

UNIVERSITY OF SOUTHAMPTON
FACULTY OF PHYSICAL AND APPLIED SCIENCES
Electronics and Computer Science

Two Dimensional Stereoscopic Mapping Robot

by

Henry S. Lovett

A project progress report submitted for the award of
MEng Electronic Engineering

Supervisor: Prof. Steve Gunn
Examiner: Prof. Mark Zwolinski

20th March, 2013

Turn off iNotes!

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL AND APPLIED SCIENCES
Electronics and Computer Science

A project report submitted for the award of MEng Electronic Engineering

TWO DIMENSIONAL STEREOSCOPIC MAPPING ROBOT

by Henry S. Lovett

Abstract Needed!

Contents

List of Symbols	xiii
1 Introduction	1
1.1 Project Management	2
2 Research	3
2.1 Hardware Research	3
2.1.1 Microcontrollers	3
2.2 Firmware	4
2.2.1 Camera	4
2.2.2 Atmel Software Framework	5
3 Hardware and Firmware Development	7
3.1 Camera	7
3.1.1 Single Camera Operation	8
3.1.2 Dual Camera Operation	9
3.2 SD Card	10
3.2.1 Storing Images	11
3.2.2 User Interface	12
3.3 Circuit Development	13
3.3.1 Stereo Camera Development	13
3.3.2 Motor Driver Development	14
3.3.2.1 Hardware	14
3.3.2.2 Firmware Development	14
3.3.2.3 Testing	16
3.3.2.4 Conclusion	16
3.4 PCB Development	18
3.4.1 Circuit Design	18
3.4.2 PCB Design	18
3.4.3 PCB Testing	19
3.4.3.1 UART Test	20
3.4.3.2 SD Card Test	21
3.4.3.3 LED Test	23
3.4.3.4 SDRAM Test	23
3.4.3.5 I^2C Test	24

3.4.3.6	Camera Test	25
3.4.3.7	Motor Driver Test	26
3.4.4	PCB Faults	27
3.4.4.1	SDRAM Footprint	27
3.4.4.2	SDRAM Chip Select	27
3.4.4.3	SDRAM Data Line Resistors	28
3.4.4.4	Camera Interrupt Line	29
3.4.4.5	Motor Driver Pinout	29
3.4.5	PCB Conclusions	30
3.5	Conclusions	31
4	Investigation into Vision Algorithms	33
4.1	Comparison	33
4.1.1	Sum of Absolute Differences	34
4.1.2	Sum of Squared Differences	34
4.1.3	NCC	34
4.1.4	Comparison	35
4.1.5	Conclusion	37
4.2	Range Finding	37
4.2.1	Derivations	37
4.2.1.1	Object is between the Cameras	37
4.2.1.2	Object is to the same side in each camera	39
4.2.1.3	Object is in front of a camera	40
4.2.1.4	Summary	40
4.3	Fourier Transform	43
4.3.1	Background Research and Uses	43
4.3.2	Fast Fourier Transform	43
4.3.3	Implementing the FFT in C	43
4.3.4	Testing of the FFT on AVR	43
4.4	Low Level Vision Algorithms	43
4.4.1	Noise Reduction	43
4.4.2	Edge Detection	43
5	Results	45
5.1	Results	45
6	Conclusions and Further Work	47
A	Gantt Chart	49
B	Circuit Diagrams	51
B.1	OV7670 Breakout Board Schematic	51
B.2	Il Matto and Dual Camera Schematic	51
B.3	The Columbus Circuit Diagram	51

C Bitmap File Format	61
C.1 Bitmap File Format	61
D Source Code	65
D.1 C Code for AVR	65
D.1.1 The Columbus Source Code	65
D.1.1.1 main.c	65
D.1.1.2 Bitmap.c	72
D.1.1.3 CustomDevices.h	73
D.1.1.4 dummy.c	74
D.1.1.5 ImageProcessor.h	74
D.1.1.6 ImageProcessor.c	75
D.1.1.7 MotorDriver.h	81
D.1.1.8 MotorDriver.c	82
D.1.1.9 OV7670.h	90
D.1.1.10 OV7670.c	96
D.1.1.11 OV7670.c	107
D.1.1.12 PCA9542A.h	110
D.1.1.13 PCA9542A.c	111
D.1.1.14 SD_Card.h	112
D.1.1.15 SD_Card.c	112
D.1.1.16 TWI.c	120
E PCB Design	123
E.1 PCB Top Side	123
E.2 PCB Bottom Side	123
References	127

List of Figures

1.1	The base of the robot	1
3.1	RGB565 pixel format	8
3.2	Signals generated to control the OV7670 capture and read	9
3.3	An Example Image taken using the OV7670 and stored as a Bitmap on the SD Card	12
3.4	Prototype of Dual Camera operation.	13
3.5	Circuit diagram of Optosensor	15
3.6	Graph of Wheel Angle against the Voltage read by the AVR	15
3.7	Dimensions of Interest for Robot Movement	17
3.8	PCB with no components. Left: Top View. Right: Bottom View	20
3.9	SDRAM Chip shown against its footprint.	27
3.10	Motor Driver error. Outputs incorrectly connected	30
3.11	Pictures of the built PCB.	32
4.1	Stereoscopic Test Images from MATLAB Examples	33
4.2	Result Graphs of Comparison Algorithms	36
4.3	Problem 1 - Object is between the Cameras	38
4.4	Problem 2 - Object is to the same side in both cameras	41
4.5	Problem 3 - Object is directly in front of a camera	42
A.1	Gantt Chart of how time will be spent in the areas of the project .	50
B.1	The circuit diagram for the OV7670 breakout board	52
B.2	The circuit diagram for Dual Cameras using the Il Matto Board .	53
B.3	The Columbus Circuit Diagram Page 1	54
B.4	The Columbus Circuit Diagram Page 2	55
B.5	The Columbus Circuit Diagram Page 3	56
B.6	The Columbus Circuit Diagram Page 4	57
B.7	The Columbus Circuit Diagram Page 5	58
B.8	The Columbus Circuit Diagram Page 6	59
E.1	The Top side of the CAD Design of the PCB	124
E.2	The Bottom side of the CAD Design of the PCB	125

List of Tables

1.1	A list of risks and the prevention steps taken to reduce their impact	2
2.1	Comparison Table of some common microcontrollers. Data of microcontrollers taken from Atmel Corporation (2012a), Atmel Corporation (2012b), Atmel Corporation (2012d) and Texas Instruments (2012). Costings from Farnell (2012)	5
3.1	A table comparing different image formats available (Fulton (2010))	11
3.2	Pin Connections of the ATMega644P for Dual Camera Operation.	13
3.3	A table of all components used and their costs.	19
3.4	A table showing examples of the incorrect data returned from the SDRAM	28
C.1	Format of a Bitmap file with values used, to write an image from the camera to an SD Card	61

List of Symbols

I^2C	Inter-Integrated Circuit
TWI	Two Wire Interface
SCCB	Serial Camera Control Bus
SPI	Serial Peripheral Interface
kB	KiloBytes
ISR	Interrupt Service Routine
φ_0	Field of view of the camera
φ_1, φ_2	Angle from camera to the object
B	Separation distance of two cameras
D	Distance from camera to the object
i, j	Pixel index of an Image
x_0	Horizontal resolution of the image
x_1, x_2	Distance of object from the normal of the camera

Chapter 1

Introduction

Talk about what I set out to do, include some definitions etc.

What I ended up doing

The uses of my robot.

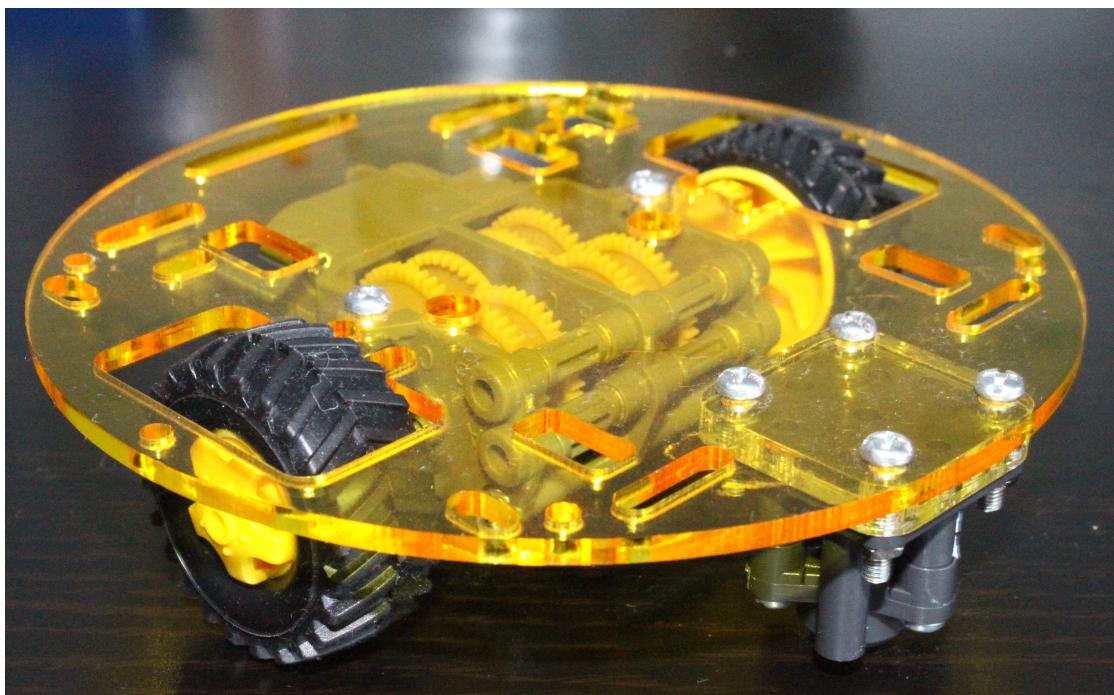


Figure 1.1: The base of the robot

Risk	Severity	Prevention
Parts not arriving on time	High	Order parts as early as possible
Project not fulfilling specification	High	Develop in stages to obtain functionality in parts. Ensure enough time is allocated to the project.
PCB Design is incorrect	Medium	Check the design carefully and get second opinion
Failure of personal computer causing data loss	Low	Keep back ups of all work on Devtrack Git repository and Dropbox.

Table 1.1: A list of risks and the prevention steps taken to reduce their impact

1.1 Project Management

In order to reduce the risk within the project, all aspects of potential issues are looked at and are summarised in table 1.1. A Gantt chart of how time will be spent can be seen in figure A.1.

The project will be designed in stages - first, gaining operation of all the basic sections; movement, image capturing, image detection algorithms etc. These will then be brought together once tested to create the final product.

Chapter 2

Research

The research for this project was split into three sections:

1. Hardware
2. Software, broken down into:
 - (a) Firmware, and
 - (b) Algorithms

Hardware and firmware research will be discussed in this section. Vision algorithms are looked at in detail in chapter [4](#).

2.1 Hardware Research

2.1.1 Microcontrollers

The robot is to be designed with a budget of £80 (not including P.C.B.). The choice of microcontroller will be an important one, as a compromise between cost, power and usability must be made. There are two main brands of microcontrollers present in the consumer market: ARM and Atmel AVRs.

ARM is an architecture which is developed by ARM Holdings. ARM devices come in a many varieties: ARM9, ARM7, Strong ARM, ARM Cortex etc. Whilst ARM Holdings do not fabricate and sell the devices themselves, many companies, such as Texas Instruments, use the architecture and manufacture their own devices.

ARM cores are based on a RISC Harvard architecture and tend to be 32-bit with a high clock speed. ARM microcontrollers have onboard support for SPI, I^2C , PWM, ADCs and can have Flash, SRAM and EEPROM memory built-in. For this comparison, the Stellaris by Texas Instruments will be examined.

Atmel have a variety of products in the microcontroller market. They range from 8-bit, low clock speed devices for the hobbyist (ATMega and ATTiny series), to an improved 8-bit variant (XMega), and a 32-bit design (AT32UC3). XMegas and AVR32s tend to have higher clock speeds than the ATMegas. The AVR core also has a Harvard RISC architecture, and is mainly 8-bit. Atmel devices often have on board peripherals such as I^2C (called TWI on AVRs), SPI and ADCs, as well as a number of different memories: Flash, EEPROM and SRAM. An AT32UC3C0512C, ATXmega256A3BU and ATMega644P will be compared in this section.

Table 2.1 shows a brief summary of some common ARM and AVR microcontrollers. The Stellaris offers the most power with the largest DMIPS performance. However, due to the necessity of floating point operations, the AT32 clearly has a distinct advantage by having a built-in floating point unit. The XMega and ATMega do not offer enough power and are restricted by a small amount of SRAM and Flash. All devices looked at use 3.3V supply and have basic communication protocols (SPI, I^2C and USART). Overall, the AT32UC3C0512C is the best choice with a high throughput, a floating point unit and a vast amount of GPIO and communications. There is no EEPROM which may be desirable, but these can be added onto an SPI or I^2C bus. This device, although slightly more costly, is best suited to this application out of the selection researched.

2.2 Firmware

2.2.1 Camera

The camera used is the OV7670 camera by OmniVision. Steve Gunn provided source code for use on the Il Matto development board which uses an ATMega644P and also has an onboard SD Card reader. The original code streamed video from the camera to a colour TFT screen. The camera is supplied on a small breakout board with a FIFO buffer. The camera operation is discussed in section 3.1. Many implementations of firmware for this camera exist.

	ARM Stelllaris	AT32UC3C0512C	XmegaA3BU	ATmega644P
Clock Speed (MHz)	80	33 or 66	32	12
DMIPS	100	91	-	20 MIPS
Package	100 LQFP or 108 BGA	64, 100, 144TQFP	64 QFP or QFN	40 DIP, 44 TQFP, 44 QFN
Cost of 1 unit(£)	10.30	15.39	6.65	6.86
Flash Size(kB)	256	512	256	64
SRAM Size (kB)	32	64	16	4
EEPROM Size(kB)	2	None internal	4	2
GPIO	64	45, 81 or 123	47	32
Operating Voltage (V)	3.3	5 or 3.3	1.6- 3.6 ¹	2.7-5.5
Communication Interfaces	SPI, I^2C , SSI, MAC, CAN, EPI, USB, US- ART, I2S	SPI, TWI, EBI, USB, Ethernet, CAN, USART, I2S	USART, TWI, USB, SPI	SPI, TWI, USART
Floating Point	None	Built in FPU	None	None
ADCs	16	16	16	8
Timers	4	3 16-bit	7 16-bit, 8 8-bit	2 8-bit, 1 16-bit

Table 2.1: Comparison Table of some common microcontrollers. Data of microcontrollers taken from [Atmel Corporation \(2012a\)](#), [Atmel Corporation \(2012b\)](#), [Atmel Corporation \(2012d\)](#) and [Texas Instruments \(2012\)](#). Costings from [Farnell \(2012\)](#)

2.2.2 Atmel Software Framework

Atmel offer a software framework which contains basic code and device drivers for many of their Xmega and AT32 devices ([Atmel Corporation \(2009\)](#)). There are also many AVR application notes which provide explanations and example code for protocols like TWI, SPI and timers. These application notes are aimed at older devices like the ATTiny and ATMega and are generally written for IAR Embedded Workbench compiler, as opposed to the AVRGCC compiler used within Atmel Studio.

Chapter 3

Hardware and Firmware Development

For initial development, the *Il Matto* board, designed by Steve Gunn, was used. The system has an ATmega644P clocked at 12MHz and has an on-board SD card socket.

The following section is broken down into the following parts:

1. Camera Code
2. SD Card
3. Circuit Development
4. PCB Development

3.1 Camera

The camera used is an OV7670 by OmniVision. It is mounted onto a break out board and connected to a AL422B FIFO Buffer. The breakout board has all passive components needed and a 24MHz clock mounted. The schematic for the device can be seen in appendix [B](#).

Original code for the camera operation was given by Steve Gunn, which I used to gain the operation required. This code streamed continuous video to a TFT screen. The operation required was to take a single photo from the camera and store the data.

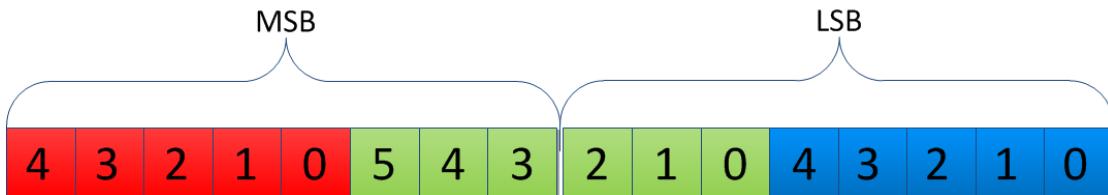


Figure 3.1: RGB565 pixel format

3.1.1 Single Camera Operation

The camera uses a SCCB Interface ([OmniVision, 2007](#)) created by OmniVision. This is almost identical to the I^2C Interface by Phillips and the two protocols are compatible. The original code used a bit-banged SCCB interface which was very slow and used up processing time. This was changed to make use of the built-in interrupt-driven I^2C interface (named TWI in Atmel AVR's)¹. This communication bus is used to set up the control registers of the OV7670 to enable operation in the correct format. RGB565 is used in my application.

RGB565 is a 16 bit pixel representation where bits 0:4 represent the blue intensity, 5:10 is the green intensity and 11:15 represent the red intensity (see figure 3.1). This is a compact way of storing data but only allows 65536 colours. Greys can also appear to be slightly green due to the inconsistent colour ratio of the green field.

The camera must use a high speed clock in order to ensure the pixels obtained are from the same time. This makes it difficult for an AVR to be able to respond to the camera quick enough (ATMegas typically clocked at 8-12MHz). This highlights the necessity for a FIFO Buffer.

The OV7670 is set up so that the VSYNC pin goes low at the beginning of every full frame of data, and HREF is high when the data being output is valid. The pixel data is then clocked out on every rising edge of PCLK. To control the buffer, WEN (write enable) is NAND with the HREF signal. When both are high, the write enable to the buffer will be active and the data will be clocked in by PCLK. In order to acquire a full frame, the first VSYNC pin is set up to interrupt the AVR to enable WEN. The camera will output an entire frame of pixel data and store it into the buffer. When the second VSYNC is received, the WEN signal is disabled, stopping any more data being stored. The FIFO buffer now contains an entire image.

¹ I^2C , SCCB and TWI are all the same but are called differently due to Phillips owning the right to the name " I^2C "

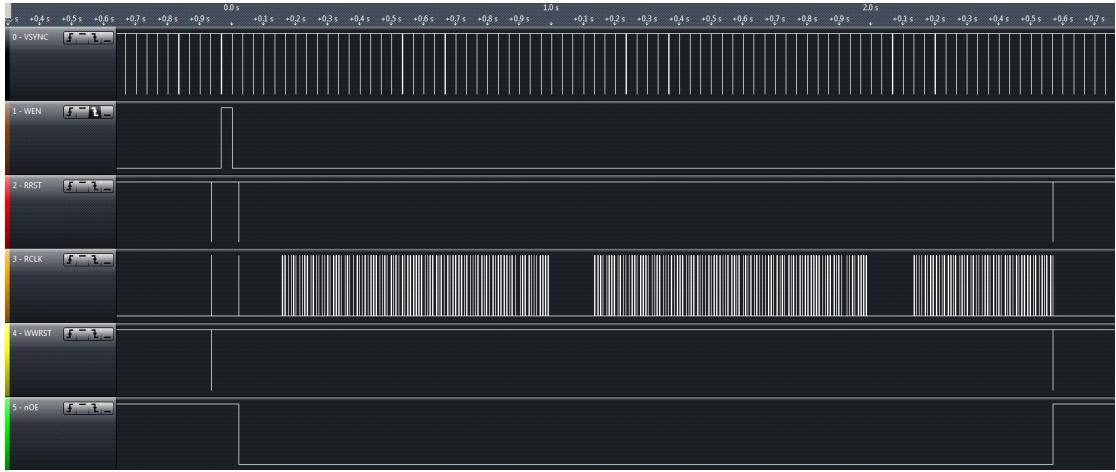


Figure 3.2: Signals generated to control the OV7670 capture and read

To obtain the data from the buffer, the AVR sets output enable and pulses the read clock. Valid data is available on the input port while RCLK is high. All the data is then read in half a pixel at a time. The entire operation can be seen in figure 3.2.

Difficulties arose at this point with the storage of the data. The ATMega644P has 4kB of internal SRAM, but 153.6kB of memory is needed to store a single image at QVGA (320 by 240 pixels, 2 bytes per pixel) quality.

Firstly, data was sent straight to a desktop computer via a COM Port using USART. A simple desktop program was written in C# to receive and store all the data, and to make a Bitmap image from the data. This method was slow, taking around 30 seconds to transmit one uncompressed image.

The second option was to use extra memory connected to the microcontroller. An SD card is used as FAT file system so that data can be looked at by a user on a computer. Text log files are also written to aid debugging. This is discussed in section 3.2.

3.1.2 Dual Camera Operation

In order for stereovision to be successful, two cameras separated by a horizontal distance (B) will need to be driven at the same time to obtain photos within a small time frame of one another.

A major problem occurred with using the I^2C interface to set up both cameras. The camera has a set I^2C address of 21_{16} , which cannot be changed. Multiple

I^2C devices with exactly the same address cannot be used on the same bus. Two solutions to this are possible: driving one from I^2C and one from SCCB, or using an I^2C multiplexer. By using two different buses, there can be no bus contention. However, SCCB is slow and processor-hungry as it deals with the protocol bit by bit in software. This takes up memory and is not reusable for other operations.

An I^2C multiplexer sits on the bus and has multiple output buses. The master can then address the multiplexer and select whether to pass the bus to bus 0, bus 1 or not allow the data to be transferred. This saves processor time, but means a write operation has to be done to select the camera bus before being able to write to the camera. This slows down the operation, but not as much as using SCCB. The main disadvantage to the I^2C MUX is the extra hardware needed; firstly the MUX itself, but also 7 extra resistors to pull up the two extra buses and the three interrupt lines must be added.

Overall, the disadvantages posed by using a MUX are small, so a multiplexer will be used as opposed to the SCCB interface. A suitable multiplexer is the Phillips PCA9542A ([Phillips, 2009](#)).

The buffers have an output enable pin so the data bus can be shared by both cameras to the AVR. The ATMega644P offers three interrupt pins, two of which are used by the two VSYNC pins for the cameras.

Two ISRs are used to control the VSYNC signals, and when taking a photo, both frames are taken at a time period close together to capture the same scenario. The data for both images are read back individually by the AVR.

Operation to read an image is identical to using one camera. However, an ID number is passed through the functions to make a decision on the pins to use to read the buffer and to enable the output. Care was taken to avoid bus contention, but no checking procedure is explicitly in place. Both images are then read back from the buffers and stored to memory.

3.2 SD Card

To use the SD card, the FATFS library ([Electronic Lives Manufacturing, 2012](#)) was used. The library supplies all the functions for writing a FAT File System in the files *ff.c*, *ff.h*, *ffconf.h*, *diskio.c*, *diskio.h* and *integer.h*. The *diskio.h* functions control what device is being used - SD/MMC Card, USB drive etc. The *ff.h* header

	Bitmap	JPEG	PNG	GIF
Extension	*.bmp	*.jpg /*.jpeg	*.png	*.gif
Compression	No	Lossless and Lossy	Lossless ZIP	Lossy
File Size of 320 by 240 pixel Im- age (kB)	225	20	23	24
Bits per Pixel	8, 16, 24 or 32	24	24, 32 or 48	24, but only 256 Colours

Table 3.1: A table comparing different image formats available ([Fulton \(2010\)](#))

contains all the functions to write to in a FAT File system.

An SD card was chosen due to it's small size, low cost and a large data storage. The cards work using an SPI bus which can be used for other devices within the system so the card only uses one extra enable pin in hardware to function.

3.2.1 Storing Images

Many image formats are common, such as Joint Photographic Expert Group (JPEG), Portable Network Graphics (PNG), Bitmap (BMP) and Graphics Interchange Format (GIF). Table 3.1 shows a summary of some common image formats.

It is clear that the best choice for images would be either PNG or JPEG. However, these require more computational time to compress the image into the correct format. To avoid compression, and thereby save processing time, bitmap was chosen at the expense of using more memory. The data in a bitmap image is also stored in RGB format so can be read back easily when processing the image. Appendix C shows the make up of a Bitmap File that was used.

By writing the image in this format, they are then able to be opened on any operating system. This aids debugging and allows the prototyping of image algorithms in a more powerful environment. Figure 3.3 shows a photo taken by the OV7670 and stored on a SD card.



Figure 3.3: An Example Image taken using the OV7670 and stored as a Bitmap on the SD Card

3.2.2 User Interface

The ATMega 664P pinout for the dual camera operation can be seen in table 3.2. Due to a lack of available GPIO pins, an ATMega168 was added on the I^2C bus to act as a port extender. The ATMega168 accepts a read or write command. A write places the written data on Port D and a read returns any button pressed that occurred on Port C. When a button is pressed, this is stored in the ATMega168 until a read has been done. This is so the master (644P) does not miss any button presses while busy doing lengthy operations such as writing an image. The code is based on Application Note AVVR311 ([Atmel Corporation, 2007](#)), written for IAR Compiler. This code was altered to compile with GCC under Atmel Studio. AVRs contain a hardware based I^2C protocol that is interrupt based in software. The interrupt service routine of the TWI vector is a state machine which loads the data to send, stores received data, responds to acknowledges and address calls and deals with bus errors that can occur.

	Port A	Port B	Port C	Port D
0	Data 0	SD Write Protect	I^2C - SCL	No Connection
1	Data 1	SD Card Detect	I^2C - SDA	No Connection
2	Data 2	USB Data Plus	Read Clock 1	VSync 0
3	Data 3	USB Data Minus	Read Reset 1	VSync 1
4	Data 4	SPI Chip Select	Write Enable 1	Read Clock 0
5	Data 5	SPI MOSI	Write Reset 1	Read Reset 0
6	Data 6	SPI MISO	Output Enable 0	Write Enable 0
7	Data 7	SPI Clock	Output Enable 1	Write Reset 0

Table 3.2: Pin Connections of the ATMega644P for Dual Camera Operation.

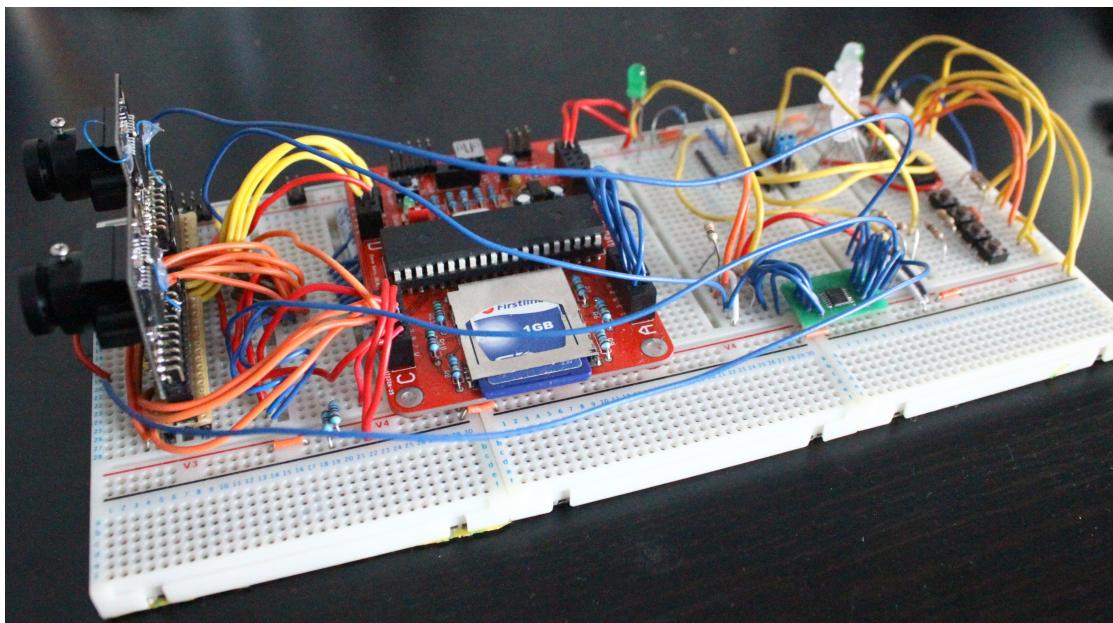


Figure 3.4: Prototype of Dual Camera operation.

3.3 Circuit Development

3.3.1 Stereo Camera Development

Figure B.2 shows the circuit diagram for the prototype. This uses the Il Matto development board for the main microcontroller. The prototype can be seen in figure 3.4. This circuit captured and stored two images from the cameras to the SD card.

3.3.2 Motor Driver Development

Testing of the Motor system - conclusion is likely to be that it is not a good method, need noise reduction

3.3.2.1 Hardware

Tachometers are devices used to measure rotational speed of a shaft. Tachometers are most commonly found in bicycles where a small magnet is attached to the wheel and a sensor is attached to the frame. The sensor can then calculate the time period between rotations and therefore can calculate the speed ((?))

Cite Needed

Here, an optosensor, TCRT1010, is used to measure rotations of the wheel and used to be able to move a distance decided by the microcontroller. The TCRT1010 package contains an IR LED and a phototransistor ([Vishay Semiconductors, 2012](#)). The schematic of a simple transistor amplifier used can be seen in figure 3.5 and was taken from ?.

Cite Needed

The wheel's rubber absorbed the IR, so a high voltage was always seen at the collector of the phototransistor. White tippex marks were applied to the wheels at regular intervals, which reflected IR and thereby giving a cheap way to detect wheel rotation. Figure 3.6 shows the voltage at the collector (read by the ADC on the AVR) against the angle of the wheel. Five white tabs were marked on the wheel, and five dips in the voltage can be seen in figure 3.6.

Maybe do some simulations of this circuit? This could dictate a maximum speed

3.3.2.2 Firmware Development

As the voltage swing from the phototransistor does not reach 0V, the AVR cannot detect this as a logical 0. The internal ADC can be used to continually read the analogue voltage from the phototransistor and detect low points from this data.

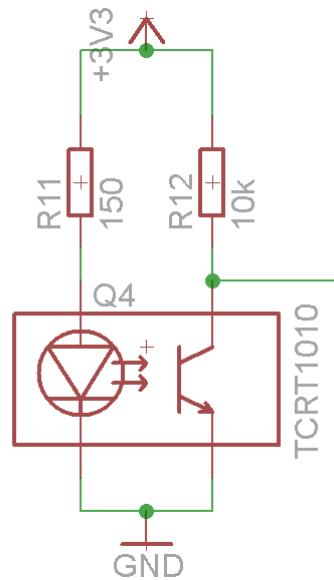


Figure 3.5: Circuit diagram of Optosensor

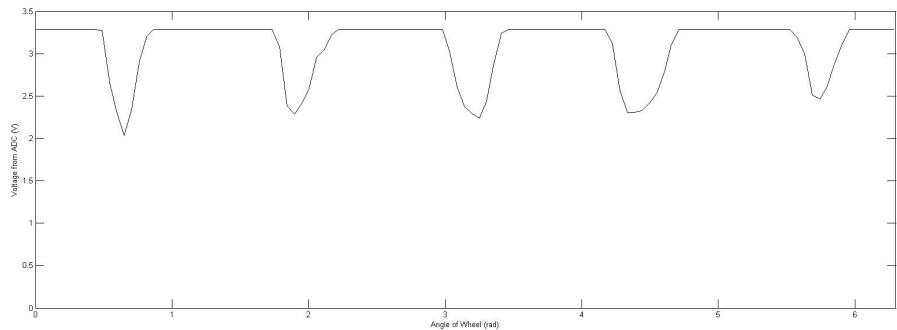


Figure 3.6: Graph of Wheel Angle against the Voltage read by the AVR

This method requires the processor to continually compare values and process the data. However, more control would be had over the noise in the data.

An alternative is to use an analogue comparator built in on most AVRs. This can be set up to run an interrupt service routine when the voltage crosses a threshold. The threshold voltage can be determined by a potentiometer. The code is primarily two methods, the set up and the ISR.

Set up is different for if a rotation or movement is wanted. Moving in a straight line takes a parameter of how far to move as a signed integer and calculates the total number of interrupts that need to occur can be calculated using (3.1). This value is put in a global variable. The PWM and input pins to set the correct direction are then set up before enabling the motor.

$$\text{Interrupts} = D \times \frac{\text{IPR}}{C_w} \quad (3.1)$$

To rotate, one of three methods can be used: Spot rotation on the centre of the robot, or pivot on either left or right wheel. For ease, the Spot rotation is the only one implemented. To calculate the distance moved, the radius to the wheels needs to be known. The circumference through the wheels is then easily calculated and the angle of rotation is then a ratio. The distance to move is calculated by equation (3.2) and the total number of interrupts can be calculated using equation (3.1). To rotate clockwise, the left motor is driven forward and the right is driven backwards. To rotate anti-clockwise, the directions are reversed

$$D_R = A \times \frac{C_b}{360} \quad (3.2)$$

Combining equations (3.1) and (3.2) gives:

$$\text{Interrupts} = A \times \frac{\text{IPR}}{C_w} \times \frac{C_b}{360} \quad (3.3)$$

Where A is the angle to rotate in degrees, IPR is the number of interrupts generated per full revolution of the wheel, C_w is the circumference of the wheel and C_b the $2\pi \times r_b$ and r_b is the distance from the centre of the robot to the centre of the wheel (see figure 3.7).

Maybe a figure to explain better?

The motor speed can be controlled by Pulse Width Modulation (PWM). The code sets up a low duty cycle PWM signal to drive the motors slowly. This removes the need for a controller to ensure the correct distance was moved.

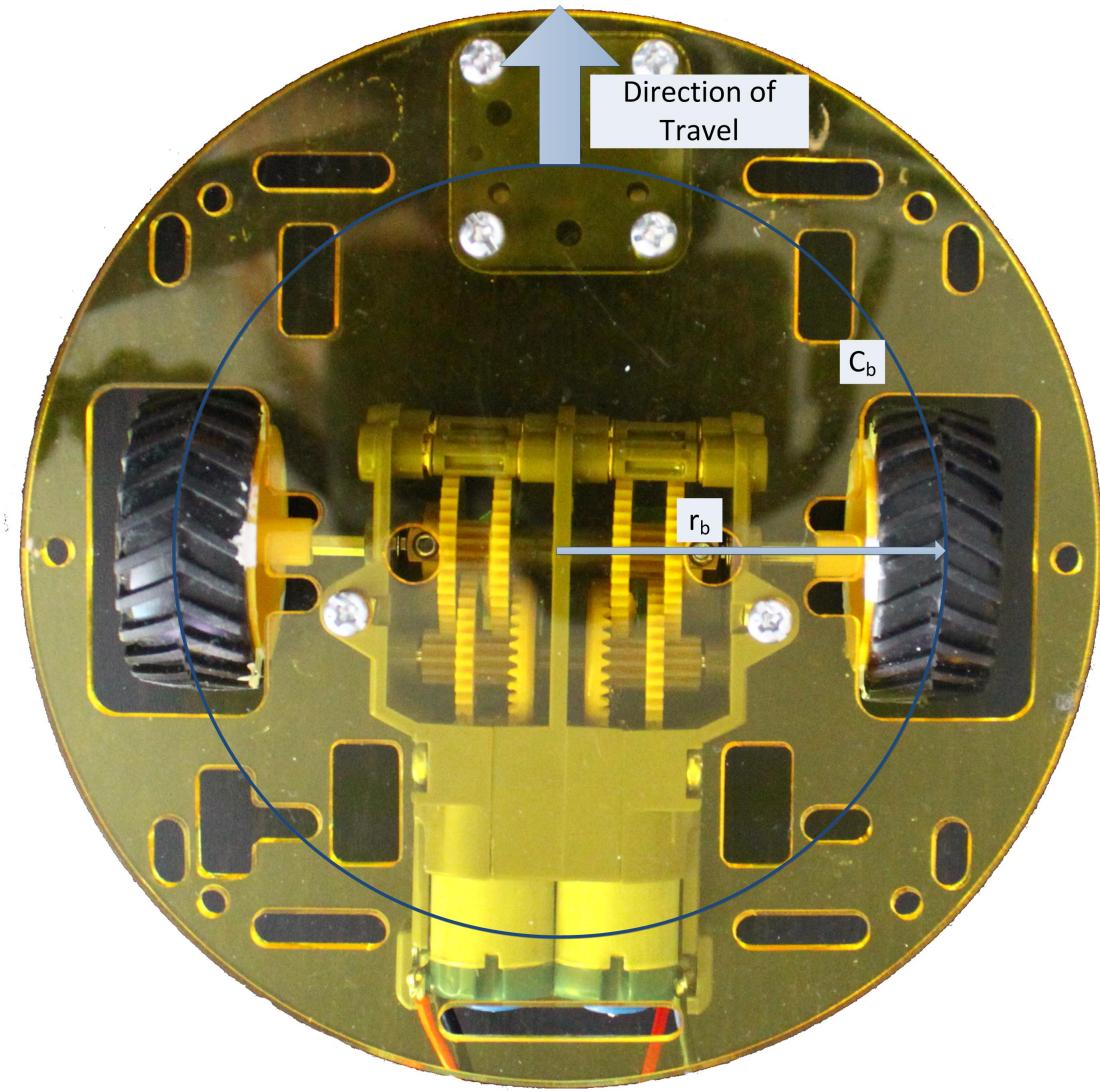
The final code can be seen in appendix D

3.3.2.3 Testing

Need to get the motors to work reasonably first

3.3.2.4 Conclusion

It doesn't work



(a) Top View of robot base showing dimensions of interest



(b) Side View of robot base showing dimensions of interest

Figure 3.7: Dimensions of Interest for Robot Movement

3.4 PCB Development

3.4.1 Circuit Design

The circuit diagram for Revision A can be seen in section [B.3](#). The schematic for the SDRAM and values and locations of decoupling capacitors were taken from the schematic of the UC3C-EK development board ([Atmel Corporation, 2012c](#)).

3.4.2 PCB Design

The PCB was designed using EAGLE CAD Software. A four layer board was decided to be used to save space when routing Power. Layer 2 is a 3V3 plane and layer 3 is a ground plane. A ground plane is also on the top and bottom layers to help eliminate any ground bounce that could occur.

The SDRAM uses the EBI protocol. In high speed systems, care is often taken to equalise track lengths ([Liu and Lin, 2004](#)). The UC3C maximum clock frequency is 33MHz (with no wait states), which is not fast enough to cause any track equalisation problems. Care, however, was taken on the USB lines to ensure correct impedance and the tracks lengths matched to each other.

Tracks were routed in order of priority, starting with the UC3C, SDRAM and cameras, and then other devices were routed, I^2C MUX, SD card, motor drivers etc. As a precaution, spare pins from the UC3C were routed to a header (J8 and J9) so that additions could be done if a pinout or connection was found to be incorrect. Also, UART, I^2C and SPI connections were routed to headers J7, J4 and J5 respectively so logic analysers and COM Port could be attached easily for debugging.

Most of the passives used were 0603 size, but some 1206 capacitors were used for decoupling the voltage regulator and a 1206 diode was used for the analogue reference circuitry. LEDs were also 1206 size. All headers were 0.1" spaced and a mini B USB socket was used.

The layout of components was important. The cameras needed to be as far apart as possible and at the front of the PCB. The motor drivers were situated toward the back of the PCB and 0.1" headers were added to connect the motors to. The optosensors were positioned such that they could be mounted directly on the PCB and be in the correct position to sense the wheels. Mounting holes were also added

Component	Cost per unit (£)	Quantity	Cost (£)	Source
Capactiors	0.155	43	6.67	Farnell
Clock	1.48	1	1.48	Farnell
Diode	0.48	1	0.48	Farnell
Headers	0.51	5	2.55	Farnell
I2C Mux PCA9542A	0.81	1	0.81	Farnell
LEDs	0.158	7	1.11	Farnell
Micro SD Card	4	1	4.00	Amazon
Micro SD Card Connector	2.04	1	2.04	Farnell
AT32UC3C0512C	15.39	1	15.39	Farnell
TB6593FNG	1.07	2	2.14	Farnell
Motors	0.42	2	0.84	Rapid
TCRT1010	0.94	2	1.88	Farnell
OV7670	17	2	34.00	
Potentiometer	0.43	2	0.86	Farnell
Resistors	0.066	46	3.04	Farnell
MT48LC4M16A2P	3.24	1	3.24	Farnell
Switch	0.45	1	0.45	Farnell
USB Socket	0.84	1	0.84	Farnell
LM1117MP	1.03	1	1.03	Farnell
		Total Cost	£82.84	

Table 3.3: A table of all components used and their costs.

onto the board so the PCB could be mounted on to the robot base easily. The overall dimensions of the PCB were $100mm \times 70mm$. A full list of components and cost of each is documented in table 3.3

Finally, the name “The Columbus” was decided on as the original application for the project was a mapping robot that would search out an unknown area, so the robot was named after Christopher Columbus who explored and navigated parts of the American continents which were unknown at the time. The Eagle CAD Diagram of the PCB can be seen in Appendix E. The PCB was manufactured by [Cart \(2013\)](#). The PCB cost £205 to manufacture and ship. A photo of the PCB can be seen in figure 3.8.

Considerations - Power consumption of devices not exceeding VReg

3.4.3 PCB Testing

A program was written to test all the devices on the PCB. The following tests are done:

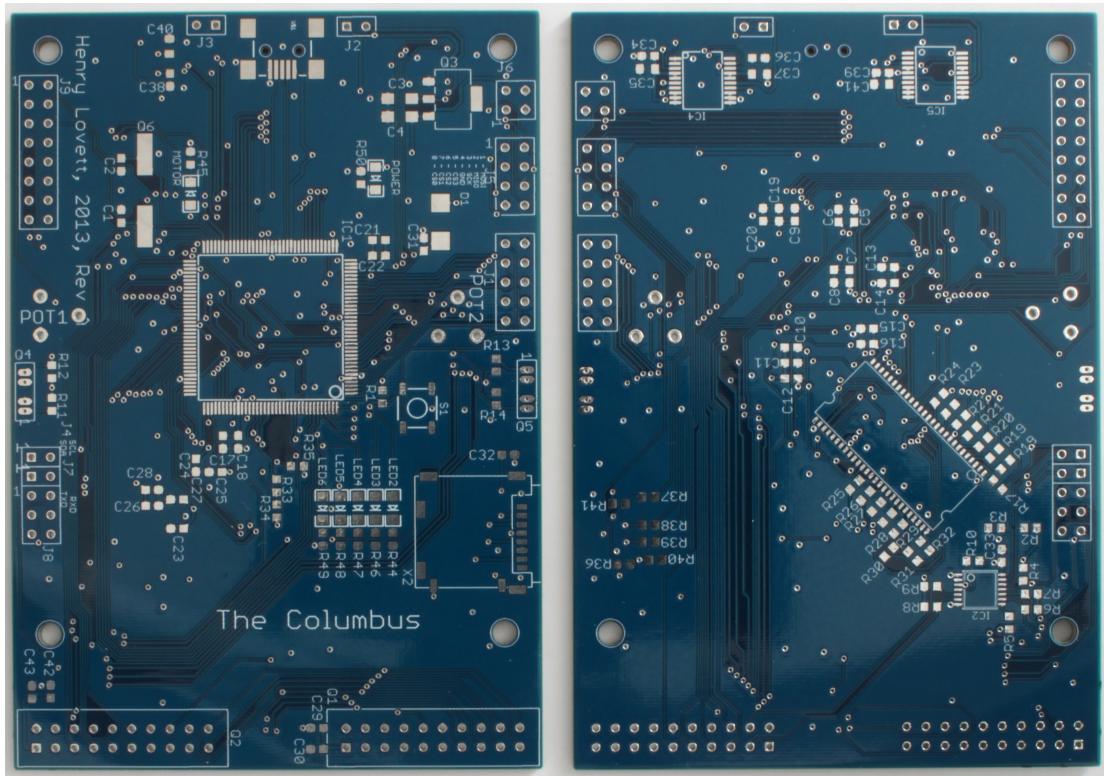


Figure 3.8: PCB with no components. Left: Top View. Right: Bottom View

1. UART Send and Receive
2. SD Card Test
3. All LEDs on and off
4. SDRAM Test
5. I^2C Test
6. Camera Test
7. Motor Test

The following sections explain the tests done to check the devices and protocols worked.

3.4.3.1 UART Test

When the test program begins, the microcontroller waits for a character input. All characters are echoed back. This enables the user to check the communications

work. Once a carriage return key is received (13_{10}), the test program continues. Listing 3.1 shows the test code for the UART protocol.

Listing 3.1: UART Test Code

```

1 ///////////////////////////////////////////////////////////////////
2 // UART Test ///////////////////////////////////////////////////////////////////
3 ///////////////////////////////////////////////////////////////////
4 uint8_t ch;
5 while (true) {
6     ch = usart_getchar(DBG_USART); // get one input character
7     if (ch) {
8         print_dbg(ch); // echo to output
9     }
10    if(ch == 13)
11        break;
12 }
```

3.4.3.2 SD Card Test

The Atmel Software Framework ([Atmel Corporation, 2009](#)) provided drivers and code for SPI communications and use of a FAT32 File System. The code was configured to use the correct Chip Select pin for the SD Card and the correct SPI Bus was also configured. The test consists of initialising the memory, reading the capacity of the card and printing it to the user.

The AVR then proceeds to delete any previous log file, create a new log file and writes “*Columbus Tester*” to it. The first 8 characters, which should be “*Columbus*” are read back and checked.

Listing 3.2: UART Test Code

```

1 ///////////////////////////////////////////////////////////////////
2 // SD Card Test ///////////////////////////////////////////////////////////////////
3 ///////////////////////////////////////////////////////////////////
4 print_dbg("\n\n\rSD Card Memory Test:\n\r");
5 // Test if the memory is ready - using the control access memory abstraction
6 // layer (/SERVICES/MEMORY/CTRL_ACCESS/)
7 if (mem_test_unit_ready(LUN_ID_SD_MMC_SPI_MEM) == CTRL_GOOD)
{
8     // Get and display the capacity
9     mem_read_capacity(LUN_ID_SD_MMC_SPI_MEM, &VarTemp);
10    print_dbg("OK:\t");
11    //printf_ulong((VarTemp + 1) >> (20 - FS_SHIFT_B_TO_SECTOR));
12    i = ((VarTemp + 1) >> (20 - FS_SHIFT_B_TO_SECTOR));
13    print_dbg_ulong(i);
14    print_dbg("MB\r\n");
15    print_dbg("SD Card Okay.\n\r");
16 }
```

```
17     else
18     {
19         // Display an error message
20         print_dbg("Not initialized: Check if memory is ready...\r\n");
21     }
22     nav_reset();
23     // Use the last drive available as default.
24     nav_drive_set(nav_drive_nb() - 1);
25     // Mount it.
26     nav_partition_mount();
27     nav_filelist_reset();
28     if(nav_filelist_findname((FS_STRING)LOG_FILE, false))
29     {
30         print_dbg("\n\rLog File Already Exists\n\rAttempting to delete...");
31         nav_setcwd((FS_STRING)LOG_FILE, true, false);
32         nav_file_del(false);

33         if(nav_filelist_findname((FS_STRING)LOG_FILE, false))
34             print_dbg("\n\rLog File Still Exists...");
35         else
36             print_dbg("\n\rLog File Deleted!");
37     }
38     print_dbg("\n\rCreating Log File.");

39     if(nav_file_create((FS_STRING)LOG_FILE) == true)
40         print_dbg("\n\rSuccess!");
41     else
42         print_dbg("\n\rNot worked...");
43     print_dbg("\n\rWriting to log file.");
44     Log_Write("Columbus Tester:\n\r", 18);
45     nav_filelist_reset();
46     nav_setcwd((FS_STRING)LOG_FILE, true, false);
47     file_open(FOPEN_MODE_R); //Open File
48     file_read_buf(Buffer, 8);
49     noErrors = 0;
50     if(Buffer[0] != 'C')
51         noErrors++;
52     if(Buffer[1] != 'o')
53         noErrors++;
54     if(Buffer[2] != 'l')
55         noErrors++;
56     if(Buffer[3] != 'u')
57         noErrors++;
58     if(Buffer[4] != 'm')
59         noErrors++;
60     if(Buffer[5] != 'b')
61         noErrors++;
62     if(Buffer[6] != 'u')
63         noErrors++;
64     if(Buffer[7] != 's')
65         noErrors++;
66     file_close();
67     if(noErrors == 0)
68         print_dbg("\n\rSD Card Read Successful\n\r");
69     else
70         print_dbg("\n\rSD Card Read Fail\n\r");
71     noErrors = 0;
```

This exercises all basic File I/O functions, creating, reading and writing and checks them on the device.

3.4.3.3 LED Test

All LEDs are turned on for 1 second, and then turned off. The user should check this occurs. It verifies that all the LEDs are functional and correctly mounted. The Power LED should be on when power is supplied to the PCB.

3.4.3.4 SDRAM Test

The SDRAM test consists of initialising the SDRAM, calculating the SDRAM Size, writing a unique test pattern to the whole memory, and then reading it back and checking it. The total number of errors are reported.

The test was adapted from an Example Application from the Atmel Software Framework ([Atmel Corporation, 2009](#)). The code can be seen in listing 3.3. It consists of two *for* loops. In the first, the iteration number is assigned to the memory location. The second loop reads back the data and checks it is correct. An int, *noErrors*, is used to count errors.

Listing 3.3: SDRAM Test Code

```

1 ///////////////////////////////////////////////////////////////////
2 // SDRAM Test ///////////////////////////////////////////////////////////////////
3 ///////////////////////////////////////////////////////////////////
4 print_dbg("\n\n\rSDRAM Test:");
5 sdram_size = SDRAM_SIZE >> 2;
6 print_dbg("\n\rSDRAM size: ");
7 print_dbg_ulong(SDRAM_SIZE >> 20);
8 print_dbg(" MB\r\n");
9 // Determine the increment of SDRAM word address requiring an update of the
10 // printed progression status.
11 progress_inc = (sdram_size + 50) / 100;
12 // Fill the SDRAM with the test pattern.
13 for (i = 0, j = 0; i < sdram_size; i++)
14 {
15     if (i == j * progress_inc)
16     {
17         print_dbg("\rFilling SDRAM with test pattern:");
18         print_dbg_ulong(j++);
19         print_dbg("%");
20     }
21     sdram[i] = i;
22 }
23 print_dbg("\rSDRAM filled with test pattern      \r\n");

```

```

25 // Recover the test pattern from the SDRAM and verify it.
26 for (i = 0, j = 0; i < sdram_size; i++)
27 {
28     if (i == j * progress_inc)
29     {
30         print_dbg("\rRecovering test pattern from SDRAM: ");
31         print_dbg_ulong(j++);
32         print_dbg("%");
33     }
34     tmp = sdram[i];
35     if (tmp != i)//failed
36     {
37         noErrors++;
38     }
39 }
40 print_dbg("\rSDRAM tested: ");
41 print_dbg_ulong(noErrors);
42 print_dbg(" corrupted word(s)      \r\n");

```

3.4.3.5 I^2C Test

The I^2C test checks the bus for devices. It prints out a table showing the address of any devices that acknowledge a probe. A probe is a set up to write to the address. If a device exists on the line, it should Acknowledge (Philips, 20012). The test is done three times, with no channel selected on the I^2C MUX, with channel 0 selected and with channel 1 selected. The two addresses expected at 21_{16} for the OV7670 Camera and 74_{16} for the I^2C MUX. The camera should only acknowledge when the I^2C MUX has the relevant channel selected. Listing 3.4 shows the test code for the I^2C bus and listing 3.4 shows the result from the full bus scan with channel 0 selected. The cameras are both checked to exist.

Listing 3.4: I^2C Test Code

```

1 ///////////////////////////////////////////////////////////////////
2 // TWI Test ///////////////////////////////////////////////////////////////////
3 ///////////////////////////////////////////////////////////////////
4 print_dbg("\n\n\rTWI Test:\n\r");
5 Log_Write("\n\n\rTWI Test:\n\r", 14);
6 for(k = 0; tkmp < 3; k++)
7 {
8     if(k == 0){
9         print_dbg("Scanning all Channels\n\r");
10    }
11    else if (k == 1){
12        //Channel 0
13        PCA9542A_Chan_Sel(I2C_CHANNEL_0);
14        print_dbg("\n\rScanning Channel 0\n\r");
15    }
16    else {

```

```

17     //Channel 1
18     PCA9542A_Chан_Sel(I2C_CHANNEL_1);
19     print_dbg("\n\rScanning Channel 1\n\r");
20 }

22     print_dbg("h 0 1 2 3 4 5 6 7 8 9 A B C D E F\n\r");
23     tmp = 0;
24     for(i = 0; i < 8; i++)
25     {
26         print_dbg_ulong(i);
27         print_dbg(" ");
28         for(j = 0; j < 16; j++){
29             int status = twim_probe(TWIM, tmp++);
30             if(status == STATUS_OK){
31                 print_dbg("A");
32             }
33             else{
34                 print_dbg("-");
35             }
36             print_dbg(" ");
37         }
38         print_dbg("\n\r");
39     }
40 }
41 noErrors = 0;
42 //Check cameras exist
43 PCA9542A_Chан_Sel(I2C_CHANNEL_0);
44 if(twim_probe(TWIM, 0x21) != STATUS_OK)
45     print_dbg("\n\rCamera 0 Not Found;");
46 PCA9542A_Chан_Sel(I2C_CHANNEL_1);
47 if(twim_probe(TWIM, 0x21) != STATUS_OK)
48     print_dbg("\n\rCamera 1 Not Found;");
```

Listing 3.5: Result of I^2C bus scan with Channel 0 of the I^2C MUX selected

```

1 Scanning Channel 0
2 h 0 1 2 3 4 5 6 7 8 9 A B C D E F
3 0 - - - - - - - - - - - - - - - -
4 1 - - - - - - - - - - - - - - - -
5 2 - A - - - - - - - - - - - - - -
6 3 - - - - - - - - - - - - - - - -
7 4 - - - - - - - - - - - - - - - -
8 5 - - - - - - - - - - - - - - - -
9 6 - - - - - - - - - - - - - - - -
10 7 - - - - A - - - - - - - - - - -
```

3.4.3.6 Camera Test

This test consists of initialising both cameras and checking it passes. Two photos are then taken and stored to the SD card. Success or Failure is displayed. Two

images should exists on the SD card from the two cameras. Listing 3.6 shows the code to conduct this test.

Listing 3.6: Camera Test Code

```

1 ///////////////////////////////////////////////////////////////////
2 // Camera Test ///////////////////////////////////////////////////////////////////
3 ///////////////////////////////////////////////////////////////////
4 print_dbg("\n\rInitialising Cameras");
5 OV7670_Init();
6 FIFO_Reset(CAMERA_LEFT | CAMERA_RIGHT);
7 if(STATUS_OK == OV7670_Status.Error)
{
8     print_dbg("\n\rCamera Initialise Okay!");
9 }
10 else
11     print_dbg("\n\rCamara Initialise Fail.");
12
13 print_dbg("\n\rTaking Photos");
14
15 TakePhoto(CAMERA_LEFT | CAMERA_RIGHT);
16 while(Photos_Ready() == false)
17     ;
18
19 if(Store_Both_Images() == true)
20     print_dbg("\n\rImages Stored Successfully!");
21 else
22     print_dbg("\n\rImages Store Fail.");
23

```

3.4.3.7 Motor Driver Test

An extensive test of the motor driver is discussed in section 3.3.2.3. The test code in this application resets the motors so that they are aligned to a white tab on the wheel. This code can be seen in listing 3.7. The robot should move no further than 2cm to reach a white tab and the motors should drive forward. This test is useful here to ensure the motors are connected the correct way around and that the potentiometers are set to an appropriate level.

Listing 3.7: Motor Test Code

```

1 ///////////////////////////////////////////////////////////////////
2 // Motor Test ///////////////////////////////////////////////////////////////////
3 ///////////////////////////////////////////////////////////////////
4 print_dbg("\n\rMotor Testing:\n\rMotor Initialised");
5 Motor_Init();
6 Motors_Reset(); //reset the motors to test them
7 while(Motors_Moving() == true)
8     ; //wait for the motors to finish moving

```

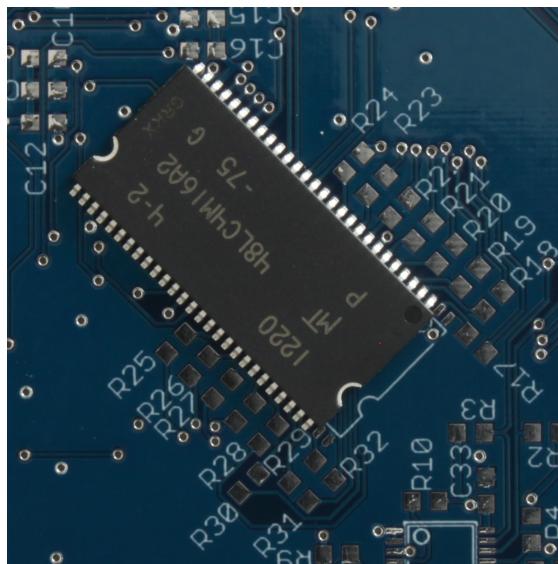


Figure 3.9: SDRAM Chip shown against its footprint.

3.4.4 PCB Faults

During the build and test of the PCB, a number of faults were found. Each is explained and the solution for the problem given.

3.4.4.1 SDRAM Footprint

The SDRAM footprint made was done exactly to the specification of the pad size and locations with no consideration for soldering to. This meant the chip fit exactly on to the footprint. This made soldering difficult as pads had to be preloaded with solder and the device's pins were heated and bound to the solder. The chip does not seat flat against the PCB. It also put the device at risk as more heat had to be used than usually necessary. Figure 3.9 shows the SDRAM chip against the footprint. There is no extra space on the pad to be able to easily solder the device.

To avoid this, existing footprints could be used from other libraries, or double checking the footprints made. The problem meant extra care during soldering had to be taken but has not impeded the operation of the device.

3.4.4.2 SDRAM Chip Select

The code was prototyped on the Atmel UC3C-EK development board prior to the PCB arriving. When the PCB was built, the code did not work, even with the

Chip Select declaration changed. To diagnose this problem, the control lines of the SDRAM were probed with a logic analyser. On the UC3C-EK, the bus was busy with refresh cycles outside of SDRAM access. On the Columbus, no activity was seen.

The reason the correct control wasn't being seen was due to the UC3C device having a dedicated SDRAM controller, attached to only Chip Select line 1. The Columbus was designed to use chip select line 0. Chip select 1 was available on an external pin, and a via through the routing was close to a via connected to the SDRAM chip select line. Therefore, to overcome the problem, a small enameled wire was soldered to join the two vias together. This solved the problem and the correct signals were then seen on the control lines. The patch can be seen in figure 3.11(b).

This fault was caused by not reading the datasheet carefully and ignoring a proven circuit diagram.

3.4.4.3 SDRAM Data Line Resistors

Once the chip select problem was solved, data returned was unreliable. The SDRAM is word (32 bit) addressed, but accessed in 16 bits. This means read cycles are done per word read. Upon investigation of this problem, the 14th, 15th, 30th and 31st (top two bits of each 16 bit access) seemed to read as a 1 the majority of the time. This result wasn't repeatable and sometimes returned correct data. The other bits of the data were always correct. Table 3.4 shows some examples of the problematic data bits. The data written should match the data read back.

Table 3.4: A table showing examples of the incorrect data returned from the SDRAM

Data Written	Data Read
00000000 00000000 00000000 00000000	11000000 00000000 11000000 00000000
00001111 00001111 00001111 00001111	11001111 00001111 11001111 00001111

The problem was traced to resistors **R31** and **R32**. They were soldered on incorrectly so that the two data lines of the SDRAM were connected together and the two AVR GPIO pins were connected together. Data was then read back from, effectively, a high impedance line and therefore varied. Once the resistors were soldered correctly, the issue no longer persisted and the whole SDRAM test passed. By utilising the soldermask more, device orientations could be added to ensure

correct placement. This can be extended to other devices, such as diodes and capacitors, especially in densely populated areas.

3.4.4.4 Camera Interrupt Line

As discussed in section 3.1, the OV7670 needs an interrupt line to synchronise quickly to the start of the frame and is done by using an interrupt line. The UC3C0512C has 9 external interrupt lines. On the PCB, interrupt lines 0 and 1 were used for this control.

Interrupt line 1 was easily configured and worked as expected. However, interrupt 0 did not seem to trigger the interrupt service routine. It was found that interrupt 0 was a “Non Maskable Interrupt” which has specific uses and cannot be used in to trigger a method.

The external interrupt 4 pin was wired to Junction 8 on the PCB. A wire was attached to the camera’s VSYNC line and attached to the relevant pin on the header. The operation was then easily obtained and the VSYNC line triggered correctly.

This issues would have been avoided with more understanding of the device before hand and checking the datasheet.

3.4.4.5 Motor Driver Pinout

An error was made in creating the device for the TB6593FNG Motor Driver in EAGLE. On the device, each motor output has two pins to drive each side of the motor. The pin assignment was mixed up when created and connected the two outputs together. Figure 3.10 shows the track errors on one of the motor drivers.

To solve this, pins 7 and 14 were lifted and removed so that output 1 and output 2 were not connected together. The devices were not damaged in the process of testing this and the motors functioned correctly after this. Double checking the footprints made against the datasheet would have avoided this problem. No impedance to the operation of the drivers has been seen, but the patch may hinder the devices ability to sink current to the motors.

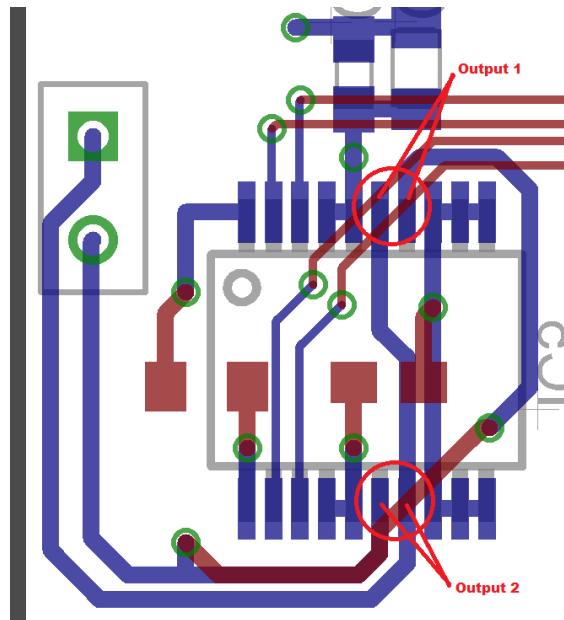


Figure 3.10: Motor Driver error. Outputs incorrectly connected

3.4.5 PCB Conclusions

A number of faults were made in the PCB design. They are:

- SDRAM footprint
- SDRAM chip select line
- SDRAM data line resistors
- Camera interrupt line
- Motor driver pinout

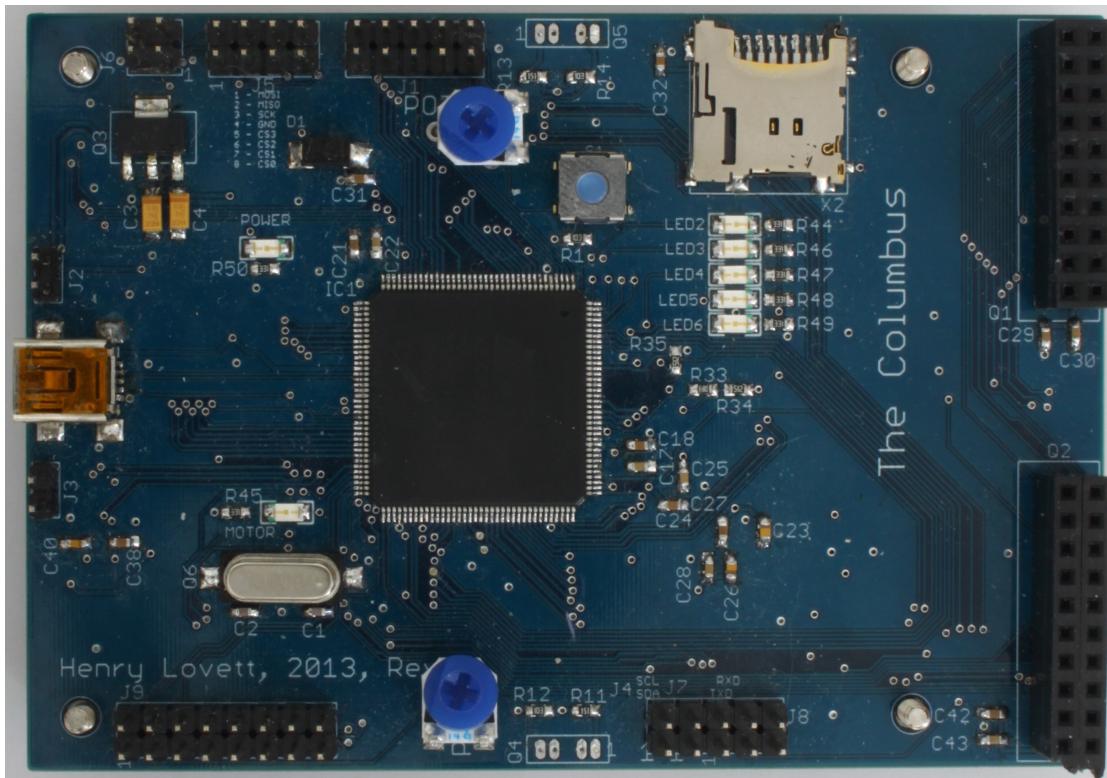
Three of the faults could have been avoided by consulting the datasheet more carefully during the circuit design stage. The footprint error was due to not being experienced in designing footprints and the data line resistors was a mistake due to lack of attention being paid.

For future PCBs, more care will be taken in circuit design, with prototyping of circuits with the hardware that will be used. This will highlight any pin specific operations (e.g. the non maskable interrupt) and reduce debugging post production. The effectiveness of a soldermask is also apparent, so more time spent on utilising this would be helpful during assembly.

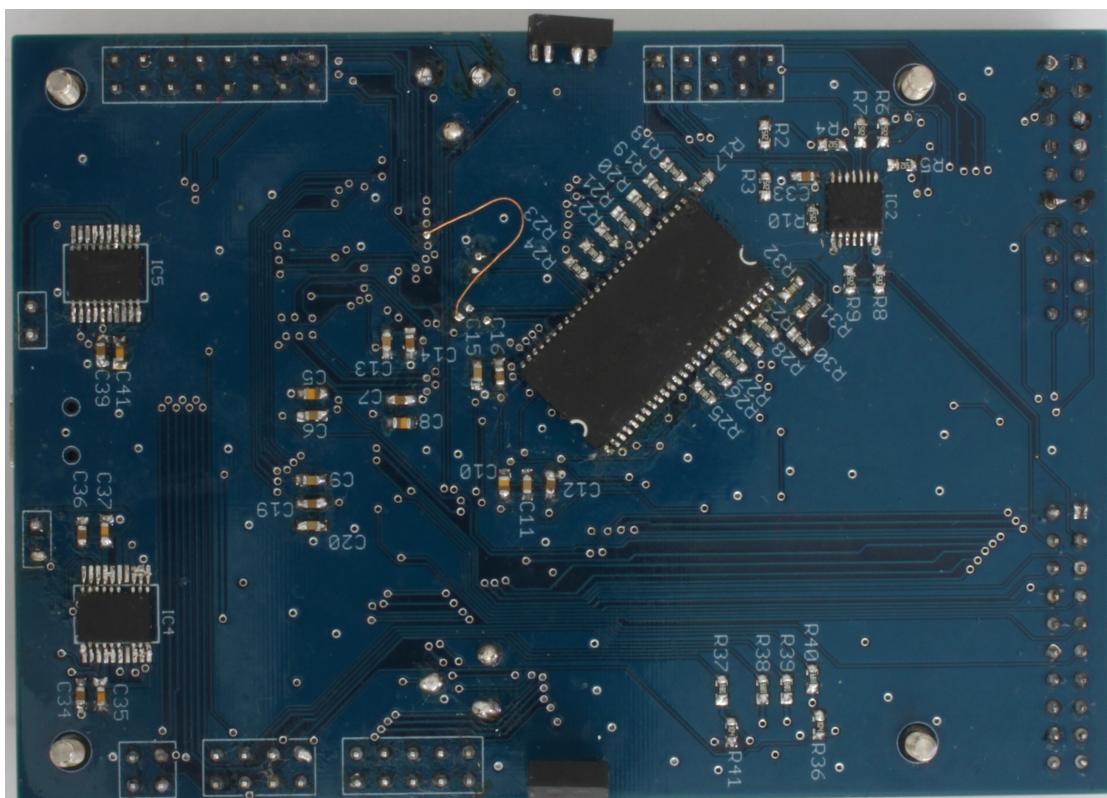
The PCB itself, was a success. It was a complex PCB with many potential things that could have gone wrong. It was the first PCB I had designed and was a four layer board, using some devices that I did not have experience with. All devices are functional (with a few small modifications) on the PCB so firmware development could continue with all hardware able to be used.

3.5 Conclusions

Overall Conclusions of the Hardware design



(a) Top view of built PCB



(b) Bottom view of built PCB with SDRAM chip select patch

Figure 3.11: Pictures of the built PCB.

Chapter 4

Investigation into Vision Algorithms

4.1 Comparison

In computer vision, there are many different ways of comparing two similar images. These include the sum of absolute differences (S.A.D.) ([Hamzah et al. \(2010\)](#)), the sum of squared differences (S.S.D.)([Mrovlje and Vrančić \(2008\)](#)) and normalised cross correlation (N.C.C.)([Zhao et al. \(2006\)](#)). Each of these methods will be explained and tested to compare them. All testing will use images seen in figure [4.1](#). Each test uses the same size window (50×50) to compare the two images.



(a) Left Image

(b) Right Image

Figure 4.1: Stereoscopic Test Images from MATLAB Examples

4.1.1 Sum of Absolute Differences

Given two identically sized two dimensional matrices, A, B , of dimensions I, J , SAD is defined as

$$SAD = \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} |A[i, j] - B[i, j]| \quad (4.1)$$

This method subtracts the observed window from the expected. All differences are then added together. This algorithm is simple and requires a small amount of computation. The algorithm returns values where a small result means the two images are well matched.

4.1.2 Sum of Squared Differences

$$SSD = \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} (A[i, j] - B[i, j])^2 \quad (4.2)$$

This is very similar to S.A.D. but adds more complexity by squaring each difference. This removes the ability of equally different but opposite differences cancelling each other out (grey to white of one pixel will cancel out a white to grey difference in the other with SAD). Again, a low result is a match in this case.

4.1.3 NCC

$$NCC = \frac{1}{n} \sum_{i,j} \frac{(A[i, j] - \bar{A})(B[i, j] - \bar{B})}{\sigma_A \cdot \sigma_B} \quad (4.3)$$

Where n is the number of pixels in A and B ,
 σ is the standard deviation of the image, and
 \bar{A} is the average pixel value.

NCC is very similar to cross correlation, but normalised to reduce the error if one image is brighter than the other. This is common in computer vision ([Tsai and Lin \(2003\)](#)) and cross correlation is often used in digital signal processing, so fast algorithms have been made to calculate this.

Unlike S.S.D. and S.A.D., the normalised cross correlation gives a high value for a match. The downside to this algorithm comes with the complexity of the equation

as it contains division and the a square root of a number in order to calculate the standard deviation. These operations are rarely implemented in hardware and are time consuming to carry out in software. They also require floating point registers and operates slowly on a microcontroller without any.

4.1.4 Comparison

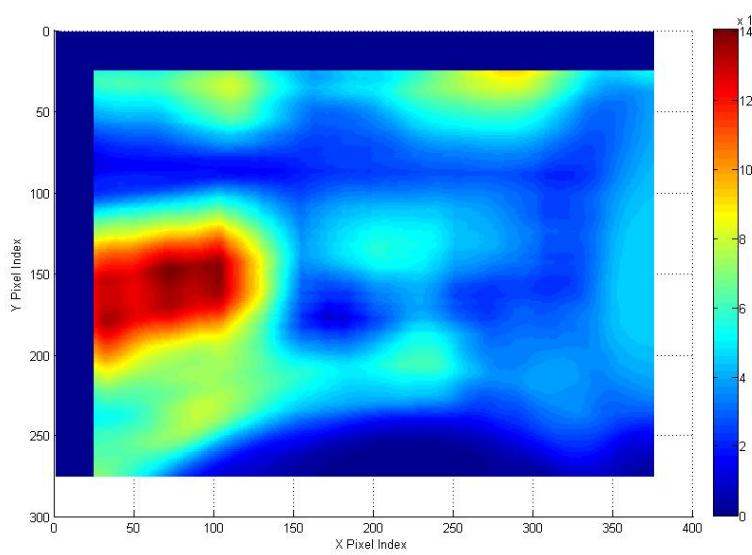
To compare these equations, a 50 by 50 window taken from the right picture was compared with the left image over the entire valid range. The coordinates on the graph give the centre pixel of the calculation.

Each graph shows the correct area being identified as a match, but this also highlights the downfalls of the SAD and SSD. The figures in figure 4.2 are rotated to match the orientation of the images in figure 4.1. Each of the images is tested by attempting to match the desk phone from the right image to the entirety of the left image. The actual match should be around (170, 176). An exact result cannot be estimated as the images are not matched perfectly - there isn't an exact integer of pixel difference between the images. This is the sub pixel problem ([Haller and Nedevschi \(2012\)](#)).

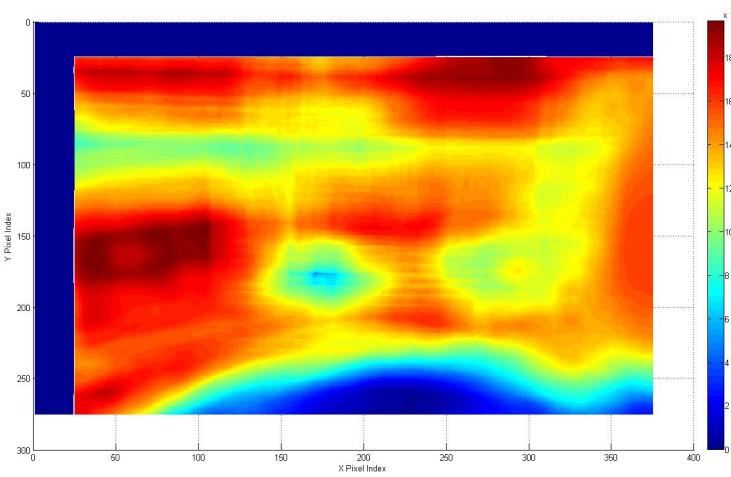
SAD results in figure 4.2(b) show large areas of matching. A minimum occurs around the location expected(170, 175) of a value of 5.66×10^4 . However, along the bottom of the image, where a dark area occurs below the desk in the lower part of figures 4.1, the SAD algorithm detects a greater comparison, with the lowest value in this area being 3370 at (227, 275). This creates a false detection here.

SSD shows matches in the same two areas: where a match should occur and the dark area beneath the desk. The minimum value where the match should occur is 4.355×10^5 at location (170, 176). However, there is a large match correlation between the dark area under the desk where the actual lowest value of 2.768×10^4 occurs at (225, 274). This, again, is a false match and is a downfall of this algorithm.

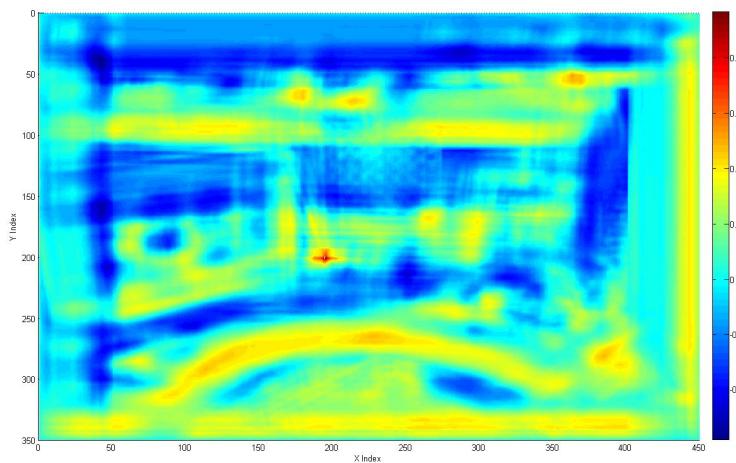
The NCC results are visible in figure 4.2(c). A match can be seen at coordinate (195, 201) with a peak value of 0.9654. The coordinate is different to the previous results because the cross correlation works over the boundary of the image creating more results. The dimensions of the image are 300×400 , but the NCC returns an data set of dimensions 350×450 when using a window size of 50×50 . To get the actual match, half of the box size must be subtracted from the returned



(a) S.A.D. Results (Low match)



(b) S.S.D. Results (Low match)



(c) N.C.C. Results (High match)

Figure 4.2: Result Graphs of Comparison Algorithms

coordinate. This means the match occurs at (170, 176). With this algorithm, there is no area of the image which is close to a false detection.

4.1.5 Conclusion

It can be seen that there is a direct correlation between the complexity of the matching algorithm to the reliability of the match returned. In brightly lit, colourful environments absent of dark colours, SAD and SSD should provide a reliable result, but this cannot be guaranteed to always be the case. Therefore further development of the matching algorithm will start with using the normalised cross correlation. A comprise between complexity and reliability needs to be reached, where reliability is the more desirable of the two. Cross correlation is also a large area of research, so optimised algorithms do exist.

4.2 Range Finding

4.2.1 Derivations

By using two images separated by a horizontal distance, B , the range of an object can be found given some characteristics of the camera. The following are derivations of the equations used to calculate distance.

The problem is broken down into three parts:

1. Object is between the cameras (Figure 4.3)
2. Object is in left or right hand sides of both images (Figure 4.4)
3. Object is directly in front of a camera (Figure 4.5)

4.2.1.1 Object is between the Cameras

Derivation from [Mrovlje and Vrančić \(2008\)](#).

$$B = B_1 + B_2 = D \tan(\varphi_1) + D \tan(\varphi_2) \quad (4.4)$$

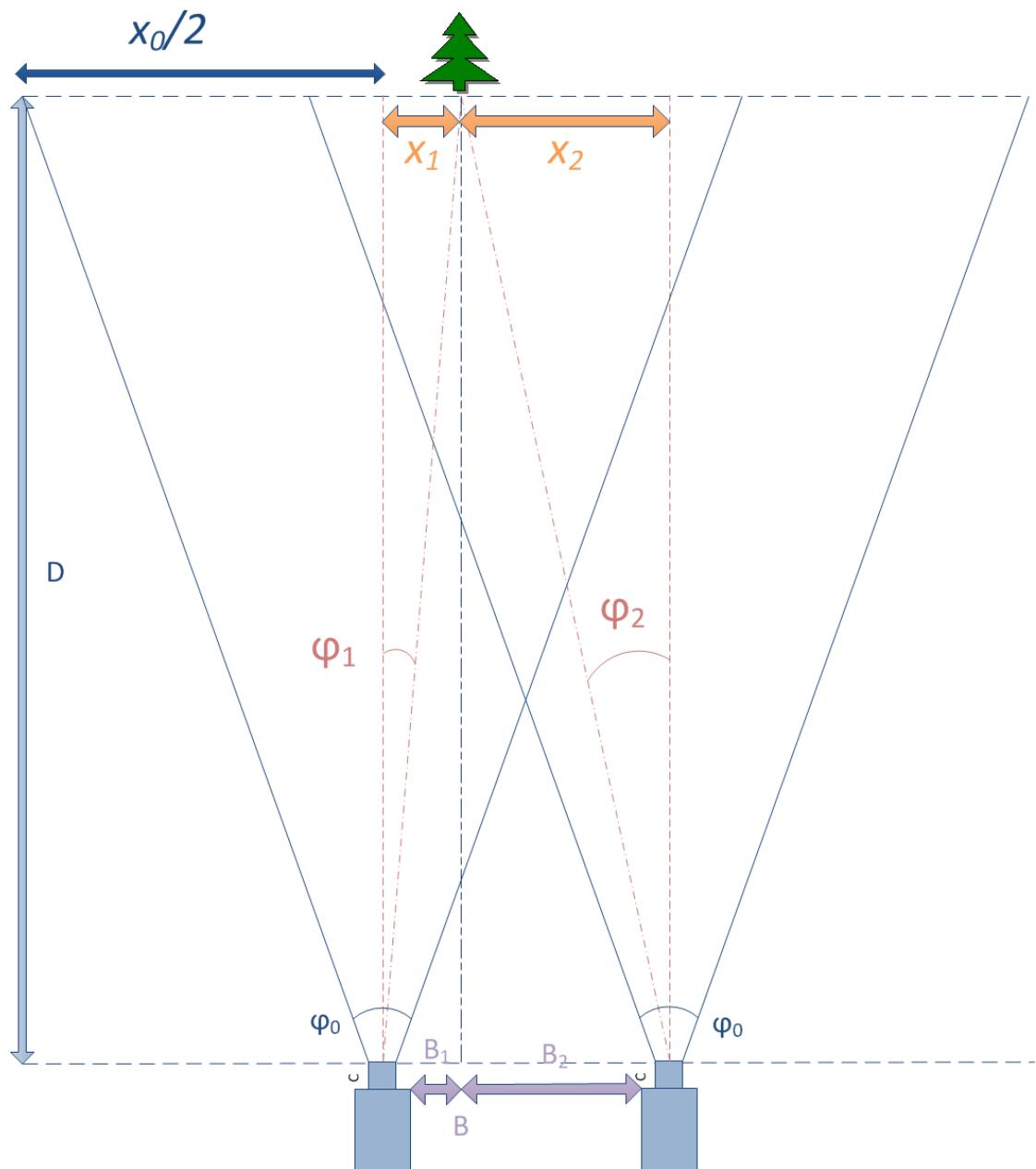


Figure 4.3: Problem 1 - Object is between the Cameras

$$D = \frac{B}{\tan(\varphi_1) + \tan(\varphi_2)} \quad (4.5)$$

$$D \tan\left(\frac{\varphi_0}{2}\right) = \frac{x_0}{2} \quad (4.6)$$

$$D \tan(\varphi_1) = x_1 \quad (4.7)$$

Dividing (4.7) by (4.6)

$$\frac{\tan(\varphi_1)}{\tan(\frac{\varphi_0}{2})} = \frac{2x_1}{x_0} \quad (4.8)$$

$$\tan(\varphi_1) = \frac{2x_1 \tan(\frac{\varphi_0}{2})}{x_0} \quad (4.9)$$

This can also be shown for the right camera:

$$\tan(\varphi_2) = \frac{-2x_2 \tan(\frac{\varphi_0}{2})}{x_0} \quad (4.10)$$

Substitution equations (4.9) and (4.10) into (4.5) gives

$$D = \frac{Bx_0}{2 \tan(\frac{\varphi_0}{2})(x_1 - x_2)} \quad (4.11)$$

4.2.1.2 Object is to the same side in each camera

Derivation is based on the derivation from Tjandranegara (2005). Using figure 4.4:

$$D \cdot \tan(\varphi_1) = x_1 \quad (4.12)$$

$$D \cdot \tan\left(\frac{\varphi_0}{2}\right) = \frac{x_0}{2} \quad (4.13)$$

$$\frac{\tan(\varphi_1)}{\tan(\frac{\varphi_0}{2})} = \frac{2x_1}{x_0} \quad (4.14)$$

$$\varphi_1 = \arctan\left(\frac{2x_1}{x_0} \tan\left(\frac{\varphi_0}{2}\right)\right) \quad (4.15)$$

and similarly

$$\varphi_2 = \arctan\left(\frac{2x_2}{x_0} \tan\left(\frac{\varphi_0}{2}\right)\right) \quad (4.16)$$

$$\theta = \varphi_2 - \varphi_1 \quad (4.17)$$

Using the sine equality rule:

$$\frac{R}{\sin(\frac{\pi}{2} - \varphi_2)} = \frac{B}{\sin(\theta)} \quad (4.18)$$

$$R = B \cdot \frac{\sin(\frac{\pi}{2} - \varphi_2)}{\sin(\theta)} = B \frac{\cos(\varphi_2)}{\sin(\theta)} \quad (4.19)$$

$$D = \cos(\varphi_1) \cdot R \quad (4.20)$$

Substituting (4.17) into (4.19), and then into (4.20):

$$D = B \cdot \frac{\cos(\varphi_2) \cdot \cos(\varphi_1)}{\sin(\varphi_2 - \varphi_1)} \quad (4.21)$$

Where φ_1 is defined in equation (4.15) and φ_2 is defined in equation (4.16).

4.2.1.3 Object is in front of a camera

The distance, D , in this problem is given by:

$$D = B \tan\left(\frac{\pi}{2} - \varphi_2\right) \quad (4.22)$$

Where φ_2 can be found from equation 4.16.

4.2.1.4 Summary

There are three situations that can occur. These are listed below with their equations.

Object is between the two cameras:

$$D = \frac{Bx_0}{2 \tan(\frac{\varphi_0}{2})(x_1 - x_2)} \quad (4.23)$$

Object is to the same side in both images:

$$D = B \cdot \frac{\cos(\varphi_2) \cdot \cos(\varphi_1)}{\sin(\varphi_2 - \varphi_1)} \quad (4.24)$$

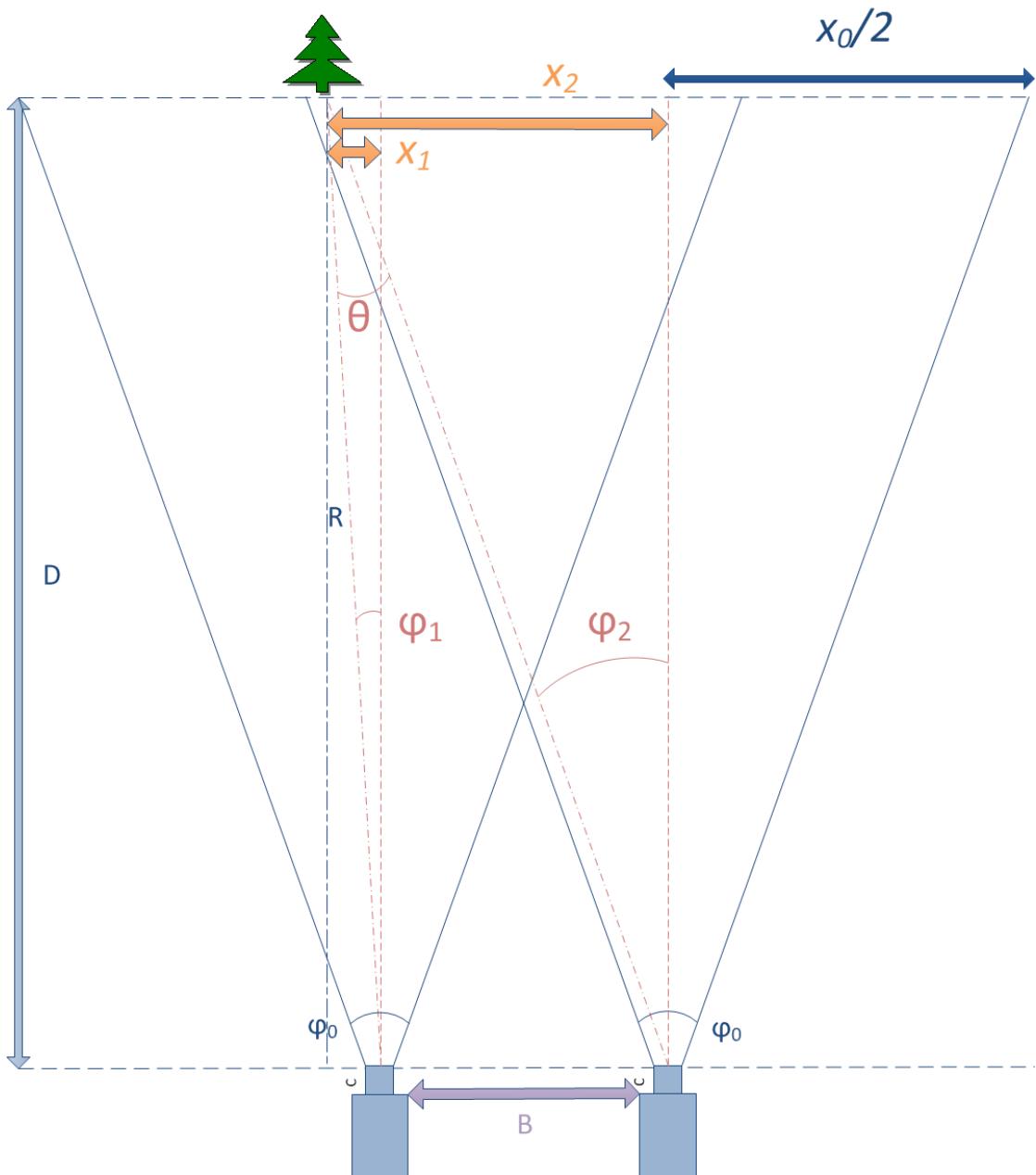


Figure 4.4: Problem 2 - Object is to the same side in both cameras

Object is directly in front of a camera:

$$D = B \tan \left(\frac{\pi}{2} - \varphi_2 \right) \quad (4.25)$$

Where φ_1 is defined in equation (4.15) and φ_2 is defined in equation (4.16).

When the images have been matched, these equations can be used to calculate the range to an object.

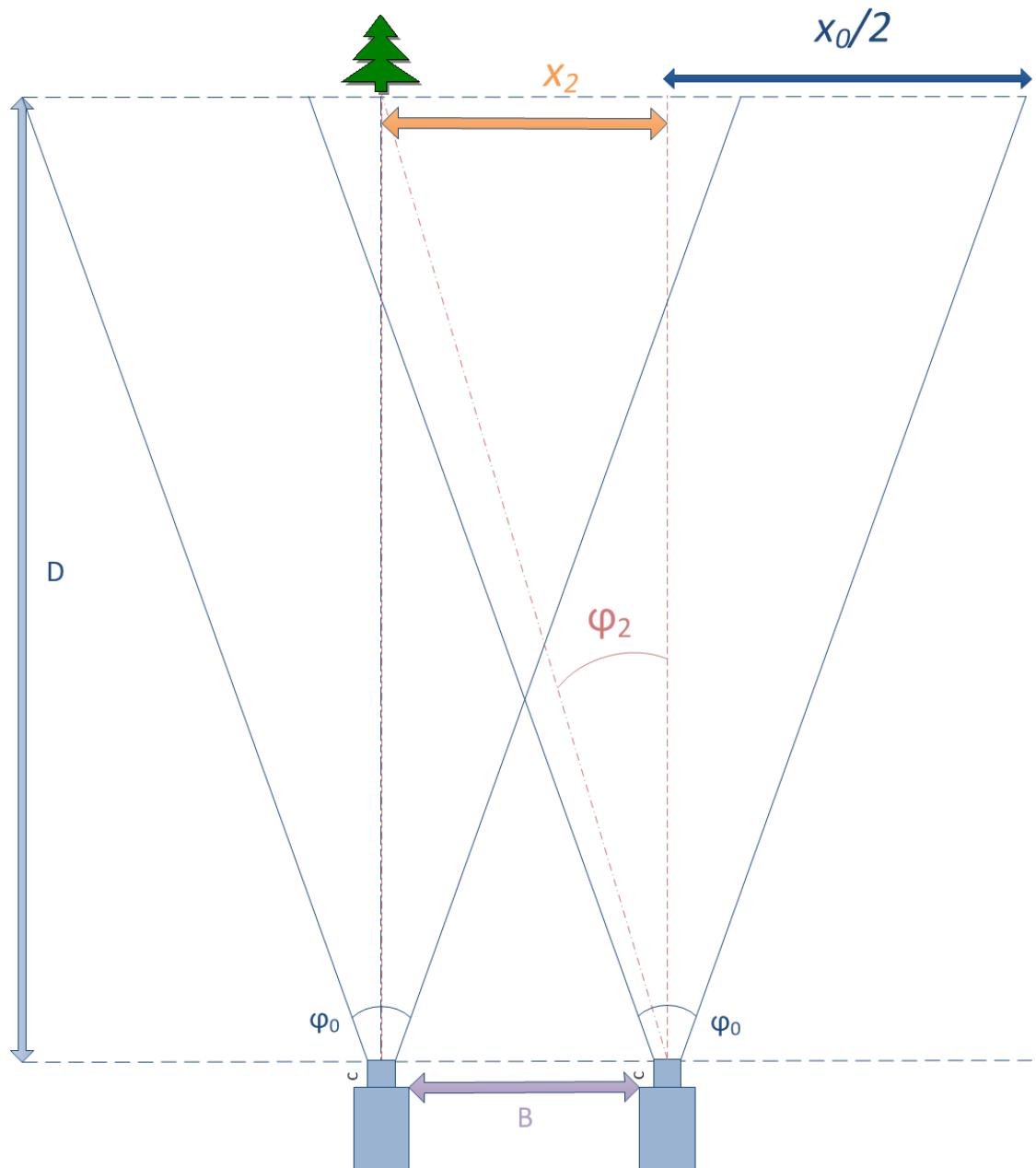


Figure 4.5: Problem 3 - Object is directly in front of a camera

4.3 Fourier Transform

4.3.1 Background Research and Uses

4.3.2 Fast Fourier Transform

4.3.3 Implementing the FFT in C

4.3.4 Testing of the FFT on AVR

4.4 Low Level Vision Algorithms

4.4.1 Noise Reduction

4.4.2 Edge Detection

Chapter 5

Results

5.1 Results

A full test of the system I have got

Summary of good and bad WITH EVIDENCE

Chapter 6

Conclusions and Further Work

What I have accomplished

What could be changed to make it better

Suggestions for further work

Appendix A

Gantt Chart



Figure A.1: Gantt Chart of how time will be spent in the areas of the project

Appendix B

Circuit Diagrams

B.1 OV7670 Breakout Board Schematic

B.2 Il Matto and Dual Camera Schematic

B.3 The Columbus Circuit Diagram

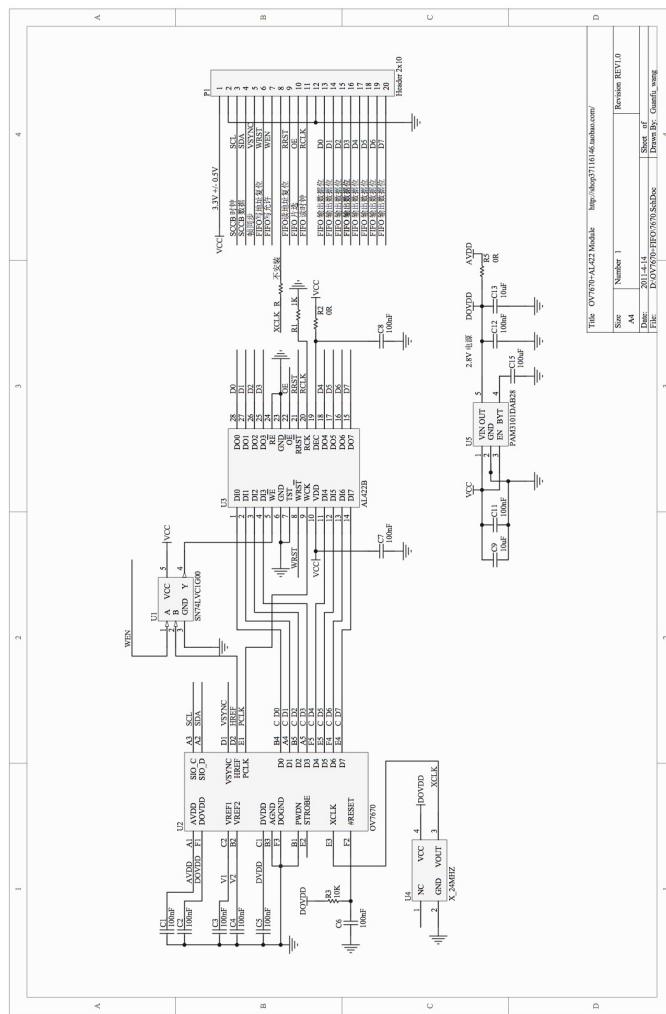


Figure B.1: The circuit diagram for the OV7670 breakout board

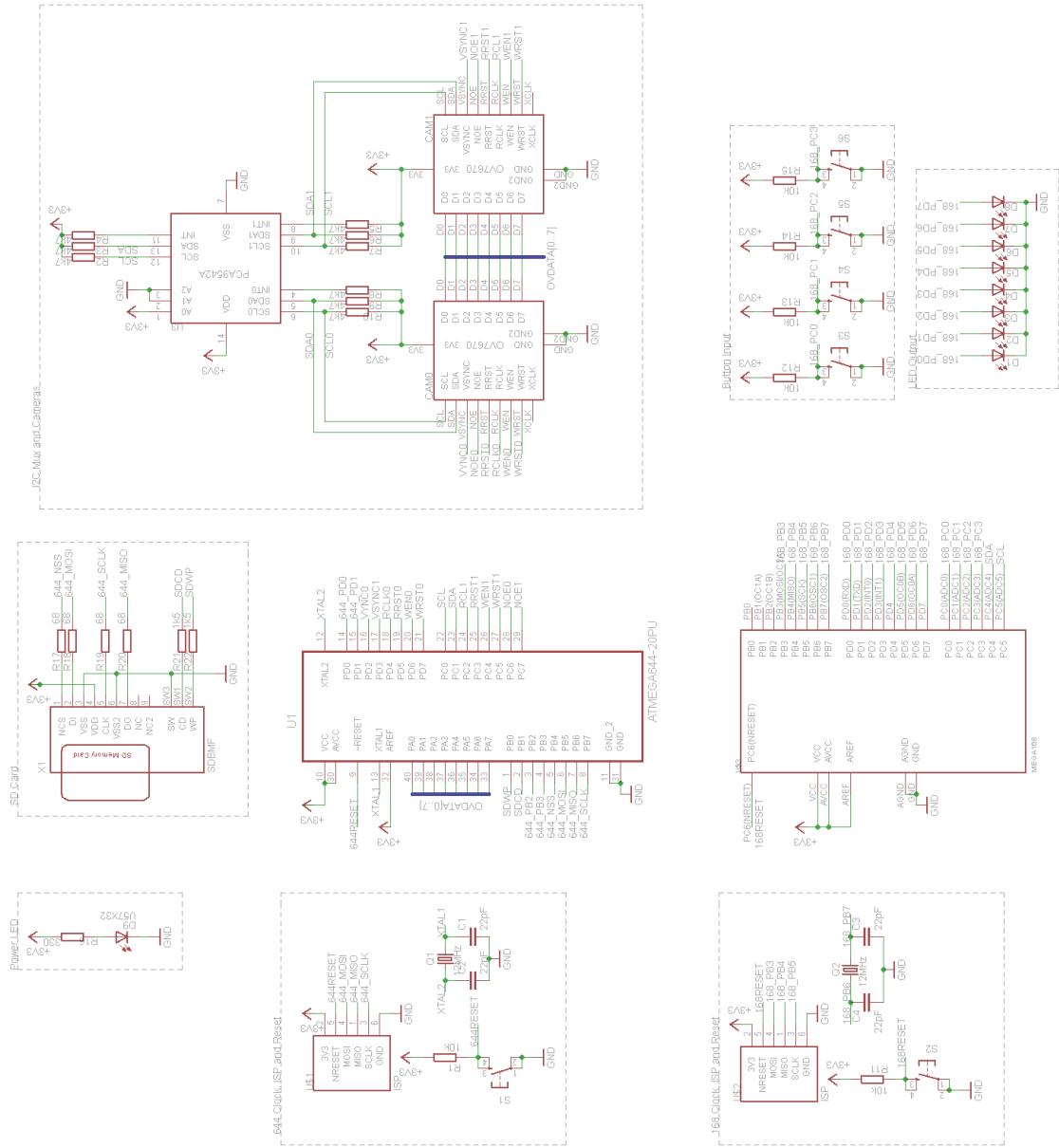


Figure B.2: The circuit diagram for Dual Cameras using the Il Matto Board

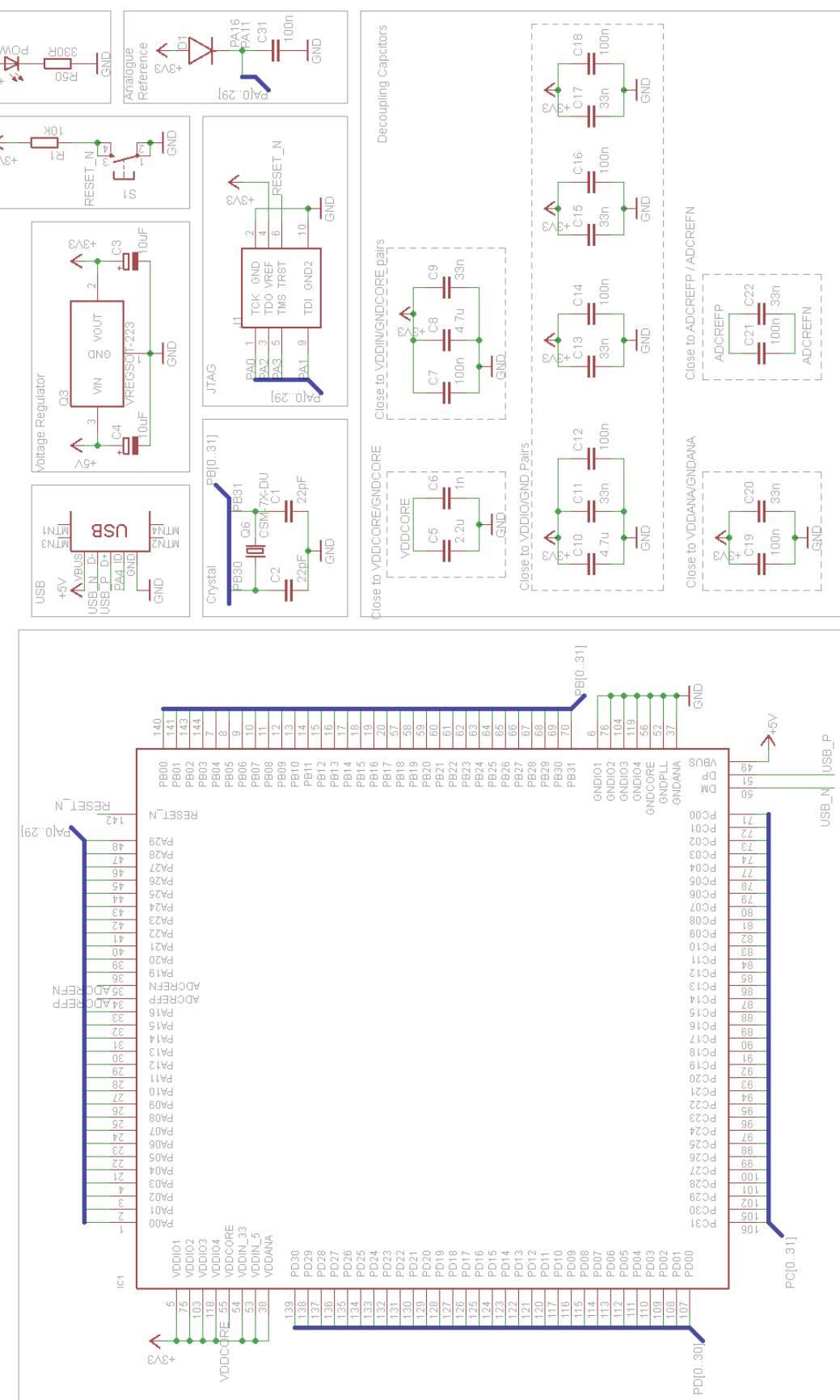


Figure B.3: The Columbus Circuit Diagram Page 1

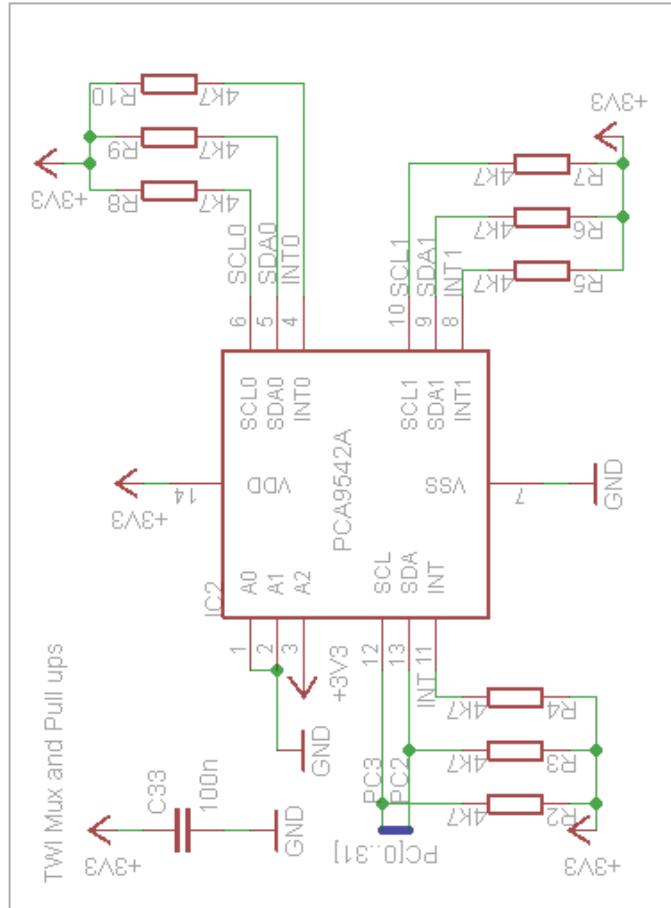
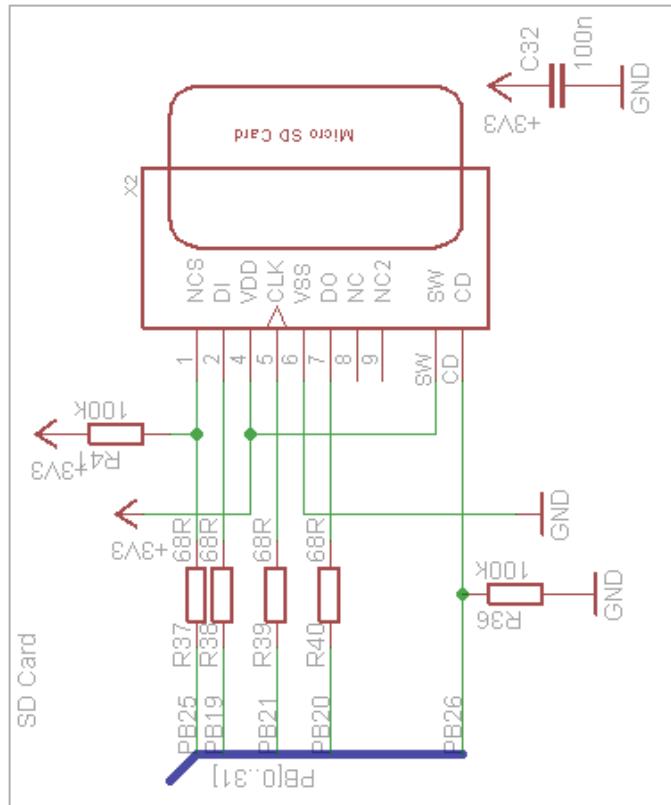


Figure B.4: The Columbus Circuit Diagram Page 2

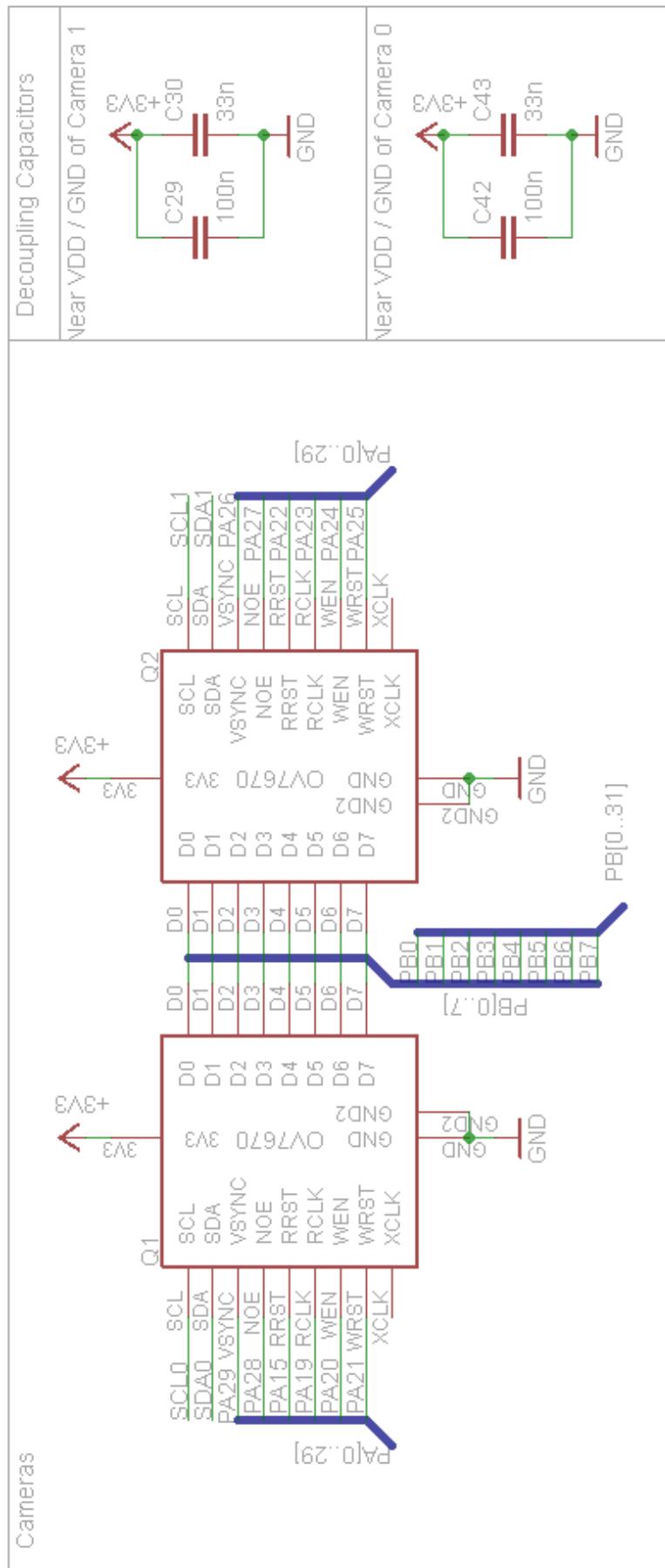


Figure B.5: The Columbus Circuit Diagram Page 3

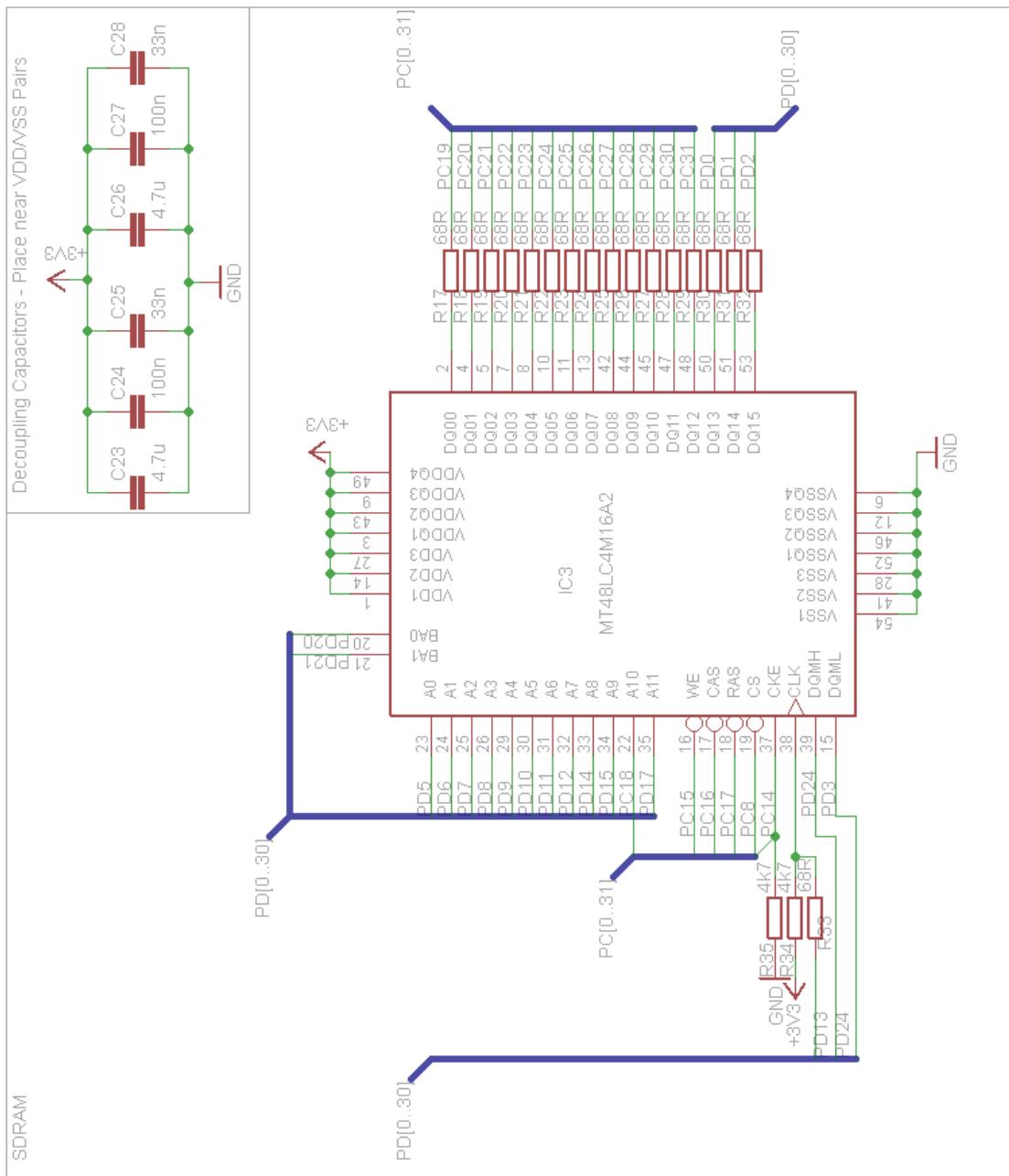


Figure B.6: The Columbus Circuit Diagram Page 4

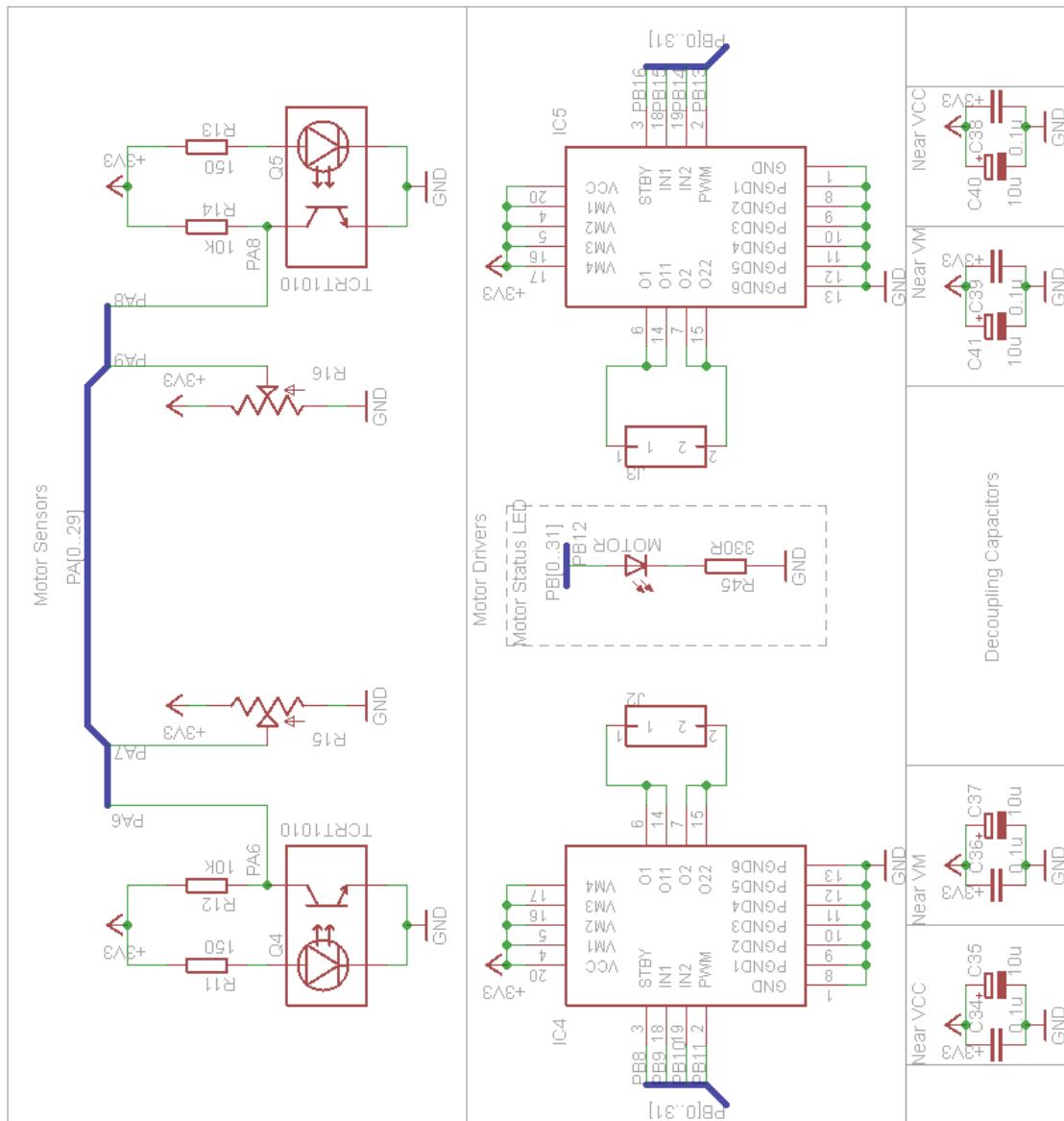


Figure B.7: The Columbus Circuit Diagram Page 5

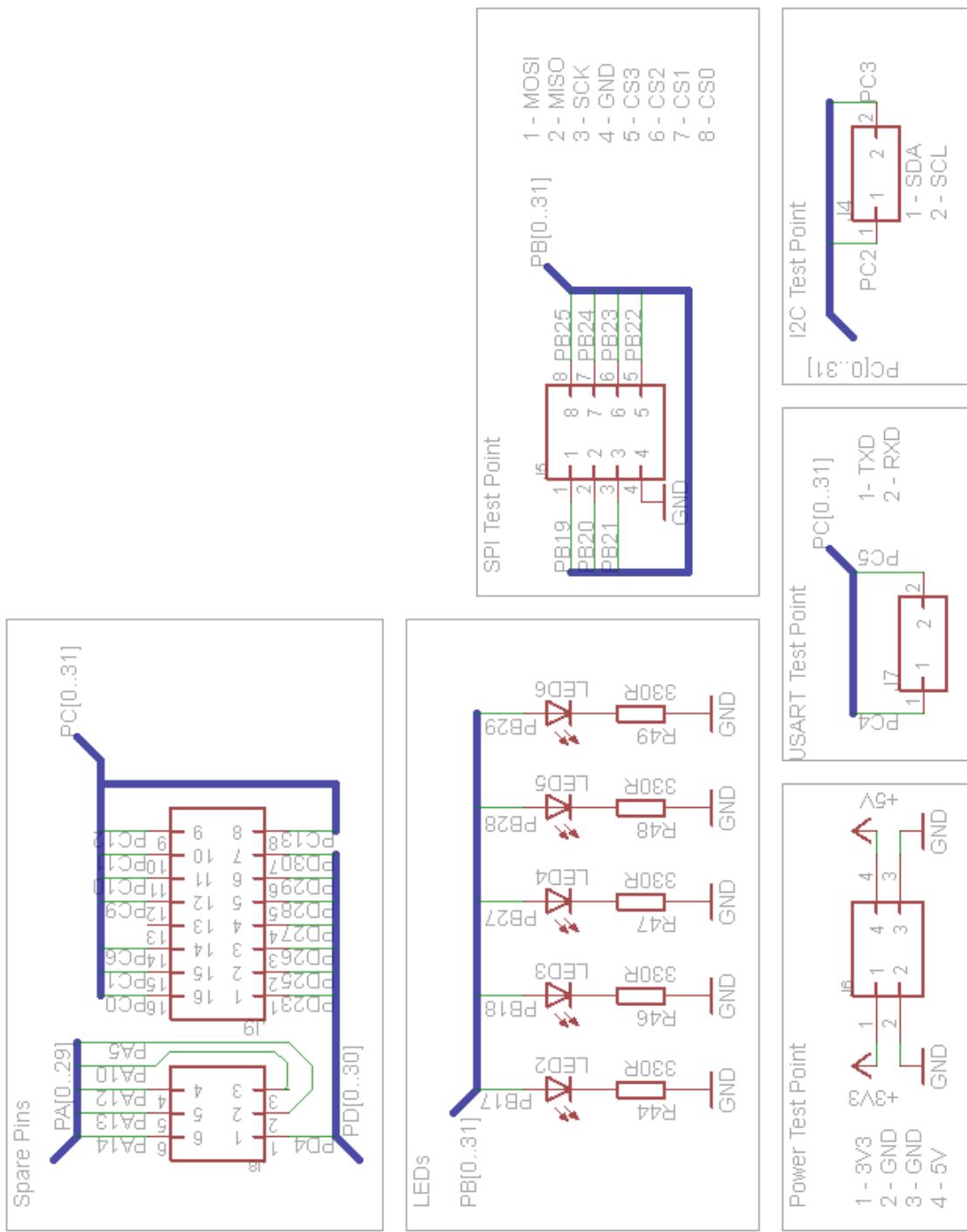


Figure B.8: The Columbus Circuit Diagram Page 6

Appendix C

Bitmap File Format

C.1 Bitmap File Format

Table C.1: Format of a Bitmap file with values used, to write an image from the camera to an SD Card

Section	Field	Description	Size (Bytes)	Value (hex)
Bitmap Header	Signature	Declares the file is a Bitmap Image	2	424D
	File Size	Size of the whole file including headers	4	36580200 (153654) ¹
	Reserved		4	00000000
	Offset to Pixel Array	The address of the start of the pixel data from the beginning of the file	4	36000000
DIB (Device Independent Bitmap) Header	Size	Size of the DIB Header (dictates the version)	4	7C000000
	Width	Width of the image (320 pixels)	4	40010000

Continued on next page

¹This is different to the 225kB stated in Table 3.1 due to omitting many optional fields

Table C.1 – continued from previous page

Section	Field	Description	Size (Bytes)	Value (hex)
	Height	Height of the image (240 pixels)	4	F0000000
	Planes	Number of colour planes	2	0100
	Bit Count	Number of bits per pixel	2	1000
	Compression	Compression Being Used, RGB Bit Fields	4	03 00 00 00
	Image Size	Size of the image	4	00 86 25 00
	X Resolution	Horizontal resolution in pixels per metre	4	13 0B 00 00
	Y Resolution	Vertical resolution in pixels per metre	4	13 0B 00 00
	Colours in Table	Number of colours in the colour table (not used)	4	00 00 00 00
	Important Colours	Number of Important Colours (0 means all colours are important)	4	00 00 00 00
	Red Mask	Bit mask of Red field	4	00 F8 00 00
	Green Mask	Bit mask of Green field	4	E0 07 00 00
	Blue Mask	Bit mask of Blue field	4	1F 00 00 00
	Alpha Mask	Bit mask of Alpha field	4	00 00 00 00
	Colour Space Type	Colour Space of the DIB	4	01 00 00 00
	Colour Space Endpoints	Sets endpoints for colours within the bitmap (not used)	36	Whole Field = 0
	Gamma Red	Gamma Value of Red Field (not used)	4	00 00 00 00

Continued on next page

Table C.1 – continued from previous page

Section	Field	Description	Size (Bytes)	Value (hex)
	Gamma Green	Gamma Value of Green Field (not used)	4	00 00 00 00
	Gamma Blue	Gamma Value of Blue Field (not used)	4	00 00 00 00
	Intent	Enum dictating the intent of the image (Picture)	4	03 00 00 00
	ICC Profile Data	Offset from the file start to the ICC Colour Profile (Not Used)	4	00 00 00 00
	ICC Profile Size	Size of the ICC Colour Profile (not used)	4	00 00 00 00
	Reserved		4	00 00 00 00
Image Data Format	Each field contains all the pixel data	Padding is used to make the table width a multiple of 4 (Not always needed)		
Pix[0, h-1]	Pix[1, h-1]	...	Pix[w-1, h-1]	Padding
:	:	:	:	:
Pix[0, 1]	Pix[1, 1]	...	Pix[w-1, 1]	Padding
Pix[0, 0]	Pix[1, 0]	...	Pix[w-1, 0]	Padding

Appendix D

Source Code

D.1 C Code for AVR

D.1.1 The Columbus Source Code

D.1.1.1 main.c

..../Code/The_Columbus/ColumbusTest/src/main.c


```
70     }
71     usart_putchar(DBG_USART, 6);
72 }

74 #define COMMAND_BUFFER_SIZE    32
75 int main (void)
76 {
77     Image_t image;
78     unsigned long i, j, tmp = 0;
79     char *Ptr;
80 // volatile unsigned long *sdram = SDRAM;
81     char CommandBuffer[COMMAND_BUFFER_SIZE];
82     int *Working_Buffer = NULL;
83     int SizeOfWorking_Buffer = 0;
84     A_ALIGNED dsp16_complex_t *ComplexBuffer;
85     int SizeOfComplex_Buffer = 0;
86     Columbus_Status.SD_Card = &SD_Status;
87     Columbus_Status.Cameras = &OV7670_Status;
88     Columbus_Status.I2CMux = &PCA9542A;
89     Columbus_Status.SD_Card = &SD_Status;
90     Columbus_Status.Motors = &Motor_Control;
91     board_init();
92     print_dbg("\n\r");
93     print_dbg(THE);
94     print_dbg(COLUMBUS);
95     print_dbg(ASCII_SHIP);
96     System_Test();

99 //print_dbg("\n\rResetting Motors.");

101 //  Motors_Reset();
102 //  while(Motors_Moving() == true)
103 //      ;
104 if(Columbus_Status.Status != STATUS_OK)
105 {
106     while(1)
107     {
108         LED2_SET;
109         LED3_SET;
110         LED4_SET;
111         LED5_SET;
112         LED6_SET;
113         delay_ms(500);
114         LED2_CLR;
115         LED3_CLR;
116         LED4_CLR;
117         LED5_CLR;
118         LED6_CLR;
119         delay_ms(500);
120     }//inifinte loop
121 }

124 print_dbg("\n\rColumbus Ready!");
125 // Insert application code here, after the board has been initialized.
126 while(1)
127 {
```

```
128     print_dbg(PROMPT);
129     Get_Line(CommandBuffer);
130     Ptr = CommandBuffer;
131     switch(*Ptr++)
132     {
133         case '?':
134             print_dbg(HELP);
135             break;
136
137         case '1'://1d FFT (w/ memallocs)
138             print_dbg("\r1D FFT; ");
139             FFT1D(Working_Buffer);
140             break;
141         case '2':
142             print_dbg("\r2D FFT; ");
143             FFT2Dabs(Working_Buffer);
144             break;
145         case '3':
146             print_dbg("\rComplex FFT2D; ");
147             SizeOfComplex_Buffer = FFT_SIZE * FFT_SIZE;
148             ComplexBuffer = mspace_malloc(sdram_msp, SizeOfComplex_Buffer *
149                                         sizeof(ComplexBuffer));
150             FFT2DCOMPLEX(Working_Buffer, ComplexBuffer, SizeOfWorking_Buffer);
151             break;
152
153         case 'B':
154             print_dbg("\rReading Bitmap; ");
155             ReadBitmap("Image_R_0.bmp", &image);
156             print_dbg("\n\rBitmap Data Returned:\n\rImage Height = ");
157             print_dbg_ulong(image.Height);
158             print_dbg("\n\rImage Width = ");
159             print_dbg_ulong(image.Width);
160             break;
161
162         case 'c':
163             print_dbg("\rConverting Working Buffer to Fixed Point");
164             for(i = 0; i < SizeOfWorking_Buffer ; i++)
165             {
166                 Working_Buffer[i] = DSP16_Q (Working_Buffer[i]);
167             }
168             break;
169
170         case 'C':
171             print_dbg("\rConverting Working Buffer back from Fixed Point");
172             j = DSP16_Q(1);
173             for(i = 0; i < SizeOfWorking_Buffer ; i++)
174             {
175                 Working_Buffer[i] = Working_Buffer[i] / j;
176             }
177             break;
178
179         case 'D':
180             print_dbg("\rFreeing Working Buffer");
181             mspace_free(sdram_msp, Working_Buffer);
182             break;
183
184         case 'i':
185             print_dbg("\rImage info:");
```

```
185     print_dbg("\n\rImage Pointer = ");
186     print_dbg_ulong(image.ImagePtr);
187     print_dbg("\n\rImage Height = ");
188     print_dbg_ulong(image.Height);
189     print_dbg("\n\rImage Width = ");
190     print_dbg_ulong(image.Width);
191     break;
192
193 case 'I':
194     print_dbg("\rInverse Fourier Transform");
195     IFFT2D(ComplexBuffer);
196     break;
197
198 case 'k':
199     print_dbg("\rComplex Buffer:\n\r[");
200     for (i = 0; i < SizeOfComplex_Buffer; i++)
201     {
202         print_dbg_ulong(ComplexBuffer[i].real);
203         print_dbg(" + j");
204         print_dbg_ulong(ComplexBuffer[i].imag);
205         print_dbg(", ");
206     }
207     print_dbg("]\n\r");
208     break;
209 case 'M': //Motor Related
210     while(*Ptr == ' ')
211         Ptr++; //Find next non - space char
212
213     switch(*(Ptr++))
214     {
215         case 'q': // Reset Motors
216             print_dbg("\rResetting Motors");
217             Motors_Reset();
218             break;
219
220         case 'F': //Move Forward
221             while(*Ptr == ' ')
222                 Ptr++; //Find next non - space char
223             i = atoi(Ptr);
224             Motors_Move(i);
225             break;
226         case 'T':
227             while(*Ptr == ' ')
228                 Ptr++; //Find next non - space char
229             i = atoi(Ptr);
230             Motors_Rotate(i);
231             break;
232         case 'l':
233             Motor_Stop(MOTOR_L);
234             break;
235         case 'L':
236             Columbus_Status.Motors->Left_Count = INTERRUPTS_PER_REVOLUTION +
237             1;
238             Columbus_Status.Motors->Left_State = FORWARD;
239             Motor_Start(MOTOR_L);
240             Motors_Execute();
241             break;
242         case 'r':
```

```

242     Motor_Stop(MOTOR_R);
243     break;
244 case 'R':
245     Columbus_Status.Motors->Right_Count = INTERRUPTS_PER_REVOLUTION +
246     1;
247     Columbus_Status.Motors->Right_State = FORWARD;
248     Motor_Start(MOTOR_R);
249     Motors_Execute();
250     break;
251 default:
252     print_dbg("\rCommand Not Recognised");
253     break;
254 }

255 break;

256 case 'p':
257     print_dbg("\rPreparing Image");
258     PrepareImage(&image);
259     print_dbg("\rImage Prepared!");
260     break;

261 case 'P'://take a photo
262     FIFO_Reset(CAMERA_LEFT | CAMERA_RIGHT);
263     print_dbg("\rTaking Photos");
264     if(TakePhoto(CAMERA_LEFT | CAMERA_RIGHT) == CAMERAS_BUSY){
265         print_dbg("Cameras Busy");
266         break;
267     }
268     while(Photos_Ready() == false)
269     ;
270
271     if(Store_Both_Images() == true)
272         print_dbg("\n\rImages Stored Successfully!");
273     break;

274 case 'r':
275     if (Working_Buffer == 0)
276     {
277         print_dbg("\rWorking Buffer Not Initialised");
278         break;
279     }
280     print_dbg("\rWorking Buffer:\n\r[");
281     for(i = 0; i < SizeOfWorking_Buffer; i++)
282     {
283         print_dbg_ulong(Working_Buffer[i]);
284         print_dbg(", ");
285     }
286     print_dbg("\b\b]\n\r");
287     break;
288 case 'R':
289     Working_Buffer = mspace_malloc(sdrum_msp, FFT_SIZE);
290     SizeOfWorking_Buffer = FFT_SIZE;
291     print_dbg("\rReading in signal.bin");
292     ReadSignal(Working_Buffer);
293     break;

294 case 's':// save the working buffer

```

```
299     print_dbg("\rSaving Working Buffer");
300     SaveBuff(Working_Buffer, SizeOfWorking_Buffer);
301     break;
302
303     case 'S':
304         print_dbg("\rSaving Bitmap");
305         SaveBitmap(image.ImagePtr, image.Width, image.Height, "ResavedImage.
306 bmp");
307         print_dbg("\rSaved Bitmap!");
308         break;
309
310     case 'T':
311         print_dbg("\rReading in 2D Signal");
312         Working_Buffer = mspace_malloc(sdram_msp, FFT_SIZE * FFT_SIZE);
313         SizeOfWorking_Buffer = FFT_SIZE * FFT_SIZE;
314         Read2DSignal(Working_Buffer);
315         break;
316     case 'v':
317         print_dbg("\rColumbus Status:");
318         print_dbg("\n\rSD Card:\n\rStatus: ");
319         print_dbg_ulong(Columbus_Status.SD_Card->Status);
320         print_dbg("\n\rMemory Size : ");
321         print_dbg_ulong(Columbus_Status.SD_Card->Memory_size);
322         print_dbg("\n\rMotors:");
323         print_dbg("\n\rLeft State : ");
324         print_dbg_ulong(Columbus_Status.Motors->Left_State);
325         print_dbg("\n\rLeft Count : ");
326         print_dbg_ulong(Columbus_Status.Motors->Left_Count);
327         print_dbg("\n\rRight State : ");
328         print_dbg_ulong(Columbus_Status.Motors->Right_State);
329         print_dbg("\n\rRight Count : ");
330         print_dbg_ulong(Columbus_Status.Motors->Right_Count);
331         print_dbg("\n\rCameras:");
332         print_dbg("\n\rStatus : ");
333         print_dbg_ulong(Columbus_Status.Cameras->Status);
334         print_dbg("\n\rVSYNC0 State : ");
335         print_dbg_ulong(Columbus_Status.Cameras->VSYNC0_State);
336         print_dbg("\n\rVSYNC1 State : ");
337         print_dbg_ulong(Columbus_Status.Cameras->VSYNC1_State);
338         print_dbg("\n\rI2C Mux:");
339         print_dbg("\n\rStatus : ");
340         print_dbg_ulong(Columbus_Status.I2CMux->Status);
341         print_dbg("\n\rChannel Selected : ");
342         print_dbg_ulong(Columbus_Status.I2CMux->ChannelSelected);
343         break;
344
345 //     case 'o'://testing storing a complex
346 //         print_dbg("\rFreeing Complex Buffer");
347 //         mspace_free(sdram_msp, ComplexBuffer);
348 //         print_dbg("\n\rAssigning Space to the Complex Buffer");
349 //         SizeOfComplex_Buffer = 10;
350 //         ComplexBuffer = mspace_malloc(sdram_msp, 10*sizeof(ComplexBuffer));
351 //         if(ComplexBuffer == NULL)
352 //         {
353 //             print_dbg("\n\rAssign Failed");
354 //             break;
355 //         }
356 //         for(i = 0; i < SizeOfComplex_Buffer; i++)
```

```

356 //      {
357 //          ComplexBuffer[i].imag = i;
358 //          ComplexBuffer[i].real = i;
359 //      }
360 //      for(i = 0; i < SizeOfComplex_Buffer; i++)
361 //      {
362 //          print_dbg("\n\r");
363 //          print_dbg_ulong(ComplexBuffer[i].real);
364 //          print_dbg(" + j");
365 //          print_dbg_ulong(ComplexBuffer[i].imag);
366 //      }
367 //      print_dbg("\n\rFreeing Complex Buffer");
368 //      mspace_free(sdram_msp, ComplexBuffer);
369 //      SizeOfComplex_Buffer = 0;
370 //      break;

372     default:
373         print_dbg("\rCommand Not Recognised;");
374         break;
375     }
376 }
377 }
```

D.1.1.2 Bitmap.c

..../Code/The_Columbus/ColumbusTest/src/CustomDevices/Bitmap.c

```

1  /*
2   * Bitmap.c
3   *
4   * Created: 16/02/2013 23:14:34
5   * Author: hslovett
6   */
7 #include "CustomDevices/CustomDevices.h"

9 const uint8_t DIBHead[DIBHEADERSIZE] = { 0x7C, 0x00, 0x00, 0x00, //Number of
   bytes
10           0x40, 0x01, 0x00, 0x00, //Width - 320
11           0xF0, 0x00, 0x00, 0x00, //Height - 240
12           0x01, 0x00, //Planes
13           0x10, 0x00, //Bits per Pixel
14           0x03, 0x00, 0x00, 0x00, //Compression
15           0x00, 0x58, 0x02, 0x00, //Size of Raw Data
16           0x13, 0x0B, 0x00, 0x00, //Horizontal Resolution
17           0x13, 0x0B, 0x00, 0x00, //Vertical Resolution
18           0x00, 0x00, 0x00, 0x00, //Colours in Palette
19           0x00, 0x00, 0x00, 0x00, //Important Colours
20           0x00, 0xF8, 0x00, 0x00, //Red Mask
21           0xE0, 0x07, 0x00, 0x00, //Green Mask
22           0x1F, 0x00, 0x00, 0x00, //Blue Mask
23           0x00, 0x00, 0x00, 0x00, //Alpha Mask
24           0x01, 0x00, 0x00, 0x00, //Colour Space Type
25           0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
26           0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
```

```

27         0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
28         0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
29         0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
30         0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
31         0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
32         0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
33         0x00, 0x00, 0x00, 0x00, //Colour Space Endpoints
34         0x00, 0x00, 0x00, 0x00, //Gamma Red
35         0x00, 0x00, 0x00, 0x00, //Gamma Green
36         0x00, 0x00, 0x00, 0x00, //Gamma Blue
37         0x03, 0x00, 0x00, 0x00, //Intent - Photo
38         0x00, 0x00, 0x00, 0x00, //ICC Profile Data
39         0x00, 0x00, 0x00, 0x00, //ICC Profile Size
40         0x00, 0x00, 0x00, 0x00}; //Reserved

42 const uint8_t BMPHeader[BMPHEADERSIZE] = { 0x42, 0x4D,
43                                         0x8A, 0x58, 0x02, 0x00, //Size
44                                         0x00, 0x00, 0x00, 0x00, //Reserved
45                                         0x8A, 0x00, 0x00, 0x00 //Offset to Pixel Array
46 };

```

D.1.1.3 CustomDevices.h

..../Code/The_Columbus/ColumbusTest/src/CustomDevices/CustomDevices.h

```

1  /*
2   * CustomDevices.h
3   *
4   * Created: 16/02/2013 14:30:50
5   * Author: hslovett
6   */
7
8
9 #ifndef CUSTOMDEVICES_H_
10 #define CUSTOMDEVICES_H_

12 //Camera
13 #include "CustomDevices/0V7670.h"
14 //I2C Mux
15 #include "CustomDevices/PCA9542A.h"
16 //MotorDriver
17 #include "CustomDevices/MotorDriver.h"
18 //SDCard
19 #include "CustomDevices/SD_Card.h"
20 //Image Processing Functions
21 #include "CustomDevices/ImageProcessor.h"

23 typedef struct {
24     int Status;
25     SD_Status_t *SD_Card;
26     Motor_Control_t *Motors;
27     OV7670_t *Cameras;
28     PCA9542A_t *I2CMux;
29 } Columbus_Status_t;

```

```

31 #define SD_ERR      0x1
32 #define CAM_ERR     0x2
33 #define I2CMux_ERR  0x4
34
35 #define FFT_SIZE   16
36
37 mspace sdram_msp;
38 Columbus_Status_t Columbus_Status;
39 //TWI Methods
40 void twim_init (void);
41 void System_Test();
42 #endif /* CUSTOMDEVICES_H_ */

```

D.1.1.4 dummy.c

..../Code/The_Columbus/ColumbusTest/src/CustomDevices/dummy.h

```

1 /*
2  *XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
3  */

```

D.1.1.5 ImageProcessor.h

..../Code/The_Columbus/ColumbusTest/src/CustomDevices/ImageProcessor.h

```

1 /*
2  *  ImageProcessor.h
3  *
4  *  Created: 28/02/2013 17:46:37
5  *  Author: hslovett
6  */
7
8
9 #ifndef IMAGEPROCESSOR_H_
10 #define IMAGEPROCESSOR_H_
11
12 #define BMP_FORMAT_RGB565      1
13 #define BMP_FORMAT_RGB555      2
14 #define BMP_FORMAT_GREYSCALE   3
15 #define BMP_FORMAT_1xUINT      4
16 #define BMP_FORMAT_2xUINT8T    5
17
18 typedef struct {
19     uint16_t *ImagePtr;
20     int Height;
21     int Width;
22     uint8_t Format;
23 } Image_t;

```

```

26 int FFT1D(int *Signal);
27 int FFT2Dabs(int *Signal);
28 int log_2(int i);

30 int* FFT2DCOMPLEX( int *Signal, dsp16_complex_t *ComplexBuffer, int size );
31 void PrepareImage(Image_t *Image);
32 //int* IFFT2D (dsp16_complex_t *Result, dsp16_complex_t *Input);
33 void IFFT2D (dsp16_complex_t *Signal); /*Need to test this! */;
34 #endif /* IMAGEPROCESSOR_H_ */

```

D.1.1.6 ImageProcessor.c

..../Code/The_Columbus/ColumbusTest/src/CustomDevices/ImageProcessor.c

```

1 /*
2  * ImageProcessor.c
3  *
4  * Created: 28/02/2013 17:46:50
5  * Author: hslovett
6  */
7 #include <asf.h>
8 #include "CustomDevices/CustomDevices.h"

10 /*#define FFT_SIZE 64*/
11 //Returns log base 2 of i - checks if it is an integer power of 2
12 int log_2(int i)
13 {
14     int ret = 0;
15     if((i & (i - 1)) != 0)
16     {
17         return -1;
18     }
19     while((i & 1) == 0) //while the bit isn't in the lowest bit (already
20     established this is a integer power of 2)
21     {
22         i >>= 1;
23         ret++;
24     }

25     return ret;
26 }
27 //One Dimensional Fast Fourier Transform
28 int FFT1D( int *Signal)
29 {
30     // int size = 64;
31     int log2Size, i =0;
32     // double a;
33     // log2Size = log_2(size);
34     // if(log2Size & 1) //if it is an odd power of two
35     //     return 0;

37     //am I making this all too complex for myself?! May just stick to defined
38     // size of 256.
39     // A_ALIGNED dsp32_complex_t *vect1;

```

```

39 // A_ALIGNED dsp32_t *vect2;
40
41 // vect1 = mspace_malloc(sdram_msp, (sizeof(dsp32_complex_t) * size));
42 // vect2 = mspace_malloc(sdram_msp, (sizeof(dsp32_t) * size));
43
44 //Defined Sizes
45
46 A_ALIGNED dsp16_complex_t vect1[FFT_SIZE];
47 A_ALIGNED dsp16_t vect2[FFT_SIZE];
48 for(i = 0; i < FFT_SIZE; i++)
49 {
50     vect2[i] = Signal[i];
51 }
52 dsp16_trans_realcomplexfft(vect1, vect2, log_2(FFT_SIZE));
53 dsp16_vect_complex_abs(vect2, vect1, FFT_SIZE);
54 for(i = 0; i < FFT_SIZE; i++)
55 {
56     Signal[i] = vect2[i] * FFT_SIZE;
57 }
58 // mspace_free(sdram_msp, vect1);
59 // mspace_free(sdram_msp, vect2);
60 return Signal;
61 }

62 int FFT2Dabs( int *Signal )
63 {
64     int i, j = 0;
65     int Ptr;
66     Ptr = 0;
67     A_ALIGNED dsp16_complex_t Input_C_1D[FFT_SIZE];
68     A_ALIGNED dsp16_complex_t Result_C_1D[FFT_SIZE];
69     A_ALIGNED dsp16_complex_t Result_C_2D[FFT_SIZE*FFT_SIZE];
70     A_ALIGNED dsp16_t Input_R_1D[FFT_SIZE];
71
72     //Stage 1 - FFT Real values from Signal. Store VERTICALLY in Result_2D
73     for(i = 0; i < FFT_SIZE; i++) //for each row
74     {
75         // print_dbg("\n\rInput to FFT: \n\r");
76         for(j = 0; j < FFT_SIZE; j++)
77         {
78             Input_R_1D[j] = Signal[Ptr++]; //copy the data across
79             // print_dbg_ulong(Input_R_1D[j]);
80             // print_dbg(" , ");
81         }
82         // print_dbg("\b\b");
83         //Do the FFT
84         dsp16_trans_realcomplexfft(Result_C_1D, Input_R_1D, log_2(FFT_SIZE));
85         //Copy data into 2D result TRANSPOSED
86         // print_dbg("\n\rOutput of FFT:\n\r");
87         for(j = 0; j < FFT_SIZE; j++)
88         {
89             Result_C_2D[i + (j * FFT_SIZE)].imag = Result_C_1D[j].imag * FFT_SIZE;// scale back up
90             Result_C_2D[i + (j * FFT_SIZE)].real = Result_C_1D[j].real * FFT_SIZE;
91             // print_dbg_ulong(Result_C_2D[i + (j * FFT_SIZE)].real);
92             // print_dbg(" + j");
93             // print_dbg_ulong(Result_C_2D[i + (j * FFT_SIZE)].imag);
94             // print_dbg(" , ");
95         }

```

```
96     }
97 //    print_dbg("\b\b]");
98 }
99 //Stage 2 - FFT Complex Values from Result_2D, put back into Rows
100
101 for(i = 0; i < FFT_SIZE; i++)//for each row
102 {
103 //    print_dbg("\n\rInput to FFT: \n\r[");
104 //    for(j = 0; j < FFT_SIZE; j++)//copy the data across
105 //
106     Input_C_1D[j].imag = Result_C_2D[j + i * FFT_SIZE].imag;
107     Input_C_1D[j].real = Result_C_2D[j + i * FFT_SIZE].real;
108 //    print_dbg_ulong(Input_C_1D[j].real);
109 //    print_dbg(" + j");
110 //    print_dbg_ulong(Input_C_1D[j].imag);
111 //    print_dbg(" , ");
112 }
113 //    print_dbg("\b\b]");
114 //Do Fourier
115 dsp16_trans_complexfft(Result_C_1D, Input_C_1D, log_2(FFT_SIZE));
116 //Copy back
117
118 //    print_dbg("\n\rOutput to FFT: \n\r[");
119 //    for(j = 0; j < FFT_SIZE; j++)//copy the data across
120 //
121 //    print_dbg_ulong(Result_C_1D[j].real);
122 //    print_dbg(" + j");
123 //    print_dbg_ulong(Result_C_1D[j].imag);
124 //    print_dbg(" , ");
125 }
126 //    print_dbg("\b\b]");
127 //Calculate Abs and put back into Signal TRANSPOSED
128 dsp16_vect_complex_abs(Input_R_1D, Result_C_1D, FFT_SIZE);
129
130 for(j = 0; j < FFT_SIZE; j++)
131 {
132     Signal[i + (j*FFT_SIZE)] = Input_R_1D[j] * FFT_SIZE;
133 }
134 }
135 return Signal;
136 }

137 //*****
138 // Method:      FFT2DCOMPLEX
139 // FullName:   FFT2DCOMPLEX
140 // Access:     public
141 // Returns:    int*
142 // Qualifier:
143 // Parameter: int * Signal
144 // Parameter: A_ALIGNED dsp16_complex_t * ComplexBuffer
145 // Parameter: int size
146 //*****
147 int* FFT2DCOMPLEX( int *Signal, dsp16_complex_t *ComplexBuffer, int size )
148 {
149     int i, j = 0;
150     int Ptr;
151     Ptr = 0;
```

```

154 A_ALIGNED dsp16_complex_t Input_C_1D[FFT_SIZE];
155 A_ALIGNED dsp16_complex_t Result_C_1D[FFT_SIZE];
156 A_ALIGNED dsp16_complex_t Result_C_2D[FFT_SIZE*FFT_SIZE];
157 A_ALIGNED dsp16_t Input_R_1D[FFT_SIZE];

160 //Stage 1 - FFT Real values from Signal. Store VERTICALLY in Result_2D
161 for(i = 0; i < FFT_SIZE; i++) //for each row
162 {
163     for(j = 0; j < FFT_SIZE; j++)
164     {
165         Input_R_1D[j] = Signal[Ptr++];
166     }

168 //Do the FFT
169 dsp16_trans_realcomplexfft(Result_C_1D, Input_R_1D, log_2(FFT_SIZE));
170 //Copy data into 2D result TRANSPOSED

172 for(j = 0; j < FFT_SIZE; j++)
173 {
174     Result_C_2D[i + (j * FFT_SIZE)].imag = Result_C_1D[j].imag * FFT_SIZE;// scale back up
175     Result_C_2D[i + (j * FFT_SIZE)].real = Result_C_1D[j].real * FFT_SIZE;
176 }

178 }
179 //Stage 2 - FFT Complex Values from Result_2D, put back into Rows

182 for(i = 0; i < FFT_SIZE; i++)//for each row
183 {

185 for(j = 0; j < FFT_SIZE; j++)//copy the data across
186 {
187     Input_C_1D[j].imag = Result_C_2D[j + i * FFT_SIZE].imag;
188     Input_C_1D[j].real = Result_C_2D[j + i * FFT_SIZE].real;

190 }

192 //Do Fourier
193 dsp16_trans_complexfft(Result_C_1D, Input_C_1D, log_2(FFT_SIZE));
194 //Copy back

197 //Calculate Abs and put back into Signal TRANSPOSED
198 //dsp16_vect_complex_abs(Input_R_1D, Result_C_1D, FFT_SIZE);

200 for(j = 0; j < FFT_SIZE; j++)
201 {
202     ComplexBuffer[i + j * FFT_SIZE].imag = Result_C_1D[j].imag;
203     ComplexBuffer[i + j * FFT_SIZE].real = Result_C_1D[j].real;
204     //Signal[i + (j*FFT_SIZE)] = Input_R_1D[j] * FFT_SIZE;
205 }
206 }

208 return Signal;
209 }

```

```
211 void PrepareImage(Image_t *Image)
212 {
213     int row, col;
214     uint16_t *PreparedImage;
215     //Allocate some memory in the RAM
216     PreparedImage = mspace_malloc(sdram_msp, 256*256 );
217
218     //print_dbg("\n\rPrepared Image Pointer = ");
219     //print_dbg_ulong(PreparedImage);
220     for(row = 0; row < 256; row++)
221     {
222         for(col = 0; col < 256; col++)
223         {
224             if(row < 240)
225                 PreparedImage[row*256 + col] = Image->ImagePtr[row * 256 + col];
226             else
227                 PreparedImage[row *256 + col] = 0;//Image->ImagePtr[(row - 240) * 256
228             + col + 32];
229         }
230     }
231
232     mspace_free(sdram_msp, Image->ImagePtr); //free up the old image
233     Image->ImagePtr = PreparedImage; //move the pointer to the prepared image
234     Image->Height = 256;
235     Image->Width = 256;
236     //SaveBitmap(PreparedImage, 256, 256, "PreparedImage.bmp");
237     //mspace_free(sdram_msp, PreparedImage);
238     //return PreparedImage;
239 }
240
241 //*****
242 // Method:    IFFT2D
243 // FullName:  IFFT2D
244 // Access:   public
245 // Returns:  void
246 // Qualifier:
247 // Parameter: dsp16_complex_t * Signal
248 //*****
249 void IFFT2D (dsp16_complex_t *Signal) //Need to test this!
250 {
251     int i, j = 0;
252     int Ptr;
253     Ptr = 0;
254     A_ALIGNED dsp16_complex_t Input_C_1D[FFT_SIZE];
255     A_ALIGNED dsp16_complex_t Result_C_1D[FFT_SIZE];
256     A_ALIGNED dsp16_complex_t Result_C_2D[FFT_SIZE*FFT_SIZE];
257     A_ALIGNED dsp16_t Input_R_1D[FFT_SIZE];
258
259
260     //Stage 1 - FFT Real values from Signal. Store VERTICALLY in Result_2D
261     for(i = 0; i < FFT_SIZE; i++) //for each row
262     {
263         for(j = 0; j < FFT_SIZE; j++)
264         {
265             Input_C_1D[j].real = Signal[Ptr].real; //copy the data across
266             Input_C_1D[j].imag = Signal[Ptr].imag;
267         }
```

```

269 //Do the FFT
270 dsp16_trans_complexifft(Result_C_1D, Input_C_1D, log_2(FFT_SIZE));
271 //Copy data into 2D result TRANSPOSED

273 for(j = 0; j < FFT_SIZE; j++)
274 {
275     Result_C_2D[i + (j * FFT_SIZE)].imag = Result_C_1D[j].imag * FFT_SIZE;//  
scale back up
276     Result_C_2D[i + (j * FFT_SIZE)].real = Result_C_1D[j].real * FFT_SIZE;
277 }

279 }
280 //Stage 2 - FFT Complex Values from Result_2D, put back into Rows

283 for(i = 0; i < FFT_SIZE; i++)//for each row
284 {

286     for(j = 0; j < FFT_SIZE; j++)//copy the data across
287     {
288         Input_C_1D[j].imag = Result_C_2D[j + i * FFT_SIZE].imag;
289         Input_C_1D[j].real = Result_C_2D[j + i * FFT_SIZE].real;

291     }

293 //Do Fourier
294 dsp16_trans_complexifft(Result_C_1D, Input_C_1D, log_2(FFT_SIZE));
295 //Copy back

298 //Put back into Signal TRANSPOSED
299 //dsp16_vect_complex_abs(Input_R_1D, Result_C_1D, FFT_SIZE);

301 for(j = 0; j < FFT_SIZE; j++)
302 {
303     Signal[i + j * FFT_SIZE].imag = Result_C_1D[j].imag;
304     Signal[i + j * FFT_SIZE].real = Result_C_1D[j].real;
305     //Signal[i + (j*FFT_SIZE)] = Input_R_1D[j] * FFT_SIZE;
306 }
307 }

309 //return Signal;
310 }

312 void ComplexMultiply(dsp16_complex_t *Result_Input1, dsp16_complex_t *Input2,
313 int size)
314 {
315     int i = 0;
316     dsp16_complex_t c;
317     for(i = 0; i < size; i++)
318     {
319         //((a+jb).(c+jd) = (ac - bd) + j(ad + bc)
320         c.real = (Result_Input1[i].real * Input2[i].real) - (Result_Input1[i].imag  
* Input2[i].imag);
321         c.imag = (Result_Input1[i].real * Input2[i].imag) + (Result_Input1[i].imag  
* Input2[i].real);
322         Result_Input1[i].imag = c.imag;
323     }
}

```

```

322     Result_Input1[i].real = c.real;
323 }
324 }
```

D.1.1.7 MotorDriver.h

..../Code/The_Columbus/ColumbusTest/src/CustomDevices/MotorDriver.h

```

1  /*
2  * MotorDriver.h
3  *
4  * Created: 10/02/2013 18:11:55
5  * Author: hslovett
6  */
7
8
9 #ifndef MOTORDRIVER_H_
10 #define MOTORDRIVER_H_
11 #include <asf.h>
12 //Definitions
13 #define MOTOR_L      ML_PWM_CHANNEL_ID
14 #define MOTOR_R      MR_PWM_CHANNEL_ID
15
16 #define FORWARD      2
17 #define BACKWARD     3
18 #define LEFT_SPOT    4
19 #define RIGHT_SPOT   5
20 #define SPOT_PIVOT   6
21 #define STOP         7
22
23
24 #define ENABLE ACA_INTERRUPT    // {AVR32_ACIFA1.ier = 1;}
25 #define DISABLE ACA_INTERRUPT  // {AVR32_ACIFA1.idr = 1;}
26 #define ENABLE ACB_INTERRUPT    // {AVR32_ACIFA1.ier = 2;}
27 #define DISABLE ACB_INTERRUPT  // {AVR32_ACIFA1.idr = 2;}
28 #define INTERRUPTS_PER_REVOLUTION 5 // Interrupts caused per full rotation of a
29 //wheel
30 #define CIRCUMFERENCE_WHEEL_MM    116 //in millimeters
31 #define CIRCUMFERENCE_WHEEL_CM    12 //in centimeters
32 #define MIN_RESOLUTION           CIRCUMFERENCE_WHEEL_CM /
33 //INTERRUPTS_PER_REVOLUTION
34 #define C_ROT_MM                 276
35 #define ROTATION_CONST          (INTERRUPTS_PER_REVOLUTION * C_ROT_MM) / (
36 //CIRCUMFERENCE_WHEEL_MM * 360)
37
38 //Type Defs
39 typedef struct {
40     int Left_State;
41     int Right_State;
42     int Left_Count;
43     int Right_Count;
44 } Motor_Control_t;
45
46 //Globals
47 pwm_opt_t pwm_opt; // PWM option config.
```

```

44  avr32_pwm_channel_t pwm_channel;
45  Motor_Control_t Motor_Control;

46  void Motor_Init();
47  void Motor_Go();
48  void Analogue_Comparator_Init();
49  void Motor_Start(int Motors);
50  void Motors_Reset(void);
51  void Motor_Stop(int Motors);
52  bool Motors_Moving();
53  void Motors_Move(int centimetres_fwd)/*Move this amount forward in centimeters
   */;
54  void Motors_Execute();
55  void Motors_Rotate(int angle_degs);
56  /*static void ACInterruptHandler(void);*/
57
58 #endif /* MOTORDRIVER_H_ */

```

D.1.1.8 MotorDriver.c

..../Code/The_Columbus/ColumbusTest/src/CustomDevices/MotorDriver.c

```

1  /*
2  * MotorDriver.c
3  *
4  * Created: 10/02/2013 18:12:07
5  * Author: hslovett
6  */
7  #include <asf.h>
8  #include "CustomDevices/CustomDevices.h"
9  #include <delay.h>
10 //Camera
11 /*#include "CustomDevices/0V7670.h"*/
12 //I2C Mux
13 /*#include "CustomDevices/PCA9542A.h"*/
14 //MotorDriver
15 /*#include "CustomDevices/MotorDriver.h"*/
16 //SDCard
17 /*#include "CustomDevices/SD_Card.h"*/

21 static void local_start_highfreq_clock(void)
22 {
23     const scif_pll_opt_t opt = {
24         .osc = SCIF_OSC0,          // Sel Osc0/PLLO or Osc1/PLL1
25         .lockcount = 16,           // lockcount in main clock for the PLL wait
26         lock
27             .div = 1,                // DIV=1 in the formula
28             .mul = 6,                // MUL=7 in the formula
29             .pll_div2 = 1,           // pll_div2 Divide the PLL output frequency
                           by 2 (this settings does not change the FVCO value)
                           .pll_wbwdisable = 0,    // pll_wbwdisable 1 Disable the Wide-Bandith
                           Mode (Wide-Bandwith mode allow a faster startup time and out-of-lock time
                           ). 0 to enable the Wide-Bandith Mode.

```



```

85             {0}, // dt
86             {0}); // dtupd ; One channel config.
87 /* unsigned int channel_id;*/

88 // Start PLL for PWM
89 local_start_highbfreq_clock();
90 // Start Enable Generic Clock with PLL as source clock
91 pwm_start_gc();

95 // gpio_enable_module_pin(EXAMPLE_PWM_L_PIN, EXAMPLE_PWM_L_FUNCTION);
96 // gpio_enable_module_pin(EXAMPLE_PWM_H_PIN, EXAMPLE_PWM_H_FUNCTION);
97 // gpio_enable_module_pin(M0_PWM_H_PIN, M0_PWM_H_FUNCTION);
98 // gpio_enable_module_pin(ADM32_PIN_PB10, ADM32_PWM_PWMH_1_1_FUNCTION); // PWM1 Low
99 gpio_enable_module_pin(ML_PWM_H_PIN, ML_PWM_H_FUNCTION);
100 gpio_enable_module_pin(MR_PWM_H_PIN, MR_PWM_H_FUNCTION); //PWM1 Low
101 //gpio_enable_module_pin(M1_PWM_H_PIN, M1_PWM_H_FUNCTION);
102 // PWM controller configuration.
103 pwm_opt.diva = ADM32_PWM_DIVA_CLK_OFF;
104 pwm_opt.divb = ADM32_PWM_DIVB_CLK_OFF;
105 pwm_opt.prea = ADM32_PWM_PREA_CCK;
106 pwm_opt.preb = ADM32_PWM_PREB_CCK;

108 pwm_opt.fault_detection_activated = false;
109 pwm_opt.sync_channel_activated = true;
110 pwm_opt.sync_update_channel_mode =
    PWM_SYNC_UPDATE_MANUAL_WRITE_MANUAL_UPDATE;
111 pwm_opt.sync_channel_select[0] = false;
112 pwm_opt.sync_channel_select[1] = false;
113 pwm_opt.sync_channel_select[2] = false;
114 pwm_opt.sync_channel_select[3] = false;
115 pwm_opt.cksel = PWM_CKSEL_GCLK;
116 pwm_init(&pwm_opt);

118 // Update the period
119 pwm_update_period_value(10);

121 // Channel configuration
122 pwm_channel.CMR.dte = 0; // Enable Deadtime for complementary Mode
123 pwm_channel.CMR.dthi = 0; // Deadtime Inverted on PWMH
124 pwm_channel.CMR.dtli = 0; // Deadtime Not Inverted on PWML
125 pwm_channel.CMR.ces = 0; // 0/1 Channel Event at the End of PWM
    Period
126 pwm_channel.CMR.calg = PWM_MODE_LEFT_ALIGNED; // Channel mode.
127 pwm_channel.CMR.cpol = PWM_POLARITY_HIGH; // Channel polarity.
128 pwm_channel.CMR.cpre = ADM32_PWM_CPRE_CCK; // Channel prescaler.
129 pwm_channel.cdtv = 50; // Channel duty cycle, should be < CPRD.
130 pwm_channel.cprd = 200; // Channel period.

132 /* channel_id = M0_PWM_CHANNEL_ID; */
133 pwm_channel_init(ML_PWM_CHANNEL_ID, &pwm_channel); // Set channel
    configuration to channel 0
134 //pwm_start_channels((1 << channel_id)); // Start channel 0 & 1.
135 /* channel_id = M1_PWM_CHANNEL_ID; */
136 pwm_channel_init(MR_PWM_CHANNEL_ID, &pwm_channel); // Set channel
    configuration to channel 0
137 //pwm_start_channels((1 << channel_id)); // Start channel 0 & 1.

```

```
138     Analogue_Comparator_Init();
139 }
140 __attribute__((__interrupt__)) static void ACInterruptHandler(void)
141 {
142     //print_dbg("\n\rACIFA Interrupt Entered.");
143     acifa_clear_flags(&AVR32_ACIFA1, 3);
144
145     if (acifa_is_acb_inp_higher(&AVR32_ACIFA1)) //LEFT MOTOR
146     {
147         LED5_CLR; //wheel not on white tab
148     }
149     else
150     {
151         LED5_SET;
152         Motor_Control.Left_Count--;
153         print_dbg("\n\rLeft Wheel Interrupt");
154         DISABLE_ACB_INTERRUPT;
155         //delay_ms(100);
156     }
157
158     if (acifa_is_aca_inp_higher(&AVR32_ACIFA1))
159     {
160
161         LED6_CLR;
162
163     }
164     else
165     {
166         LED6_SET;
167         Motor_Control.Right_Count--;
168         print_dbg("\n\rRight Wheel Interrupt");
169         //delay_ms(100);
170         DISABLE ACA_INTERRUPT;
171     }
172
173     int temp = 0;
174     if(Motor_Control.Left_Count <= 0) //if we have reached the end of the
175         movement on left wheel
176     temp |= MOTOR_L;
177
178     if(Motor_Control.Right_Count <= 0)
179         temp |= MOTOR_R;
180     if(temp != 0)
181         Motor_Stop(temp); //Stop the Motor
182     //delay_ms(100);
183 }
184 void Analogue_Comparator_Init()
185 {
186     static const gpio_map_t ACIFA_GPIO_MAP =
187     {
188         {POTO_AC1AP1_PIN, POTO_AC1AP1_FUNCTION},
189         {POT1_AC1BP1_PIN, POT1_AC1BP1_FUNCTION},
190         {SENSEO_AC1AN1_PIN, SENSEO_AC1AN1_FUNCTION},
191         {SENSE1_AC1BN1_PIN, SENSE1_AC1BN1_FUNCTION},
192     };
}
```

```

194     gpio_enable_module(ACIFA_GPIO_MAP, sizeof(ACIFA_GPIO_MAP) / sizeof(
195         ACIFA_GPIO_MAP[0]));
196     //Make it an interrupt
197     Disable_global_interrupt();
198
199     //INTC_init_interrupts();
200     acifa_configure_hysteresis(&AVR32_ACIFA1, ACIFA_COMP_SELA, 2);
201     acifa_configure(&AVR32_ACIFA1,
202         ACIFA_COMP_SELA,
203         POTO_AC1AP1_INPUT,
204         SENSEO_AC1AN1_INPUT,
205         FOSCO);
206
207     acifa_configure_hysteresis(&AVR32_ACIFA1, ACIFA_COMP_SELB, 2);
208     acifa_configure(&AVR32_ACIFA1,
209         ACIFA_COMP_SELB,
210         POT1_AC1BP1_INPUT,
211         SENSE1_AC1BN1_INPUT,
212         FOSCO);
213
214     //    //Reset Wheels
215     /*      Motor_Go(FORWARD); */
216     //MO_IN1_CLR;
217     //    M1_IN1_CLR;
218     //    while(acifa_is_aca_inp_higher(&AVR32_ACIFA1) == false)
219     //        ;
220     //    MO_IN1_CLR;
221     //
222     //    M1_IN1_SET;
223     //    while(!acifa_is_acb_inp_higher(&AVR32_ACIFA1))
224     //        ;
225     //    M1_IN1_CLR;
226
227
228     //Motor_Go(S)
229     //acifa_enable_interrupt(&AVR32_ACIFA1, (1 << AVR32_ACIFA_ACBINT )| (1 <<
230     //    AVR32_ACIFA_ACINT));//Enable ACBINT and ACAINT
231     ENABLE_ACA_INTERRUPT;
232     ENABLE_ACB_INTERRUPT;
233     AVR32_ACIFA1.iер = 3; //enable interrupts
234     //acifa_enable_interrupt_toggle(&AVR32_ACIFA1, ACIFA_COMP_SELA);
235     //acifa_enable_interrupt_toggle(&AVR32_ACIFA1, ACIFA_COMP_SELB);
236     acifa_enable_interrupt_inp_lower(&AVR32_ACIFA1, ACIFA_COMP_SELA);
237     acifa_enable_interrupt_inp_lower(&AVR32_ACIFA1, ACIFA_COMP_SELB);
238     acifa_start(&AVR32_ACIFA1, (ACIFA_COMP_SELA|ACIFA_COMP_SELB));
239
240
241     INTC_register_interrupt(&ACIInterruptHandler, AVR32_ACIFA1_IRQ , AVR32_INTC_INTO);
242
243     Enable_global_interrupt();
244 }
245 void Motor_Start(int Motors)
246 {
247     if(Motors & MOTOR_L)
248     {

```

```
249     if(Motor_Control.Left_State == FORWARD)
250     {
251         ML_IN1_SET;
252         ML_IN2_CLR;
253     }
254     else if (Motor_Control.Left_State == BACKWARD)
255     {
256         ML_IN1_CLR;
257         ML_IN2_SET;
258     }
259     else //Somethings gone wrong
260     {
261         ML_IN1_CLR;
262         ML_IN2_CLR;
263         return; //don't start any pwm channel
264     }
265     ML_GO;
266     pwm_start_channels((1 << MOTOR_L)); //Start PWM Channel on M0 line
267 }

269     if(Motors & MOTOR_R)
270     {
271         if(Motor_Control.Right_State == FORWARD)
272         {
273             MR_IN1_SET;
274             MR_IN2_CLR;
275         }
276         else if (Motor_Control.Right_State == BACKWARD)
277         {
278             MR_IN1_CLR;
279             MR_IN2_SET;
280         }
281         else //Somethings gone wrong
282         {
283             MR_IN1_CLR;
284             MR_IN2_CLR;
285             return; //don't start any pwm channel
286         }
287         MR_GO;
288         pwm_start_channels((1 << MOTOR_R));
289     }
290 }
291 void Motors_Execute()
292 {
293     while(Motors_Moving())
294     {
295         ENABLE ACA_INTERRUPT;
296         ENABLE ACB_INTERRUPT;
297         for(int i = 0; i < 750; i++)
298         {
299             delay_ms(1);
300         }
301     }
302 }
303 void Motor_Stop(int Motors)
304 {
305     if(Motors & MOTOR_L)
306     {
```

```
307     ML_STANDBY;
308     Motor_Control.Left_State = STOP;
309     pwm_stop_channels((1 << MOTOR_L)); //Start PWM Channel on M0 line
310 }
311
312 if(Motors & MOTOR_R)
313 {
314     MR_STANDBY;
315     Motor_Control.Right_State = STOP;
316     pwm_stop_channels((1 << MOTOR_R));
317 }
318
319 void Motors_Move(int centimetres_fwd)//Move this amount forward in centimeters
320 {
321     //Calculate number of interrupts of each wheel
322     int number_interrupts;
323     if(centimetres_fwd > 0)
324     {
325         Motor_Control.Left_State = FORWARD;
326         Motor_Control.Right_State = FORWARD;
327     }
328     else
329     {
330         centimetres_fwd = Abs(centimetres_fwd);
331         Motor_Control.Left_State = BACKWARD;
332         Motor_Control.Right_State = BACKWARD;
333     }
334     number_interrupts = (centimetres_fwd * (int)INTERRUPTS_PER_REVOLUTION) / (
335         int)CIRCUMFERENCE_WHEEL_CM;
336     print_dbg("\n\rNumber of interrupts to move = ");
337     print_dbg_ulong(number_interrupts);
338
339     Motor_Control.Left_Count = number_interrupts;
340     Motor_Control.Right_Count = number_interrupts;
341     Motor_Start(MOTOR_L | MOTOR_R);
342     Motors_Execute();
343 }
344
345 void Motors_Reset(void)
346 {
347     Motor_Control.Left_State = FORWARD;
348     Motor_Control.Left_Count = 1;
349     Motor_Control.Right_State = FORWARD;
350     Motor_Control.Right_Count = 1;
351     Motor_Start(MOTOR_L | MOTOR_R);
352 }
353
354 bool Motors_Moving()
355 {
356 //    if(Motor_Control.Left_State != STOP)
357 //    {
358 //        if(Motor_Control.Right_State != STOP)
359 //        {
360 //            return true;
361 //        }
362 //        else
363 //            return false;
364 //    }
365 }
```

```
364 //    else
365 //    {
366 //        return false;
367 //    }
368 if(Motor_Control.Left_State != STOP) //Left is moving
369 {
370     return true;
371 }
372 else if (Motor_Control.Right_State != STOP) //Right is moving
373 {
374     return true;
375 }
376 else
377 {
378     return false;
379 }
380 }

383 void Motors_Rotate(int angle_degs)
384 {
385     int interrupts_to_move = 0;
386     //calculate interrupts to move
387     interrupts_to_move = angle_degs * ROTATION_CONST;
388 //    if(Pivot_Type == LEFT_SPOT)
389 //    {
390 //        //Right wheel moves
391 //        Motor_Control.Left_Count = 0;
392 //        Motor_Control.Left_State = STOP;
393 //        if(interrupts_to_move > 0)
394 //        {
395 //            Motor_Control.Right_State = FORWARD;
396 //        }
397 //        else
398 //        {
399 //            Motor_Control.Right_State = BACKWARD;
400 //        }
401 //        Motor_Control.Right_Count = Abs(interrupts_to_move);
402 //    }
403 //    else if (Pivot_Type == RIGHT_SPOT)
404 //    {
405 //        //Left Wheel Moves
406 //        Motor_Control.Right_Count = 0;
407 //        Motor_Control.Right_State = STOP;
408 //        if(interrupts_to_move > 0)
409 //        {
410 //            Motor_Control.Left_State = FORWARD;
411 //        }
412 //        else
413 //        {
414 //            Motor_Control.Left_State = BACKWARD;
415 //        }
416 //        Motor_Control.Left_Count = Abs(interrupts_to_move);
417 //    }
418 //    else if (Pivot_Type == SPOT_PIVOT)
419 //    {
420 //        //Both Wheels Move
421 //        if(interrupts_to_move > 0)
```

```

422 {
423     Motor_Control.Left_State = FORWARD;
424     Motor_Control.Right_State = BACKWARD;
425 }
426 else
427 {
428     Motor_Control.Right_State = FORWARD;
429     Motor_Control.Left_State = BACKWARD;
430 }
431 Motor_Control.Left_Count = Abs(interrupts_to_move);
432 Motor_Control.Right_Count = Abs(interrupts_to_move);
433 Motor_Start(MOTOR_L | MOTOR_R);
434 // }
435 }
```

D.1.1.9 OV7670.h

..../Code/The_Columbus/ColumbusTest/src/CustomDevices/OV7670.h

```

1  /*
2   * OV7670.h
3   *
4   * Created: 15/02/2013 13:12:00
5   * Author: hslovett
6   */
7
8
9 #ifndef OV7670_H_
10 #define OV7670_H_
11 #include <asf.h>
12 ///////////////////////////////////////////////////////////////////
13 // Constants
14 ///////////////////////////////////////////////////////////////////
15 #define HEIGHT          240
16 #define WIDTH           320
17 #define PIXELSIZE       2
18 #define SETTINGS_LENGTH 167
19 #define OV7670_ADDR     0x21
20
21 #define CAMERA_LEFT      1
22 #define CAMERA_RIGHT     2
23
24 #define CAMERA_LEFT_ERR   0x10
25 #define CAMERA_RIGHT_ERR  0x20
26
27 #define BMPHEADERSIZE 14
28 #define DIBHEADERSIZE 124 //v5
29 #define FILESIZE 153738
30 ///////////////////////////////////////////////////////////////////
31 // Globals
32 ///////////////////////////////////////////////////////////////////
33 const char default_settings[SETTINGS_LENGTH][2];
34 const uint8_t DIBHead[DIBHEADERSIZE];
35 const uint8_t BMPHeader[BMPHEADERSIZE];
```

```

36  typedef struct {
37      uint8_t Status;
38      bool Camera_0_Found;
39      bool Camera_1_Found;
40      bool Camera_0_Error;
41      bool Camera_1_Error;
42      uint8_t VSYNC0_State;
43      uint8_t VSYNC1_State;
44  } OV7670_t ;
45
46 OV7670_t OV7670_Status;
47
48 #define IDLE          0
49 #define TAKE_PHOTO     1
50 #define TAKING_PHOTO   2
51 #define TAKEN_PHOTO    3
52 #define CAMERAS_BUSY   4
53
54 #define Image0Name    "Image_L_%d.bmp"
55 #define Image1Name    "Image_R_%d.bmp"
56 ///////////////////////////////////////////////////////////////////
57 // Methods
58 ///////////////////////////////////////////////////////////////////
59 void OV7670_Init(void);           //Initialises Camera
60 void FIFO_Init();
61 int TakePhoto(uint8_t Cameras);
62 bool Photos_Ready(void);
63 void Store_Image_0();
64 void Store_Image_1();
65 void FIFO_Reset(uint8_t CameraID);
66 bool Store_Both_Images();
67 //void FIFO_Reset(uint8_t CameraID);
68 ///////////////////////////////////////////////////////////////////
69 // Pins & Macros
70 ///////////////////////////////////////////////////////////////////
71 #define FIFO_0_RCLK    AVR32_PIN_PA19
72 #define FIFO_0_nRRST   AVR32_PIN_PA15
73 #define FIFO_0_WEN     AVR32_PIN_PA20
74 #define FIFO_0_WRST    AVR32_PIN_PA21
75 #define FIFO_0_nOE    AVR32_PIN_PA28
76 #define FIFO_0_VSYNC   AVR32_PIN_PA29
77
78 #define FIFO_1_RCLK    AVR32_PIN_PA23
79 #define FIFO_1_nRRST   AVR32_PIN_PA22
80 #define FIFO_1_WEN     AVR32_PIN_PA24
81 #define FIFO_1_WRST    AVR32_PIN_PA25
82 #define FIFO_1_nOE    AVR32_PIN_PA27
83
84 #define VSYNC_1_PIN      AVR32_EIC_EXTINT_1_2_PIN
85 #define VSYNC_1_FUNCTION AVR32_EIC_EXTINT_1_2_FUNCTION
86 #define VSYNC_1_LINE     1
87 #define VSYNC_1_ENABLE_INTERRUPT {eic_enable_interrupt_line(&AVR32_EIC,
88                             VSYNC_1_LINE);}
88 #define VSYNC_1_DISABLE_INTERRUPT {eic_disable_interrupt_line(&AVR32_EIC,
89                             VSYNC_1_LINE);}
90
91 #define VSYNC_0_PIN      AVR32_EIC_EXTINT_4_0_PIN
92 #define VSYNC_0_FUNCTION AVR32_EIC_EXTINT_4_0_FUNCTION

```

```

92 #define VSYNC_0_LINE          4
93 #define VSYNC_0_ENABLE_INTERRUPT {eic_enable_interrupt_line(&AVR32_EIC,
94                               VSYNC_0_LINE);}
95 #define VSYNC_0_DISABLE_INTERRUPT {eic_disable_interrupt_line(&AVR32_EIC,
96                               VSYNC_0_LINE);}

97 #define FIFO_0_RCLK_SET      {gpio_set_gpio_pin(FIFO_0_RCLK);}
98 #define FIFO_0_nRRST_SET     {gpio_set_gpio_pin(FIFO_0_nRRST);}
99 #define FIFO_0_WEN_SET       {gpio_set_gpio_pin(FIFO_0_WEN);}
100 #define FIFO_0_WRST_SET      {gpio_set_gpio_pin(FIFO_0_WRST);}
101 #define FIFO_0_nOE_SET       {gpio_set_gpio_pin(FIFO_0_nOE);}

103 #define FIFO_0_RCLK_CLR      {gpio_clr_gpio_pin(FIFO_0_RCLK);}
104 #define FIFO_0_nRRST_CLR     {gpio_clr_gpio_pin(FIFO_0_nRRST);}
105 #define FIFO_0_WEN_CLR       {gpio_clr_gpio_pin(FIFO_0_WEN);}
106 #define FIFO_0_WRST_CLR      {gpio_clr_gpio_pin(FIFO_0_WRST);}
107 #define FIFO_0_nOE_CLR       {gpio_clr_gpio_pin(FIFO_0_nOE);}

111 #define FIFO_1_RCLK_SET      {gpio_set_gpio_pin(FIFO_1_RCLK);}
112 #define FIFO_1_nRRST_SET     {gpio_set_gpio_pin(FIFO_1_nRRST);}
113 #define FIFO_1_WEN_SET       {gpio_set_gpio_pin(FIFO_1_WEN);}
114 #define FIFO_1_WRST_SET      {gpio_set_gpio_pin(FIFO_1_WRST);}
115 #define FIFO_1_nOE_SET       {gpio_set_gpio_pin(FIFO_1_nOE);}

117 #define FIFO_1_RCLK_CLR      {gpio_clr_gpio_pin(FIFO_1_RCLK);}
118 #define FIFO_1_nRRST_CLR     {gpio_clr_gpio_pin(FIFO_1_nRRST);}
119 #define FIFO_1_WEN_CLR       {gpio_clr_gpio_pin(FIFO_1_WEN);}
120 #define FIFO_1_WRST_CLR      {gpio_clr_gpio_pin(FIFO_1_WRST);}
121 #define FIFO_1_nOE_CLR       {gpio_clr_gpio_pin(FIFO_1_nOE);}

124 #define CAMERA_INPUT  {(uint8_t)((AVR32_GPIO.port[1].pvr) & 0xFF);}
125 ///////////////////////////////////////////////////////////////////
126 //Camera Register Address definitions
127 ///////////////////////////////////////////////////////////////////
128 #define OV_GAIN      0x00 //Gain Control Setting - ACG[7:0]
129 #define OV_BLUE      0x01 //Blue Channel Gain
130 #define OV_RED       0x02 //Red Channel Gain
131 #define OV_VREF      0x03 //Vertical Frame Control & ACG[9:8]
132 #define OV_COM1      0x04 //CCIR656 enable, AEC low bits (AECHH, AECH)
133 #define OV_BAVE      0x05 //U/B Average level - AUTO UPDATED
134 #define OV_GbAVE     0x06 //Y/Gb Average Level - AUTO UPDATED
135 #define OV_AECHH     0x07 //Exposure value [15:10] (AECH, COM1)
136 #define OV_RAVE      0x08 //V/R Average level - AUTO UPDATED
137 #define OV_COM2      0x09 //Soft Sleep, Output drive capability
138 #define OV_PID       0x0A //Product ID MSB Read only
139 #define OV_VER       0x0B //Product ID LSB Read Only
140 #define OV_COM3      0x0C //Output data MSB/LSB swap + other stuff
141 #define OV_COM4      0x0D //Average values - MUST BE SAME AS COM1
142 #define OV_COM5      0x0E //RESERVED
143 #define OV_COM6      0x0F //COM6
144 #define OV_AECH      0x10 //Exposure value [9:2] (see AECHH, COM1)
145 #define OV_CLKRC     0x11 //Internal Clock options
146 #define OV_COM7      0x12 //RESET, Output format
147 #define OV_COM8      0x13 //Common control 8

```

```

148 #define OV_COM9      0x14 //Automatic Gain Ceiling
149 #define OV_COM10     0x15 //PCLK, HREF and VSYNC options
150 #define OV_RSVD      0x16 //RESERVED
151 #define OV_HSTART    0x17 //Output format Horizontal Frame start
152 #define OV_HSTOP     0x18 //Output format Horizontal Frame end
153 #define OV_VSTRT     0x19 //Output format Vertical Frame start
154 #define OV_VSTOP      0x1A //Output format Vertical Frame Stop
155 #define OV_PSHFT     0x1B //Pixel Delay Select
156 #define OV_MIDH      0x1C //Manufacturer ID MSB - READ ONLY
157 #define OV_MIDL      0x1D //Manufacturer ID LSB - READ ONLY
158 #define OV_MVFP      0x1E //Mirror / Vflip Enable
159 #define OV_LAEC      0x1F //RESERVED
160 #define OV_ADCCTR0   0x20 //ADC Control
161 #define OV_ADCCTR1   0x21 //RESERVED
162 #define OV_ADCCTR2   0x22 //RESERVED
163 #define OV_ADCCTR3   0x23 //RESERVED
164 #define OV_AEW       0x24 //ACG/AEC Stable Operating Region Upper Limit
165 #define OV_AEB       0x25 //ACG/AEC Stable Operation Region Lower Limit
166 #define OV_VPT       0x26 //ACG/AEC Fast Mode Operation Region
167 #define OV_BBIAS     0x27 //B Channel Signal Output Bias
168 #define OV_GbBIAS    0x28 //Gb Channel Output Bias
169 #define OV_RSVD1     0x29 //RESERVED
170 #define OV_EXHCH     0x2A //Dummy Pixel Insert MSB
171 #define OV_EXHCL     0x2B //Dummy Pixel Insert LSB
172 #define OV_RBIAIS    0x2C //R Channel Signal Output Bias
173 #define OV_ADVFL     0x2D //LSB of insert dummy line in vertical direction
174 #define OV_AdVFH     0x2E //MSB of insert dummy line in vertical direction
175 #define OV_YAVE      0x2F //Y/G Channel Average Value
176 #define OV_HSYST     0x30 //HSYNC Rising Edge Delay (low 8 bits)
177 #define OV_HSYEN     0x31 //HSYNCE Falling Edge Delay (low 8 bits)
178 #define OV_HREF      0x32 //HREF Control
179 #define OV_CHLF      0x33 //Array Current Control - RESERVED
180 #define OV_ARBLM     0x34 //Array Reference Control - RESERVED
181 #define OV_RSVD2     0x35 //RESERVED
182 #define OV_RSVD3     0x36 //RESERVED
183 #define OV_ADCCTRL   0x37 //ADC Control - RESERVED
184 #define OV_ACOM      0x38 //ADC and Analog Common Mode Control - RESERVED
185 #define OV_OFON      0x39 //ADC Offset Control
186 #define OV_TSLB      0x3A //Line Buffer Test Option
187 #define OV_COM11     0x3B //COM11
188 #define OV_COM12     0x3C //COM12
189 #define OV_COM13     0x3D //COM13
190 #define OV_COM14     0x3E //COM14
191 #define OV_EDGE      0x3F //Edge Detection Adjustment
192 #define OV_COM15     0x40 //COM15
193 #define OV_COM16     0x41 //COM16
194 #define OV_COM17     0x42 //COM17
195 #define OV_AWBC1     0x43
196 #define OV_AWBC2     0x44
197 #define OV_AWBC3     0x45
198 #define OV_AWBC4     0x46
199 #define OV_AWBC5     0x47
200 #define OV_AWBC6     0x48
201 #define OV_RSVD4     0x49
202 #define OV_RSVD5     0x40
203 #define OV_RSVD6     0x4A
204 #define OV_REG4B     0x4B
205 #define OV_DNSTH    0x4C

```

```
206 #define OV_RSVD7      0x4D
207 #define OV_RSVD8      0x4E
208 #define OV_MTX1       0x4F
209 #define OV_MTX2       0x50
210 #define OV_MTX3       0x51
211 #define OV_MTX4       0x52
212 #define OV_MTX5       0x53
213 #define OV_MTX6       0x54
214 #define OV_BRIGHT      0x55
215 #define OV CONTRAS     0x56
216 #define OV CONTRASCNTR 0x57
217 #define OV_MTXS        0x58
218 #define OV_RSVD9       0x59
219 #define OV_RSVD9_1     0x5A
220 #define OV_RSVD9_2     0x5B
221 #define OV_RSVD9_3     0x5C
222 #define OV_RSVD9_4     0x5D
223 #define OV_RSVD9_5     0x5E
224 #define OV_RSVD9_6     0x5F
225 #define OV_RSVD10      0x60
226 #define OV_RSVD11      0x61
227 #define OV_LCC1        0x62
228 #define OV_LCC2        0x63
229 #define OV_LCC3        0x64
230 #define OV_LCC4        0x65
231 #define OV_LCC5        0x66
232 #define OV_MANU        0x67
233 #define OV_MANV        0x68
234 #define OV_GFIX        0x69
235 #define OV_GGAIN        0x6A
236 #define OV_DBLV        0x6B
237 #define OV_AWBCTR3      0x6C
238 #define OV_AWBCTR2      0x6D
239 #define OV_AWBCTR1      0x6E
240 #define OV_AWBCTR0      0x6F
241 #define OV_SCALING_XSC  0x70
242 #define OV_SCALING_YSC  0x71
243 #define OV_SCALING_DCWCTR 0x72
244 #define OV_SCALING_PCLK_DIV 0x73
245 #define OV_REG74        0x74
246 #define OV_REG75        0x75
247 #define OV_REG76        0x76
248 #define OV_REG77        0x77
249 #define OV_RSVD12      0x78
250 #define OV_RSVD13      0x79
251 #define OV_GAM1         0x7A
252 #define OV_GAM2         0x7B
253 #define OV_GAM3         0x7C
254 #define OV_GAM4         0x7D
255 #define OV_GAM5         0x7E
256 #define OV_GAM6         0x7F
257 #define OV_GAM7         0x80
258 #define OV_GAM8         0x81
259 #define OV_GAM9         0x82
260 #define OV_GAM10        0x83
261 #define OV_GAM11        0x84
262 #define OV_GAM12        0x85
263 #define OV_GAM13        0x86
```

```
264 #define OV_GAM14      0x87
265 #define OV_GAM15      0x88
266 #define OV_GAM16      0x89
267 #define OV_RSVD14     0x8A
268 #define OV_RSVD15     0x8B
269 #define OV_RSVD16     0x8C
270 #define OV_RSVD17     0x8D
271 #define OV_RSVD18     0x8E
272 #define OV_RSVD19     0x8F
273 #define OV_RSVD20     0x90
274 #define OV_RSVD21     0x91
275 #define OV_DM_LNL      0x92
276 #define OV_DM_LNH      0x93
277 #define OV_LCC6       0x94
278 #define OV_LCC7       0x95
279 #define OV_RSVD22     0x96
280 #define OV_RSVD23     0x97
281 #define OV_RSVD24     0x98
282 #define OV_RSVD25     0x99
283 #define OV_RSVD26     0x9A
284 #define OV_RSVD27     0x9B
285 #define OV_RSVD28     0x9C
286 #define OV_BD50ST    0x9D
287 #define OV_BD60ST    0x9E
288 #define OV_HIST0      0x9F
289 #define OV_HIST1      0xA0
290 #define OV_HIST2      0xA1
291 #define OV_HIST3      0xA2
292 #define OV_HIST4      0xA3
293 #define OV_HIST5      0xA4
294 #define OV_HIST6      0xA5
295 #define OV_HIST7      0xA6
296 #define OV_HIST8      0xA7
297 #define OV_HIST9      0xA8
298 #define OV_HIST10     0xA9
299 #define OV_HIST11     0xAA
300 #define OV_HIST12     0xAB
301 #define OV_STR_OPT     0xAC
302 #define OV_STR_R       0xAD
303 #define OV_STR_G       0xAE
304 #define OV_STR_B       0xAF
305 #define OV_RSVD28_1    0xB0
306 #define OV_RSVD29     0xB1
307 #define OV_RSVD30     0xB2
308 #define OV_THL_ST      0xB3
309 #define OV_RSVD31     0xB4
310 #define OV_THL_DLT     0xB5
311 #define OV_RSVD32     0xB6
312 #define OV_RSVD33     0xB7
313 #define OV_RSVD34     0xB8
314 #define OV_RSVD35     0xB9
315 #define OV_RSVD36     0xBA
316 #define OV_RSVD37     0xBB
317 #define OV_RSVD38     0xBC
318 #define OV_RSVD39     0xBD
319 #define OV_AD_CHB      0xBE
320 #define OV_AD_CHR      0xBF
321 #define OV_AD_CHGb     0xC0
```

```

322 #define OV_AD_CHGr      0xC1
323 #define OV_RSVD40      0xC2
324 #define OV_RSVD41      0xC3
325 #define OV_RSVD42      0xC4
326 #define OV_RSVD43      0xC5
327 #define OV_RSVD44      0xC6
328 #define OV_RSVD45      0xC7
329 #define OV_RSVD46      0xC8
330 #define OV_SATCTR     0xC9

334 #endif /* OV7670_H_ */

```

D.1.1.10 OV7670.c

..../Code/The_Columbus/ColumbusTest/src/CustomDevices/OV7670.c

```

1  /*
2   * OV7670.c
3   *
4   * Created: 15/02/2013 13:12:12
5   * Author: hslovett
6   */

9  #include <asf.h>
10 #include "CustomDevices/CustomDevices.h"
11 #include "stdio.h"
12 #include "delay.h"
13 // Camera
14 // #include "CustomDevices/OV7670.h"
15 // I2C Mux
16 // #include "CustomDevices/PCA9542A.h"
17 // MotorDriver
18 // /*#include "CustomDevices/MotorDriver.h"*/
19 // SDCard
20 // #include "CustomDevices/SD_Card.h"

22 __attribute__((__interrupt__)) static void VSYNC0_Handler (void)
23 {
24     //print_dbg("\n\rVSYNC0 Detected!");
25     eic_clear_interrupt_line(&AVR32_EIC, VSYNC0_LINE);
26     //VSYNC0_DISABLE_INTERRUPT;
27     switch(OV7670_Status.VSYNC0_State)
28     {
29         case TAKE_PHOTO:
30             FIFO0_WEN_SET;
31             OV7670_Status.VSYNC0_State = TAKING_PHOTO;
32             break;
33
34         case TAKING_PHOTO:
35             FIFO0_WEN_CLR;
36             OV7670_Status.VSYNC0_State = TAKEN_PHOTO;

```

```
37         break;
38
39     case (TAKEN_PHOTO):
40         FIFO_0_WEN_CLR;
41         break;
42
43     case (IDLE):
44     default:
45         VSYNC_0_DISABLE_INTERRUPT;
46         FIFO_0_WEN_CLR;
47         OV7670_Status.VSYNC0_State = IDLE;
48         break;
49     }
50 }
51
52 __attribute__((__interrupt__)) static void VSYNC1_Handler (void)
53 {
54     //print_dbg("\n\rVSYNC1 Detected!");
55     eic_clear_interrupt_line(&AVR32_EIC, VSYNC_1_LINE);
56     //VSYNC_1_DISABLE_INTERRUPT;
57     switch(OV7670_Status.VSYNC1_State)
58     {
59         case (TAKE_PHOTO):
60             FIFO_1_WEN_SET;
61             OV7670_Status.VSYNC1_State = TAKING_PHOTO;
62             //print_dbg("\n\rCase: Take Photo;");
63             break;
64
65         case (TAKING_PHOTO):
66             FIFO_1_WEN_CLR;
67             OV7670_Status.VSYNC1_State = TAKEN_PHOTO;
68             //print_dbg("\n\rCase: Taking Photo;");
69             break;
70
71         case (TAKEN_PHOTO):
72             FIFO_1_WEN_CLR;
73             //print_dbg("\n\rCase: Taken Photo;");
74             break;
75
76         case (IDLE):
77         default:
78             VSYNC_1_DISABLE_INTERRUPT;
79             FIFO_1_WEN_CLR;
80             OV7670_Status.VSYNC1_State = IDLE;
81             //print_dbg("\n\rCase: Idle;");
82             break;
83     }
84 }
85 unsigned char Write_Reg(unsigned char Register, unsigned char Data)
86 {
87     /* I2C Traffic Generated:
88      * S | OV_7670 + W | A | RegID | A | Data | A | P |
89      */
90     uint8_t Buff[2] = {Register, Data};
91     int status = twim_write(&AVR32_TWIMO, &Buff, 2, OV7670_ADDR, false);
92     return status;
93 }
94 unsigned char Read_Reg(unsigned char Register, unsigned char *Data)
```

```
95  {
96  /* I2C Traffic Generated:
97  * S | OV_ADDR + W | A | RegID | A | P |
98  * S | OV_ADDR + R | A | Data |~A | P |
99  */
100 unsigned char Buff[2] = {Register, 0};
101 int status = twim_write(&AVR32_TWIMO, &Buff, 1, OV7670_ADDR, false);
102 if(status != STATUS_OK)
103     return status;
104
105 status = twim_read(&AVR32_TWIMO, &Buff, 1, OV7670_ADDR, false);
106 *Data = Buff[0];
107
108 return status;
109}
110
111 void OV7670_Init()
112 {
113
114 //Check Cameras Exist
115 PCA9542A_Chан_Sel(I2C_CHANNEL_0);
116 if (twim_probe(&AVR32_TWIMO, OV7670_ADDR) == STATUS_OK)
117     OV7670_Status.Camera_0_Found = true;
118 else
119     OV7670_Status.Camera_0_Found = false;
120
121 PCA9542A_Chан_Sel(I2C_CHANNEL_1);
122 if (twim_probe(&AVR32_TWIMO, OV7670_ADDR) == STATUS_OK)
123     OV7670_Status.Camera_1_Found = true;
124 else
125     OV7670_Status.Camera_1_Found = false;
126
127 //Initialise Cameras
128 if(OV7670_Status.Camera_0_Found)
129 {
130     PCA9542A_Chан_Sel(I2C_CHANNEL_0);
131     //Reset Camera
132     if(STATUS_OK != Write_Reg(OV_COM7, 0x80))
133     {
134         print_dbg("\n\rCamera Reset Fail");
135         OV7670_Status.Camera_0_Error = true;
136         OV7670_Status.Status = ERR_DEVICE;
137         //return FAIL;
138     }
139     delay_ms(10); //wait for Camera to reset
140     for (int i = 0; i < SETTINGS_LENGTH; i++)
141     {
142         if(STATUS_OK != Write_Reg(default_settings[i][0], default_settings[i][1]))
143         {
144             print_dbg("\n\rCamera Initialise Fail");
145             //return FAIL;
146             OV7670_Status.Camera_0_Error = true;
147             OV7670_Status.Status = ERR_DEVICE;
148             break;
149         }
150     }
151     delay_ms(1);
152 }
```

```
152     }
153 }
154 if(OV7670_Status.Camera_1_Found)
155 {
156     PCA9542A_Chан_Sel(I2C_CHANNEL_1);

158 //Reset Camera
159 if(STATUS_OK != Write_Reg(OV_COM7, 0x80))
160 {
161     print_dbg("\n\rCamera Reset Fail");
162     OV7670_Status.Camera_1_Error = true;
163     OV7670_Status.Status = ERR_DEVICE;
164     //return FAIL;
165 }
166 delay_ms(10); //wait for Camera to reset
167 for (int i = 0; i < SETTINGS_LENGTH; i++)
168 {
169     if(STATUS_OK != Write_Reg(default_settings[i][0], default_settings[i]
170 [1]))
171     {
172         print_dbg("\n\rCamera Initialise Fail");
173         //return FAIL;
174         OV7670_Status.Camera_1_Error = true;
175         OV7670_Status.Status = ERR_DEVICE;
176         break;
177     }
178     delay_ms(1);
179 }
180 PCA9542A_Chан_Sel(NO_SELECT);

182 //Initialise VSYNC Interrupts
183 eic_options_t eic_options;
184 eic_options.eic_mode = EIC_MODE_EDGE_TRIGGERED;
185 eic_options.eic_edge = EIC_EDGE_FALLING_EDGE;
186 eic_options.eic_async = EIC_SYNCH_MODE;
187 eic_options.eic_line = VSYNC_1_LINE;
188 //eic_options.eic_line = VSYNC_0_LINE;

190 Disable_global_interrupt();
191 gpio_enable_module_pin(VSYNC_1_PIN, VSYNC_1_FUNCTION);
192 gpio_enable_module_pin(VSYNC_0_PIN, VSYNC_0_FUNCTION);

194 gpio_enable_pin_pull_up(VSYNC_1_PIN); //Enable pull up as it is a low level
195     interrupt
196 gpio_enable_pin_pull_up(VSYNC_0_PIN);
197 //Initialise EIC
198 eic_init(&AVR32_EIC, &eic_options, 1);
199 eic_options.eic_line = VSYNC_0_LINE;
200 eic_init(&AVR32_EIC, &eic_options, 1);

201 INTC_register_interrupt(&VSYNC1_Handler, AVR32_EIC_IRQ_1, AVR32_INTC_INTO);
202 INTC_register_interrupt(&VSYNC0_Handler, AVR32_EIC_IRQ_4, AVR32_INTC_INTO);
203 //Enable interrupt on VSYNC1
204 eic_enable_line(&AVR32_EIC, VSYNC_1_LINE);
205 eic_enable_line(&AVR32_EIC, (VSYNC_0_LINE));
206 VSYNC_1_ENABLE_INTERRUPT;
207 VSYNC_0_ENABLE_INTERRUPT;
```

```
209     FIFO_Init();
210     Enable_global_interrupt();

212 }
213 void FIFO_Init()
214 {
215     //Disable both outputs
216     FIFO_0_nOE_SET
217     FIFO_1_nOE_SET

219     //Reset Buffer 0
220     FIFO_0_WRST_CLR;
221     FIFO_0_RCLK_CLR;
222     FIFO_0_nRRST_SET;
223     FIFO_0_WEN_CLR;
224     delay_us(10);
225     FIFO_0_RCLK_SET;
226     delay_us(10);
227     FIFO_0_RCLK_CLR;
228     FIFO_0_nRRST_CLR;
229     delay_us(10);
230     FIFO_0_RCLK_SET;
231     delay_us(10);
232     FIFO_0_RCLK_CLR;
233     FIFO_0_nRRST_SET;
234     delay_us(10);
235     FIFO_0_WRST_SET;

237     //Reset Buffer 1
238     FIFO_1_WRST_CLR;
239     FIFO_1_RCLK_CLR;
240     FIFO_1_nRRST_SET;
241     FIFO_1_WEN_CLR;
242     delay_us(10);
243     FIFO_1_RCLK_SET;
244     delay_us(10);
245     FIFO_0_RCLK_CLR;
246     FIFO_1_nRRST_CLR;
247     delay_us(10);
248     FIFO_1_RCLK_SET;
249     delay_us(10);
250     FIFO_1_RCLK_CLR;
251     FIFO_1_nRRST_SET;
252     delay_us(10);
253     FIFO_1_WRST_SET;
254 }

256 void FIFO_Reset(uint8_t CameraID)
257 {
258     FIFO_0_nOE_SET;
259     FIFO_1_nOE_SET;
260     if(CameraID & CAMERA_LEFT)
261     {
262         FIFO_0_WRST_CLR;
263         FIFO_0_nRRST_CLR;
264         FIFO_0_RCLK_SET;
265         delay_us(10);
266 }
```

```
266     FIFO_0_RCLK_CLR;
267     FIFO_0_nRRST_SET;
268     FIFO_0_WRST_SET;
269 }
270 if(CameraID & CAMERA_RIGHT)
271 {
272     FIFO_1_WRST_CLR;
273     FIFO_1_nRRST_CLR;
274     FIFO_1_RCLK_SET;
275     delay_us(10);
276     FIFO_1_RCLK_CLR;
277     FIFO_1_nRRST_SET;
278     FIFO_1_WRST_SET;
279 }
280 }

283 int TakePhoto(uint8_t Cameras)
284 {
285
286     //Only want to take pictures on cameras found
287     if(((OV7670_Status.VSYNC0_State != IDLE) || !OV7670_Status.Camera_0_Found)
288         && ((OV7670_Status.VSYNC1_State != IDLE) || !OV7670_Status.Camera_1_Found)
289     )
290         return CAMERAS_BUSY; //wait for cameras to be idle if they are found
291
292     if(Cameras & CAMERA_LEFT)
293         OV7670_Status.VSYNC0_State = TAKE_PHOTO;
294
295     if(Cameras & CAMERA_RIGHT)
296         OV7670_Status.VSYNC1_State = TAKE_PHOTO;
297     eic_clear_interrupt_line(&AVR32_EIC, VSYNC_1_LINE);
298     eic_clear_interrupt_line(&AVR32_EIC, VSYNC_0_LINE);
299     VSYNC_0_ENABLE_INTERRUPT;
300     VSYNC_1_ENABLE_INTERRUPT;
301
302     return TAKING_PHOTO;
303 }
304
305 bool Photos_Ready(void)
306 {
307     int status = 0;
308     if(OV7670_Status.Camera_0_Found == true) //If camera is there
309     {
310         if(OV7670_Status.Camera_0_Error == false)//and has no errors
311         {
312             if(OV7670_Status.VSYNC0_State == TAKEN_PHOTO)
313             {
314                 status |= 1; //camera0 has taken photo
315             }
316         }
317     }
318     else
319     {
320         status |= 1;
321     }
322     else
323     {
324         status |= 1;
325     }
326 }
```

```
322     if(0V7670_Status.Camera_1_Found == true) //If camera is there
323     {
324         if(0V7670_Status.Camera_1_Error == false)//and has no errors
325         {
326             if(0V7670_Status.VSYNC1_State == TAKEN_PHOTO)
327             {
328                 status |= 1; //camera0 has taken photo
329             }
330         }
331     }
332     else
333     {
334         status |= 1;
335     }
336     else
337     {
338         status |= 1;
339     }
340 }
341 }

345 bool Store_Both_Images()
346 {
347     if(Photos_Ready() == false)
348         return false;

350     Store_Image_1();
351     FIFO_Reset(CAMERA_RIGHT);

353     Store_Image_0();
354     FIFO_Reset(CAMERA_LEFT);

356     0V7670_Status.VSYNC0_State = IDLE;
357     0V7670_Status.VSYNC1_State = IDLE;
358     return true;
359 }
360 // void Store_Image_0()
361 // {
362 //     int i,j;
363 //     //Image0
364 //     //make file
365 //     //delete file if it exists already
366 //     char Filename_buff[15];
367 //     i = 0;
368 //     while(1)
369 //     {
370 //         nav_filelist_reset();
371 //         sprintf(&Filename_buff, Image0Name, i++);
372 //         if(nav_filelist_findname((FS_STRING)Filename_buff, false))
373 //         {
374 //             //nav_setcwd((FS_STRING)Image0Name, true, false);
375 //             // print_dbg("\n\r File Exists");
376 //             // print_dbg(&Filename_buff);
377 //             //nav_file_del(false);
378 //         }
379 //     }
380 }
```

```
380 //      {
381 //          break;
382 //      }
383 //  nav_file_create((FS_STRING)Filename_buff); //create file
384 //
385 //
386 //  file_open(FOPEN_MODE_W);
387 //  //write bitmap headers
388 //  file_write_buf(BMPHeader, BMPHEADERSIZE);
389 //  file_write_buf(DIBHead, DIBHEADERSIZE);
390 //
391 //  //read and write image data
392 //  //Image0
393 //  //reset read pointer
394 //  FIFO_0_nRRST_CLR;
395 //  FIFO_0_RCLK_SET;
396 //
397 //  FIFO_0_RCLK_CLR;
398 //  FIFO_0_nRRST_SET;
399 //  delay_us(10);
400 //  //enable output
401 //  FIFO_0_nOE_CLR;
402 //  uint8_t buffer[WIDTH * 2];
403 //
404 //  for(j = 0; j < HEIGHT; j++)
405 //  {
406 //      for(i = 0; i < WIDTH*2; i+=2)
407 //      {
408 //          FIFO_0_RCLK_SET;
409 //          delay_us(10);
410 //          buffer[i+1] = ((AVR32_GPIO.port[1].pvr) & 0xFF); //CAMERA_INPUT;
411 //          delay_us(10);
412 //          FIFO_0_RCLK_CLR;
413 //          delay_us(10);
414 //          FIFO_0_RCLK_SET;
415 //          delay_us(10);
416 //          buffer[i] = ((AVR32_GPIO.port[1].pvr) & 0xFF); //CAMERA_INPUT;
417 //          delay_us(10);
418 //          FIFO_0_RCLK_CLR;
419 //          delay_us(10);
420 //      }
421 //      file_write_buf(&buffer, WIDTH * 2);
422 //  }
423 //  FIFO_0_nOE_SET;
424 //  file_close();
425 //
426 //
427 // }

429 // void Store_Image_1()
430 //
431 //  int i, j;
432 //  uint8_t buffer[WIDTH * 2];
433 //  char Filename_buff[15];
434 //  //uint8_t *Buffer_ram;
435 //  //Buffer_ram = mspace_malloc(sdram_msp, WIDTH * 2);
436 //  // if(Buffer_ram == NULL)
437 //  // {
```

```
438 // //      print_dbg("\n\rBuffer allocation fail.\n\r");
439 // //
440 // //      return;
441 // //      }
442 // //make file
443 // //delete file if it exits already
444 // nav_filelist_reset();
445 // while(1)
446 //
447 //   sprintf(&Filename_buff, Image1Name, i++);
448 //   if(nav_filelist_findname((FS_STRING)Filename_buff, false))
449 //   {
450 //     //nav_setcwd((FS_STRING)Image1Name, true, false);
451 //     //print_dbg("\n\rImage1.bmp File Exists");
452 //     //nav_file_del(false);
453 //   }
454 //   else
455 //   {
456 //     break;
457 //   }
458 //}
459 // nav_file_create((FS_STRING)Filename_buff); //create file
460 // file_open(FOPEN_MODE_W);
461 // //write bitmap headers
462 // file_write_buf(BMPHeader, BMPHEADERSIZE);
463 // file_write_buf(DIBHead, DIBHEADERSIZE);
464 // //Image1
465 // //reset read pointer
466 // FIFO_1_nRRST_CLR;
467 //
468 // FIFO_1_RCLK_SET;
469 // delay_us(10);
470 // FIFO_1_RCLK_CLR;
471 // FIFO_1_nRRST_SET;
472 //
473 // //enable output
474 // FIFO_1_nOE_CLR;
475 // // uint8_t buffer[WIDTH * 2];
476 //
477 // for(j = 0; j < HEIGHT; j++)
478 //
479 //   for(i = 0; i < WIDTH*2; i+=2)
480 //
481 //     FIFO_1_RCLK_SET;
482 //     delay_us(10);
483 //     buffer[i+1] = ((AVR32_GPIO.port[1].pvr) & 0xFF); //CAMERA_INPUT;
484 //     delay_us(10);
485 //     FIFO_1_RCLK_CLR;
486 //     delay_us(10);
487 //     FIFO_1_RCLK_SET;
488 //     delay_us(10);
489 //     buffer[i] = ((AVR32_GPIO.port[1].pvr) & 0xFF); //CAMERA_INPUT;
490 //     delay_us(10);
491 //     FIFO_1_RCLK_CLR;
492 //     delay_us(10);
493 //
494 //   file_write_buf(&buffer, WIDTH * 2);
495 // }
```

```
496 //  
497 // FIFO_1_nOE_SET;//disable output  
498 // file_close();  
499 // //mspace_free(sDRAM_msp, Buffer_ram);  
500 // /* mspace_free(sDRAM_msp, Buffer_ram);*/  
501 // }  
  
503 void Store_Image_1()  
{  
505     int i, j;  
506     //uint8_t buffer[WIDTH * 2];  
507     char Filename_buff[15];  
508     uint8_t *Buffer_ram;  
509     Buffer_ram = mspace_malloc(sDRAM_msp, HEIGHT * WIDTH * 2);  
510     i = 0;  
511     //make file  
512     //delete file if it exists already  
513     nav_filelist_reset();  
514     while(1)  
515     {  
516         sprintf(&Filename_buff, Image1Name, i++);  
517         if(nav_filelist_findname((FS_STRING)Filename_buff, false))  
518         {  
519             ;  
520         }  
521         else  
522         {  
523             break;  
524         }  
525     }  
  
527     //Image1  
528     //reset read pointer  
529     FIFO_1_nRRST_CLR;  
  
531     FIFO_1_RCLK_SET;  
532     delay_us(10);  
533     FIFO_1_RCLK_CLR;  
534     FIFO_1_nRRST_SET;  
  
536     //enable output  
537     FIFO_1_nOE_CLR;  
538     // uint8_t buffer[WIDTH * 2];  
  
540     for(j = 0; j < HEIGHT * WIDTH * 2; j+= 2)  
541     {  
542         FIFO_1_RCLK_SET;  
543         delay_us(10);  
544         Buffer_ram[j+1] = ((AVR32_GPIO.port[1].pvr) & 0xFF); //CAMERA_INPUT;  
545         delay_us(10);  
546         FIFO_1_RCLK_CLR;  
547         delay_us(10);  
548         FIFO_1_RCLK_SET;  
549         delay_us(10);  
550         Buffer_ram[j] = ((AVR32_GPIO.port[1].pvr) & 0xFF); //CAMERA_INPUT;  
551         delay_us(10);  
552         FIFO_1_RCLK_CLR;  
553         delay_us(10);
```

```

554     }
555
556     FIFO_1_nOE_SET;//disable output
557 /*  file_close();*/
558     SaveBitmap(Buffer_ram, WIDTH, HEIGHT, Filename_buff);
559     mspace_free(sdram_msp, Buffer_ram);
560 }
561
562 void Store_Image_0()
563 {
564     int i, j;
565 //uint8_t buffer[WIDTH * 2];
566     char Filename_buff[15];
567     uint16_t *Buffer_ram;
568     Buffer_ram = mspace_malloc(sdram_msp, HEIGHT * WIDTH );
569     i = 0;
570     //make file
571     //delete file if it exists already
572     nav_filelist_reset();
573     while(1)
574     {
575         sprintf(&Filename_buff, Image0Name, i++);
576         if(nav_filelist_findname((FS_STRING)Filename_buff, false))
577         {
578             ;
579         }
580         else
581         {
582             break;
583         }
584     }
585
586 //Image1
587 //reset read pointer
588 FIFO_0_nRRST_CLR;
589
590 FIFO_0_RCLK_SET;
591 delay_us(10);
592 FIFO_0_RCLK_CLR;
593 FIFO_0_nRRST_SET;
594
595 //enable output
596 FIFO_0_nOE_CLR;
597 //  uint8_t buffer[WIDTH * 2];
598
599 for(j = 0; j < HEIGHT * WIDTH; j++)
600 {
601     FIFO_0_RCLK_SET;
602     delay_us(10);
603     Buffer_ram[j] = ((AVR32_GPIO.port[1].pvr) & 0xFF); //CAMERA_INPUT;
604     delay_us(10);
605     FIFO_0_RCLK_CLR;
606     delay_us(10);
607     FIFO_0_RCLK_SET;
608     delay_us(10);
609     Buffer_ram[j] |= (((AVR32_GPIO.port[1].pvr) & 0xFF) << 8); //CAMERA_INPUT;
610     delay_us(10);
611     FIFO_0_RCLK_CLR;

```

```
612     delay_us(10);
613 }
614
615 FIFO_0_nOE_SET; // disable output
616 /* file_close(); */
617 SaveBitmap(Buffer_ram, WIDTH, HEIGHT, Filename_buff);
618 mspace_free(sdram_msp, Buffer_ram);
619 }
```

D.1.1.11 OV7670.c

..../Code/The_Columbus/ColumbusTest/src/CustomDevices/OV7670_Setup.c

```
/*
 * OV7670_Setup.c
 *
 * Created: 15/02/2013 13:14:09
 * Author: hslovett
 */

#include "CustomDevices/CustomDevices.h"

const char default_settings[SETTINGS_LENGTH][2]=
{
{OV_TSLB, 0x04},
{OV_COM15, 0xd0}, //RGB565 / RGB555
{OV_COM7, 0x14},
{OV_HREF, 0x80},
{OV_HSTART, 0x16},
{OV_HSTOP, 0x04},
{OV_VSTRT, 0x02},
{OV_VSTOP, 0x7b}, //0x7a,
{OV_VREF, 0x06}, //0x0a,
{OV_COM3, 0x00}, //MSB and LSB swapped
{OV_COM14, 0x00}, //
{OV_SCALING_XSC, 0x00},
{OV_SCALING_YSC, 0x00},
{OV_SCALING_DCWCTR, 0x11},
{OV_SCALING_PCLK_DIV, 0x00}, //
{0xa2, 0x02},
{OV_CLKRC, 0x01},
{OV_GAM1, 0x20},
{OV_GAM2, 0x1c},
{OV_GAM3, 0x28},
{OV_GAM4, 0x3c},
{OV_GAM5, 0x55},
{OV_GAM6, 0x68},
{OV_GAM7, 0x76},
{OV_GAM8, 0x80},
{OV_GAM9, 0x88},
{OV_GAM10, 0x8f},
{OV_GAM11, 0x96},
{OV_GAM12, 0xa3},
{OV_GAM13, 0xaf},
```

```
42 {OV_GAM14, 0xc4},  
43 {OV_GAM15, 0xd7},  
44 {OV_GAM16, 0xe8},  
45 {OV_COM8, 0xe0},  
46 {OV_GAIN, 0x00}, //AGC  
47 {OV_AECH, 0x00},  
48 {OV_COM4, 0x00},  
49 {OV_COM9, 0x20}, //0x38, limit the max gain  
50 {OV_HIST6, 0x05},  
51 {OV_HIST12, 0x07},  
52 {OV_AEW, 0x75},  
53 {OV_AEB, 0x63},  
54 {OV_VPT, 0xA5},  
55 {OV_HIST0, 0x78},  
56 {OV_HIST1, 0x68},  
57 {OV_HIST2, 0x03}, //0x0b,  
58 {OV_HIST7, 0xdf}, //0xd8,  
59 {OV_HIST8, 0xdf}, //0xd8,  
60 {OV_HIST9, 0xf0},  
61 {OV_HIST10, 0x90},  
62 {OV_HIST11, 0x94},  
63 {OV_COM8, 0xe5},  
64 {OV_COM5, 0x61},  
65 {OV_COM6, 0x4b},  
66 {0x16, 0x02},  
67 {OV_MVFP, 0x27}, //0x37,  
68 {0x21, 0x02},  
69 {0x22, 0x91},  
70 {0x29, 0x07},  
71 {0x33, 0x0b},  
72 {0x35, 0x0b},  
73 {0x37, 0x1d},  
74 {0x38, 0x71},  
75 {OV_OFON, 0x2a}, //  
76 {OV_COM12, 0x78},  
77 {0x4d, 0x40},  
78 {0x4e, 0x20},  
79 {OV_GFIX, 0x0c}, ////////////////////////////  
80 {OV_DBLV, 0x60}, //PLL  
81 {OV_REG74, 0x19},  
82 {0x8d, 0x4f},  
83 {0x8e, 0x00},  
84 {0x8f, 0x00},  
85 {0x90, 0x00},  
86 {0x91, 0x00},  
87 {OV_DM_LNL, 0x00}, //0x19, //0x66  
88 {0x96, 0x00},  
89 {0x9a, 0x80},  
90 {0xb0, 0x84},  
91 {0xb1, 0x0c},  
92 {0xb2, 0x0e},  
93 {OV_THL_ST, 0x82},  
94 {0xb8, 0xa},  
95 {OV_AWBC1, 0x14},  
96 {OV_AWBC2, 0xf0},  
97 {OV_AWBC3, 0x34},  
98 {OV_AWBC4, 0x58},  
99 {OV_AWBC5, 0x28},
```

```
100 {OV_AWBC6, 0x3a},
101 {0x59, 0x88},
102 {0x5a, 0x88},
103 {0x5b, 0x44},
104 {0x5c, 0x67},
105 {0x5d, 0x49},
106 {0x5e, 0x0e},
107 {OV_LCC3, 0x04},
108 {OV_LCC4, 0x20},
109 {OV_LCC5, 0x05},
110 {OV_LCC6, 0x04},
111 {OV_LCC7, 0x08},
112 {OV_AWBCTR3, 0x0a},
113 {OV_AWBCTR2, 0x55},
114 {OV_AWBCTR1, 0x11},
115 {OV_AWBCTR0, 0x9f}, //0x9e for advance AWB
116 {OV_GGAIN, 0x40},
117 {OV_BLUE, 0x40},
118 {OV_RED, 0x40},
119 {OV_COM8, 0xe7},
120 {OV_COM10, 0x02}, //VSYNC negative
121 {OV_MTX1, 0x80},
122 {OV_MTX2, 0x80},
123 {OV_MTX3, 0x00},
124 {OV_MTX4, 0x22},
125 {OV_MTX5, 0x5e},
126 {OV_MTX6, 0x80},
127 {OV_MT XS, 0x9e},
128 {OV_COM16, 0x08},
129 {OV_EDGE, 0x00},
130 {OV_REG75, 0x05},
131 {OV_REG76, 0xe1},
132 {OV_DNSTH, 0x00},
133 {OV_REG77, 0x01},
134 {OV_COM13, 0xc2}, //0xc0,
135 {OV_REG4B, 0x09},
136 {OV_SATCTR, 0x60},
137 {OV_COM16, 0x38},
138 {OV_CONTRAS, 0x40},
139 {0x34, 0x11},
140 {OV_COM11, 0x02}, //0x00, //0x02,
141 {OV_HIST5, 0x89}, //0x88,
142 {0x96, 0x00},
143 {0x97, 0x30},
144 {0x98, 0x20},
145 {0x99, 0x30},
146 {0x9a, 0x84},
147 {0x9b, 0x29},
148 {0x9c, 0x03},
149 {OV_BD50ST, 0x4c},
150 {OV_BD60ST, 0x3f},
151 {0x78, 0x04},
152 {0x79, 0x01}, //Some weird thing with reserved registers.
153 {0xc8, 0xf0},
154 {0x79, 0x0f},
155 {0xc8, 0x00},
156 {0x79, 0x10},
157 {0xc8, 0x7e},
```

```
158 {0x79, 0xa},  
159 {0xc8, 0x80},  
160 {0x79, 0xb},  
161 {0xc8, 0x01},  
162 {0x79, 0xc},  
163 {0xc8, 0xf},  
164 {0x79, 0xd},  
165 {0xc8, 0x20},  
166 {0x79, 0x09},  
167 {0xc8, 0x80},  
168 {0x79, 0x02},  
169 {0xc8, 0xc0},  
170 {0x79, 0x03},  
171 {0xc8, 0x40},  
172 {0x79, 0x05},  
173 {0xc8, 0x30},  
174 {0x79, 0x26},  
175 {OV_COM2, 0x03},  
176 {OV_BRIGHT, 0x00},  
177 {OV_CONTRAS, 0x40},  
178 {OV_COM11, 0x42}, //0x82, //0xc0, //0xc2, //night mode  
180 };
```

D.1.1.12 PCA9542A.h

..../Code/The_Columbus/ColumbusTest/src/CustomDevices/PCA9542A.h

```
1  /*
2   *  PCA9542A.h
3   *
4   *  * Created: 15/02/2013 12:21:46
5   *  Author: hslovett
6   */
7
8
9 #ifndef PCA9542A_H_
10#define PCA9542A_H_
11
12#define A0 0
13#define A1 0
14#define A2 1
15#define PCA9542A_ADDR (0x70 | (A2 << 2) | (A1 << 1) | A0)
16
17#define NO_SELECT 0x00
18//#define ERROR 0x01
19#define I2C_CHANNEL_0 0x04
20#define I2C_CHANNEL_1 0x05
21
22//Status Codes
23#define SUCCESS 0
24#define DEVICE_NOT_FOUND 2
25
26typedef struct {
```

```
27     uint8_t Status;
28     uint8_t ChannelSelected;
29 } PCA9542A_t;

31 PCA9542A_t PCA9542A;
32 int PCA9542A_Init();
33 //void PCA9542A_Channel_Select(uint8_t Channel);
34 void PCA9542A_Chан_Sel(unsigned char Channel);
35 #endif /* PCA9542A_H_ */
```

D.1.1.13 PCA9542A.c

..../Code/The_Columbus/ColumbusTest/src/CustomDevices/PCA9542A.c

```
/*
 * PCA9542A.c
 *
 * Created: 15/02/2013 12:21:36
 * Author: hslovett
 */

#include <asf.h>
#include "CustomDevices/CustomDevices.h"
//Camera
/*#include "CustomDevices/OV7670.h"*/
//I2C Mux
/*#include "CustomDevices/PCA9542A.h"*/
//MotorDriver
/*#include "CustomDevices/MotorDriver.h"*/
//SDCard
/*#include "CustomDevices/SD_Card.h"*/

int PCA9542A_Init()
{
    int status = twim_probe(&AVR32_TWIMO, PCA9542A_ADDR);
    if (status != STATUS_OK)
    {
        PCA9542A.Status = DEVICE_NOT_FOUND;
        return DEVICE_NOT_FOUND;
    }
    char buff[2] = {NO_SELECT, 0};
    status = twim_write(&AVR32_TWIMO, &buff, 1, PCA9542A_ADDR, false);
    PCA9542A.Status = STATUS_OK;
    PCA9542A.ChannelSelected = NO_SELECT;
    return status;
}

void PCA9542A_Chан_Sel(unsigned char Channel)
{
    int status = 0;
    char buff[2] = {Channel, 0};
    status = twim_write(&AVR32_TWIMO, &buff, 1, PCA9542A_ADDR, false);
    if (status == STATUS_OK)
```

```

41  {
42      PCA9542A.ChannelSelected = Channel;
43  }
44  else
45  {
46      PCA9542A.Status = ERR_PROTOCOL;
47  }
48 }
```

D.1.1.14 SD_Card.h

..../Code/The_Columbus/ColumbusTest/src/CustomDevices/SD_Card.h

```

1  /*
2  * SD_Card.h
3  *
4  * Created: 10/02/2013 17:11:51
5  * Author: hslovett
6  */
7
8
9 #ifndef SD_CARD_H_
10 #define SD_CARD_H_
11 #include "ImageProcessor.h"
12 #define SIGNAL_FILE "signal.bin"
13 #define TWOD_SIGNAL_FILE "signal2d.bin"
14
15 typedef struct {
16     uint8_t Status;
17     uint32_t Memory_size;
18 } SD_Status_t;
19 SD_Status_t SD_Status;
20
21 void local_pdca_init(void);
22 void sd_mmc_resources_init(void);
23 static void pdca_int_handler(void);
24 void wait();
25 void Log_Write_ulong(unsigned long n);
26 void Log_Write(char *buff, int length);
27 void SaveBuff( int * WorkingBuffer , int size);
28 int Read2DSignal( int * WorkingBuffer );
29 int ReadSignal( int * WorkingBuffer );
30 void SaveBitmap(uint16_t *Image, int width, int height, char *FileName);
31 //void ReadBitmap(char *filename);
32 void ReadBitmap(char *Filename, Image_t *image);
33 #endif /* SD_CARD_H_ */
```

D.1.1.15 SD_Card.c

..../Code/The_Columbus/ColumbusTest/src/CustomDevices/SD_Card.c

```
1  /*
2   * SD_Card.c
3   *
4   * Created: 10/02/2013 17:11:58
5   * Author: hslovett
6   */
7 //Camera
8 /*#include "CustomDevices/OV7670.h"*/
9 //I2C Mux
10 /*#include "CustomDevices/PCA9542A.h"*/
11 //MotorDriver
12 /*#include "CustomDevices/MotorDriver.h"*/
13 //SDCard
14 /*#include "CustomDevices/SD_Card.h"*/
15 #include "CustomDevices/CustomDevices.h"
16 #include "conf_sd_mmc_spi.h"
17 #include <asf.h>

19 // Dummy char table
20 const char dummy_data[] =
21 #include "dummy.h"
22 ;
23 // PDCA Channel pointer
24 volatile avr32_pdca_channel_t* pdca_channelrx ;
25 volatile avr32_pdca_channel_t* pdca_channeltx ;
26 // Used to indicate the end of PDCA transfer
27 volatile bool end_of_transfer;
28 // Local RAM buffer for the example to store data received from the SD/MMC
29 // card
30 volatile char ram_buffer[1000];

32 void wait()
33 {
34     volatile int i;
35     for(i = 0 ; i < 5000; i++);
36 }
37 /* interrupt handler to notify if the Data reception from flash is
38 * over, in this case lunch the Memory(ram_buffer) to USART transfer and
39 * disable interrupt*/
40
41 static void pdca_int_handler(void)
42 {
43     // Disable all interrupts.
44     Disable_global_interrupt();

46     // Disable interrupt channel.
47     pdca_disable_interrupt_transfer_complete(AVR32_PDCA_CHANNEL_SPI_RX);

49     sd_mmc_spi_read_close_PDCA(); //unselects the SD/MMC memory.
50     wait();
51     // Disable unnecessary channel
52     pdca_disable(AVR32_PDCA_CHANNEL_SPI_TX);
53     pdca_disable(AVR32_PDCA_CHANNEL_SPI_RX);

55     // Enable all interrupts.
56     Enable_global_interrupt();
```

```
58     end_of_transfer = true;
59 }

61 /*! \brief Initializes SD/MMC resources: GPIO, SPI and SD/MMC.
62 */
63 void sd_mmc_resources_init(void)
64 {
65     // GPIO pins used for SD/MMC interface
66     static const gpio_map_t SD_MMC_SPI_GPIO_MAP =
67     {
68         {SD_MMC_SPI_SCK_PIN, SD_MMC_SPI_SCK_FUNCTION}, // SPI Clock.
69         {SD_MMC_SPI_MISO_PIN, SD_MMC_SPI_MISO_FUNCTION}, // MISO.
70         {SD_MMC_SPI_MOSI_PIN, SD_MMC_SPI_MOSI_FUNCTION}, // MOSI.
71         {SD_MMC_SPI_NPCS_PIN, SD_MMC_SPI_NPCS_FUNCTION} // Chip Select NPCS.
72     };

74     // SPI options.
75     spi_options_t spiOptions =
76     {
77         .reg          = SD_MMC_SPI_NPCS,
78         .baudrate    = SD_MMC_SPI_MASTER_SPEED, // Defined in conf_sd_mmc_spi.h.
79         .bits         = SD_MMC_SPI_BITS,           // Defined in conf_sd_mmc_spi.h.
80         .spck_delay   = 0,
81         .trans_delay  = 0,
82         .stay_act     = 1,
83         .spi_mode     = 0,
84         .modfdis     = 1
85     };

87     // Assign I/Os to SPI.
88     gpio_enable_module(SD_MMC_SPI_GPIO_MAP,
89                         sizeof(SD_MMC_SPI_GPIO_MAP) / sizeof(SD_MMC_SPI_GPIO_MAP
90                         [0]));

91     // Initialize as master.
92     spi_initMaster(SD_MMC_SPI, &spiOptions);

94     // Set SPI selection mode: variable_ps, pcs_decode, delay.
95     spi_selectionMode(SD_MMC_SPI, 0, 0, 0);

97     // Enable SPI module.
98     spi_enable(SD_MMC_SPI);

100    // Initialize SD/MMC driver with SPI clock (PBA).
101    sd_mmc_spi_init(spiOptions, PBA_HZ);
102}

104 /*! \brief Initialize PDCA (Peripheral DMA Controller A) resources for the SPI
105     transfer and start a dummy transfer
106 */
107 void local_pdca_init(void)
108 {
109     // this PDCA channel is used for data reception from the SPI
110     pdca_channel_options_t pdca_options_SPI_RX ={ // pdca channel options
111         .addr = ram_buffer,
```

```

112     // memory address. We take here the address of the string dummy_data. This
113     // string is located in the file dummy.h
114
114     .size = 512,                                     // transfer counter: here the
115     size of the string
115     .r_addr = NULL,                                    // next memory address after 1st
116     transfer complete
116     .r_size = 0,                                      // next transfer counter not
117     used here
117     .pid = AVR32_PDCA_CHANNEL_USED_RX,               // select peripheral ID - data
118     are on reception from SPI1 RX line
118     .transfer_size = PDCA_TRANSFER_SIZE_BYTE // select size of the transfer:
119     8,16,32 bits
119 }

121 // this channel is used to activate the clock of the SPI by sending a dummy
122 // variables
122 pdca_channel_options_t pdca_options_SPI_TX ={ // pdca channel options

124     .addr = (void *)&dummy_data,                   // memory address.
125                                         // We take here the address of
126                                         // the string dummy_data.
126                                         // This string is located in the
127                                         // file dummy.h
127     .size = 512,                                     // transfer counter: here the
128     size of the string
128     .r_addr = NULL,                                    // next memory address after 1st
129     transfer complete
129     .r_size = 0,                                      // next transfer counter not
130     used here
130     .pid = AVR32_PDCA_CHANNEL_USED_TX,               // select peripheral ID - data
131     are on reception from SPI1 RX line
131     .transfer_size = PDCA_TRANSFER_SIZE_BYTE // select size of the transfer:
132     8,16,32 bits
132 }

134 // Init PDCA transmission channel
135 pdca_init_channel(AVR32_PDCA_CHANNEL_SPI_TX, &pdca_options_SPI_TX);

137 // Init PDCA Reception channel
138 pdca_init_channel(AVR32_PDCA_CHANNEL_SPI_RX, &pdca_options_SPI_RX);

140 //!\ brief Enable pdca transfer interrupt when completed
141 INTC_register_interrupt(&pdca_int_handler, AVR32_PDCA_IRQ_0, AVR32_INTC_INT1
141 ); // pdca_channel_spi1_RX = 0

143 }

145 #define BUFFER_FILENAME "Buffer.bin"
146 void SaveBuff( int * WorkingBuffer , int size)
147 {
148     //If the file exists, delete it
149     if(nav_filelist_findname((FS_STRING)BUFFER_FILENAME, false))
150     {
151         nav_setcwd((FS_STRING)BUFFER_FILENAME, false, false);
152         nav_file_del(false);
153     }
154     nav_file_create((FS_STRING)BUFFER_FILENAME);

```

```
155     nav_setcwd((FS_STRING)BUFFER_FILENAME, false, true);
156     file_open(FOPEN_MODE_APPEND);
157     file_write_buf(WorkingBuffer, size * sizeof(WorkingBuffer));
158     file_close();
159 }

161 void Log_Write(char *buff, int length)
162 {
163     nav_setcwd((FS_STRING)LOG_FILE, true, false);
164     file_open(FOPEN_MODE_APPEND);
165     if(length == -1)
166         length = sizeof(buff);
167     file_write_buf(buff, length);
168     file_close();
169 }
170 void Log_Write_ulong(unsigned long n)
171 {
172     char tmp[11];
173     int i = sizeof(tmp) - 1;

175     // Convert the given number to an ASCII decimal representation.
176     tmp[i] = '\0';
177     do
178     {
179         tmp[--i] = '0' + n % 10;
180         n /= 10;
181     } while (n);

183     // Transmit the resulting string with the given USART.
184     Log_Write(tmp + i, -1);
185 }

187 int ReadSignal( int * WorkingBuffer )
188 {
189     bool status_b;
190     int Status, temp;
191     char c = 0;
192     if(Columbus_Status.SD_Card->Status != STATUS_OK)
193         return ERR_IO_ERROR;
194     nav_filelist_reset();
195     nav_setcwd((FS_STRING)SIGNAL_FILE, false, false);
196     status_b = file_open(FOPEN_MODE_R);
197     if(status_b == false)
198     {
199         print_dbg("File Open Error");
200         return ERR_IO_ERROR;
201     }

204     //Status = file_read_buf(WorkingBuffer, 16);
205     for(Status = 0; Status < FFT_SIZE; Status++)
206     {
207         //    print_dbg("\n\r Read from file: ");
208         c = 0;
209         temp = 0;
210         temp |= file_getc() << 24;
211         temp |= file_getc() << 16;
212         temp |= file_getc() << 8;
```

```
213     temp |= file_getc();
215 //     print_dbg_char(c);
217     WorkingBuffer[Status] = temp;
218 //     print_dbg(" Working Buff = ");
219 //     print_dbg_char(WorkingBuffer[Status]);
220 }
221 file_close();
222 return STATUS_OK;
223 }

225 int Read2DSignal( int * WorkingBuffer )
226 {
227     bool status_b;
228     int Status, temp;
229     char c = 0;
230     if(Columbus_Status.SD_Card->Status != STATUS_OK)
231         return ERR_IO_ERROR;
232     nav_filelist_reset();
233     nav_setcwd((FS_STRING)TWOD_SIGNAL_FILE, false, false);
234     status_b = file_open(FOPEN_MODE_R);
235     if(status_b == false)
236     {
237         print_dbg("File Open Error");
238         return ERR_IO_ERROR;
239     }

242 //Status = file_read_buf(WorkingBuffer, 16);
243 for(Status = 0; Status < FFT_SIZE * FFT_SIZE; Status++)
244 {
245 //     print_dbg("\n\r Read from file: ");
246     c = 0;
247     temp = 0;
248     temp |= file_getc() << 24;
249     temp |= file_getc() << 16;
250     temp |= file_getc() << 8;
251     temp |= file_getc();

253 //     print_dbg_char(c);

255     WorkingBuffer[Status] = temp;
256 //     print_dbg(" Working Buff = ");
257 //     print_dbg_char(WorkingBuffer[Status]);
258 }
259 file_close();
260 return STATUS_OK;
261 }

263 void SaveBitmap(uint16_t *Image, int width, int height, char *FileName)
264 {
265     int i, j, k;
266     uint8_t *Buffer;

268     nav_filelist_reset();
269     if(nav_filelist_findname((FS_STRING)FileName, false))
270     {
```

```
271     nav_setcwd((FS_STRING)FileName, true, false);
272     nav_file_del(false);
273 }
274 nav_file_create((FS_STRING)FileName);
275 file_open(FOPEN_MODE_W);
276 //write a modified bitmap header
277 //Calculate which is the biggest:
278 i = width * 2;
279 if(height > i)
280     i = height;
281 if(DIBHEADERSIZE > i)
282     i = DIBHEADERSIZE;

284 Buffer = malloc(i);

286 for(i = 0; i < BMPHEADERSIZE; i++)//copy all the header
287 {
288     Buffer[i] = BMPHeader[i];
289 }
290 //edit the size field
291 j = width * height * 2 + BMPHEADERSIZE + DIBHEADERSIZE;
292 for(i = 0; i < 4; i++)
293 {
294     Buffer[i + 2] = (uint8_t)(j >> 8*i);
295 }

297 file_write_buf(Buffer, BMPHEADERSIZE);

299 //DIB Header
300 for(i = 0; i < DIBHEADERSIZE; i++)
301 {
302     Buffer[i] = DIBHead[i];
303 }
304 Buffer[4] = (uint8_t)(width & 0xFF);
305 Buffer[5] = (uint8_t)((width >> 8) & 0xFF);
306 Buffer[6] = (uint8_t)((width >> 16) & 0xFF);
307 Buffer[7] = (uint8_t)((width >> 24) & 0xFF);

309 Buffer[8] = (uint8_t)(height & 0xFF);
310 Buffer[9] = (uint8_t)((height >> 8) & 0xFF);
311 Buffer[10] = (uint8_t)((height >> 16) & 0xFF);
312 Buffer[11] = (uint8_t)((height >> 24) & 0xFF);

314 file_write_buf(Buffer, DIBHEADERSIZE);

316 for(i = 0; i < height ; i++)
317 {
318     for(j = 0; j < width ; j++)
319     {
320         //Copy the data across.

322         /*Buffer[j] = Image[i*width + j];*/
323         Buffer[(2 * j) + 1] = (uint8_t)(Image[i*width + j]);
324         Buffer[(2 * j)] = (uint8_t)(Image[i*width + j] >> 8);
325     }
326     if(file_write_buf(Buffer, width * 2) != (width * 2))
327     {
328         print_dbg("\n\rFile write error.");
329     }
330 }
```

```
329     }
330
331 //    j = width % 4;
332 //    if(j != 0)
333 //    { //Padding is needed to make things 4 byte aligned
334 //        file_write_buf(Buffer, j);
335 //    }
336 }
337
338
339
340     free(Buffer);
341     file_close();
342 }
343
344 #define BMP_HEADER_FILESIZE_OFFSET      2
345 #define BMP_HEADER_OFFSETTOARRAY_OFFSET 10
346 #define DIB_V5_WIDTH_OFFSET           4
347 #define DIB_V5_HEIGHT_OFFSET          8
348 #define DIB_V5_BITCOUNT_OFFSET        14
349 #define DIB_V5_IMAGESIZE_OFFSET       20
350
351 int ReadBigEndian(uint8_t *Buffer, int Offset, uint size)
352 {
353     int retVal, i;
354     retVal = 0; //initialise value
355     for(i = 0; i < size; i++)
356     {
357         retVal |= Buffer[Offset + i] << (i * 8);
358     }
359     return (Buffer[Offset]) | (Buffer[Offset + 1] << 8) | (Buffer[Offset + 2] <<
360         16) | (Buffer[Offset + 3] << 24);
361 }
362 void ReadBitmap(char *Filename, Image_t *image)
363 {
364     // Image_t image;
365     int i, j, FileSize, OffsetToArray, temp, BitCount, ImageSize;
366     uint8_t Buffer[128];
367     nav_filelist_reset();
368     if(nav_filelist_findname((FS_STRING)Filename, false) == false)//if the file
369     // doesn't exist
370     {
371         print_dbg("\n\rFile ");
372         print_dbg(Filename);
373         print_dbg("\n\r does not exist;");
374         return;
375     }
376     nav_setcwd((FS_STRING)Filename, false, false);
377     file_open(FOPEN_MODE_R);
378     //Read Header
379     file_read_buf(Buffer, BMPHEADERSIZE);
380     //Check for BM to confirm it is a Bitmap
381     if((Buffer[0] != 'B') || (Buffer[1] != 'M'))
382     {
383         print_dbg("\n\rBitmap Parse Fail 'BM';");
384         return;
385     }
386     //Extract file size and offset to pixel array
```

```

385     FileSize = ReadBigEndian(Buffer, BMP_HEADER_FILESIZE_OFFSET, 4);
386     OffsetToArray = ReadBigEndian(Buffer, BMP_HEADER_OFFSETTOARRAY_OFFSET, 4);

388     file_read_buf(Buffer, DIBHEADERSIZE);
389     temp = ReadBigEndian(Buffer, 0, 4);
390     if(temp != 0x7C) //check it is a V5 BMP DIB Header
391     {
392         print_dbg("\n\rBMP Parse: DIB Header not V5;");
393         return;
394     }
395     image->Width= ReadBigEndian(Buffer, DIB_V5_WIDTH_OFFSET, 4);
396     image->Height = ReadBigEndian(Buffer, DIB_V5_HEIGHT_OFFSET, 4);
397     BitCount = ReadBigEndian(Buffer, DIB_V5_BITCOUNT_OFFSET, 2);
398     ImageSize = ReadBigEndian(Buffer, DIB_V5_IMAGESIZE_OFFSET, 4);
399     print_dbg("\n\rBitmap Width = ");
400     print_dbg_ulong(image->Width);
401     print_dbg("\n\rBitmap Height = ");
402     print_dbg_ulong(image->Height);
403     print_dbg("\n\rBitmap File Size = ");
404     print_dbg_ulong(FileSize);
405     print_dbg("\n\rBitmap Offset to Array = ");
406     print_dbg_ulong(OffsetToArray);
407     print_dbg("\n\rBitmap Image Bitcount = ");
408     print_dbg_ulong(BitCount);
409     print_dbg("\n\rBitmap Image Size = ");
410     print_dbg_ulong(ImageSize);

412     file_seek(OffsetToArray, FS_SEEK_SET);
413     j = 0;
414     image->ImagePtr = mspace_malloc(sdram_msp, image->Height * image->Width);
415     for(i = 0; i < ImageSize; i += 2)
416     {
417         image->ImagePtr[j++] = (file_getc()<<8) | (file_getc());
418     }
419     file_close();
420     nav_filelist_reset();
421     return;
422 }
```

D.1.1.16 TWI.c

..../Code/The_Columbus/ColumbusTest/src/CustomDevices/TWI.c

```

1  /*
2   * TWI.c
3   *
4   * Created: 27/02/2013 10:51:19
5   * Author: hslovett
6   */
7
8 #include "CustomDevices/CustomDevices.h"
9 #include <asf.h>
10
11 #define TARGET_ADDRESS 0x0          //! Target's TWI address
```

```
12 #define TARGET_ADDR_LGT      3           //! Internal Address length
13 #define VIRTUALMEM_ADDR     0x123456    //! Internal Address
14 #define TWIM_MASTER_SPEED   50000       //! Speed of TWI

17 void twim_init (void)
18 {
19     int8_t status;
20     /**
21     * \internal
22     * PIN 2 & 3 in Header J24 can be used in EVK1104
23     * PIN 1 & 2 in Header J44 can be used in UC3C_EK
24     * \endinternal
25     */
26     const gpio_map_t TWIM_GPIO_MAP = {
27     {AVR32_TWIMSO_TWCK_0_0_PIN, AVR32_TWIMSO_TWCK_0_0_FUNCTION},
28     {AVR32_TWIMSO_TWD_0_0_PIN, AVR32_TWIMSO_TWD_0_0_FUNCTION}
29 };

31 // Set TWIM options
32 const twi_options_t TWIM_OPTIONS = {
33     .pba_hz = FOSC0,
34     .speed = TWIM_MASTER_SPEED,
35     .chip = TARGET_ADDRESS,
36     .smbus = false,
37 };
38 // TWIM gpio pins configuration
39 gpio_enable_module (TWIM_GPIO_MAP,
40     sizeof (TWIM_GPIO_MAP) / sizeof (TWIM_GPIO_MAP[0]));
42 // Initialize as master.
43 status = twim_master_init (&AVR32_TWIMO, &TWIM_OPTIONS);

45 }
```


Appendix E

PCB Design

E.1 PCB Top Side

E.2 PCB Bottom Side

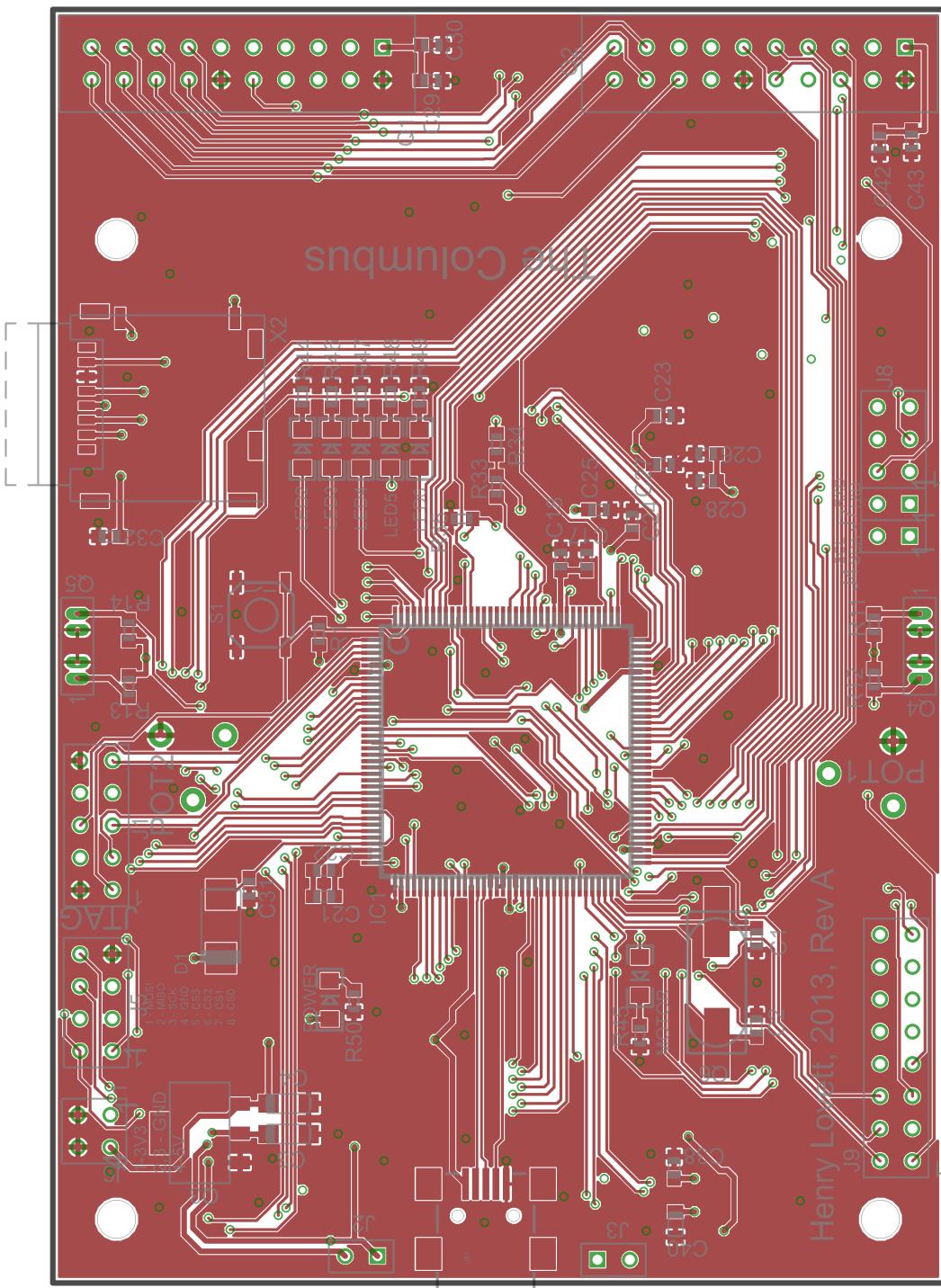


Figure E.1: The Top side of the CAD Design of the PCB

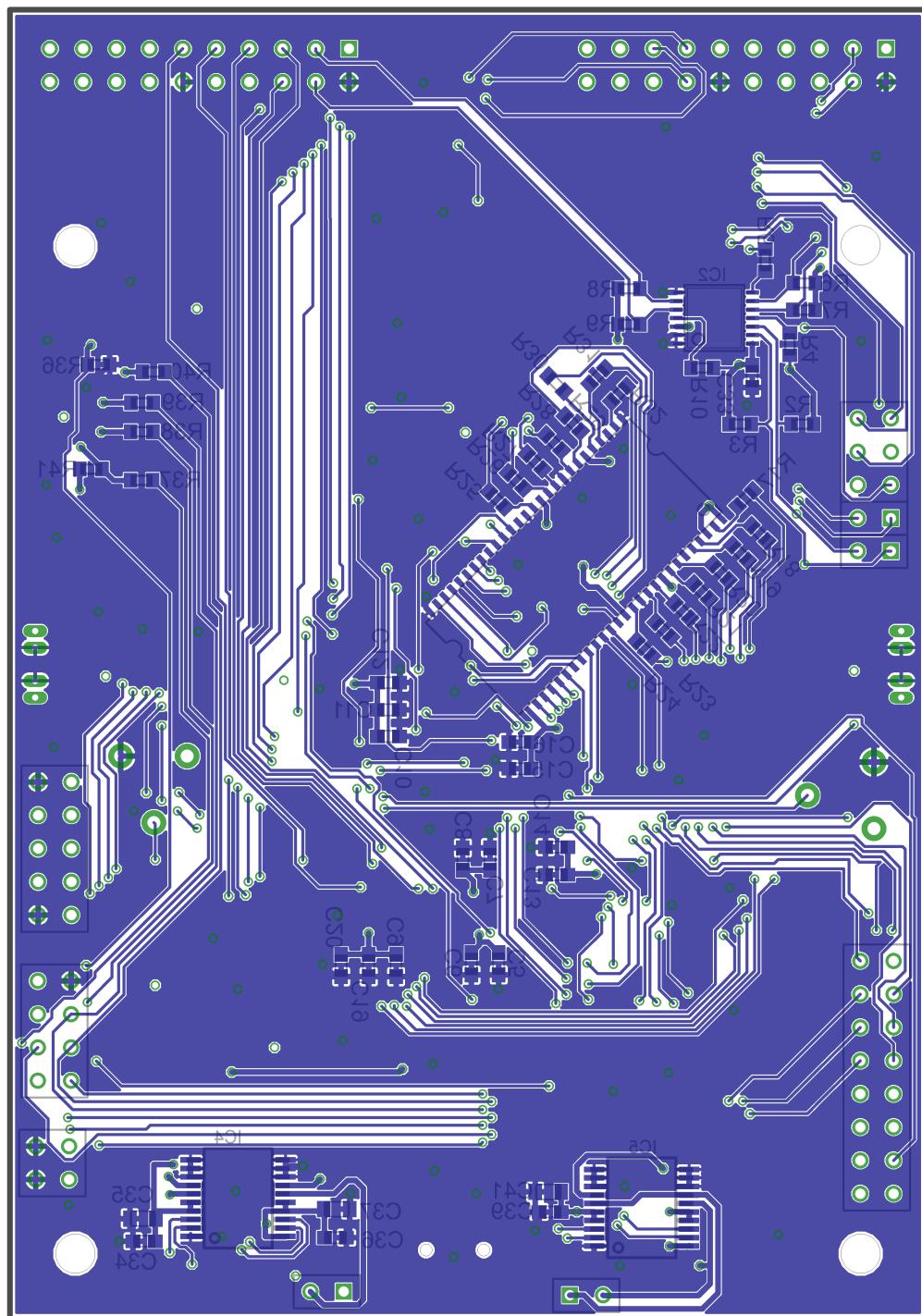


Figure E.2: The Bottom side of the CAD Design of the PCB

References

- Atmel Corporation. *AVR311: TWI Slave*, 2007.
- Atmel Corporation. [Uc3c-ek rev 2 schematic](#), 2009.
- Atmel Corporation. *AT32UC3C0512C Datasheet*, 2012a.
- Atmel Corporation. *ATMega644P Datasheet*, 2012b.
- Atmel Corporation. [Atmel software framework](#), 2012c.
- Atmel Corporation. *ATXMega256A3BU Datasheet*, 2012d.
- PCB Cart. [Pcb manufacturer](#), 2013.
- Electronic Lives Manufacturing. [Fatfs - generic fat file system module](#), 2012.
- Farnell. [Farnell online store](#), 2012.
- Wayne Fulton. [Image file formats - jpg, tif, png, gif. which to use?](#), 2010.
- I. Haller and S. Nedevschi. Design of interpolation functions for subpixel-accuracy stereo-vision systems. *Image Processing, IEEE Transactions on*, 21(2):889–898, 2012.
- Rostam Affendi Hamzah, Sani Irwan Md Salim, and Hasrul Nisham Rosly. An effective distance detection of obstacles in stereo vision application. *Canadian Journal on Electrical and Electronics Engineering*, 1(3):49–53, 2010.
- Jin Liu and Xiaofeng Lin. Equalization in high-speed communication systems. *Circuits and Systems Magazine, IEEE*, 4(2):4–17, 2004.
- Jernej Mrovlje and Damir Vrančić. Distance measuring based on stereoscopic pictures. Technical report, University of Ljubljana, 2008.
- OmniVision. *OmniVision Serial Camera Control Bus (SCCB) Functional Specification*, 2007.

- Philips. *UM10204*, 20012.
- Phillips. *PCA9542A : 2-channel I2C-bus multiplexer and interrupt logic*, 2009.
- Texas Instruments. *Stellaris LM3S9B96 Datasheet*, 2012.
- Edwin Tjandranegara. *Distance estimation algorithm for stereo pair images*, 2005.
- D.M. Tsai and C.T. Lin. *Fast normalized cross correlation for defect detection*, 2003.
- Vishay Semiconductors. *TCRT1000, TCRT1010 Datasheet*, 2012.
- F. Zhao, Q. Huang, and W. Gao. Image matching by normalized cross-correlation. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 2, pages II–II. IEEE, 2006.