stats_html, "\n<table width=420 border=1, cellpadding=2 cellspacing=0>\n");
fprintf (fp_stats_html, "\n<tr><td colspan=2 align=center bgcolor=\"#c0c0c0\"><b>Inspection Results</b></td></tr>\n");
fprintf (fp_stats_html, "\n<tr><td><b>Good Capsules</b></td><td>%d (%.3g%%)</td></tr>\n",
    cap_count_good, good_cap_percentage);

fprintf (fp_stats_html, "\n<tr><td><b>Bad Capsules</b></td><td>%d (%.3g%%)</td></tr>\n",
    cap_count_bad, bad_cap_percentage);

fprintf (fp_stats_html, "\n<tr><td><b>Empty Holders</b></td><td>%d (%.3g%%)</td></tr>\n",
    empty_holder_count, empty_holder_percentage);

fprintf (fp_stats_html, "\n<tr><td><b>Total Capsules</b></td><td>%d</td></tr>\n",
    capsule_count);

fprintf (fp_stats_html, "\n<tr><td><b>Misaligned Images</b></td><td>%d</td></tr>\n",
    misalignment_counter);

fprintf (fp_stats_html, "\n<tr><td colspan=2 align=center bgcolor=\"#c0c0c0\"><b>Inspection Rate</b></td></tr>\n");
fprintf (fp_stats_html, "\n<tr><td><b>Elapsed Time</b></td><td>%.5g s</td></tr>\n",
    ((inspect_rate_end_time - inspect_rate_start_time)));

fprintf (fp_stats_html, "\n<tr><td><b>Inspection Rate</b></td><td>%.5g caps/min</td></tr>\n",
    0);

fprintf (fp_stats_html, "\n<tr><td><b>Effective Inspection Rate</b></td><td>%.5g caps/min</td></tr>\n",
    0);

fprintf (fp_stats_html, "\n</table>");

/* HTML Closing */
fprintf (fp_stats_html, "\n<p><br></p>\n\n<hr>\n\n<p><i>Last Updated: %s</i></p>", "");
fprintf (fp_stats_html, "\n</body>\n");
fprintf (fp_stats_html, "</html>\n");

fclose (fp_stats_html);

```

---

---

```

/* CSV File Output */
if (cam_master_index != 0xff)
    fprintf (fp_stats_csv, "Quadrant, %d\n", pcam[cam_master_index]->get_cam_quadrant() );
else
    fprintf (fp_stats_csv, "Quadrant, UNKNOWN\n");

fprintf (fp_stats_csv, "Start Time, %s\n", "");
fprintf (fp_stats_csv, "Last Updated, %s\n", "");
fprintf (fp_stats_csv, "\nInspection Results, Count, Percentage\n");
fprintf (fp_stats_csv, "Good Capsules, %d, %.3g%%\n", cap_count_good, good_cap_percentage);
fprintf (fp_stats_csv, "Bad Capsules, %d, %.3g%%\n", cap_count_bad, bad_cap_percentage);
fprintf (fp_stats_csv, "Empty Holders, %d, %.3g%%\n", empty_holder_count, empty_holder_percentage);
fprintf (fp_stats_csv, "Total Capsules, %d\n", capsule_count);
fprintf (fp_stats_csv, "Misaligned Images, %d\n", misalignment_counter);
fprintf (fp_stats_csv, "\nInspection Rate\n");
fprintf (fp_stats_csv, "Elapsed Time, %.3g, seconds\n", 0);
fprintf (fp_stats_csv, "Inspection Rate, %.3g, caps/min\n", 0);
fprintf (fp_stats_csv, "Effective Inspection Rate, %.3g, caps/min\n", 0);

fclose (fp_stats_csv);
}

```

### E.1.4 inspect.conf

Configuration file used by inspect.

```

# Configuration File for inspect
# Used to define application parameters including
# camera sensor gains

#Application Parameters
stats_html_file = /opt/pill_machine/statistics.html
stats_csv_file = /opt/pill_machine/statistics.csv
image_dir = /images

#Camera Gains
left_gain = 0x0a2f
right_gain = 0x0a0f
center_gain = 0x0a0a
bottom_gain = 0x0a0a

```

## E.2 test\_ip

An offline test application to verify the functionality of the image processing algorithm.

### E.2.1 test\_ip.cc

```

/* test_ip.cc
 * =====
 * Utility to verify the functionality of the image processing library
 * offline. Input files are specified along with a position ID.
 *
 * Author: Neil Scott
 */

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <tiffio.h>

#include "ip.h"

void
usage (void)

```

---

```

{
    fprintf (stderr, "Usage:\n");
    fprintf (stderr, "  ip_test -p [POSITION] file.tiff\n\n");
    fprintf (stderr, "Position IDs:\n");
    fprintf (stderr, "  1 - Center\n");
    fprintf (stderr, "  2 - Left\n");
    fprintf (stderr, "  3 - Right\n");
    fprintf (stderr, "  4 - Bottom\n");
}

int
main (int argc, char *argv[])
{
    TIFF *img;
    unsigned char *Y, *U, *V;
    int CAM_POS = 1, DETAILS = 1;
    pmIP **pm_ip;

    int width, height;
    char filename[80];
    unsigned char R, G, B;
    uint32 *raster;
    int ret;
    int pos;

    fprintf (stdout, "Image Processor Test App.\n");

    if (argc < 4) {
        usage ();
        return -1;
    }

    for (int i = 0; i < argc; i++) {
        if (!strcmp(argv[i], "-p")) {
            sscanf (argv[i+1], "%d", &pos);
            i++;
        }
    }

    strcpy (filename, argv[argc-1]);

    if ( (img = TIFFOpen (filename, "r")) == NULL) {
        fprintf (stderr, "Error loading image file!\n");
        return -1;
    }

    TIFFGetField (img, TIFFTAG_IMAGEWIDTH, &width);
    TIFFGetField (img, TIFFTAG_IMAGELENGTH, &height);

    /* Allocate Memory */
    raster = (uint32 *) malloc (sizeof(uint32) * width * height);

    /* Load image to buffer */
    TIFFReadRGBAImage (img, width, height, raster, 0);

    /* Convert RGB image to Grayscale */
    /* Allocate memory for grayscale buffer */
    Y = (unsigned char *) malloc(sizeof (char) * width * height);

    int m = 0;
    unsigned int pix;

    for (int e = height - 1; e != -1; e--) {
        for (int c = 0; c < width; c++) {
            R = TIFFGetR(raster[e * width + c]);
            G = TIFFGetG(raster[e * width + c]);
            B = TIFFGetB(raster[e * width + c]);

            pix = (unsigned int) (0.299 * (double) R + 0.587 * (double) G + 0.114 * (double) B + 0.5);

            if (pix > 255) pix = 255;
            if (pix < 0) pix = 0;
        }
    }
}

```

---

```
        Y[m++] = (unsigned char) pix;
    }
}

pm_ip = new pmIP*[1];
pm_ip[0] = new pmIP (width, height, pos, DETAILS);

ret = pm_ip[0]->inspect (Y, NULL, NULL);

//ret = inspect (Y, NULL, NULL, width, height, pos, DETAILS);

if (ret)
    fprintf (stdout, "Inspection FAILED\n");
else
    fprintf (stdout, "Inspection PASSED\n");

fprintf (stdout, "ret = %d\n", ret);

/* cleanup */
free (Y);
free (raster);
TIFFClose (img);

return 0;
}
```

## E.3 cam\_init

This application is used to initialize the camera for first time use. The quadrant, position and master flag are set using this utility. The ability to read/write any arbitrary register of the camera EEPROM exists with this utility.

### E.3.1 Makefile

```
TARGET      = cam_init
CC          = g++
LDFLAGS     = -lusb
INCLUDE     = -I../firmware/fx2_revB/firmware/include
CFLAGS      = -O2

all: $(TARGET)

$(TARGET): $(TARGET).cc
    $(CC) $(TARGET).cc $(CFLAGS) $(LDFLAGS) $(INCLUDE) -o $(TARGET)

clean:
    rm $(TARGET)
```

### E.3.2 cam\_init.cc

```
/* cam_init.cc
 * =====
 * Load USB PID/VID information into camera EEPROM. Also writes camera
 * location information, ie. quadrant and camera position.
 *
 *
 * Author:   Neil Scott
 * Date:     July 28, 2007
 */

#include <iostream>
#include <usb.h>
#include "fx2cam_commands.h"
```

```

#define VERSION                "0.01"

#define FX2_VID                 0xabcd
#define FX2_PID                 0x0201

#define VENDOR_REQUEST_IN      0xC0
#define VENDOR_REQUEST_OUT     0x40

#define VRQ_EEPROM_READ_LG     0xE6
#define VRQ_EEPROM_WRITE_LG    0xE7
#define VRQ_I2C_READ           0x81

#define EEPROM_HEADER_SIZE      8 //bytes
#define EEPROM_SIZE             16*1024 // 128kbits = 16kB

#define EEPROM_QUADPOS_START_ADDR 0x10
#define EEPROM_QUADPOS_LENGTH   3

/* EEPROM Header contains VID/PID/DID according to FX2 TRM (pg. 3.4)
 *
 * For a C0 Load (Only VID/PID/DID - Host loads firmware)
 *   EEPROM ADDR      CONTAINS
 *   0                 0xC0
 *   1                 VID (low)
 *   2                 VID (high)
 *   3                 PID (low)
 *   4                 PID (high)
 *   5                 DID (low)
 *   6                 DID (high)
 *   7                 Configuration Byte (Set to 0) (FX2 TRM pg. 3.8)
 *
 *   VID: 0xABCD;   PID: 0x0201;   DID: 0x0101
 */
unsigned char EEPROM_HEADER[] = {0xc0, 0xcd, 0xab, 0x01, 0x02, 0x01, 0x01, 0x00};

char *pos_str[] = {"UNDEFINED", "CENTER", "LEFT", "RIGHT", "BOTTOM"};

/* List all devices on the USB bus. Mark matching VID/PID
 */
void
show_bus(void)
{
    for (usb_bus *bus = usb_busses; bus; bus = bus->next) {
        for (struct usb_device *dev = bus->devices; dev; dev = dev->next) {
            fprintf(stdout, "bus %s dev %s: ID %04x:%04x", bus->dirname,
                    dev->filename,
                    dev->descriptor.idVendor,
                    dev->descriptor.idProduct);

            if (dev->descriptor.idVendor == FX2_VID &&
                dev->descriptor.idProduct == FX2_PID) {
                std::cout << " <--";
            }

            std::cout << std::endl;
        }
    }
}

/* Search bus for specific device
 */
struct usb_device *
find_device (const char *bus, const char *dev)
{
    for (usb_bus *b = usb_busses; b; b = b->next) {
        if (!strcmp (b->dirname, bus)) {
            for (struct usb_device *d = b->devices; d; d = d->next) {
                if (!strcmp (d->filename, dev))
                    return (d);
            }
        }
    }
}

```

```

    }

    return (NULL);
}

/* Sequentially write data to EEPROM starting at addr
 */
int
write_eeeprom (struct usb_dev_handle *usb_hdl, int addr, char *data, int len)
{
    int ret;
    /* Verify len is no more than 64 bytes */
    if (len > 64)
        return -1;

    ret = usb_control_msg (usb_hdl,
        VENDOR_REQUEST_OUT,
        VRQ_EEPROM_WRITE_LG,
        addr,
        0,
        (char *) data,
        len,
        500);

    return ret;
}

/* Output application usage
 */
void
usage (void)
{
    std::cout << "Usage: cam_init -d=[BUS.DEV] [OPTIONS]..." << std::endl;
    std::cout << "Camera first time setup." << std::endl << std::endl;
    std::cout << "  -d, --device          Specify device to target as BUS.DEV" << std::endl;
    std::cout << "  -q, --quadrant       Specify camera quadrant" << std::endl;
    std::cout << "  -p, --position       Specify camera position" << std::endl;
    std::cout << "  -m, --master         Specify camera as quadrant master (responsible for" << std::endl;
    std::cout << "      comm. with control board" << std::endl;
    std::cout << "  -r, --read           Retrieve ALL EEPROM data to screen" << std::endl;
    std::cout << "  --quickread          Retrieve the camera quadrant, position and master" << std::endl;
    std::cout << "  -s, --save <FILENAME> Read data from EEPROM and save to <FILENAME>" << std::endl;
    std::cout << "  -l, --list           List all devices on USB bus" << std::endl;
    std::cout << "  --blank              Clear EEPROM memory with 0xff" << std::endl;
    std::cout << "  -w, --write_eeeprom [ADDR] [VAL] Write a value to a specific memory location of the" << std::endl;
    std::cout << "      EEPROM" << std::endl;
    std::cout << "  -rb, --read_eeeprom [ADDR] Read a specific byte of EEPROM memory" << std::endl;
    std::cout << "  -rp, --reload_params Reload window parameters from EEPROM" << std::endl;
    std::cout << "  *ADDR and VAL are decimal numbers" << std::endl;
    std::cout << std::endl << "cam_init v" << VERSION << std::endl;
}

int
main (int argc, char *argv[])
{
    char *usb_bus_dev = NULL;
    char bus[4], dev[4];
    int quadrant, position, master = 0;
    bool device_p = false;
    bool bus_dev_p = false;
    bool list_only_p = false;
    bool quadrant_p = false;
    bool position_p = false;
    bool master_p = false;
    bool read_p = false;
    bool save_p = false;
    bool quick_read_p = false;
    bool blank_p = false;
    bool write_eeeprom_p = false;

```

---

```

bool  read_eeeprom_p = false;
bool  reload_params_p = false;
char  we_data;
int  we_addr;
int  re_addr;
char  data[64];
int  ret;
int  byte_count, len;
char  filename[128];
FILE  *fp;

if (argc < 2) {
    usage ();
    return -1;
}

/* Loop through command line arguments */
for (int i = 0; i < argc; i++) {
    if (!strncmp(argv[i], "--device=", 9)) {
        usb_bus_dev = argv[i] + 9;
    }

    if (!strncmp(argv[i], "-d=", 3)) {
        usb_bus_dev = argv[i] + 3;
    }

    if (!strcmp("--quadrant", argv[i]) || !strcmp("-q", argv[i])) {
        quadrant_p = true;
        sscanf (argv[i+1], "%d", &quadrant);
    }

    if (!strcmp("--position", argv[i]) || !strcmp("-p", argv[i])) {
        position_p = true;
        sscanf (argv[i+1], "%d", &position);
    }

    if (!strcmp("--master", argv[i]) || !strcmp("-m", argv[i])) {
        master_p = true;
        sscanf (argv[i+1], "%d", &master);

        if (master)
            master = 1;
    }

    if (!strcmp("--read", argv[i]) || !strcmp("-r", argv[i])) {
        read_p = true;
    }

    if (!strcmp("--quickread", argv[i]) ) {
        quick_read_p = true;
    }

    if (!strcmp("--save", argv[i]) || !strcmp("-s", argv[i])) {
        save_p = true;

        if (argc < i + 2) {
            std::cerr << "You must specify a valid filename!" << std::endl;
            return -1;
        }

        sscanf (argv[i+1], "%s", &filename);
    }

    if (!strcmp("--list", argv[i]) || !strcmp("-l", argv[i])) {
        list_only_p = true;
    }

    if (!strcmp("--blank", argv[i]) ) {
        blank_p = true;
    }

    if (!strcmp("--write_eeeprom", argv[i]) || !strcmp("-w", argv[i])) {
        if (argc < (i+3)) {
            std::cerr << "Memory location and value must be specified!" << std::endl;

```

---



---

```

        return -1;
    }

    write_eeprom_p = true;
    sscanf (argv[i+1], "%d", &we_addr);
    sscanf (argv[i+2], "%d", &we_data);
    fprintf (stdout, "WRITE EEPROM ADDRESS: ADDR = 0x%02x, DATA = 0x%02x\n\n", we_addr, we_data);
    i+=2;
}

if (!strcmp("--read_eeprom", argv[i]) || !strcmp ("-rb", argv[i])) {
    if (argc < (i+2)) {
        std::cerr << "Memory location must be specified!" << std::endl;
        return -1;
    }

    read_eeprom_p = true;
    sscanf (argv[i+1], "%d", &re_addr);
    i++;
}

if (!strcmp("--reload_params", argv[i]) || !strcmp ("-rp", argv[i])) {
    reload_params_p = true;
}
}

/* Initialize libusb */
usb_init ();
usb_find_busses ();
usb_find_devices ();

if (list_only_p) {
    show_bus ();
    return -1;
}

if (usb_bus_dev != NULL) {
    /* Extract BUS and DEVICE from command line arguments */
    const char *p = strchr (usb_bus_dev, '.');
    if (!p) {
        std::cerr << "Illegal/nonexistent device " << usb_bus_dev << "." << std::endl;
        return -1;
    }
    strncpy (bus, usb_bus_dev, p - usb_bus_dev);
    bus[p - usb_bus_dev] = '\0';
    strcpy (dev, p+1);
}

else {
    std::cerr << "USB bus and device not specified!" << std::endl;
    usage ();
    return -1;
}

struct usb_device *usb_dev;
/* Find device at location specified */
usb_dev = find_device (bus, dev);

if (usb_dev == NULL) {
    std::cerr << "Unable to find device!" << std::endl;
    return -1;
}

std::cout << "Device Found. (ID: " << std::hex;
std::cout << usb_dev->descriptor.idVendor;
std::cout << ":" << usb_dev->descriptor.idProduct << ")" << std::endl;

struct usb_dev_handle *usb_hdl;

usb_hdl = usb_open (usb_dev);
if (!usb_hdl) {
    std::cerr << "Error Opening Device: " << usb_strerror() << std::endl;
    return -1;
}

```

---

---

```

if (usb_set_configuration (usb_hdl, 1) < 0) {
    std::cerr << "Error setting configuration: " << usb_strerror() << std::endl;
    return -1;
}

if (usb_claim_interface (usb_hdl, 0) < 0) {
    std::cerr << "Error claiming interace: " << usb_strerror() << std::endl;
    return -1;
}

if (usb_set_altinterface (usb_hdl, 0) < 0) {
    std::cerr << "Error setting alternative interface: " << usb_strerror() << std::endl;
    return -1;
}

if (save_p) {
    fp = fopen (filename, "wb");

    if (fp == NULL) {
        std::cerr << "Error opening file: " << filename << std::endl;
        usb_close (usb_hdl);
        return -1;
    }
}

/* Read Contents */
if (read_p | save_p) {
    /* Read Contents of Serial EEPROM - VRQ_EEPROM_READ_LG (16-bit address)
     * wValue: Start Address
     * wLength: Length of data (64 bytes max because using EP0)
     */
    /* Read ALL data */
    len = EEPROM_SIZE;
    while (len) {
        if (len < 64)
            byte_count = len;
        else
            byte_count = 64;

        ret = usb_control_msg (usb_hdl,
            VENDOR_REQUEST_IN,
            VRQ_EEPROM_READ_LG,
            (EEPROM_SIZE - len),
            0,
            data,
            byte_count,
            1500);
        if (ret < 0) {
            std::cerr << "Error Reading EEPROM Data: " << usb_strerror() << std::endl;
            usb_close (usb_hdl);
            return -1;
        }

        if (read_p) {
            for (int i = 0; i < byte_count; i++) {
                //std::cout << std::hex << data[i] << " ";
                fprintf (stdout, "%02x ", (unsigned char) data[i]);
                if (!(i+1) % 16) {
                    std::cout << std::endl;
                }
            }
        }

        if (save_p) {
            fwrite (data, byte_count, 1, fp);
        }

        len -= byte_count;
    }

    if (save_p) {
        fclose (fp);
    }
}

```

---

```
std::cout << std::endl;
usb_close (usb_hdl);
return 0;
}

if (blank_p) {
    int err_count = 0;

    for (int i = 0; i < 64; i++)
        data[i] = 0xff;

    for (int i = 0; i < (EEPROM_SIZE/64) ; i++) {
        ret = usb_control_msg (usb_hdl,
            VENDOR_REQUEST_OUT,
            VRO_EEPROM_WRITE_LG,
            i*64,
            0,
            (char *) data,
            64,
            1500);

        if (ret < 0) {
            fprintf (stderr, "Error writing data to EEPROM - %s\n", usb_strerror());
            err_count++;
        }

        fprintf (stdout, "%.4g%% complete.\n", ((double) (i+1) / (double) (EEPROM_SIZE/64))*100.0);
    }

    if (err_count)
        fprintf (stderr, "\n\nErrors occurred while blanking EEPROM!\n");
    else
        fprintf (stdout, "\n\nBlanking EEPROM successful!\n");

    usb_close (usb_hdl);
    return 0;
}

if (quick_read_p) {
    /* Read Quadrant, Position and Master flag from EEPROM */
    ret = usb_control_msg (usb_hdl,
        VENDOR_REQUEST_IN,
        VRO_EEPROM_READ_LG,
        EEPROM_QUADPOS_START_ADDR,
        0,
        (char *)data,
        EEPROM_QUADPOS_LENGTH,
        500);

    /* verify data */
    if ( (unsigned) data[0] > 4) {
        fprintf (stderr, "Camera not configured!\n");
        usb_close (usb_hdl);
        return (-1);
    }

    if ( (unsigned) data[1] > 4) {
        fprintf (stderr, "Invalid Position!\n");
        usb_close (usb_hdl);
        return (-1);
    }

    std::cout << "Quadrant:      " << (int) data[0] << std::endl;
    std::cout << "Position:      " << pos_str[data[1]] << std::endl;
    std::cout << "Master:        " << ( (data[2]) ? "Yes" : "No") << std::endl;
    usb_close (usb_hdl);
    return 0;
}

if (write_eeprom_p) {
    ret = write_eeprom (usb_hdl, we_addr, &we_data, 1);
    if (ret < 0)
        std::cerr << "Error writing EEPROM data!" << std::endl;
    else

```

---

```

        std::cout << "Data successfully written to EEPROM..." << std::endl;

        usb_close (usb_hdl);
        return 0;
    }

    if (read_eeeprom_p) {
        ret = usb_control_msg (usb_hdl,
            VENDOR_REQUEST_IN,
            VRQ_EEPROM_READ_LG,
            re_addr,
            0,
            (char *)data,
            1,
            500);

        fprintf (stdout, "0x%02x: %d\n\n", re_addr, (unsigned char) data[0]);
        usb_close (usb_hdl);
        return 0;
    }

    if (reload_params_p) {
        ret = usb_control_msg (usb_hdl,
            VENDOR_REQUEST_IN,
            VRQ_GET_WINDOW_PARAM,
            0,
            VRQ_UPDATE_PARAMS,
            (char *)data,
            1,
            500);
        if (ret < 0)
            fprintf (stderr, "Error reloading window parameters: %s\n", usb_strerror());
        else {
            if (data[0] != 0x08)
                fprintf (stderr, "Error reloading window parameters: ACK not received!\n");
            else
                fprintf (stdout, "Successfully reloaded window parameters...\n");
        }

        usb_close (usb_hdl);
        return 0;
    }

    if (!(quadrant_p && position_p)) {
        std::cerr << "Quadrant and Position must be specified at command line!" << std::endl;
        usb_close (usb_hdl);
        return -1;
    }
    else {
        std::cout << "Quadrant:      " << quadrant << std::endl;
        std::cout << "Position:    " << pos_str[position] << std::endl;
        std::cout << "Master:      " << ( (master) ? "Yes" : "No") << std::endl;
    }

    /* Write Header Data */
    ret = usb_control_msg (usb_hdl,
        VENDOR_REQUEST_OUT,
        VRQ_EEPROM_WRITE_LG,
        0,
        0,
        (char *) EEPROM_HEADER,
        EEPROM_HEADER_SIZE,
        500);

    if (ret < 0) {
        std::cerr << "Error writing to EEPROM: " << usb_strerror() << std::endl;
        usb_close (usb_hdl);
        return -1;
    }

    /* Write Position and Quadrant Data - Stored at address - 0x0010 */
    data[0] = quadrant;
    data[1] = position;
    data[2] = master;

```

---

```
ret = usb_control_msg (usb_hdl,
    VENDOR_REQUEST_OUT,
    VRQ_EEPROM_WRITE_LG,
    0x10,
    0,
    (char *) data,
    0x3,
    500);

if (ret < 0) {
    std::cerr << "Error writing to EEPROM: " << usb_strerror() << std::endl;
    usb_close (usb_hdl);
    return -1;
}

usb_close (usb_hdl);

return 0;
}
```

## E.4 fpga\_loader\_ss

This application is used to load a .bin/.bit FPGA configuration file generated using Xilinx ISE to the FPGA of the USB2.0 camera.

### E.4.1 Makefile

```
TOP_SRC    = ../..
CC          = g++
CFLAGS      = -O2
LDFLAGS     = -lusb
INCLUDE     = -I$(TOP_SRC)/firmware/fx2/firmware/include
CLEANFILES = fpga_loader_ss

all: fpga_loader_ss

fpga_loader_ss: fpga_loader_ss.cc
    $(CC) $(CFLAGS) $(LIB) $(LDFLAGS) $(INCLUDE) $(OBJS) $(DEFS) -o $@

clean:
    rm $(CLEANFILES)

# Dependencies
fpga_loader_ss: $(TOP_SRC)/firmware/fx2/firmware/include/fx2cam_commands.h
```

### E.4.2 fpga\_loader\_ss.cc

```
/*
 * Filename:
 *   fpga_load_ss.cc
 *
 * Description:
 *   application to load FPGA firmware using slave-serial method.
 *
 * Author:
 *   Neil Scott
 *
 * Date:
 *   May 4, 2007
 *   May 21, 2007 - Added error handling for unspecified bit file
 */

#include <stdio.h>
```

---

```

#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>
#include <usb.h>

#include "fx2cam_commands.h"
#include "fx2cam_i2c_addr.h"

#define DEBUG 0

#define VERSION "0.01"

#define FX2_MAX_DEVICES 16
#define FX2_CAM_VENDOR_ID 0xabcd
#define FX2_CAM_PRODUCT_ID 0x0201

#define FPGA_POWER_ON 0x01
#define FPGA_POWER_OFF 0x00

#define FPGA_RESET_ENABLE 0x01
#define FPGA_RESET_DISABLE 0x00

/* Global Variables */
char *app_title={"Xilinx Slave-Serial FPGA Loader"};

void
show_bus (void)
{
    for (usb_bus *bus = usb_busses; bus; bus = bus->next) {
        for (struct usb_device *dev = bus->devices; dev; dev = dev->next) {
            fprintf (stdout, "bus %s dev %s: ID %04x:%04x",
                    bus->dirname,
                    dev->filename,
                    dev->descriptor.idVendor,
                    dev->descriptor.idProduct);

            if (dev->descriptor.idVendor == FX2_CAM_VENDOR_ID &&
                dev->descriptor.idProduct == FX2_CAM_PRODUCT_ID) {
                fprintf (stdout, " <--");
            }
            fprintf (stdout, "\n");
        }
    }
}

struct usb_device *
find_device (const char *bus, const char *dev)
{
    for (usb_bus *b = usb_busses; b; b = b->next) {
        if (!strcmp (b->dirname, bus)) {
            for (struct usb_device *d = b->devices; d; d = d->next) {
                if (!strcmp (d->filename, dev))
                    return (d);
            }
        }
    }

    return (NULL);
}

void
usage (void)
{
    fprintf (stderr, "Usage: fpga_load_ss [OPTIONS]... [BIT FILE]\n");
    fprintf (stderr, "Options:\n");
    fprintf (stderr, "  -h, --help           Display this help screen\n");
    fprintf (stderr, "  -v, --verbose        Increase verbosity\n");
    fprintf (stderr, "  -d, --device=BUS.DEV Select the bus and device to use\n");
    fprintf (stderr, "  -l, --list           List the devices on the USB bus\n");
    fprintf (stderr, "  -p, --power <state> Soft power control to FPGA [on / off]\n");
    fprintf (stderr, "  -r, --reset          Soft reset to FPGA (once configured)\n");
}

```

---

---

```

    fprintf (stderr, " -V, --version           Display version information\n");
    fprintf (stderr, "\n\n");
    fprintf (stderr, "[BIT FILE] is the path to the BIT file generated using ISE\n");
    fprintf (stderr, " 'path/foo.bit'\n");
}

void
print_version (void)
{
    fprintf (stdout, "%s v%s\n", app_title, VERSION);
    fprintf (stdout, "2007 Neil Scott, University of Windsor\n");
}

int
main (char argc, char **argv)
{
    int          ret;
    char         data[64];
    struct       usb_bus *usb_bus;
    struct       usb_device *fx2_dev_list[FX2_MAX_DEVICES];
    struct       usb_dev_handle *usb_hdl;
    struct       usb_device *usb_dev;
    int          dev_count = 0;
    int          h, i;
    FILE         *fp;
    char         filename[80];
    int          fsize, byte_count, rval;
    bool         p_verbose = false;
    bool         p_list_only = false;
    bool         p_fpga_power_only = false;
    bool         p_fpga_reset_only = false;
    int          fpga_power_state;
    char         *usb_bus_dev = NULL;
    int          option_count = 0;
    char         msg[80];

    /* Check argument count */
    if (argc < 2) {
        usage ();
        return (1);
    }

    /* Parse command line arguments */
    for (int i = 0; i < argc; i++) {
        if ( (!strcmp (argv[i], "--help")) || (!strcmp (argv[i], "-h")) ) {
            usage ();
            return 1;
        }

        if ( (!strcmp (argv[i], "--version")) || (!strcmp (argv[i], "-V")) ) {
            print_version ();
            return 1;
        }

        if ( (!strcmp (argv[i], "--verbose")) || (!strcmp (argv[i], "-v")) ) {
            p_verbose = true;
            option_count++;
        }

        if ( (!strcmp (argv[i], "--list")) || (!strcmp (argv[i], "-l")) ) {
            p_list_only = true;
            option_count++;
        }

        if (!strcmp (argv[i], "-d=", 3)) {
            usb_bus_dev = argv[i] + 3;
            option_count++;
        }

        if (!strcmp (argv[i], "--device=", 9)) {
            usb_bus_dev = argv[i] + 9;
            option_count++;
        }
    }

```

---

```

    }

    if ( (!strcmp (argv[i], "--power")) || (!strcmp (argv[i], "-p")) ) {
        if ( (!strcmp (argv[i+1], "on")) || (!strcmp (argv[i+1], "off")) ) {
            p_fpga_power_only = true;
            if (!strcmp (argv[i+1], "on")) {
                fpga_power_state = FPGA_POWER_ON;
            }
            else {
                fpga_power_state = FPGA_POWER_OFF;
            }
        }
        i++;
        option_count++;
    }
}

if ( (!strcmp (argv[i], "--reset")) || (!strcmp (argv[i], "-r")) ) {
    p_fpga_reset_only = true;
    option_count++;
}
}

/* Initialize libusb */
usb_init();
usb_find_busses();
usb_find_devices();

/* Print Application Name */
if (p_verbose) {
    sprintf (msg, "%s (v%s)\n", app_title, VERSION);

    fprintf (stdout, "%s", msg);
    for (int j = 0; j < strlen (msg) - 1; j++) {
        fprintf (stdout, "=");
    }
    fprintf (stdout, "\n\n");
}

/* Check if only listing bus */
if (p_list_only) {
    show_bus ();
    return (0);
}

/* Ensure a bus and device were specified */
if (usb_bus_dev == NULL) {
    fprintf (stderr, "You must specify a device to program!\n");
    return (1);
}

/* Search the USB bus for the Device Specified */
char bus[FX2_MAX_DEVICES], dev[FX2_MAX_DEVICES];

const char *p = strchr (usb_bus_dev, '.');
if (!p) {
    fprintf (stderr, "Illegal/nonexistent device %s.\n", usb_bus_dev);
    return (1);
}

strcpy (bus, usb_bus_dev, p - usb_bus_dev);
bus[p - usb_bus_dev] = '\0';
strcpy (dev, p+1);
usb_dev = find_device (bus, dev);

if (usb_dev == NULL) {
    fprintf (stderr, "Illegal/nonexistent device: %s.\n", usb_bus_dev);
    return (1);
}

if (p_verbose) {
    fprintf (stdout, "Device Found. (ID: %04x:%04x)\n",
        usb_dev->descriptor.idVendor,
        usb_dev->descriptor.idProduct);
}

```



---

```

    }

    if (p_verbose) {
        fprintf (stdout, "Claiming Device...\n");
    }

    usb_hdl = usb_open (usb_dev);
    if (!usb_hdl) {
        fprintf (stderr, "Error Opening Device: %s\n", usb_strerror() );
        return (1);
    }

    /* Set Configuration */
    if (usb_set_configuration (usb_hdl, 1) < 0) {
        fprintf (stderr, "Error setting configuration: %s\n", usb_strerror() );
        return (1);
    }

    /* Claim Device */
    if (usb_claim_interface (usb_hdl, 0) < 0) {
        fprintf (stderr, "Error claiming device interface: %s\n", usb_strerror() );
        return (1);
    }

    /* Set Alt interface */
    if (usb_set_altinterface (usb_hdl, 0) < 0) {
        fprintf (stderr, "Error setting alternative interface: %s\n", usb_strerror() );
        return (1);
    }

    /* FPGA Power Only */
    if (p_fpga_power_only) {
        /* Control Message to FX2 to Power on/off FPGA */
        ret = usb_control_msg ( usb_hdl,
            VENDOR_REQUEST_IN,
            VRQ_FPGA_POWER,
            fpga_power_state,
            0,
            data,
            1,
            500);

        if (ret < 0) {
            fprintf (stderr, "Error setting FPGA power state: %s\n",
                (fpga_power_state == FPGA_POWER_ON) ? "on" : "off");
        }
        else {
            if (data[0] == 0x08) {
                fprintf (stdout, "FPGA power state set: %s\n",
                    (fpga_power_state == FPGA_POWER_ON) ? "on" : "off");
            }
            else {
                fprintf (stderr, "Error setting FPGA power state: NACK received\n");
            }
        }
    }

    usb_close (usb_hdl);
    return 0;
}

/* FPGA Reset Only */
if (p_fpga_reset_only) {
    /* Control Message to FX2 to Soft-Reset FPGA */
    ret = usb_control_msg ( usb_hdl,
        VENDOR_REQUEST_IN,
        VRQ_FPGA_RESET,
        FPGA_RESET_ENABLE,
        0,
        data,
        1,
        500);

    if (ret < 0) {
        fprintf (stderr, "Error on FPGA soft-reset\n");
    }
}

```

---

---

```

    }
    else if (data[0] == 0x08) {
        fprintf (stdout, "FPGA successfully put into reset\n");
    }
    else {
        fprintf (stderr, "Error on FPGA soft-reset\n");
    }
}

usleep (250000);          /*250ms delay */

ret = usb_control_msg ( usb_hdl,
    VENDOR_REQUEST_IN,
    VRQ_FPGA_RESET,
    FPGA_RESET_DISABLE,
    0,
    data,
    1,
    500);

if (ret < 0) {
    printf (stderr, "Error on FPGA soft-reset\n");
}
else if (data[0] == 0x08) {
    fprintf (stdout, "FPGA successfully taken out of reset\n");
}
else {
    printf (stderr, "Error on FPGA soft-reset\n");
}

if (p_verbose)
    fprintf (stdout, "Closing USB device...\n");

/* Close USB handle */
usb_close (usb_hdl);
return 0;
}

/* Check for program file */
if (option_count > (argc - 2) ) {
    fprintf (stderr, "Bit File not specified!\n");
    usage ();
    usb_close (usb_hdl);
    return 1;
}

/* Get File */
if (!p_list_only) {
    strcpy (filename, argv[argc-1]);

    fp = fopen (filename, "rb");

    if (fp == NULL) {
        fprintf (stderr, "Error opening file (%s)!\n");
        usb_close (usb_hdl);
        return 1;
    }
}

/* get bit file size */
fseek (fp, 0, SEEK_END);
fsize = ftell (fp);
rewind (fp);

if (p_verbose) fprintf (stdout, "File Size: %dkb (%d bytes)\n", (fsize+1) / 1024, fsize+1);

if (p_verbose) fprintf (stdout, "Starting FPGA Configuration...\n");

if (p_verbose) fprintf (stdout, "Enabling Power to FPGA...\n");

/* send command to enable power to FPGA */
ret = usb_control_msg ( usb_hdl,
    VENDOR_REQUEST_IN,
    VRQ_FPGA_POWER,
    FPGA_POWER_ON,
    0,
    data,
    1,
    500);

```

---

---

```

    0,
    data,
    1,
    500);

if (ret < 0)
    fprintf (stderr, "ERROR: usb_control_msg() - %s\n", usb_strerror() );
else {
    /* ensure FPGA was powered */
    if (data[0] != 0x08) {
        fprintf (stderr, "Acknowledge not received, unable to power FGPA!\n");
        usb_close (usb_hdl);
        return (1);
    }
}

/* Allow FPGA to power up */
usleep (250000);

if (p_verbose) fprintf (stdout, "Holding FPGA in reset...\n");

/* send command to hold FPGA in reset after configuration */
ret = usb_control_msg ( usb_hdl,
    VENDOR_REQUEST_IN,
    VRQ_FPGA_RESET,
    FPGA_RESET_ENABLE,
    0,
    data,
    1,
    500);

if (ret < 0)
    fprintf (stderr, "ERROR: usb_control_msg() - %s\n", usb_strerror() );
else {
    /* ensure FPGA was powered */
    if (data[0] != 0x08) {
        fprintf (stderr, "Acknowledge not received, unable to hold FGPA in reset!\n");
        usb_close (usb_hdl);
        return (1);
    }
}

/* send fpga load start to FX2 */
ret = usb_control_msg ( usb_hdl,
    VENDOR_REQUEST_IN,
    VRQ_FPGA_LOAD_SS,
    FPGA_LOAD_START,
    data,
    2,
    500);

if (ret < 0)
    fprintf (stderr, "ERROR: usb_control_msg() - %s\n", usb_strerror() );

/* check return to ensure start was successful */
if (data[1] != 0x08) {
    fprintf (stderr, "Acknowledge not received!\n");
}
if (data[0] != 0x01) {
    fprintf (stderr, "Error starting serial mode configuration!\n");
    usb_close (usb_hdl);
    return (1);
}

byte_count = fsize;
bool        p_toggle = false;
int         progress_update;
int         progress_count = 0;

progress_update = (fsize / 64) / 50;

if (p_verbose) {
    fprintf (stdout, "Configuration Progress:\n");

```

---

---

```

    fprintf (stdout, "|0-----25-----50-----75-----100|\n ");
}

/* build 64 byte packets to send to FX2 */
while (byte_count > 64) {
    if ( (rval = fread (data, 1, 64, fp)) > 0) {
        ret = usb_control_msg ( usb_hdl,
            VENDOR_REQUEST_OUT,
            VRQ_FPGA_LOAD_SS,
            rval,
            FPGA_LOAD_DATA,
            data,
            rval,
            1500);

        if (ret < 0)
            fprintf (stderr, "ERROR on VRQ_FPGA_LOAD_SS - FPGA_LOAD_DATA: usb_control_msg() - %s\n",
                usb_strerror() );

        if (p_verbose) {
            if (progress_count == progress_update) {
                fprintf (stdout, "=");
                fflush (stdout);
                progress_count = 0;
            }
            else
                progress_count++;
        }
    }
    byte_count -= rval;

    if (DEBUG) fprintf (stdout, "byte_count = %d\n", byte_count);

    usleep(5);
}

if (byte_count) {
    if ( (rval = fread (data, 1, byte_count, fp)) > 0) {
        ret = usb_control_msg ( usb_hdl,
            VENDOR_REQUEST_OUT,
            VRQ_FPGA_LOAD_SS,
            rval,
            FPGA_LOAD_DATA,
            data,
            rval,
            200);
        if (ret < 0)
            fprintf (stderr, "ERROR on VRQ_FPGA_LOAD_SS - FPGA_LOAD_DATA: usb_control_msg() - %s\n",
                usb_strerror() );

        byte_count -= rval;

        if (DEBUG) fprintf (stdout, "byte_count = %d\n", byte_count);
    }
}

if (p_verbose) fprintf(stdout, "\n");

usleep(200000);

/* ensure the done bit is set */
ret = usb_control_msg ( usb_hdl,
    VENDOR_REQUEST_IN,
    VRQ_FPGA_LOAD_SS,
    0,
    FPGA_LOAD_CHECK_DONE,
    data,
    2,
    30000);

if (ret < 0)

```

---

```
fprintf (stderr, "ERROR on VRQ_FPGA_LOAD_SS - FPGA_LOAD_CHECK_DONE: usb_control_msg() - %s\n",
        usb_strerror() );

/* check return */
if (data[1] != 0x08) {
    fprintf (stderr, "Acknowledge not received!\n");
}
if (data[0] != 0x01) {
    fprintf (stderr, "Done indication not received by device!\n");
}
else {
    if (p_verbose) fprintf (stdout, "Programming Successful!\n");
}

/* Control Message to FX2 to Soft-Reset FPGA */
ret = usb_control_msg ( usb_hdl,
    VENDOR_REQUEST_IN,
    VRQ_FPGA_RESET,
    FPGA_RESET_ENABLE,
    0,
    data,
    1,
    500);
if (ret < 0) {
    fprintf (stderr, "Error on FPGA soft-reset\n");
}
else if (data[0] == 0x08) {
    fprintf (stdout, "FPGA successfully put into reset\n");
}
else {
    fprintf (stderr, "Error on FPGA soft-reset\n");
}

usleep (250000);          /*250ms delay */

ret = usb_control_msg ( usb_hdl,
    VENDOR_REQUEST_IN,
    VRQ_FPGA_RESET,
    FPGA_RESET_DISABLE,
    0,
    data,
    1,
    500);

if (ret < 0) {
    fprintf (stderr, "Error on FPGA soft-reset\n");
}
else if (data[0] == 0x08) {
    fprintf (stdout, "FPGA successfully taken out of reset\n");
}
else {
    fprintf (stderr, "Error on FPGA soft-reset\n");
}

if (p_verbose) fprintf (stdout, "Closing USB device...\n");

usb_close (usb_hdl);

return 0;
}
```

## E.5 pyWindowConfig

### E.5.1 pyWindowConfig.py

```
#!/usr/bin/env python

import pygtk
import gtk
import os
```

---

```

import pexpect

PM_EEPROM_WIDTH_HIGH_ADDR = 0x20
PM_EEPROM_WIDTH_LOW_ADDR = 0x21
PM_EEPROM_LENGTH_HIGH_ADDR = 0x22
PM_EEPROM_LENGTH_LOW_ADDR = 0x23
PM_EEPROM_COL_START_HIGH_ADDR = 0x24
PM_EEPROM_COL_START_LOW_ADDR = 0x25
PM_EEPROM_ROW_START_HIGH_ADDR = 0x26
PM_EEPROM_ROW_START_LOW_ADDR = 0x27
PM_EEPROM_COL_SKIP_ADDR = 0x28
PM_EEPROM_ROW_SKIP_ADDR = 0x29

USBlist=[None]*16
USBposition=[None]*16
cmd=''
options=''

class gEEPROMConfig:
    def on_close(self, widget, event, data=None):
        gtk.main_quit()
        return False

    def list_usb (self):
        global USBposition
        x = os.popen ('lsusb', "r")
        y = 0;
        found = 0;
        while 1:
            line = x.readline()
            line = line.rstrip()
            USBlist[y] = line

            if line.count('abcd'):
                USBlist[found] = line
                print found
                print USBlist[found]

                # Get position
                bus = USBlist[found][4:7]
                dev = USBlist[found][15:18]
                z = os.popen('./cam_init -d=' + bus + '.' + dev + ' --quickread')
                quad = z.readline()
                quad = z.readline()
                quad = quad[14:]
                pos = z.readline()
                pos = pos[14:]
                pos = pos.rstrip()
                USBposition[found] = 'Q' + quad + ':' + pos

                print 'Q:' + quad
                print 'P:' + pos

                self.USBcbUSB.insert_text(found, line + ' - [Q' + quad + ':' + pos + ']\n')
                found = found + 1

            y = y+1

            if not line: break;
        if not found:
            self.error = gtk.MessageDialog(self.options, gtk.DIALOG_MODAL, gtk.MESSAGE_INFO,
                                           gtk.BUTTONS_OK, 'No Devices Found!')
            self.error.connect("response", self.on_close)
            self.error.show_all()

        else:
            self.USBcbUSB.set_active(0)
            self.usblist.show_all()

    def read_eeprom_data (self):
        global cmd

        opts = ' -rb ' + str(PM_EEPROM_WIDTH_HIGH_ADDR)
        z = os.popen (cmd + opts)

```

---

```
r = z.readline()
r = z.readline()
val_hi = int(r[5:])

opts = ' -rb ' + str(PM_EEPROM_WIDTH_LOW_ADDR)
z = os.popen (cmd + opts)
r = z.readline()
r = z.readline()
val_lo = int(r[5:])

val = (val_hi << 8) + val_lo
print 'RESULT: ' + str(val)
self.OPTentryWidth.set_text(str(val))

opts = ' -rb ' + str(PM_EEPROM_LENGTH_HIGH_ADDR)
z = os.popen (cmd + opts)
r = z.readline()
r = z.readline()
val_hi = int(r[5:])

opts = ' -rb ' + str(PM_EEPROM_LENGTH_LOW_ADDR)
z = os.popen (cmd + opts)
r = z.readline()
r = z.readline()
val_lo = int(r[5:])

val = (val_hi << 8) + val_lo
print 'RESULT: ' + str(val)
self.OPTentryLength.set_text(str(val))

opts = ' -rb ' + str(PM_EEPROM_COL_START_HIGH_ADDR)
z = os.popen (cmd + opts)
r = z.readline()
r = z.readline()
val_hi = int(r[5:])

opts = ' -rb ' + str(PM_EEPROM_COL_START_LOW_ADDR)
z = os.popen (cmd + opts)
r = z.readline()
r = z.readline()
val_lo = int(r[5:])

val = (val_hi << 8) + val_lo
print 'RESULT: ' + str(val)
self.OPTentryXStart.set_text(str(val))

opts = ' -rb ' + str(PM_EEPROM_ROW_START_HIGH_ADDR)
z = os.popen (cmd + opts)
r = z.readline()
r = z.readline()
val_hi = int(r[5:])

opts = ' -rb ' + str(PM_EEPROM_ROW_START_LOW_ADDR)
z = os.popen (cmd + opts)
r = z.readline()
r = z.readline()
val_lo = int(r[5:])

val = (val_hi << 8) + val_lo
print 'RESULT: ' + str(val)
self.OPTentryYStart.set_text(str(val))

opts = ' -rb ' + str(PM_EEPROM_COL_SKIP_ADDR)
z = os.popen (cmd + opts)
r = z.readline()
r = z.readline()
val = int(r[5:])
```

---

```

print 'RESULT: ' + str(val)
self.OPTentryXBin.set_text(str(val))

opts = ' -rb ' + str(PM_EEPROM_ROW_SKIP_ADDR)
z = os.popen (cmd + opts)
r = z.readline()
r = z.readline()
val = int(r[5:])

print 'RESULT: ' + str(val)
self.OPTentryYBin.set_text(str(val))

def options_on_ok (self, widget):
    #write eeprom data
    global cmd, options

    val = int(self.OPTentryWidth.get_text())
    val_hi = val >> 8
    val_lo = val & 0xff
    options = ' -w ' + str(PM_EEPROM_WIDTH_HIGH_ADDR) + ' ' + str(val_hi)
    print cmd + options
    os.system(cmd + options)
    options = ' -w ' + str(PM_EEPROM_WIDTH_LOW_ADDR) + ' ' + str(val_lo)
    print cmd + options
    os.system(cmd + options)

    val = int(self.OPTentryLength.get_text())
    val_hi = val >> 8
    val_lo = val & 0xff
    options = ' -w ' + str(PM_EEPROM_LENGTH_HIGH_ADDR) + ' ' + str(val_hi)
    print cmd + options
    os.system(cmd + options)
    options = ' -w ' + str(PM_EEPROM_LENGTH_LOW_ADDR) + ' ' + str(val_lo)
    print cmd + options
    os.system(cmd + options)

    val = int(self.OPTentryXStart.get_text())
    val_hi = val >> 8
    val_lo = val & 0xff
    options = ' -w ' + str(PM_EEPROM_COL_START_HIGH_ADDR) + ' ' + str(val_hi)
    os.system(cmd + options)
    print cmd + options
    options = ' -w ' + str(PM_EEPROM_COL_START_LOW_ADDR) + ' ' + str(val_lo)
    print cmd + options
    os.system(cmd + options)

    val = int(self.OPTentryYStart.get_text())
    val_hi = val >> 8
    val_lo = val & 0xff
    options = ' -w ' + str(PM_EEPROM_ROW_START_HIGH_ADDR) + ' ' + str(val_hi)
    print cmd + options
    os.system(cmd + options)
    options = ' -w ' + str(PM_EEPROM_ROW_START_LOW_ADDR) + ' ' + str(val_lo)
    print cmd + options
    os.system(cmd + options)

    val = int(self.OPTentryXBin.get_text())
    options = ' -w ' + str(PM_EEPROM_ROW_SKIP_ADDR) + ' ' + str(val)
    print cmd + options
    os.system(cmd + options)

    val = int(self.OPTentryYBin.get_text())
    options = ' -w ' + str(PM_EEPROM_COL_SKIP_ADDR) + ' ' + str(val)
    print cmd + options
    os.system(cmd + options)

    #reload EEPROM data
    options = ' -rp'
    os.system(cmd + options)

```

---



---

```

def options_on_cancel (self, widget):
    gtk.main_quit()

def on_error_ok (self, widget, event):
    self.error.hide_all()

def usb_on_ok (self, widget):
    global cmd
    global USBposition

    selected = self.USBcbUSB.get_active()
    bus = USBlist[selected][4:7]
    dev = USBlist[selected][15:18]

    cmd = './cam_init -d=' + bus + ',' + dev

    self.options.set_title(self.options.get_title() + ' - [' + USBposition[selected] + ']')
    self.read_eeprom_data()

    self.usblist.hide_all()

def __init__(self):
    self.threads = 0
    self.options = gtk.Window()
    self.usblist = gtk.Window()
    self.usblist.set_modal(True)
    self.usblist.set_transient_for(self.options)

    self.options.connect("delete_event", self.on_close, None)
    self.options.connect("destroy", self.on_close, None)

    #Options List
    self.options.set_border_width(5)
    self.options.set_title("Window Options")

    #Frame
    self.OPTvbTop = gtk.VBox(False, 5)
    self.options.add(self.OPTvbTop)

    self.OPTframe = gtk.Frame(label="Window Options")
    self.OPTvbTop.add (self.OPTframe)

    self.OPTvbFrame = gtk.VBox(False, 5)
    self.OPTframe.add (self.OPTvbFrame)

    #width
    self.OPThbWidth = gtk.HBox(True, 5)
    self.OPTvbFrame.add (self.OPThbWidth)

    self.OPTlblWidth = gtk.Label("Width")
    self.OPThbWidth.add (self.OPTlblWidth)

    self.OPTentryWidth = gtk.Entry()
    self.OPThbWidth.add (self.OPTentryWidth)
    #self.OPTentryWidth.set_text('2048')

    #length
    self.OPThbLength = gtk.HBox(True, 5)
    self.OPTvbFrame.add (self.OPThbLength)

    self.OPTlblLength = gtk.Label("Length")
    self.OPThbLength.add (self.OPTlblLength)

    self.OPTentryLength = gtk.Entry()
    self.OPThbLength.add (self.OPTentryLength)
    #self.OPTentryLength.set_text('1536')

    #xstart
    self.OPThbXStart = gtk.HBox(True, 5)
    self.OPTvbFrame.add (self.OPThbXStart)

    self.OPTlblXStart = gtk.Label("X-Start")
    self.OPThbXStart.add (self.OPTlblXStart)

```

---

---

```

self.OTPentryXStart = gtk.Entry()
self.OTPthbXStart.add (self.OTPentryXStart)
self.OTPentryXStart.set_text('28')

#ystart
self.OTPthbYStart = gtk.HBox(True, 5)
self.OTVbFrame.add (self.OTPthbYStart)

self.OTlbyYStart = gtk.Label("Y-Start")
self.OTPthbYStart.add (self.OTlbyYStart)

self.OTPentryYStart = gtk.Entry()
self.OTPthbYStart.add (self.OTPentryYStart)
self.OTPentryYStart.set_text('16')

#xbin
self.OTPthbXBin = gtk.HBox(True, 5)
self.OTVbFrame.add (self.OTPthbXBin)

self.OTlbyXBin = gtk.Label("X-Bin")
self.OTPthbXBin.add (self.OTlbyXBin)

self.OTPentryXBin = gtk.Entry()
self.OTPthbXBin.add (self.OTPentryXBin)
self.OTPentryXBin.set_text('1')

#ybin
self.OTPthbYBin = gtk.HBox(True, 5)
self.OTVbFrame.add (self.OTPthbYBin)

self.OTlbyYBin = gtk.Label("Y-Bin")
self.OTPthbYBin.add (self.OTlbyYBin)

self.OTPentryYBin = gtk.Entry()
self.OTPthbYBin.add (self.OTPentryYBin)
self.OTPentryYBin.set_text('1')

#buttons
self.OTPthbBtns = gtk.HBox (True, 5)
self.OTVbTop.add (self.OTPthbBtns)

self.OTbtnCancel = gtk.Button("_Cancel")
self.OTbtnCancel.connect ("clicked", self.options_on_cancel)
self.OTPthbBtns.add (self.OTbtnCancel)

self.OTbtnOK = gtk.Button("_Ok")
self.OTbtnOK.connect ("clicked", self.options_on_ok)
self.OTPthbBtns.add (self.OTbtnOK)

# self.OTPframe = gtk.Frame(label="Window Options")
# self.OTVbTop.add (self.OTPframe)

#USB List window
self.usblist.set_border_width(5)
self.usblist.set_title ("Select USB Device...")
self.USBvbTop = gtk.VBox(False, 3)
self.usblist.add(self.USBvbTop)

self.USBcbUSB = gtk.combo_box_new_text();
self.USBvbTop.add (self.USBcbUSB)

self.USBbtnOK = gtk.Button("_Ok")
self.USBvbTop.add (self.USBbtnOK)
self.USBbtnOK.connect ("clicked", self.usb_on_ok)

self.list_usb()

self.options.show_all()

def main(self):
    gtk.main()

```

```
if __name__ == "__main__":
    app = gEEPROMConfig()
    app.main()
```

## E.6 pyCamCal

### E.6.1 pyCamCal.py

```
#!/usr/bin/env python

import pygtk
import gtk
import os
import pexpect

PM_EEPROM_WIDTH_HIGH_ADDR = 0x20
PM_EEPROM_WIDTH_LOW_ADDR = 0x21
PM_EEPROM_LENGTH_HIGH_ADDR = 0x22
PM_EEPROM_LENGTH_LOW_ADDR = 0x23
PM_EEPROM_COL_START_HIGH_ADDR = 0x24
PM_EEPROM_COL_START_LOW_ADDR = 0x25
PM_EEPROM_ROW_START_HIGH_ADDR = 0x26
PM_EEPROM_ROW_START_LOW_ADDR = 0x27
PM_EEPROM_COL_SKIP_ADDR = 0x28
PM_EEPROM_ROW_SKIP_ADDR = 0x29

USBlist=[None]*16
USBposition=[None]*16
cmd=''
options=''

class gEEPROMConfig:
    def on_close(self, widget, event, data=None):
        gtk.main_quit()
        return False

    def list_usb (self):
        global USBposition
        x = os.popen ('lsusb', "r")
        y = 0;
        found = 0;
        while 1:
            line = x.readline()
            line = line.rstrip()
            USBlist[y] = line

            if line.count('abcd'):
                USBlist[found] = line
                print found
                print USBlist[found]

                # Get position
                bus = USBlist[found][4:7]
                dev = USBlist[found][15:18]
                z = os.popen('./cam_init -d=' + bus + '.' + dev + ' --quickread')
                quad = z.readline()
                quad = z.readline()
                quad = quad[14]
                pos = z.readline()
                pos = pos[14:]
                pos = pos.rstrip()
                USBposition[found] = 'Q' + quad + ':' + pos

                print 'Q:' + quad
                print 'P:' + pos

                self.USBcbUSB.insert_text(found, line + ' - [Q' + quad + ':' + pos + ']\n')
                found = found + 1

        y = y+1
```

---

```

        if not line: break;
    if not found:
        self.error = gtk.MessageDialog(self.options, gtk.DIALOG_MODAL, gtk.MESSAGE_INFO,
                                       gtk.BUTTONS_OK, 'No Devices Found!')
        self.error.connect("response", self.on_close)
        self.error.show_all()

    else:
        self.USBcbUSB.set_active(0)
        self.usblist.show_all()

def read_eeeprom_data (self):
    global cmd

    opts = ' -rb ' + str(PM_EEPROM_WIDTH_HIGH_ADDR)
    z = os.popen (cmd + opts)
    r = z.readline()
    r = z.readline()
    val_hi = int(r[5:])

    opts = ' -rb ' + str(PM_EEPROM_WIDTH_LOW_ADDR)
    z = os.popen (cmd + opts)
    r = z.readline()
    r = z.readline()
    val_lo = int(r[5:])

    val = (val_hi << 8) + val_lo
    print 'RESULT: ' + str(val)
    self.OTPentryWidth.set_text(str(val))

    opts = ' -rb ' + str(PM_EEPROM_LENGTH_HIGH_ADDR)
    z = os.popen (cmd + opts)
    r = z.readline()
    r = z.readline()
    val_hi = int(r[5:])

    opts = ' -rb ' + str(PM_EEPROM_LENGTH_LOW_ADDR)
    z = os.popen (cmd + opts)
    r = z.readline()
    r = z.readline()
    val_lo = int(r[5:])

    val = (val_hi << 8) + val_lo
    print 'RESULT: ' + str(val)
    self.OTPentryLength.set_text(str(val))

    opts = ' -rb ' + str(PM_EEPROM_COL_START_HIGH_ADDR)
    z = os.popen (cmd + opts)
    r = z.readline()
    r = z.readline()
    val_hi = int(r[5:])

    opts = ' -rb ' + str(PM_EEPROM_COL_START_LOW_ADDR)
    z = os.popen (cmd + opts)
    r = z.readline()
    r = z.readline()
    val_lo = int(r[5:])

    val = (val_hi << 8) + val_lo
    print 'RESULT: ' + str(val)
    self.OTPentryXStart.set_text(str(val))

    opts = ' -rb ' + str(PM_EEPROM_ROW_START_HIGH_ADDR)
    z = os.popen (cmd + opts)
    r = z.readline()
    r = z.readline()
    val_hi = int(r[5:])

```

---

```
opts = ' -rb ' + str(PM_EEPROM_ROW_START_LOW_ADDR)
z = os.popen (cmd + opts)
r = z.readline()
r = z.readline()
val_lo = int(r[5:])

val = (val_hi << 8) + val_lo
print 'RESULT: ' + str(val)
self.OPTentryYStart.set_text(str(val))

opts = ' -rb ' + str(PM_EEPROM_COL_SKIP_ADDR)
z = os.popen (cmd + opts)
r = z.readline()
r = z.readline()
val = int(r[5:])

print 'RESULT: ' + str(val)
self.OPTentryXBin.set_text(str(val))

opts = ' -rb ' + str(PM_EEPROM_ROW_SKIP_ADDR)
z = os.popen (cmd + opts)
r = z.readline()
r = z.readline()
val = int(r[5:])

print 'RESULT: ' + str(val)
self.OPTentryYBin.set_text(str(val))

def options_on_ok (self, widget):
    #write eeprom data
    global cmd, options

    val = int(self.OPTentryWidth.get_text())
    val_hi = val >> 8
    val_lo = val & 0xff
    options = ' -w ' + str(PM_EEPROM_WIDTH_HIGH_ADDR) + ' ' + str(val_hi)
    print cmd + options
    os.system(cmd + options)
    options = ' -w ' + str(PM_EEPROM_WIDTH_LOW_ADDR) + ' ' + str(val_lo)
    print cmd + options
    os.system(cmd + options)

    val = int(self.OPTentryLength.get_text())
    val_hi = val >> 8
    val_lo = val & 0xff
    options = ' -w ' + str(PM_EEPROM_LENGTH_HIGH_ADDR) + ' ' + str(val_hi)
    print cmd + options
    os.system(cmd + options)
    options = ' -w ' + str(PM_EEPROM_LENGTH_LOW_ADDR) + ' ' + str(val_lo)
    print cmd + options
    os.system(cmd + options)

    val = int(self.OPTentryXStart.get_text())
    val_hi = val >> 8
    val_lo = val & 0xff
    options = ' -w ' + str(PM_EEPROM_COL_START_HIGH_ADDR) + ' ' + str(val_hi)
    os.system(cmd + options)
    print cmd + options
    options = ' -w ' + str(PM_EEPROM_COL_START_LOW_ADDR) + ' ' + str(val_lo)
    print cmd + options
    os.system(cmd + options)

    val = int(self.OPTentryYStart.get_text())
    val_hi = val >> 8
    val_lo = val & 0xff
    options = ' -w ' + str(PM_EEPROM_ROW_START_HIGH_ADDR) + ' ' + str(val_hi)
    print cmd + options
    os.system(cmd + options)
    options = ' -w ' + str(PM_EEPROM_ROW_START_LOW_ADDR) + ' ' + str(val_lo)
```

---

```

print cmd + options
os.system(cmd + options)

val = int(self.OPTentryXBin.get_text())
options = ' -w ' + str(PM_EEPROM_ROW_SKIP_ADDR) + ' ' + str(val)
print cmd + options
os.system(cmd + options)

val = int(self.OPTentryYBin.get_text())
options = ' -w ' + str(PM_EEPROM_COL_SKIP_ADDR) + ' ' + str(val)
print cmd + options
os.system(cmd + options)

#reload EEPROM data
options = ' -rp'
os.system(cmd + options)

def options_on_cancel (self, widget):
    gtk.main_quit()

def on_error_ok (self, widget, event):
    self.error.hide_all()

def usb_on_ok (self, widget):
    global cmd
    global USBposition

    selected = self.USBcbUSB.get_active()
    bus = USBlist[selected][4:7]
    dev = USBlist[selected][15:18]

    cmd = './cam_init -d=' + bus + '.' + dev

    self.options.set_title(self.options.get_title() + ' - [' + USBposition[selected] + ']')
    self.read_eeeprom_data()

    self.usblist.hide_all()

def __init__(self):
    self.threads = 0
    self.options = gtk.Window()
    self.usblist = gtk.Window()
    self.usblist.set_modal(True)
    self.usblist.set_transient_for(self.options)

    self.options.connect("delete_event", self.on_close, None)
    self.options.connect("destroy", self.on_close, None)

    #Options List
    self.options.set_border_width(5)
    self.options.set_title("Window Options")

    #Frame
    self.OPTvbTop = gtk.VBox(False, 5)
    self.options.add(self.OPTvbTop)

    self.OPTframe = gtk.Frame(label="Window Options")
    self.OPTvbTop.add (self.OPTframe)

    self.OPTvbFrame = gtk.VBox(False, 5)
    self.OPTframe.add (self.OPTvbFrame)

    #width
    self.OPThbWidth = gtk.HBox(True, 5)
    self.OPTvbFrame.add (self.OPThbWidth)

    self.OPTlblWidth = gtk.Label("Width")
    self.OPThbWidth.add (self.OPTlblWidth)

    self.OPTentryWidth = gtk.Entry()
    self.OPThbWidth.add (self.OPTentryWidth)
    #self.OPTentryWidth.set_text('2048')

    #length

```

---

---

```

self.OTvbLength = gtk.HBox(True, 5)
self.OTvbFrame.add (self.OTvbLength)

self.OTlblLength = gtk.Label("Length")
self.OTvbLength.add (self.OTlblLength)

self.OTentryLength = gtk.Entry()
self.OTvbLength.add (self.OTentryLength)
#self.OTentryLength.set_text('1536')

#xstart
self.OTvbXStart = gtk.HBox(True, 5)
self.OTvbFrame.add (self.OTvbXStart)

self.OTlblXStart = gtk.Label("X-Start")
self.OTvbXStart.add (self.OTlblXStart)

self.OTentryXStart = gtk.Entry()
self.OTvbXStart.add (self.OTentryXStart)
self.OTentryXStart.set_text('28')

#ystart
self.OTvbYStart = gtk.HBox(True, 5)
self.OTvbFrame.add (self.OTvbYStart)

self.OTlblYStart = gtk.Label("Y-Start")
self.OTvbYStart.add (self.OTlblYStart)

self.OTentryYStart = gtk.Entry()
self.OTvbYStart.add (self.OTentryYStart)
self.OTentryYStart.set_text('16')

#xbin
self.OTvbXBin = gtk.HBox(True, 5)
self.OTvbFrame.add (self.OTvbXBin)

self.OTlblXBin = gtk.Label("X-Bin")
self.OTvbXBin.add (self.OTlblXBin)

self.OTentryXBin = gtk.Entry()
self.OTvbXBin.add (self.OTentryXBin)
self.OTentryXBin.set_text('1')

#ybin
self.OTvbYBin = gtk.HBox(True, 5)
self.OTvbFrame.add (self.OTvbYBin)

self.OTlblYBin = gtk.Label("Y-Bin")
self.OTvbYBin.add (self.OTlblYBin)

self.OTentryYBin = gtk.Entry()
self.OTvbYBin.add (self.OTentryYBin)
self.OTentryYBin.set_text('1')

#buttons
self.OTvbBtns = gtk.HBox (True, 5)
self.OTvbTop.add (self.OTvbBtns)

self.OTbtnCancel = gtk.Button("_Cancel")
self.OTbtnCancel.connect ("clicked", self.options_on_cancel)
self.OTvbBtns.add (self.OTbtnCancel)

self.OTbtnOK = gtk.Button("_Ok")
self.OTbtnOK.connect ("clicked", self.options_on_ok)
self.OTvbBtns.add (self.OTbtnOK)

# self.OTframe = gtk.Frame(label="Window Options")
# self.OTvbTop.add (self.OTframe)

#USB List window
self.usblist.set_border_width(5)
self.usblist.set_title ("Select USB Device...")
self.USBvbTop = gtk.VBox(False, 3)

```

---

```

self.usblist.add(self.USBvbTop)

self.USBcbUSB = gtk.combo_box_new_text();
self.USBvbTop.add (self.USBcbUSB)

self.USBbtnOK = gtk.Button("_Ok")
self.USBvbTop.add (self.USBbtnOK)
self.USBbtnOK.connect ("clicked", self.usb_on_ok)

self.list_usb()

self.options.show_all()

def main(self):
    gtk.main()

if __name__ == "__main__":
    app = gEEPROMConfig()
    app.main()

```

## E.7 Human Machine Interface (w32)

The HMI application was developed in Microsoft Visual Basic 6.

### E.7.1 frmMain.frm

```

VERSION 5.00
Object = "{5E9E78A0-531B-11CF-91F6-C2863C385E30}#1.0#0"; "MSFLXGRD.OCX"
Object = "{648A5603-2C6E-101B-82B6-000000000014}#1.1#0"; "MSCOMM32.OCX"
Object = "{48E59290-9880-11CF-9754-00AA00C00908}#1.0#0"; "MSINET.OCX"
Begin VB.Form frmMain
    BorderStyle = 0 'None
    Caption = "PM Control Panel"
    ClientHeight = 9000
    ClientLeft = 0
    ClientTop = 0
    ClientWidth = 12000
    BeginProperty Font
        Name = "Tahoma"
        Size = 8.25
        Charset = 0
        Weight = 400
        Underline = 0 'False
        Italic = 0 'False
        Strikethrough = 0 'False
    EndProperty
    LinkTopic = "Form1"
    MaxButton = 0 'False
    MinButton = 0 'False
    ScaleHeight = 9000
    ScaleWidth = 12000
    StartUpPosition = 2 'CenterScreen
    Begin VB.Frame frameShutdownVerify
        Caption = "Question"
        BeginProperty Font
            Name = "Tahoma"
            Size = 14.25
            Charset = 0
            Weight = 400
            Underline = 0 'False
            Italic = 0 'False
            Strikethrough = 0 'False
        EndProperty
        Height = 3615
        Left = 4920
        TabIndex = 23
        Top = 3720
    End

```



```

Width          = 7695
Begin VB.PictureBox Picture1
    BorderStyle  = 0 'None
    Height       = 3135
    Left         = 120
    ScaleHeight  = 3135
    ScaleWidth   = 7335
    TabIndex     = 24
    Top          = 360
    Width        = 7335
Begin VB.CommandButton cmdShutdownNo
    Caption      = "NO"
    BeginProperty Font
        Name      = "Tahoma"
        Size      = 15.75
        Charset   = 0
        Weight    = 700
        Underline  = 0 'False
        Italic     = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height       = 855
    Left         = 4200
    TabIndex     = 27
    Top          = 2040
    Width        = 1935
End
Begin VB.CommandButton cmdShutdownYes
    Caption      = "YES"
    BeginProperty Font
        Name      = "Tahoma"
        Size      = 15.75
        Charset   = 0
        Weight    = 700
        Underline  = 0 'False
        Italic     = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height       = 855
    Left         = 1080
    TabIndex     = 26
    Top          = 2040
    Width        = 1935
End
Begin VB.Label lblShutdownVerify
    Alignment    = 2 'Center
    Caption      = "Are you sure you wish to shutdown?"
    BeginProperty Font
        Name      = "Tahoma"
        Size      = 24
        Charset   = 0
        Weight    = 400
        Underline  = 0 'False
        Italic     = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height       = 1575
    Left         = 0
    TabIndex     = 25
    Top          = 360
    Width        = 7215
End
End
Begin VB.Frame frameStats
    Caption      = "Statistics"
    BeginProperty Font
        Name      = "Tahoma"
        Size      = 14.25
        Charset   = 0
        Weight    = 400
        Underline  = 0 'False
        Italic     = 0 'False
        Strikethrough = 0 'False

```

```

EndProperty
Height      = 5775
Left       = 5040
TabIndex    = 18
Top         = 1200
Width      = 5295
Begin InetCtrlsObjects.Inet inetDownload
    Left     = 240
    Top      = 1440
    _ExtentX = 1005
    _ExtentY = 1005
    _Version = 393216
    Protocol = 4
    URL      = "http://"
    RequestTimeout = 4
End
Begin VB.Timer tmrStats
    Interval = 2000
    Left     = 840
    Top      = 1560
End
Begin MSFlexGridLib.MSFlexGrid gridStats
    Height      = 1455
    Index       = 0
    Left        = 0
    TabIndex    = 21
    Top         = 0
    Width       = 5055
    _ExtentX    = 8916
    _ExtentY    = 2566
    _Version    = 393216
    AllowBigSelection = 0 'False
    ScrollBars  = 0
    Appearance  = 0
    BeginProperty Font {0BE35203-8F91-11CE-9DE3-00AA004BB851}
        Name      = "Arial"
        Size     = 14.25
        Charset   = 0
        Weight    = 400
        Underline = 0 'False
        Italic    = 0 'False
        Strikethrough = 0 'False
    EndProperty
End
End
Begin MSCommLib.MSComm Comm
    Left       = 8640
    Top        = 120
    _ExtentX   = 1005
    _ExtentY   = 1005
    _Version   = 393216
    DTREnable  = -1 'True
    BaudRate   = 57600
End
Begin VB.Frame frameShutdownWait
    Height      = 2055
    Left       = 4680
    TabIndex    = 28
    Top         = 480
    Width      = 6615
    Begin VB.Timer tmrShutdown
        Enabled  = 0 'False
        Interval = 350
        Left     = 0
        Top      = 1560
    End
    Begin VB.Label lblShutdownWait
        Alignment = 2 'Center
        BackStyle = 0 'Transparent
        Caption    = "Please wait while the system shuts down..."
        BeginProperty Font
            Name      = "Tahoma"
            Size     = 24
            Charset   = 0

```

```

        Weight      = 400
        Underline   = 0   'False
        Italic      = 0   'False
        Strikethrough = 0   'False
    EndProperty
    Height      = 1215
    Left        = 240
    TabIndex    = 29
    Top         = 120
    Width       = 6135
End
Begin VB.Shape shapeShutdownAnimation
    BackColor   = &H00004000&
    BackStyle   = 1   'Opaque
    Height      = 255
    Index       = 5
    Left        = 5640
    Top         = 1200
    Visible     = 0   'False
    Width       = 255
End
Begin VB.Shape shapeShutdownAnimation
    BackColor   = &H00004000&
    BackStyle   = 1   'Opaque
    Height      = 255
    Index       = 6
    Left        = 5880
    Top         = 1200
    Visible     = 0   'False
    Width       = 255
End
Begin VB.Shape shapeShutdownAnimation
    BackColor   = &H00004000&
    BackStyle   = 1   'Opaque
    Height      = 255
    Index       = 4
    Left        = 6120
    Top         = 1560
    Width       = 255
End
Begin VB.Shape shapeShutdownAnimation
    BackColor   = &H00004000&
    BackStyle   = 1   'Opaque
    Height      = 255
    Index       = 3
    Left        = 5760
    Top         = 1560
    Width       = 255
End
Begin VB.Shape shapeShutdownAnimation
    BackColor   = &H00004000&
    BackStyle   = 1   'Opaque
    Height      = 255
    Index       = 2
    Left        = 5400
    Top         = 1560
    Width       = 255
End
Begin VB.Shape shapeShutdownAnimation
    BackColor   = &H00004000&
    BackStyle   = 1   'Opaque
    Height      = 255
    Index       = 1
    Left        = 5040
    Top         = 1560
    Width       = 255
End
Begin VB.Shape shapeShutdownAnimation
    BackColor   = &H00004000&
    BackStyle   = 1   'Opaque
    Height      = 255
    Index       = 0
    Left        = 4680
    Top         = 1560

```

```

        Width          = 255
    End
End
Begin VB.CommandButton cmdShutdown
    Caption          = "SHUTDOWN"
    BeginProperty Font
        Name          = "Tahoma"
        Size          = 14.25
        Charset       = 0
        Weight        = 700
        Underline     = 0 'False
        Italic        = 0 'False
        Strikethrough  = 0 'False
    EndProperty
    Height          = 495
    Left            = 4800
    TabIndex       = 19
    Top            = 7800
    Width          = 3255
End
Begin VB.CommandButton cmdCalMode
    Caption          = "CALIBRATION MODE"
    Enabled         = 0 'False
    BeginProperty Font
        Name          = "Tahoma"
        Size          = 14.25
        Charset       = 0
        Weight        = 700
        Underline     = 0 'False
        Italic        = 0 'False
        Strikethrough  = 0 'False
    EndProperty
    Height          = 495
    Left            = 4800
    TabIndex       = 20
    Top            = 7200
    Width          = 3255
End
Begin VB.CommandButton cmdControlScr
    Caption          = "CONTROL/MONITOR"
    BeginProperty Font
        Name          = "Tahoma"
        Size          = 14.25
        Charset       = 0
        Weight        = 700
        Underline     = 0 'False
        Italic        = 0 'False
        Strikethrough  = 0 'False
    EndProperty
    Height          = 1095
    Left            = 1080
    TabIndex       = 6
    Top            = 6720
    Width          = 3255
End
Begin VB.Frame frameQuadMonitor
    Caption          = "Quadrant Monitor"
    BeginProperty Font
        Name          = "Tahoma"
        Size          = 14.25
        Charset       = 0
        Weight        = 400
        Underline     = 0 'False
        Italic        = 0 'False
        Strikethrough  = 0 'False
    EndProperty
    Height          = 3855
    Left            = 120
    TabIndex       = 12
    Top            = 3240
    Width          = 4575
    Begin VB.Timer tmrUART
        Interval      = 1000
        Left          = 3720
    End
End

```

```

Top          = 360
End
Begin VB.Label lblQPercent
Alignment    = 2 'Center
Caption      = "0%"
BeginProperty Font
    Name      = "Arial"
    Size      = 18
    Charset   = 0
    Weight    = 400
    Underline = 0 'False
    Italic    = -1 'True
    Strikethrough = 0 'False
EndProperty
Height      = 375
Index       = 3
Left        = 840
TabIndex    = 33
Top         = 1320
Width       = 975
End
Begin VB.Label lblQPercent
Alignment    = 2 'Center
Caption      = "0%"
BeginProperty Font
    Name      = "Arial"
    Size      = 18
    Charset   = 0
    Weight    = 400
    Underline = 0 'False
    Italic    = -1 'True
    Strikethrough = 0 'False
EndProperty
Height      = 375
Index       = 2
Left        = 1080
TabIndex    = 32
Top         = 2040
Width       = 975
End
Begin VB.Label lblQPercent
Alignment    = 2 'Center
Caption      = "0%"
BeginProperty Font
    Name      = "Arial"
    Size      = 18
    Charset   = 0
    Weight    = 400
    Underline = 0 'False
    Italic    = -1 'True
    Strikethrough = 0 'False
EndProperty
Height      = 375
Index       = 1
Left        = 720
TabIndex    = 31
Top         = 2760
Width       = 975
End
Begin VB.Label lblQPercent
Alignment    = 2 'Center
Caption      = "0%"
BeginProperty Font
    Name      = "Arial"
    Size      = 18
    Charset   = 0
    Weight    = 400
    Underline = 0 'False
    Italic    = -1 'True
    Strikethrough = 0 'False
EndProperty
Height      = 375
Index       = 0
Left        = 2160

```

```

    TabIndex      = 30
    Top           = 480
    Width         = 975
End
Begin VB.Label lblM
    Alignment      = 2 'Center
    BackStyle      = 0 'Transparent
    Caption        = "M"
    BeginProperty Font
        Name        = "Tahoma"
        Size        = 14.25
        Charset     = 0
        Weight      = 700
        Underline   = 0 'False
        Italic      = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height         = 375
    Left           = 1800
    TabIndex      = 17
    Top           = 1920
    Width         = 615
End
Begin VB.Label lblQuad
    BackStyle      = 0 'Transparent
    Caption        = "Label1"
    BeginProperty Font
        Name        = "Tahoma"
        Size        = 14.25
        Charset     = 0
        Weight      = 700
        Underline   = 0 'False
        Italic      = -1 'True
        Strikethrough = 0 'False
    EndProperty
    Height         = 375
    Index          = 3
    Left           = 0
    TabIndex      = 16
    Top           = 0
    Width         = 1335
End
Begin VB.Label lblQuad
    BackStyle      = 0 'Transparent
    Caption        = "Label1"
    BeginProperty Font
        Name        = "Tahoma"
        Size        = 14.25
        Charset     = 0
        Weight      = 700
        Underline   = 0 'False
        Italic      = -1 'True
        Strikethrough = 0 'False
    EndProperty
    Height         = 375
    Index          = 2
    Left           = 0
    TabIndex      = 15
    Top           = 0
    Width         = 1335
End
Begin VB.Label lblQuad
    BackStyle      = 0 'Transparent
    Caption        = "Label1"
    BeginProperty Font
        Name        = "Tahoma"
        Size        = 14.25
        Charset     = 0
        Weight      = 700
        Underline   = 0 'False
        Italic      = -1 'True
        Strikethrough = 0 'False
    EndProperty
    Height         = 375

```

```

    Index      = 1
    Left       = 0
    TabIndex   = 14
    Top        = 0
    Width      = 1335
End
Begin VB.Label lblQuad
    BackStyle   = 0 'Transparent
    Caption     = "Label1"
    BeginProperty Font
        Name     = "Tahoma"
        Size     = 14.25
        Charset  = 0
        Weight   = 700
        Underline = 0 'False
        Italic   = -1 'True
        Strikethrough = 0 'False
    EndProperty
    Height      = 375
    Index       = 0
    Left        = 2400
    TabIndex    = 13
    Top         = 1680
    Width       = 1335
End
Begin VB.Shape shapeMotor
    BackStyle   = 1 'Opaque
    BorderWidth = 2
    Height      = 855
    Left        = 1680
    Shape       = 2 'Oval
    Top         = 1680
    Width       = 855
End
Begin VB.Shape boxQuad
    BackStyle   = 1 'Opaque
    Height      = 2175
    Index       = 3
    Left        = 360
    Top         = 1320
    Width       = 2175
End
Begin VB.Shape boxQuad
    BackColor   = &H0000C000&
    BackStyle   = 1 'Opaque
    Height      = 2175
    Index       = 2
    Left        = 120
    Top         = 360
    Width       = 2175
End
Begin VB.Shape boxQuad
    BackStyle   = 1 'Opaque
    Height      = 2175
    Index       = 1
    Left        = 2400
    Top         = 480
    Width       = 2175
End
Begin VB.Shape boxQuad
    BackStyle   = 1 'Opaque
    Height      = 2175
    Index       = 0
    Left        = 2160
    Top         = 1320
    Width       = 2175
End
End
Begin VB.Timer tmrERR
    Interval    = 750
    Left        = 240
    Top         = 4440
End
Begin VB.CommandButton cmdExit

```

```

Caption          = "EXIT"
BeginProperty Font
    Name          = "Tahoma"
    Size          = 21.75
    Charset       = 0
    Weight        = 700
    Underline     = 0 'False
    Italic        = 0 'False
    Strikethrough = 0 'False
EndProperty
Height          = 1095
Left           = 8520
TabIndex       = 5
Top            = 6960
Width          = 1815
End
Begin VB.Frame frameMotorControl
Caption        = "Motor Control"
BeginProperty Font
    Name          = "Tahoma"
    Size          = 14.25
    Charset       = 0
    Weight        = 400
    Underline     = 0 'False
    Italic        = 0 'False
    Strikethrough = 0 'False
EndProperty
Height         = 2415
Left           = 120
TabIndex       = 0
Top            = 840
Width          = 4455
Begin VB.PictureBox pbMotorControl
Appearance     = 0 'Flat
BackColor      = &H80000005&
BorderStyle    = 0 'None
BeginProperty Font
    Name          = "MS Sans Serif"
    Size          = 8.25
    Charset       = 0
    Weight        = 400
    Underline     = 0 'False
    Italic        = 0 'False
    Strikethrough = 0 'False
EndProperty
ForeColor      = &H80000008&
Height         = 1935
Left           = 120
ScaleHeight    = 1935
ScaleWidth     = 4215
TabIndex       = 1
Top            = 360
Width          = 4215
Begin VB.CommandButton cmdStop
BackColor      = &H000000C0&
Caption        = "STOP"
BeginProperty Font
    Name          = "Tahoma"
    Size          = 15.75
    Charset       = 0
    Weight        = 700
    Underline     = 0 'False
    Italic        = 0 'False
    Strikethrough = 0 'False
EndProperty
Height         = 735
Left           = 2280
Style          = 1 'Graphical
TabIndex       = 11
Top            = 1200
Width          = 1695
End
Begin VB.CommandButton cmdStart
BackColor      = &H0000C000&

```



```

Caption          = "START"
BeginProperty Font
    Name          = "Tahoma"
    Size          = 15.75
    Charset       = 0
    Weight        = 700
    Underline     = 0 'False
    Italic        = 0 'False
    Strikethrough = 0 'False
EndProperty
Height          = 735
Left            = 240
Style           = 1 'Graphical
TabIndex       = 10
Top            = 1200
Width          = 1695
End
Begin VB.CommandButton cmdMotorSpeedUP
Caption          = "+"
BeginProperty Font
    Name          = "Tahoma"
    Size          = 14.25
    Charset       = 0
    Weight        = 700
    Underline     = 0 'False
    Italic        = 0 'False
    Strikethrough = 0 'False
EndProperty
Height          = 495
Left            = 3600
TabIndex       = 4
Top            = 0
Width          = 615
End
Begin VB.CommandButton cmdMotorSpeedDOWN
Caption          = "-"
BeginProperty Font
    Name          = "Tahoma"
    Size          = 14.25
    Charset       = 0
    Weight        = 700
    Underline     = 0 'False
    Italic        = 0 'False
    Strikethrough = 0 'False
EndProperty
Height          = 495
Left            = 3600
TabIndex       = 3
Top            = 600
Width          = 615
End
Begin VB.TextBox txtMotorSpeed
Alignment        = 2 'Center
BackColor        = &H0080FFFF&
BeginProperty Font
    Name          = "Tahoma"
    Size          = 14.25
    Charset       = 0
    Weight        = 700
    Underline     = 0 'False
    Italic        = 0 'False
    Strikethrough = 0 'False
EndProperty
Height          = 480
Left            = 2280
TabIndex       = 2
Text            = "1000"
Top            = 360
Width          = 1215
End
Begin VB.Label lblMotorSpeed
BackStyle        = 0 'Transparent
Caption          = "SPEED (Hz):"
BeginProperty Font

```

```

        Name      = "Tahoma"
        Size      = 15.75
        Charset   = 0
        Weight    = 700
        Underline = 0 'False
        Italic    = 0 'False
        Strikethrough = 0 'False
    EndProperty
    Height      = 495
    Left        = 120
    TabIndex    = 9
    Top         = 360
    Width       = 2175
End
End
Begin VB.Line lineSBMain
    Visible     = 0 'False
    X1          = 360
    X2          = 6480
    Y1          = 8160
    Y2          = 8160
End
Begin VB.Label sbMain
    Appearance  = 0 'Flat
    BackColor   = &H80000005&
    BorderStyle = 1 'Fixed Single
    Caption     = "Label1"
    BeginProperty Font
        Name      = "Arial"
        Size      = 14.25
        Charset   = 0
        Weight    = 400
        Underline = 0 'False
        Italic    = 0 'False
        Strikethrough = 0 'False
    EndProperty
    ForeColor   = &H80000008&
    Height      = 375
    Left        = 720
    TabIndex    = 22
    Top         = 8040
    Width       = 4935
End
Begin VB.Shape shapeBorder
    Height      = 855
    Left        = 0
    Top         = 0
    Width       = 1335
End
Begin VB.Label lblScrTitle
    Alignment   = 2 'Center
    BackStyle   = 0 'Transparent
    Caption     = "Control/Monitor"
    BeginProperty Font
        Name      = "Tahoma"
        Size      = 21.75
        Charset   = 0
        Weight    = 400
        Underline = 0 'False
        Italic    = 0 'False
        Strikethrough = 0 'False
    EndProperty
    ForeColor   = &H00FFFFFF&
    Height      = 615
    Left        = 480
    TabIndex    = 7
    Top         = 0
    Width       = 9975
End
Begin VB.Label lblScrTitleShadow
    Alignment   = 2 'Center
    Appearance  = 0 'Flat
    BackColor   = &H8000000D&

```

```

    BorderStyle      = 1 'Fixed Single
    Caption          = "Control/Monitor"
    BeginProperty Font
        Name          = "Tahoma"
        Size          = 21.75
        Charset       = 0
        Weight        = 400
        Underline     = 0 'False
        Italic        = 0 'False
        Strikethrough  = 0 'False
    EndProperty
    ForeColor        = &H80000008&
    Height           = 615
    Left             = 120
    TabIndex         = 8
    Top              = 480
    Width            = 9975
End
Begin VB.Line LineBottom
    BorderWidth      = 2
    Visible          = 0 'False
    X1               = 1080
    X2               = 10800
    Y1               = 6120
    Y2               = 6120
End
Begin VB.Shape shapeControlBox
    BackColor        = &H8000000C&
    BackStyle        = 1 'Opaque
    Height           = 735
    Left             = 0
    Top              = 7200
    Width            = 2655
End
End
Attribute VB_Name = "frmMain"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit
Private Declare Function InitCommonControls Lib "comctl32.dll" () As Long
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

Const MACHINE_NAME As String = "PharmaSorter"

Const MOTOR_SPEED_INC = 100
Const MOTOR_SPEED_MAX = 2000
Const motor_speed_min = 500

Const PC_STATE_OFF = 1
Const PC_STATE_BOOTING = 2
Const PC_STATE_READY = 3

Dim pc_states(4) As Integer
'Information Messages
Const INFO_READY As String = " READY..."
'Warning Messages
Const WARNING_PCS_NOT_READY As String = " WAITING FOR INSPECTION PCs TO BOOT!"
'Error Messages

Private Sub cmdControlScr_Click()
    lblScrTitle.Caption = "Control/Monitor"
    lblScrTitleShadow.Caption = lblScrTitle.Caption
    frameMotorControl.Visible = True
    frameQuadMonitor.Visible = True
    frameStats.Visible = True
    frameShutdownVerify.Visible = False
    frameShutdownWait.Visible = False
End Sub

Private Sub cmdExit_Click()
    End

```

```
End Sub

Private Sub cmdMotorSpeedDOWN_Click()
    Dim MotorSpeed As Integer
    Dim Speed_Low As Integer
    Dim Speed_High As Integer

    MotorSpeed = CInt(txtMotorSpeed.Text)

    If Not (MotorSpeed Mod 100) = 0 Then
        MotorSpeed = MotorSpeed - MotorSpeed Mod 100
    End If

    If MotorSpeed > motor_speed_min Then
        MotorSpeed = MotorSpeed - MOTOR_SPEED_INC
        txtMotorSpeed.Text = MotorSpeed
    Else
        Beep
    End If

    ' Send UART Command to set speed
    Speed_Low = MotorSpeed And &HFF
    Speed_High = (MotorSpeed And &HFF00) / 256

    uart_cmd_set UARTCMD_SET_MOTOR_FREQ, Speed_High, Speed_Low
End Sub

Private Sub cmdMotorSpeedUP_Click()
    Dim MotorSpeed As Integer
    Dim Speed_Low As Integer
    Dim Speed_High As Integer

    MotorSpeed = CInt(txtMotorSpeed.Text)
    If Not (MotorSpeed Mod 100) = 0 Then
        MotorSpeed = MotorSpeed - MotorSpeed Mod 100
    End If

    If MotorSpeed < MOTOR_SPEED_MAX Then
        MotorSpeed = MotorSpeed + MOTOR_SPEED_INC
        txtMotorSpeed.Text = MotorSpeed
    Else
        Beep
        Exit Sub
    End If

    ' Send UART Command to set speed
    Speed_Low = MotorSpeed And &HFF
    Speed_High = (MotorSpeed And &HFF00) / 256

    uart_cmd_set UARTCMD_SET_MOTOR_FREQ, Speed_High, Speed_Low
End Sub

Private Sub cmdShutdown_Click()
    lblScrTitle.Caption = "System Shutdown"
    lblScrTitleShadow.Caption = lblScrTitle.Caption
    frameMotorControl.Visible = False
    frameQuadMonitor.Visible = False
    frameStats.Visible = False
    frameShutdownVerify.Visible = True
End Sub

Private Sub cmdShutdownNo_Click()
    cmdControlScr_Click
End Sub

Private Sub cmdShutdownYes_Click()
    frameShutdownVerify.Visible = False
    frameShutdownWait.Visible = True
    tmrShutdown.Enabled = True

    ' Send Shutdown Command
```

---

```

cmdStop_Click
uart_cmd_set UARTCMD_SHUTDOWN, &HFF, &HFF

End Sub

Private Sub cmdStart_Click()
Dim iRetVal As Integer

'Send Start Motor command to system controller
iRetVal = uart_cmd_set(UARTCMD_SET_MOTOR_STAT, &HFF, &HFF)

shapeMotor.BackColor = GUI_GREEN_BRIGHT
End Sub

Private Sub cmdStop_Click()
Dim iRetVal As Integer

'Send Stop Motor command to system controller
iRetVal = uart_cmd_set(UARTCMD_SET_MOTOR_STAT, &H0, &H0)

shapeMotor.BackColor = GUI_RED_BRIGHT
End Sub

Private Sub Form_Initialize()
InitCommonControls
End Sub

Private Sub Form_Load()
Dim I As Integer
Dim data() As Byte
Dim buffer As Variant
Dim ret As Integer

'Get current motor speed
ret = uart_cmd_get(UARTCMD_GET_MOTOR_FREQ, 0, 0, buffer)

If (ret) Then
MsgBox "Unable to connect to " & MACHINE_NAME, vbInformation + vbOKOnly
End
End If

data = buffer

txtMotorSpeed.Text = data(2) * 256 + data(4)

'Get current motor state
ret = uart_cmd_get(UARTCMD_GET_MOTOR_STAT, 0, 0, buffer)

data = buffer

If data(2) = &HFF Then
shapeMotor.BackColor = GUI_GREEN_BRIGHT
Else
shapeMotor.BackColor = GUI_RED_BRIGHT
End If

tmrERR.Enabled = True
sbMain.Caption = " WAITING FOR INSPECTION PCs TO BOOT!"

' Set PC states to red for now
For I = boxQuad.LBound To boxQuad.UBound
boxQuad(I).BackColor = GUI_RED
Next I

' Fill in labels for flexgrid
With gridStats(0)
.Rows = 9
.Cols = 5

.Col = 0
For I = 0 To .Rows - 1
.Row = I
.CellFontBold = True
Next I

```

---

```

        .Row = 0
        For I = 0 To .Cols - 1
            .Col = I
            .CellFontBold = True
        Next I

        .Row = 0
        For I = 1 To 4
            .Col = I
            .Text = "Q" & I
        Next I

        .Col = 0
        .Row = 1
        .Text = "GOOD"
        .Row = .Row + 1
        .Text = "BAD"
        .Row = .Row + 1
        .Text = "EMPTY"
        .Row = .Row + 1
        .Text = "TOTAL"
        .Row = .Row + 1
        .Text = "GOOD (%)"
        .Row = .Row + 1
        .Text = "MISALIGN"
        .Row = .Row + 1
        .Text = "RATE (cap/m)"
        .Row = .Row + 1
        .Text = "EFF. RATE"
    End With

    'Set Default Screen
    cmdControlScr_Click
End Sub

Private Sub Form_Resize()
    'On Error GoTo ErrHdlr

    Dim l As Integer

    ' Position Objects
    sbMain.Top = Me.Height - sbMain.Height
    sbMain.Left = 0
    sbMain.Width = Me.Width

    lblScrTitleShadow.Left = BORDER_SPACE
    lblScrTitleShadow.Top = BORDER_SPACE
    lblScrTitleShadow.Width = Me.Width - BORDER_SPACE * 2
    lblScrTitle.Left = BORDER_SPACE - 20
    lblScrTitle.Top = BORDER_SPACE - 20
    lblScrTitle.Width = Me.Width - BORDER_SPACE * 2

    cmdExit.Left = Me.Width - cmdExit.Width - BORDER_SPACE
    cmdExit.Top = Me.Height - cmdExit.Height - sbMain.Height - BORDER_SPACE * 3 / 2

    cmdControlScr.Left = BORDER_SPACE
    cmdControlScr.Top = Me.Height - cmdExit.Height - sbMain.Height - BORDER_SPACE * 3 / 2

    cmdShutdown.Left = cmdControlScr.Left + cmdControlScr.Width + BORDER_SPACE
    cmdShutdown.Top = cmdControlScr.Top + cmdShutdown.Height + BORDER_SPACE / 2

    cmdCalMode.Left = cmdControlScr.Left + cmdControlScr.Width + BORDER_SPACE
    cmdCalMode.Top = cmdControlScr.Top

    frameMotorControl.Left = BORDER_SPACE
    frameMotorControl.Top = lblScrTitle.Top + lblScrTitle.Height + BORDER_SPACE

    frameQuadMonitor.Left = BORDER_SPACE
    frameQuadMonitor.Top = frameMotorControl.Top + frameMotorControl.Height + BORDER_SPACE
    frameQuadMonitor.Height = cmdExit.Top - frameQuadMonitor.Top - BORDER_SPACE * 2
    frameQuadMonitor.Width = frameMotorControl.Width

```

```

'Size quadrant boxes
For I = boxQuad.LBound To boxQuad.UBound
    If (frameQuadMonitor.Height > frameQuadMonitor.Width) Then
        boxQuad(I).Width = (frameQuadMonitor.Width - BORDER_SPACE / 2) / 2
        boxQuad(I).Height = boxQuad(I).Width
    Else
        boxQuad(I).Width = (frameQuadMonitor.Height - BORDER_SPACE * 2) / 2
        boxQuad(I).Height = boxQuad(I).Width
    End If
Next I

'Arrange quadrant boxes
' Q3 | Q2
' ----+----
' Q4 | Q1
boxQuad(0).Left = frameQuadMonitor.Width / 2
boxQuad(1).Left = frameQuadMonitor.Width / 2
boxQuad(2).Left = boxQuad(0).Left - boxQuad(0).Width
boxQuad(3).Left = boxQuad(0).Left - boxQuad(0).Width

boxQuad(0).Top = frameQuadMonitor.Height / 2 + BORDER_SPACE / 2
boxQuad(3).Top = frameQuadMonitor.Height / 2 + BORDER_SPACE / 2
boxQuad(1).Top = boxQuad(0).Top - boxQuad(0).Height ' - BORDER_SPACE / 8
boxQuad(2).Top = boxQuad(0).Top - boxQuad(0).Height ' - BORDER_SPACE / 8

'Quadrant labels
lblQuad(0).Left = boxQuad(0).Left + boxQuad(0).Width - lblQuad(0).Width - BORDER_SPACE / 4
lblQuad(0).Top = boxQuad(0).Top + boxQuad(0).Height - lblQuad(0).Height
lblQuad(0).Alignment = 1 ' vbAlignRight

lblQuad(1).Left = boxQuad(1).Left + boxQuad(1).Width - lblQuad(1).Width - BORDER_SPACE / 4
lblQuad(1).Top = boxQuad(1).Top ' + lblQuad(1).Height
lblQuad(1).Alignment = 1 ' vbAlignRight

lblQuad(2).Left = boxQuad(2).Left + BORDER_SPACE / 4
lblQuad(2).Top = boxQuad(2).Top
lblQuad(2).Alignment = 0 ' vbAlignLeft

lblQuad(3).Left = boxQuad(3).Left + BORDER_SPACE / 4
lblQuad(3).Top = boxQuad(3).Top + boxQuad(3).Height - lblQuad(3).Height
lblQuad(3).Alignment = 0 ' vbAlignLeft

'Good cap percentage labels
For I = lblQPercent.LBound To lblQPercent.UBound
    lblQPercent(I).BackStyle = vbTransparent
    lblQPercent(I).Left = (boxQuad(I).Left + boxQuad(I).Width / 2) - lblQPercent(I).Width / 2
    lblQPercent(I).Top = (boxQuad(I).Top + boxQuad(I).Height / 2) - lblQPercent(I).Height / 2
Next I

For I = lblQuad.LBound To lblQuad.UBound
    lblQuad(I).Caption = "Q" & CInt(I + 1)
Next I

shapeMotor.Left = (frameQuadMonitor.Width - shapeMotor.Width) / 2
shapeMotor.Top = boxQuad(0).Top - (shapeMotor.Height / 2)

lblM.Left = (frameQuadMonitor.Width - lblM.Width) / 2
lblM.Top = boxQuad(0).Top - (lblM.Height / 2)

'Statistics
frameStats.Top = frameMotorControl.Top
frameStats.Height = cmdExit.Top - frameStats.Top - BORDER_SPACE * 2

frameStats.Left = frameMotorControl.Left + frameMotorControl.Width + BORDER_SPACE
frameStats.Width = Me.Width - frameStats.Left - BORDER_SPACE

For I = gridStats.LBound To gridStats.UBound
    gridStats(I).Left = BORDER_SPACE / 2
    gridStats(I).Width = frameStats.Width - BORDER_SPACE
    gridStats(I).Height = (frameStats.Height) / gridStats.count - (BORDER_SPACE * 3)
    gridStats(I).Top = (I * (gridStats(gridStats.LBound).Height) + BORDER_SPACE) + BORDER_SPACE
Next I

'Data Grid

```

---

```

With gridStats(0)
    .ColWidth(0) = 2000

    For I = 1 To .Cols - 1
        .ColWidth(I) = (.Width - .ColWidth(0)) / (.Cols - 1) - 20
    Next I

    For I = 0 To .Rows - 1
        .RowHeight(I) = (.Height) / (.Rows) - 10
    Next I
End With

'Shutdown
frameShutdownVerify.Left = (Me.Width - frameShutdownVerify.Width) / 2
frameShutdownVerify.Top = (cmdExit.Top - lblScrTitle.Top + lblScrTitle.Height - frameShutdownVerify
    .Height) / 2
frameShutdownWait.Left = (Me.Width - frameShutdownWait.Width) / 2
frameShutdownWait.Top = (cmdExit.Top - lblScrTitle.Top + lblScrTitle.Height - frameShutdownWait.
    Height) / 2

LineBottom.Y1 = cmdExit.Top - BORDER_SPACE
LineBottom.Y2 = cmdExit.Top - BORDER_SPACE
LineBottom.X1 = BORDER_SPACE
LineBottom.X2 = Me.Width - BORDER_SPACE

lineSBMain.BorderWidth = 3
lineSBMain.Y1 = sbMain.Top
lineSBMain.Y2 = sbMain.Top
lineSBMain.X1 = 0
lineSBMain.X2 = Me.Width

shapeControlBox.Left = BORDER_SPACE / 2
shapeControlBox.Width = Me.Width - BORDER_SPACE
shapeControlBox.Top = cmdExit.Top - BORDER_SPACE
shapeControlBox.Height = cmdExit.Height + BORDER_SPACE * 2

shapeBorder.Left = 0
shapeBorder.Top = 0
shapeBorder.Width = Me.Width
shapeBorder.Height = Me.Height

' Colours
pbMotorControl.BackColor = frameMotorControl.BackColor

ErrHdlr:
    ' I don't care if it gets here
End Sub

Private Sub tmrERR_Timer()
    Dim pc_ready As Integer
    Dim bWarning As Boolean
    Dim bError As Boolean
    Dim I As Integer

    'Are all inspection PCs up and running?
    For I = 0 To 3
        If (pc_states(I) = PC_STATE_READY) Then
            pc_ready = pc_ready + 1
        End If
    Next I

    If Not pc_ready = 4 Then
        bWarning = True
        sbMain.Caption = WARNING_PCS_NOT_READY
    Else
        bWarning = False
        sbMain.Caption = INFO_READY
    End If

    ' Flash Banner
    If bWarning = True Then
        If (sbMain.BackColor = vbYellow) Then
            sbMain.BackColor = Me.BackColor
        Else

```

---



---

```

        sbMain.BackColor = vbYellow
    End If
ElseIf bError = True Then
    If (sbMain.BackColor = vbRed) Then
        sbMain.BackColor = Me.BackColor
    Else
        sbMain.BackColor = vbRed
    End If
Else
    If Not (sbMain.BackColor = Me.BackColor) Then
        sbMain.BackColor = Me.BackColor
    End If
End If
End Sub

Private Function uart_cmd_get(cmd As Integer, value_hi As Integer, value_low As Integer, ByRef data As
Variant) As Integer
On Error GoTo ErrHdlr

    Dim sData As String
    Dim count As Integer
    Dim buffer As Variant

    count = 0

    Dim x As Byte
    Dim check(3) As Byte

    check(0) = cmd
    check(0) = Not check(0)
    check(1) = value_hi
    check(1) = Not check(1)
    check(2) = value_low
    check(2) = Not check(2)

    'Send data to serial port
    If (Comm.PortOpen = False) Then
        Comm.PortOpen = True
    End If

    ' command
    Comm.Output = Chr(cmd)
    Comm.Output = Chr(value_hi)
    Comm.Output = Chr(value_low)

    ' data check
    Comm.Output = Chr(check(0))
    Comm.Output = Chr(check(1))
    Comm.Output = Chr(check(2))

    Sleep 45

    ' check for ack

    While (Comm.InBufferCount > 0)
        count = count + Comm.InBufferCount
        buffer = buffer & Comm.Input
        'DoEvents
    Wend

    If (count < 2) Then
        uart_cmd_get = 1
    Else
        uart_cmd_get = 0
    End If

    data = buffer

    ' If (buffer(0) <> 6) Then
    '     MsgBox "Error during communication (NACK received)!", vbCritical
    ' End If

```

---

---

```

Comm.PortOpen = False

Exit Function
ErrHdlr:
If Err.Number = 8002 Then
    MsgBox "Invalid Port Selected...", vbCritical + vbOKOnly
End If

uart_cmd_get = 1
End Function

Private Function uart_cmd_set(cmd As Integer, value_hi As Integer, value_low As Integer) As Integer
    Dim buffer As Variant
    Dim data() As Byte
    Dim x As Byte
    Dim check(3) As Byte

    check(0) = cmd
    check(0) = Not check(0)
    check(1) = value_hi
    check(1) = Not check(1)
    check(2) = value_low
    check(2) = Not check(2)

    'Send data to serial port
    If (Comm.PortOpen = False) Then
        Comm.PortOpen = True
    End If

    ' command
    Comm.Output = Chr(cmd)
    Comm.Output = Chr(value_hi)
    Comm.Output = Chr(value_low)

    ' data check
    Comm.Output = Chr(check(0))
    Comm.Output = Chr(check(1))
    Comm.Output = Chr(check(2))

    Sleep 65

    While (Comm.InBufferCount > 0)
        buffer = Comm.Input
        'DoEvents
    Wend

    data = buffer
    ' check for ack

    x = data(0)
    If (x <> 6) Then
        uart_cmd_set = 1
    Else
        uart_cmd_set = 0
    End If

    Comm.PortOpen = False

End Function

Private Sub tmrShutdown_Timer()
    Static I, J, F As Integer
    Dim buffer As Variant
    Dim data() As Byte
    Dim c As Integer

    I = I + 1
    J = J + 1
    If I > shapeShutdownAnimation.UBound Then
        I = shapeShutdownAnimation.LBound
    End If

    If J > shapeShutdownAnimation.UBound Then

```

```

        J = shapeShutdownAnimation.LBound
    End If

    If F = 0 Then
        I = shapeShutdownAnimation.LBound
        J = shapeShutdownAnimation.UBound
        F = 1
    End If

    shapeShutdownAnimation(J).BackColor = shapeShutdownAnimation(I).BackColor
    shapeShutdownAnimation(I).BackColor = vbGreen

    uart_cmd_get UARTCMD_GET_SHUTDOWN_STATUS, 0, 0, buffer

    data() = buffer

    If data(2) Then
        ' Shutdown complete
        lblShutdownWait.Caption = "System is now safe to power down..."
        For c = shapeShutdownAnimation.LBound To shapeShutdownAnimation.UBound
            shapeShutdownAnimation(c).BackColor = vbGreen
        Next c

        tmrShutdown.Enabled = False
        'Sleep (4000)
        'Shell "shutdown -s -t 10", vbNormalFocus
    End If
End Sub

Private Sub tmrStats_Timer()
    On Error Resume Next

    Dim bytes() As Byte
    Dim fnum As Integer

    ' Grab Files from Inspection PCs
    ' Check state of inspection PCs
    If (pc_states(0) = PC_STATE_READY) Then
        inetDownload.Protocol = icHTTP
        inetDownload.URL = "HTTP://192.168.1.101/statistics.csv"
        bytes() = inetDownload.OpenURL(inetDownload.URL, icByteArray)
        'bytes() = inetDownload.OpenURL("http://192.168.1.101/statistics.csv", icByteArray)

        fnum = FreeFile
        Open "C:\tmp\stats_Q1.csv" For Binary Access Write As #fnum
            Put #fnum, , bytes()
        Close #fnum
    End If

    If (pc_states(1) = PC_STATE_READY) Then
        bytes() = inetDownload.OpenURL("http://192.168.1.102/statistics.csv", icByteArray)

        fnum = FreeFile
        Open "C:\tmp\stats_Q2.csv" For Binary Access Write As #fnum
            Put #fnum, , bytes()
        Close #fnum
    End If

    If (pc_states(2) = PC_STATE_READY) Then
        bytes() = inetDownload.OpenURL("http://192.168.1.103/statistics.csv", icByteArray)

        fnum = FreeFile
        Open "C:\tmp\stats_Q3.csv" For Binary Access Write As #fnum
            Put #fnum, , bytes()
        Close #fnum
    End If

    If (pc_states(3) = PC_STATE_READY) Then
        bytes() = inetDownload.OpenURL("http://192.168.1.104/statistics.csv", icByteArray)

        fnum = FreeFile
        Open "C:\TMP\stats_Q4.csv" For Binary Access Write As #fnum
            Put #fnum, , bytes()
    End If

```

---

```

        Close #fnum
    End If

    Dim fieldArray() As String
    Dim I, J As Integer
    Dim lineRead As String
    Dim QuadNum As Integer
    Dim sFileName As String

    For QuadNum = 1 To 4
        If pc_states(QuadNum - 1) = PC_STATE_READY Then

            'Open File
            sFileName = "C:\tmp" & "\stats_Q" & QuadNum & ".csv"

            Open sFileName For Input As #1

            While Not EOF(1)
                Line Input #1, lineRead

                lineRead = Replace(lineRead, vbCrLf, ",")
                fieldArray = parseCsv(lineRead)
                'Parse through CSV looking for specific parameters
                If lineRead <> "" Then
                    With gridStats(0)
                        For I = 0 To UBound(fieldArray)
                            .Col = QuadNum
                            If InStr(1, fieldArray(I), "Good", vbTextCompare) Then
                                .Row = 1
                                .Text = fieldArray(I + 1)
                            End If
                            If InStr(1, fieldArray(I), "Good", vbTextCompare) Then
                                .Row = 5
                                .Text = fieldArray(I + 2)
                                lblQPercent(QuadNum - 1).Caption = fieldArray(I + 2)
                            End If
                            If InStr(1, fieldArray(I), "Bad", vbTextCompare) Then
                                .Row = 2
                                .Text = fieldArray(I + 1)
                            End If
                            If InStr(1, fieldArray(I), "Empty", vbTextCompare) Then
                                .Row = 3
                                .Text = fieldArray(I + 1)
                            End If
                            If InStr(1, fieldArray(I), "Total", vbTextCompare) Then
                                .Row = 4
                                .Text = fieldArray(I + 1)
                            End If
                            If InStr(1, fieldArray(I), "Effective Inspection Rate", vbTextCompare) Then
                                .Row = 8
                                .Text = fieldArray(I + 1)
                            End If
                            If (InStr(1, fieldArray(I), "Inspection Rate", vbTextCompare) And _
                                InStr(1, fieldArray(I), "Effective", vbTextCompare) = False) Then
                                .Row = 7
                                .Text = fieldArray(I + 1)
                            End If
                            If InStr(1, fieldArray(I), "Misaligned", vbTextCompare) Then
                                .Row = 6
                                .Text = fieldArray(I + 1)
                            End If
                        Next
                    End With
                End If

                Wend

            Close #1

        Else
            ' Blank entire column
            gridStats(0).Col = QuadNum
            For J = 1 To gridStats(0).Rows - 1
                gridStats(0).Row = J
            Next
        End If
    Next

```

---

---

```

        gridStats(0).Text = ""
    Next J
End If

Next QuadNum

End Sub

Private Sub tmrUART_Timer()
On Error GoTo ErrHdlr

    Dim cmd As Integer
    Dim value_hi As Integer
    Dim value_low As Integer
    Dim sData As String
    Dim buffer As Variant
    Dim data() As Byte
    Dim count As Integer
    Dim ret As Integer
    Dim x As Byte

    count = 0

    ret = uart_cmd_get(UARTCMD_GET_PC_PWR_STATE, 0, 0, buffer)

    If (ret) Then
        GoTo ErrHdlr
    End If

    data = buffer

    If (data(0) <> 6) Then
        MsgBox "Error during communication (NACK received)!", vbCritical
    End If

    x = data(2)

    If ((x And &H3) = &H3) Then
        'lblPCStatus(0).Caption = "Online"
        boxQuad(0).BackColor = vbGreen
        pc_states(0) = PC_STATE_READY
    ElseIf (x And &H1) Then
        'lblPCStatus(0).Caption = "Booting Up..."
        boxQuad(0).BackColor = vbYellow
        pc_states(0) = PC_STATE_BOOTING
    Else
        'lblPCStatus(0).Caption = "Offline"
        boxQuad(0).BackColor = vbRed
        pc_states(0) = PC_STATE_OFF
    End If

    If ((x And &HC) = &HC) Then
        'lblPCStatus(1).Caption = "Online"
        boxQuad(1).BackColor = vbGreen
        pc_states(1) = PC_STATE_READY
    ElseIf (x And &H4) Then
        'lblPCStatus(1).Caption = "Booting Up..."
        boxQuad(1).BackColor = vbYellow
        pc_states(1) = PC_STATE_BOOTING
    Else
        'lblPCStatus(1).Caption = "Offline"
        boxQuad(1).BackColor = vbRed
        pc_states(1) = PC_STATE_OFF
    End If

    If ((x And &H30) = &H30) Then
        'lblPCStatus(2).Caption = "Online"
        boxQuad(2).BackColor = vbGreen
        pc_states(2) = PC_STATE_READY
    ElseIf (x And &H10) Then
        'lblPCStatus(2).Caption = "Booting Up..."
        boxQuad(2).BackColor = vbYellow
        pc_states(2) = PC_STATE_BOOTING
    Else

```

---

```

        'lblPCStatus(2).Caption = "Offline"
        boxQuad(2).BackColor = vbRed
        pc_states(2) = PC_STATE_OFF
    End If

    If ((x And &HC0) = &HC0) Then
        'lblPCStatus(3).Caption = "Online"
        boxQuad(3).BackColor = vbGreen
        pc_states(3) = PC_STATE_READY
    ElseIf (x And &H40) Then
        'lblPCStatus(3).Caption = "Booting Up..."
        boxQuad(3).BackColor = vbYellow
        pc_states(3) = PC_STATE_BOOTING
    Else
        'lblPCStatus(3).Caption = "Offline"
        boxQuad(3).BackColor = vbRed
        pc_states(3) = PC_STATE_OFF
    End If

    'Get current motor state
    ret = uart_cmd_get(UARTCMD_GET_MOTOR_STAT, 0, 0, buffer)

    data = buffer

    If data(2) = &HFF Then
        shapeMotor.BackColor = GUI_GREEN_BRIGHT
    Else
        shapeMotor.BackColor = GUI_RED_BRIGHT
    End If

    Exit Sub

ErrHdlr:

End Sub

```

## E.7.2 modGUIConsts.bas

```

Attribute VB_Name = "modGUIConsts"
Global Const BORDER_SPACE As Integer = 250

Global Const GUI_GREEN_BRIGHT As Long = &H30FF30
Global Const GUI_YELLOW_BRIGHT As Long = &H30FFFF
Global Const GUI_RED_BRIGHT As Long = &H3030FF

Global Const GUI_GREEN As Long = &HE000&
Global Const GUI_YELLOW As Long = &HF0F0&
Global Const GUI_RED As Long = &HE0&

```

## E.7.3 modCSVParser.bas

```

Attribute VB_Name = "modCSVParser"

Function parseCsv(lineIn As String) As String()
    Dim s As String
    Dim local_s As String
    Dim n1 As Integer
    Dim n2 As Integer
    Dim str2 As String
    Dim i As Integer
    Dim strTemp() As String
    i = 0
    s = Trim(lineIn) ' remove spaces, if any

    While Len(s)
        ReDim Preserve strTemp(i) ' Didnt want to have fixed fields,
        ' dont know how to dynamically allocate, hence ...
        If Mid$(s, 1, 1) = """" Then ' if already in quotes ...
            strTemp(i) = Mid$(s, 2, InStr(2, s, """")) - 2) 'find 2nd ""
        Else: If InStr(s, ",") Then strTemp(i) = Mid$(s, 1, InStr(s, ",") - 1)

```

```

End If
's = Mid$(s, Len(strTemp(i)) + 1)
n2 = InStr(Len(strTemp(i)) Or 1, s, ",")
If n2 Then s = Trim(Mid$(s, n2 + 1)) Else s = "" ' clip till next
I = I + 1
Wend
parseCsv = strTemp
End Function

```

## E.7.4 modUARTCommands.bas

Attribute VB\_Name = "modUARTCommands"

```

'/* uart_commands.h
' * =====
' * Definitions for UART commands.
' *
' * Author: Neil Scott
' * Date: August 10, 2007
' */

'/* System Control - Set Commands */
Global Const UARTCMD_SET_MOTOR_STAT = &H90 '/* Enable or Disable Motor */

#define UARTCMD_SET_MOTOR_STAT 0x90 '/* Enable or Disable Motor */
Global Const UARTCMD_SET_MOTOR_FREQ = &H91 '/* Set Motor Speed */

#define UARTCMD_SET_MOTOR_FREQ 0x91

#define UARTCMD_SET_BLO_WIDTH 0x92 '/* Set pulse width for BLO */
#define UARTCMD_SET_BLI_WIDTH 0x93 '/* Set pulse width for BLI */
#define UARTCMD_SET_FLO_WIDTH 0x94 '/* Set pulse width for FLO */
#define UARTCMD_SET_FLI_WIDTH 0x95 '/* Set pulse width for FLI */

#define UARTCMD_SET_CAM0_PULSE_POS 0x9A '/* Set motor pulse position for CAM0 */
#define UARTCMD_SET_CAM1_PULSE_POS 0x9B '/* Set motor pulse position for CAM1 */
#define UARTCMD_SET_ACCEPT_ON_PULSE_POS 0x9C '/* Set motor pulse position for ACCEPT ON */
'/*
#define UARTCMD_SET_ACCEPT_OFF_PULSE_POS 0x9D '/* Set motor pulse position for ACCEPT OFF */

Global Const UARTCMD_POWER_ON_PCS = &H9E '/* Toggle power SW for 400ms to PC MBs */
Global Const UARTCMD_SHUTDOWN = &H9F '/* Put system in shutdown mode */

'/* System Control - Get Commands */
Global Const UARTCMD_GET_MOTOR_STAT = &H10 '/* Enable or Disable Motor */
Global Const UARTCMD_GET_MOTOR_FREQ = &H11 '/* Get Motor Speed */
Global Const UARTCMD_GET_BLO_WIDTH = &H12 '/* Get pulse width for BLO */
Global Const UARTCMD_GET_BLI_WIDTH = &H13 '/* Get pulse width for BLI */
Global Const UARTCMD_GET_FLO_WIDTH = &H14 '/* Get pulse width for FLO */
Global Const UARTCMD_GET_FLI_WIDTH = &H15 '/* Get pulse width for FLI */

#define UARTCMD_GET_CAM0_PULSE_POS 0x1A '/* Get motor pulse position for CAM0 */
#define UARTCMD_GET_CAM1_PULSE_POS 0x1B '/* Get motor pulse position for CAM1 */
#define UARTCMD_GET_ACCEPT_ON_PULSE_POS 0x1C '/* Get motor pulse position for ACCEPT ON */
#define UARTCMD_GET_ACCEPT_OFF_PULSE_POS 0x1D '/* Get motor pulse position for ACCEPT OFF */

Global Const UARTCMD_GET_PC_PWR_STATE = &H1E '/* Retrieve the power state of all PCs */
Global Const UARTCMD_GET_SHUTDOWN_STATUS = &H1F '/* Retrieve status of shutdown (complete?) */

'/* System Monitors */
Global Const UARTCMD_GET_TEMP_SP = &H30 '/* Get Temperature from Side Panel Sensor */
Global Const UARTCMD_GET_TEMP_FP = &H31 '/* Get Temperature from Front Panel Sensor */

'/* Inspection Status */
Global Const UARTCMD_GOOD_COUNT = &H21 '/* Get good capsule count for specified quadrant */
'/*
Global Const UARTCMD_BAD_COUNT = &H22 '/* Get reject capsule count for specified quadrant */
Global Const UARTCMD_TOTAL_COUNT = &H23 '/* Get total capsule count from specified quadrant */
'/*
#define UARTCMD_RESET_COUNTERS 0xAF '/* Reset the counters */

```

```
/* Fault Registers */
#define UARTCMD_FAULT_COUNT      0xF0      /* Get fault count */
#define UARTCMD_FAULT_CODE      0xF1      /* Get fault code of previous fault */

/* Debug Modes */

Global Const UARTCMD_DEBUG_MODE = &HDD      /* Set in debug mode so images are acquired
      when motor is off */

Global Const UARTCMD_ONE_STEP = &HDE      /* Make system step one capsule, fires BL and
      trigger */
```



---

## *VITA AUCTORIS*

---

Neil Scott was born in Sarnia, Ontario, Canada. He received his Bachelor of Applied Science degree in Electrical Engineering from the University of Windsor, Ontario, Canada in 2006. He is currently working towards a Master of Applied Science degree in Electrical Engineering. His primary area of research is the development of a high throughput inspection system for quality control of pharmaceutical capsules. His main area of expertise lies in hardware design, software and firmware development. His research involves USB device development, embedded system design with a focus on high-level programming languages, primarily C and C++.