

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

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

**Two Dimensional Stereoscopic Mapping Robot**

by

**Henry S. Lovett**

A project report for a Part Three Project for the award of BEng Electronic  
Engineering

December 7, 2012



Turn off iNotes!

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL AND APPLIED SCIENCES  
Electronics and Computer Science

TWO DIMENSIONAL STEREOSCOPIC MAPPING ROBOT

by Henry S. Lovett

This paper describes the research, design, build and testing of a stereoscopic mapping robot. Two cameras are used to locate objects and calculate distances to solid items. This allows an occupancy map to be built up of the unknown area. The object detection is done by a normalised cross correlation and uses derived equations to find distances. The camera used is an OV7670 by OmniVision delivering QVGA sized images. The end product will be a mobile wheeled robot that explores an unknown area. Multiple modules of the design have been developed and tested, all aspects will be brought together onto an AVR32 for the final product.



# Contents

<b>Nomenclature</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Research</b>	<b>3</b>
2.1 Hardware Research . . . . .	3
2.1.1 Microcontrollers . . . . .	3
2.2 Firmware . . . . .	5
2.2.1 Camera . . . . .	5
2.2.2 Atmel Software Framework . . . . .	5
<b>3 Initial Hardware and Firmware Development</b>	<b>7</b>
3.1 Camera . . . . .	7
3.1.1 Single Camera Operation . . . . .	7
3.1.2 Dual Camera Operation . . . . .	9
3.2 SD Card . . . . .	10
3.2.1 Storing Images . . . . .	10
3.2.2 User Interface . . . . .	11
3.3 Circuit and PCB Development . . . . .	12
3.3.1 Il Matto Development . . . . .	12
<b>4 Investigation into Vision Algorithms</b>	<b>13</b>
4.1 Comparison . . . . .	13
4.1.1 Sum of Absolute Differences . . . . .	14
4.1.2 Sum of Squared Differences . . . . .	14
4.1.3 NCC . . . . .	14
4.1.4 Comparison . . . . .	15
4.1.5 Conclusion . . . . .	15
4.2 Range Finding . . . . .	17
4.2.1 Derivations . . . . .	17
4.2.1.1 Object is between the Cameras . . . . .	18
4.2.1.2 Object is to the same side in each camera . . . . .	19
4.2.1.3 Object is in front of a camera . . . . .	20
4.2.1.4 Summary . . . . .	20
<b>5 Conclusions and Further Work</b>	<b>23</b>
<b>A Circuit Diagrams</b>	<b>25</b>

<b>B Bitmap File Format</b>	<b>27</b>
B.1 Bitmap File Format . . . . .	27
<b>C Source Code</b>	<b>31</b>
C.1 C Code for AVR . . . . .	31
C.1.1 Dual Camera Operation . . . . .	31
C.1.1.1 main.c . . . . .	31
C.1.1.2 Bitmap.h . . . . .	35
C.1.1.3 Bitmap.c . . . . .	35
C.1.1.4 Config.h . . . . .	37
C.1.1.5 Config.c . . . . .	38
C.1.1.6 DualCameras.h . . . . .	39
C.1.1.7 DualCameras.c . . . . .	44
C.1.1.8 PCA9542A.h . . . . .	52
C.1.1.9 PCA9542A.c . . . . .	53
C.1.1.10 TWI_Master.h . . . . .	54
C.1.1.11 TWI_Master.c . . . . .	56
C.1.1.12 Usart.h . . . . .	61
C.1.1.13 Usart.c . . . . .	62
C.1.2 Dual Camera User Interface . . . . .	64
C.1.2.1 DualCamera_UI.c . . . . .	64
C.1.2.2 TWI_slave.h . . . . .	66
C.1.2.3 TWI_slave.c . . . . .	68
C.2 MATLAB Code for Image Algorithm Prototyping . . . . .	75
C.2.1 Image Matching Algorithms . . . . .	75
C.2.1.1 loadimages.m . . . . .	75
C.2.1.2 GetSubImage.m . . . . .	75
C.2.1.3 SADAll.m . . . . .	76
C.2.1.4 SSDAll.m . . . . .	77
C.2.1.5 NCC.m . . . . .	78
<b>References</b>	<b>81</b>

# List of Figures

1.1	The base of the robot will use . . . . .	2
3.1	An Example Image taken using the OV7670 and stored as a Bitmap on the SD Card . . . . .	11
3.2	Prototype of Dual Camera operation. . . . .	12
4.1	Stereoscopic Test Images from MATLAB Examples . . . . .	13
4.2	Result Graphs of Comparison Algorithms . . . . .	16
4.3	Problem 1 - Object is between the Cameras . . . . .	17
4.4	Problem 1 : Left Camera Simplified . . . . .	18
4.5	Problem 2 - Object is to the same side in both cameras . . . . .	20
4.6	Problem 3 - Object is directly in front of a camera . . . . .	21
A.1	The circuit diagram for the OV7670 breakout board . . . . .	25
A.2	The circuit diagram for Dual Cameras using the Il Matto Board . . . . .	26



# List of Tables

2.1	Comparison Table of some common microcontrollers. Data of microcontroller taken from Atmel Corporation (2012a), Atmel Corporation (2012b), Atmel Corporation (2012d) and Texas Instruments (2012). Costings from Farnell (2012) . . . . .	4
3.1	A table comparing different image formats available (Fulton (2010)) . . . . .	10
3.2	Pin Connections of the ATMega 644P for Dual Camera Operation. . . . .	11
B.1	Format of a Bitmap file with values used, to write an image from the camera to an SD Card . . . . .	27



# Nomenclature

$I^2C$	Inter-Integrated Circuit
TWI	Two Wire Interface
SCCB	Serial Camera Control Bus
SPI	Serial Peripheral Interface
kB	KiloBytes
$\varphi_0$	Field of View of the Camera
$\varphi_1, \varphi_2$	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

This report documents the design, test and build of a stereoscopic mapping robot. The end product will be a small, two wheeled robot with a roller ball, that will autonomously search and map an unknown area and build up an occupancy map of the searched area.

Stereoscopy in computer vision is the ability to calculate the locations and depths using images from two cameras that are used to triangulate and estimate distances ([Saxena et al. \(2007\)](#)). By using two cameras on the same plane and separated by a set horizontal distance, depth of the observed scene can be perceived by the system.

Cite Occupancy Map

An occupancy map is a representation of an area where a location is given a value of either *free*, *occupied* or *unknown*. Three dimensional occupancy maps can also be generated, such as the OctoMap ([Wurm et al. \(2010\)](#)). Objects can also be tracked using an occupancy map and statistics ([Fleuret et al. \(2007\)](#)).

The purpose of a mapping robot is to build a representation of the area around it. This then leads on to being able to conduct an application that requires knowledge of the area. One algorithm used to build up an occupancy map is the S.L.A.M. algorithm ([Thrun and Montemerlo \(2006\)](#)) and is used in other mapping robots([Se et al. \(2002\)](#)). Accurate mapping robots tend to use laser range finders ([Ruhnke et al. \(2011\)](#)).

Stereovision is a small section of computer vision which is widely used in many applications, including Microsofts Xbox Kinect ([Microsoft \(2012\)](#)) where stereo vision is used to locate a game player to involve them more in the game. [Mrovlje and Vrančić \(2008\)](#) uses stereovision to be able to locate the distance to a marker.

The stereovision mapping robot discussed in this report is a low cost alternative to other robots which use laser range finders or high quality cameras ([Se et al. \(2002\)](#)). The robot discussed will use the base seen in figure 1.1 and use two OmniVision OV7670 cameras delivering up to VGA format images.



Figure 1.1: The base of the robot will use

# Chapter 2

## Research

The research done for this project is split down into three sections:

1. Hardware
2. Software, broken down into:
  - (a) Firmware
  - (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 AVR.

ARM is an architecture which is developed by ARM Holdings. ARM devices come in a many varieties, ARM9, ARM7, Strong ARM, ARM Cortex etc. While ARM Holdings do not fabricate and sell the devices themselves, many companies, such as Texas Instruments, use the architecture and fabricate their own devices. For this comparison, the Stellaris by Texas Instruments will be looked at. ARM cores tend to be 32-bit and have a high clock speed. They are a RISC Harvard architecture. ARM microcontrollers have onboard support for SPI,  $I^2C$ , PWN, ADCs and have Flash, SRAM and EEPROM memory.

	ARM Stellaris	AT32UC3C0512C	XmegaA3BU	ATmega644P
Clock Speed (MHz)	80	33 or 66	32	12
DMIPS	100	91	-	20 MIPS
Package	100 LQFP or 108 BGA	64, 144TQFP	100, 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 <sup>1</sup>	2.7-5.5
Communication Interfaces	SPI, $I^2C$ , SSI, MAC, CAN, EPI, USB, USART, 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 microcontroller taken from [Atmel Corporation \(2012a\)](#), [Atmel Corporation \(2012b\)](#), [Atmel Corporation \(2012d\)](#) and [Texas Instruments \(2012\)](#). Costings from [Farnell \(2012\)](#)

Atmel have a variety of products in the microcontroller market. They range from 8-bit, low clock speed devices for the hobbyist, the ATmega and ATTiny series, an improved 8-bit variant, Xmega, and a 32-bit design, AT32UC3C. XMegas and AVR32s tend to have higher clock speeds than the ATMegas. The AVR core is also a Harvard RISC architecture, but mainly 8-bit. Atmel devices often have on board peripherals such as  $I^2C$ (called TWI on AVRs), SPI, 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 with a small amount of SRAM and Flash. All devices looked at use 3.3V supply and have basic communication protocols (SPI,  $I^2C$ ). 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 desired to have, but these can be added onto and 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, which has an onboard SD Card reader. The original code streamed video from the camera to a colour TFT screen. It 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

### 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 \(2012c\)](#). There are also many AVR application notes which provide explanations and example code for things 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

## Initial Hardware and Firmware Development

For initial development, an *Il Matto* board, designed by Steve Gunn, which has an ATMega644P, was used. The system is clocked at 16MHz and has an on-board SD card connector.

The following section is broken down into parts listed below:

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

### 3.1 Camera

The camera that was used was an OV7670 by OmniVision. It is mounted onto a break out board and connected to a AL422B FIFO Buffer. The breakout board also had all passive components and a 24MHz clock mounted. The schematic for the device can be seen in Appendix A.

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

#### 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. 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 $s$ )<sup>1</sup>. 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 contain green intensity and 11:15 represent the red intensity. This is a compact way of storing data but only allows 65536 colours. Greys can also appear to be slightly green due to an inconsistent colour ratio of the green field.

figure to show this better?

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 $s$  (ATMegas typically clocked at 12-16MHz) to be able to respond to the camera quick enough. This highlights the importance of the necessity of the 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 operation is then automatic and all the data is clocked into the buffer until the second interrupt of VSYNC where WEN is disabled. At this point, the entire frame of data is stored in the buffer.

To obtain the data from the buffer, the AVR sets output enable and manually pulses the read clock. Valid data is available on the input port chosen. All the data is then read in half a pixel at a time.

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

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

The second option then was to use extra memory connected to the microcontroller. An SD card was decided to be used as a FAT file system. This will allow data to be looked

---

<sup>1</sup> $I^2C$  , SCCB and TWI are all the same but are called differently due to Phillips owning the right to the name “ $I^2C$ ”

at by a user on a computer of image files and text log files could also be written. This is discussed in section 3.2.

Put a figure of the pin timings in?

### 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 of the same time frame.

The buffers have an output enable pin so the data bus can be shared by both cameras to the AVR. All buffer function pins are driven from pins, although a demultiplexer could be used if pins are short. 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 method 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 from one and then the other by the AVR.

A major problem now 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. Space for the code then has to be made and this code cannot be reused.

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  $I^2CMUX$  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 and simplify the operation and reduce the code size so an  $I^2CMUX$  will be used. A suitable multiplexer is the Phillips PCA9542A ([Phillips \(2009\)](#)).

Operation to read an image is identical to using one camera. 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.

	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 Image (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 \(2010\)](#))

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

An SD card was chosen to be used 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 much computational time to compress the image to obtain the correct format. To avoid compression, and thereby save processing time, Bitmap was decided to be used 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 data. Appendix B shows the make up of a Bitmap File that was used.

By writing the image in this format, the images 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 3.1 shows a photo taken by the OV7670 and stored on a SD card.



Figure 3.1: An Example Image taken using the OV7670 and stored as a Bitmap on the SD Card

	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 ATMega 644P for Dual Camera Operation.

### 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 GPIO pins remaining, an ATMega 168 was added on the  $I^2C$ bus to act as a port extender. The ATMega168 accepts a read or write, places the written data on Port D and reads in the lower nibble of Port C. When a button is pressed, this is stored in the ATMega168 until a read has been done. This is so the master doesn't miss any button presses while busy doing lengthy operations such as writing an image. The code is based on [Atmel Corporation \(2007\)](#), written for IAR Compiler. This code was altered to compile with GCC under Atmel Studio. AVR's 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 acknowledgements and address calls and deals with bus errors that can occur.

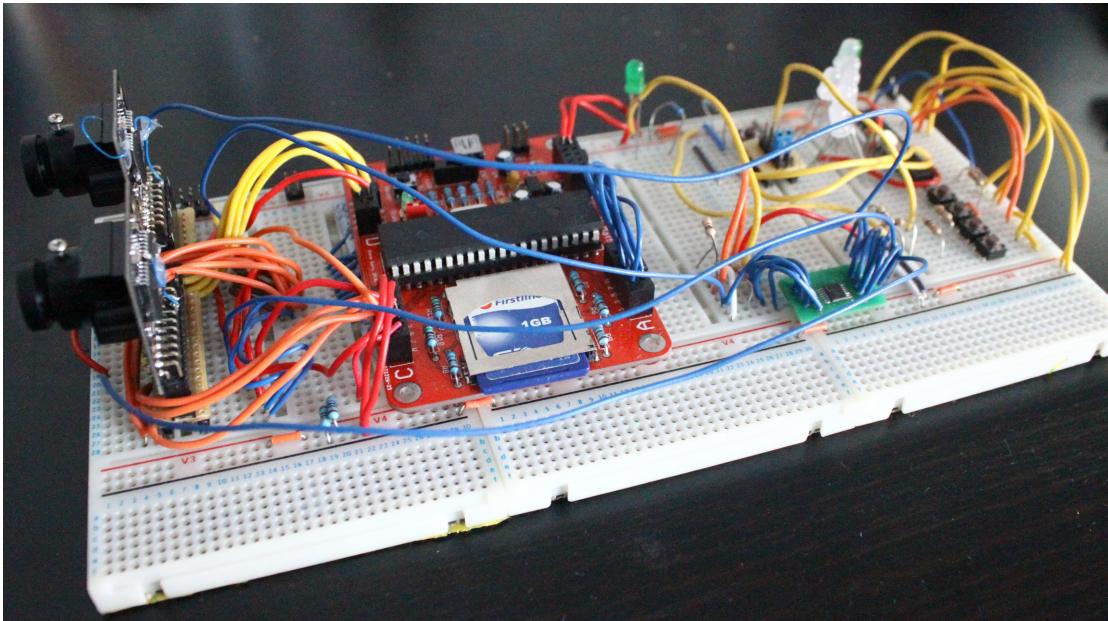


Figure 3.2: Prototype of Dual Camera operation.

### 3.3 Circuit and PCB Development

#### 3.3.1 Il Matto Development

Figure A.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 3.2.

# Chapter 4

## Investigation into Vision Algorithms

### 4.1 Comparison

find some references to back these claims up

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.) and normalised cross correlation (N.C.C.). 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 of image to compare to of  $50 \times 50$  pixels of the same part of the 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 takes each sub image and subtracts the observed sub image 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  $B$ ,

$\sigma$  is the standard deviation of the image, and

$\bar{A}$  is the average pixel value.

Find a source for this equation

NCC is very similar to cross correlation, but normalised to reduce the error if one image is brighter than the other. It is common in computer vision ([Tsai and Lin \(2003\)](#)) as cross correlation is a common operation in DSP 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 with division in it and a square root to calculate the standard deviation. These operations are rarely implemented in hardware and are time consuming to carry out. They also

require floating point registers and operations slow on a microcontroller with a small amount of floating point registers.

#### 4.1.4 Comparison

To compare these equations, a 50 by 50 image 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 of the graphs show the correct area being indentified as a match, but this also highlights the downfalls of the SAD and SSD. The figures in figure 4.2 are orientated to match the orientation of the images in figure 4.1. Each of the images is tested by attempting to match the phone from the test figure. 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 \(2012\)](#)).

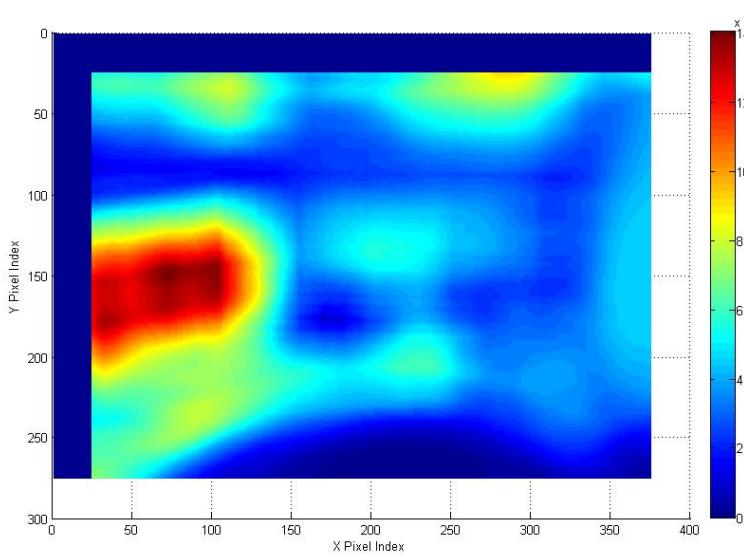
SAD results in figure 4.2(b) show large areas of matching. The actual match is at (170, 175) and a minimum does occur at this position as expected, of a value of  $5.66 \times 10^4$ . However, along the bottom of the image where a dark area occurs in the lower part of figures 4.1 below the desk, the 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 values where the match should occur is  $4.355 \times 10^5$  at location (170, 176). However, again, there is thought to be a large match correlation between the dark area under the desk where the actual lowest value of 2.768 is at (225, 274). This, again, is a false match and is a downfall of this algorithm.

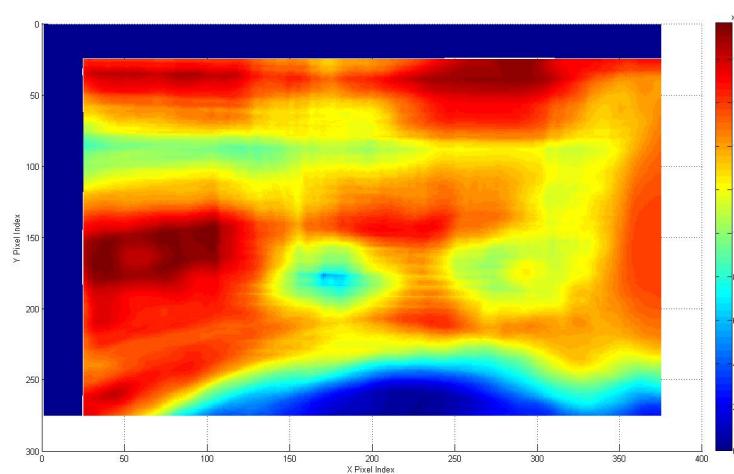
The NCC results are visible in figure 4.2(c). A match can be seen at coordinate (195, 201) with a peak value of 0.9654. The coordinate is different to the previous results because the cross correlation works over the boundary of the image creating more results. The dimensions of the image are  $300 \times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

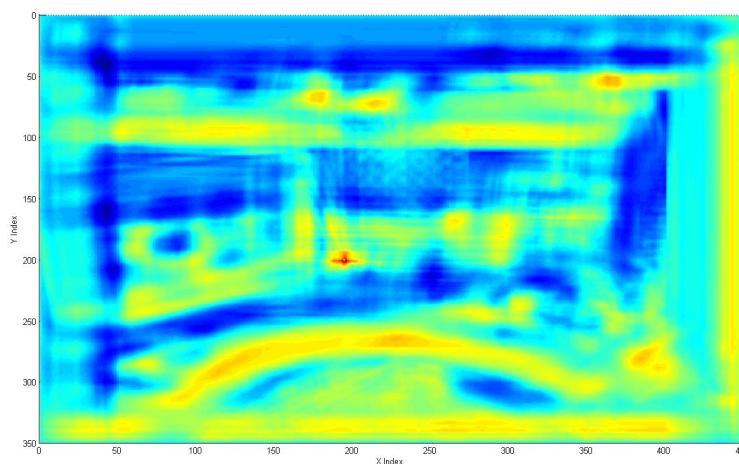
It can be seen there is a direct correlation between the complexity of the matching algorithm to the reliability of the match returned. In brightly lit, colourful environments



(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

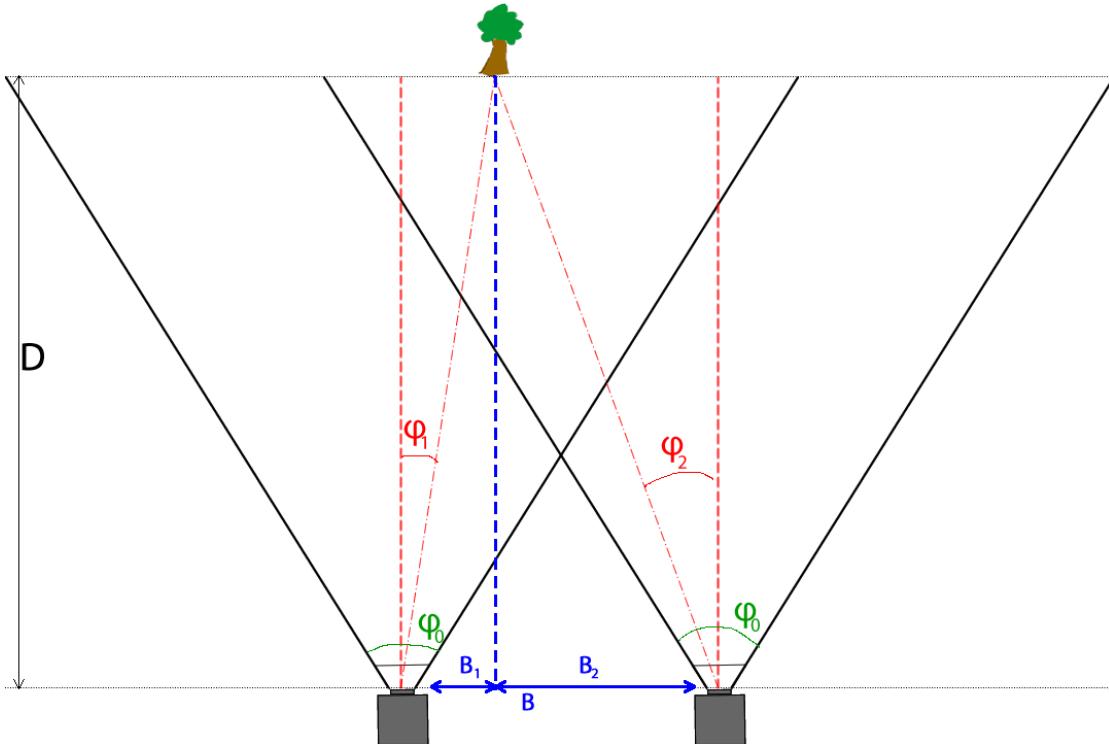


Figure 4.3: Problem 1 - Object is between the Cameras

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. There is a compromise of complexity for reliability to be made, of which reliability is more desirable. 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 difference, the range of an object can be found given some characteristics of the camera. The following is a derivation of the equations used to calculate distance.

The problem is broken down into 3

1. Object is between the cameras (Figure 4.3)
2. Object is in left or right hand sides of both images (Figure 4.5)
3. Object is directly in front of a camera (Figure 4.6)

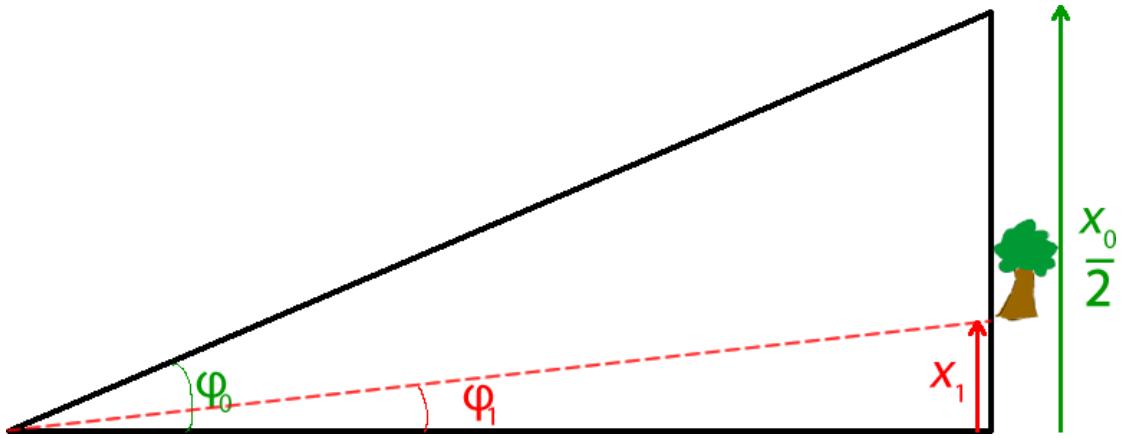


Figure 4.4: Problem 1 : Left Camera Simplified

#### 4.2.1.1 Object is between the Cameras

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

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

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

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

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

Dividing (4.7) by (4.6)

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

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

It can also be shown that for the right camera:

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

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 4.5:

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

$$D \cdot \tan(\frac{\varphi_1}{2}) = \frac{x_0}{2} \quad (4.13)$$

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

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

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  $\varphi_1$  is defined in equation (4.15) and  $\varphi_2$  is defined in equation (4.16).

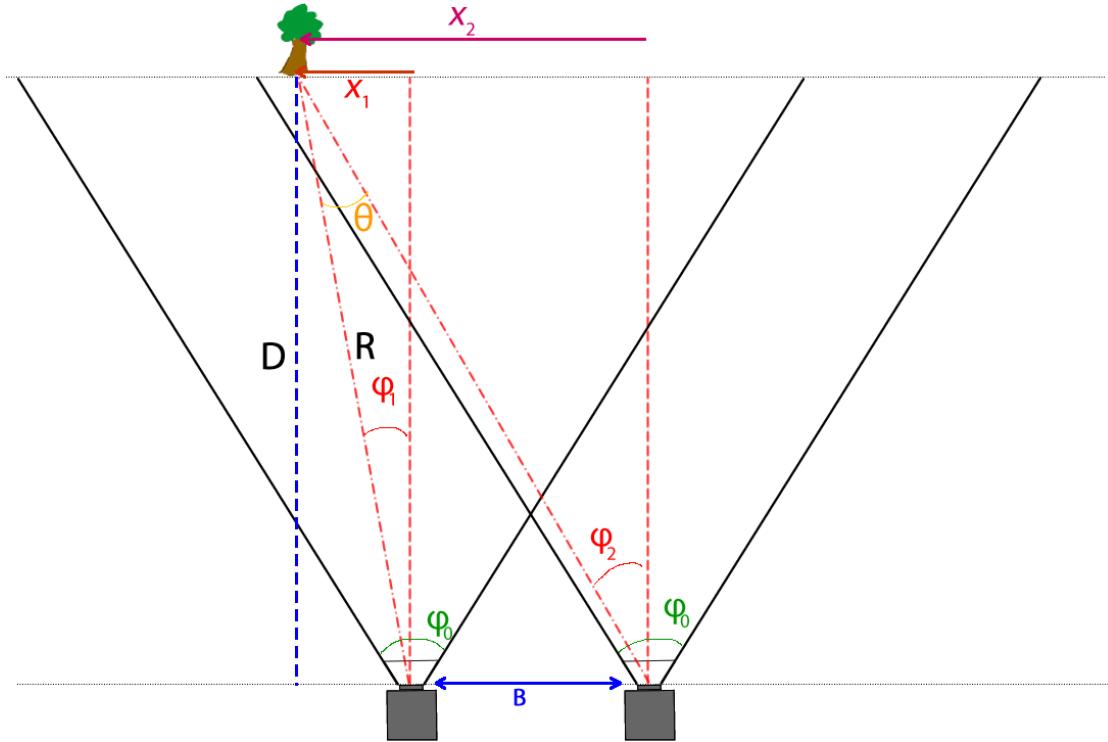


Figure 4.5: Problem 2 - Object is to the same side in both cameras

#### 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  $\varphi_2$  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:

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

Object is to the same side in both images:

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

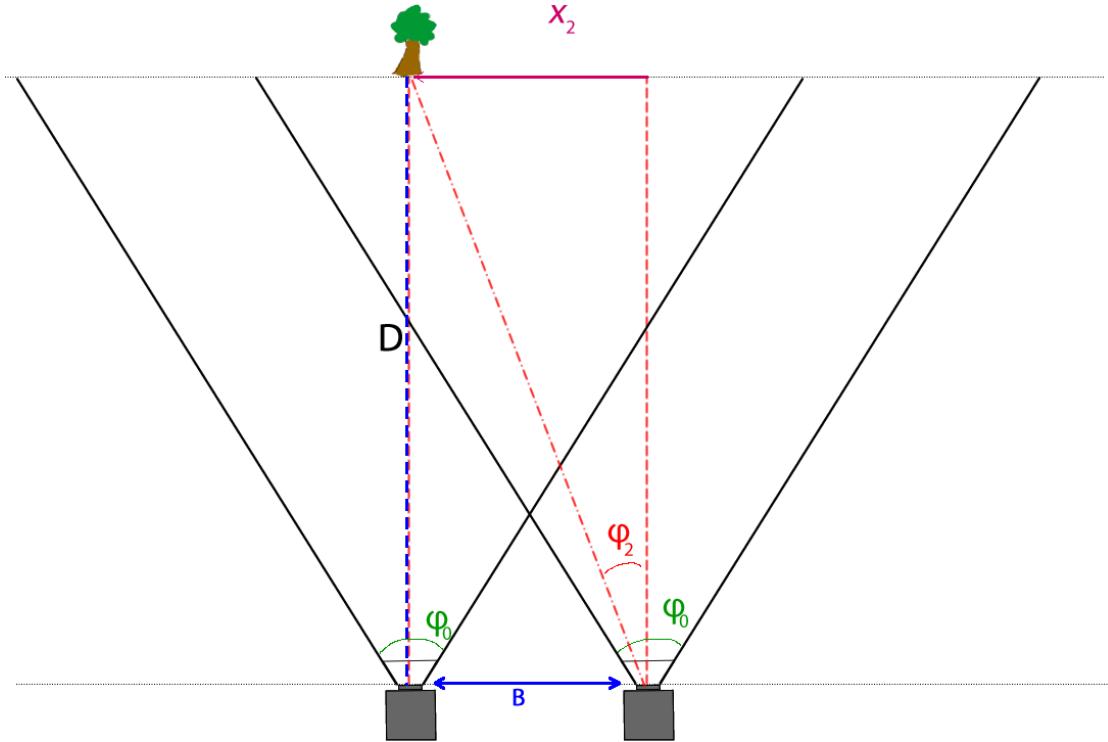


Figure 4.6: Problem 3 - Object is directly in front of a camera

Object is directly in front of a camera:

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

Where  $\varphi_1$  is defined in equation (4.15) and  $\varphi_2$  is defined in equation (4.16).

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



## Chapter 5

# Conclusions and Further Work

So far the project has completed three aspects of the whole design: image acquisition, object matching a range finding. All these aspects have been developed separately on different platforms; on an ATMega644P or in MATLAB.

A full image can be read from a camera and stored on an SD card in a FAT32 filesystem. This gives the ability of removable memory that can be viewed on a computer to see any internal logs that have been written. Images are stored in QVGA format (320 by 240 pixels) as a bitmap image.

Multiple comparison algorithms have been investigated and compared. They were then all tested using the same image. It was clear that, although at a sacrifice of computations, the normalised cross correlation is the best measure to use for reliability.

Range finding equations were then derived. These use the characteristics of the camera and the separation distance between them.

The next steps will be bringing the comparison and range equations together so as to be able to choose an object in a scene and find the range to said object. Object detection algorithms need to be looked into so that an object in the view can be detected and decided upon where it is.

Robot movement also needs developing. An accurate method to move a distance will be implemented with a proportional or PID controller.

A printed circuit board will also be designed with an AVR32 as the microcontroller and all extra hardware on the board will be included. This will then mean all modules (image acquisition, comparison, range finding etc.) can be ported to the AVR32 and the robot can then be made mobile. Further parts can then be developed such as the occupancy map and search algorithms.

All modules will be tested before being implemented. The final hardware design and development is to be completed by 15th March 2013 and the project report hand-in date is 1st May 2013.

## Appendix A

# Circuit Diagrams

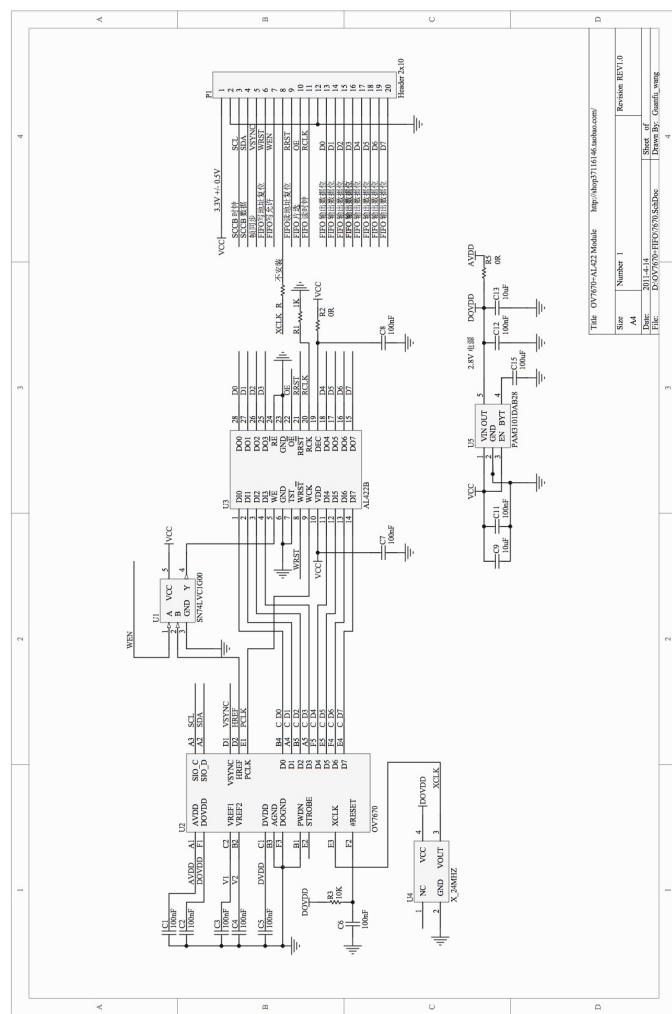


Figure A.1: The circuit diagram for the OV7670 breakout board

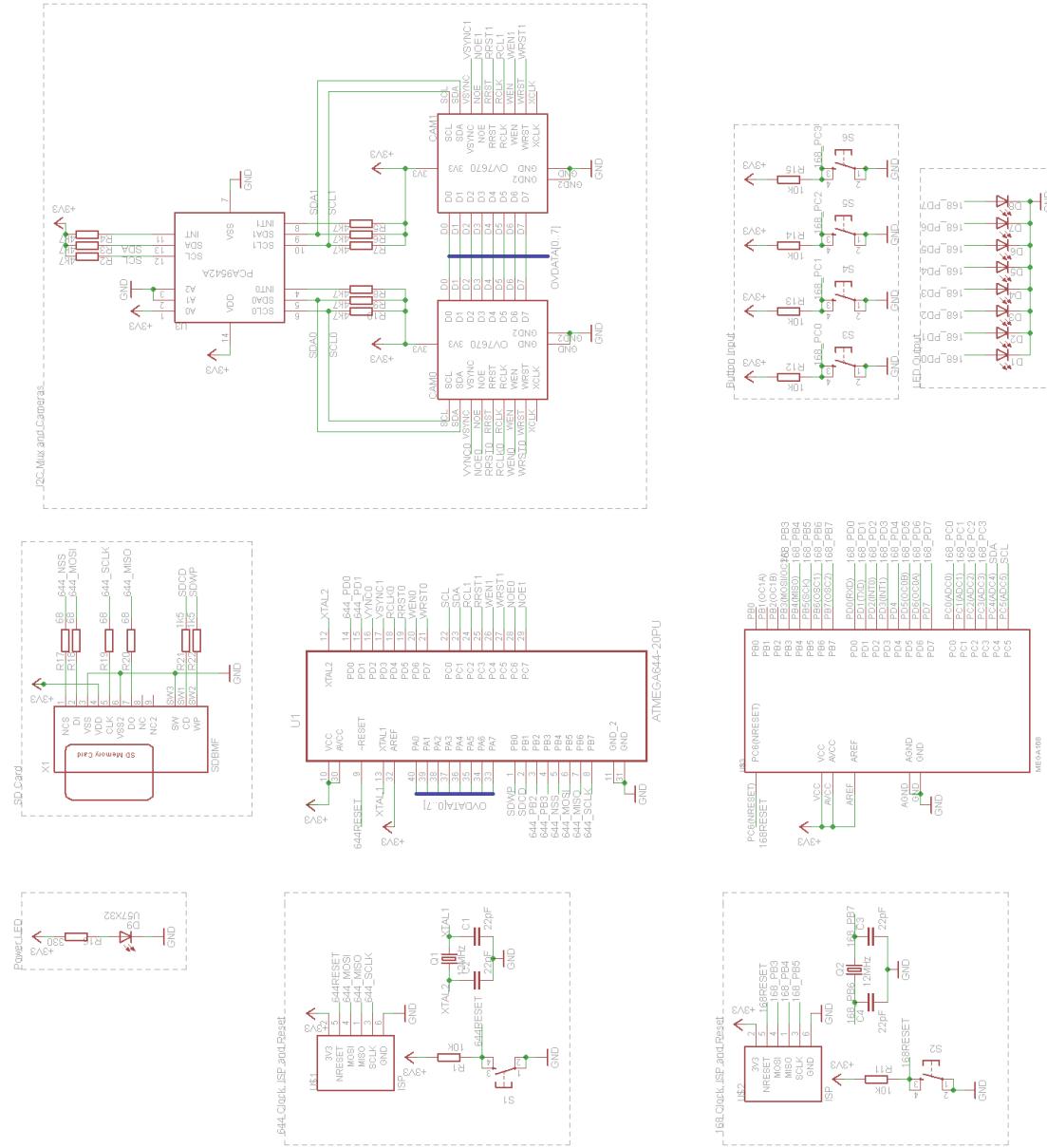


Figure A.2: The circuit diagram for Dual Cameras using the II Matto Board

## Appendix B

# Bitmap File Format

### B.1 Bitmap File Format

Table B.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 Bitamp Image	2	424D
	File Size	Size of the whole file including headers	4	36580200 (153654) <sup>1</sup>
	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 Independant Bitmap) Header	Size	Size of the DIB Header (dictates the version)	4	7C000000
	Width	Width of the image (320 pixels)	4	40010000
	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

Continued on next page

---

<sup>1</sup>This is different to the 225kB said in Table 3.1 due to omitting many optional fields

Table B.1 – continued from previous page

Section	Field	Description	Size (Bytes)	Value (hex)
	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	Colour Space of the DIB	4	01 00 00 00
	Type			
	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
	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

Continued on next page

Table B.1 – continued from previous page

Section	Field	Description	Size (Bytes)	Value (hex)
Image Format	Data	Each field contains all the pixel data		
Pix[0, h-1]	Pix[1, h-1]	Padding ... ⋮	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 C

## Source Code

### C.1 C Code for AVR

#### C.1.1 Dual Camera Operation

##### C.1.1.1 main.c

..../Code/DualOV7670/main.c

```
1  /*
2   * DualOV7670.c
3   *
4   * Created: 09/11/2012 11:43:13
5   * Author: hl13g10
6   */
7  #include "Config.h"

11 //static FILE mystdout = FDEV_SETUP_STREAM(File_Write_Printf, NULL,
12 //                                         _FDEV_SETUP_WRITE);
13 //FatFS Variables
14 FILINFO Finfo;
15 FATFS Fatfs[_VOLUMES];      /* File system object for each logical drive */
16 //FIL Files[2];           /* File object */
17 uint8_t StatusReg;
18 // char Line[100];          /* Console input buffer */
19 //char Buff[100];           /* Working buffer */
20 char ImageRName[20];
21 char ImageLName[20];
22 #define STATUS_OKAY        0x01
23 #define STATUS_SDOokay     0x02
24 #define STATUS_CAM00okay    0x04
25 #define STATUS_CAM10okay    0x08
26 #define STATUS_READY        0x10
27 #define STATUS_CAPTURING    0x20
28 #define STATUS_Exit_Bad     0x80
```

```

29 #define Button_Capture    0
30 #define Button_Exit       3
31 unsigned char UI_LEDs(uint8_t LED)
32 {
33     unsigned char mesbuf[TWI_BUFFER_SIZE];
34     mesbuf[0] = (0x15 << TWI_ADR_BITS) | (FALSE << TWI_READ_BIT);
35     mesbuf[1] = 0x10;
36     mesbuf[2] = LED;
37     TWI_Start_Transceiver_With_Data(mesbuf, 3);
38     while(TWI_Transceiver_Busy()) ;
39     return TWI_StatusReg.lastTransOK;
40 }
41 unsigned char UI.Buttons()
42 {
43     unsigned char messageBuf[TWI_BUFFER_SIZE]; //Initialise a buffer
44     messageBuf[0] = (0x15<<TWI_ADR_BITS) | (FALSE<<TWI_READ_BIT); // The first
        byte must always consist of General Call code or the TWI slave address.
45     messageBuf[1] = 0x20;           // The first byte is used for the command
46     TWI_Start_Transceiver_With_Data( messageBuf, 2 );
47     _delay_us(250);
48     // Request/collect the data from the Slave
49     messageBuf[0] = (0x15<<TWI_ADR_BITS) | (TRUE<<TWI_READ_BIT); // The first
        byte must always consist of General Call code or the TWI slave address.
50     TWI_Start_Transceiver_With_Data( messageBuf, 2 );

52     // Get the received data from the transceiver buffer
53     TWI_Get_Data_From_Transceiver( messageBuf, 2 );
54     return messageBuf[1];
55 }
56 ISR(TIMERO_COMPA_vect)
57 {
58     disk_timerproc(); /* Drive timer procedure of low level disk I/O module */
59     // if(!TWI_StatusReg.lastTransOK) //if the last TWI transmission failed,
        reset the protocol
60     //    TWI_Start_Transceiver();
61     //    if(!TWI_Transceiver_Busy())
62     //        UI_LEDs(StatusReg);
63 }
64 int main(void)
65 {
66     unsigned long int a = 0;
67     uint8_t b = 0;
68     HRESULT fr;
69     uint8_t PhotoCount = 0;
70     TWI_Master_Initiate();
71     IO_Init();
72     sei();
73     PCA9542A_Init();

75     StatusReg = STATUS_OKAY;
76     UI_LEDs(StatusReg);

79     fr = f_mount(0, &Fatfs[0]);
80     if(fr != FR_OK)
81     {
82         StatusReg |= (STATUS_Exit_Bad);
83         StatusReg &= ~(STATUS_OKAY);
84         UI_LEDs(StatusReg);

```

```

85     return 0;
86 }
87 else
88     StatusReg |= STATUS_SD0kay;
89 UI_LEDs(StatusReg);

91 fr = f_open(&Files[0], "/log.txt", FA_WRITE|FA_CREATE_ALWAYS);
92 if(fr != FR_OK)
93 {
94     StatusReg |= (STATUS_Exit_Bad);
95     StatusReg &= ~(1<<STATUS_SD0kay) | (1<<STATUS_OKAY);
96     UI_LEDs(StatusReg);
97     return 0;
98 }
99 UI_LEDs(StatusReg);

101 f_close(&Files[0]);
102 f_open(&Files[0], "/log.txt", FA_WRITE);
103 //stdout = &mystdout;
104 b = MCUSR;
105 MCUSR = 0;
106 f_write(&Files[0],"Il Matto Dual Camera\n", sizeof("Il Matto Dual Camera\n")
107         , &a);

108 /*f_write(&Files[0], "System Startup Complete.\n", 26, &a);*/

110 PCA9542A_SetChannel(CH1);
111 b = OV7670_init();
112 if(b == 0)
113     StatusReg |= STATUS_CAM10kay;
114 PCA9542A_SetChannel(NO_SELECT);
115 UI_LEDs(StatusReg);
116 sprintf(Buff, "OV7670_1 Initialise result : %d\n", b);
117 f_write(&Files[0], &Buff, 33, &a);

119 PCA9542A_SetChannel(CH0);
120 b = OV7670_init();
121 if(b == 0)
122     StatusReg |= STATUS_CAM00kay;
123 UI_LEDs(StatusReg);
124 PCA9542A_SetChannel(NO_SELECT);
125 sprintf(Buff, "OV7670_0 Initialise result : %d\n", b);
126 f_write(&Files[0], &Buff, 33, &a);
127 FIFO_init();

129 //f_close(&Files[0]);
130 StatusReg |= STATUS_READY;
131 UI_LEDs(StatusReg);
132 _delay_ms(250);
133 uint8_t Input;

135 while(1)
136 {
137     Input = (~UI.Buttons() & 0x0F); //Data is received negative
138     if(Input)//if a button has been pressed
139     {
140         _delay_ms(250);
141         sprintf(Buff, "Button Received : %d\n", Input);
142         f_write(&Files[0], Buff, 21, &a);

```

```

144     StatusReg&= ~(STATUS_READY); //no longer ready
145
146     switch(Input)
147     {
148         case (1<<Button_Capture):
149             StatusReg |= STATUS_CAPTURING;
150             UI_LEDs(StatusReg);
151             //Reset both buffers
152             FIFO_Reset(0);
153             FIFO_Reset(1);
154             f_write(&Files[0], "Capturing Images...\n", 20, &a);
155             LoadImagesToBuffer(); //Load both images
156
157             //Create Bitmap for image 0
158             //PSTR("Image_r.bmp");
159
160             f_open(&Files[1], "Image_r.bmp", FA_CREATE_ALWAYS | FA_WRITE);
161             f_write(&Files[0], "Created image0 file.\n", 22, &a);
162             f_lseek(&Files[1], BMPFileSize);
163             f_lseek(&Files[1], 0);
164             f_close(&Files[1]);
165             f_write(&Files[0], "Extended image0 file.\n", 22, &a);
166
167             //Create Bitmap for image 1
168             f_open(&Files[1], "image_l.bmp", FA_CREATE_ALWAYS | FA_WRITE);
169             f_write(&Files[0], "Created image1 file.\n", 22, &a);
170             f_lseek(&Files[1], BMPFileSize);
171             f_lseek(&Files[1], 0);
172             f_close(&Files[1]);
173             f_write(&Files[0], "Extended image1 file.\n", 22, &a);
174             //Get image 0
175             f_open(&Files[1], "Image_r.bmp", FA_WRITE);
176             while (2 == GetImageIfAvailable(&Files[1], 0)) ;
177             f_close(&Files[1]);
178             f_write(&Files[0], "Captured image0.\n", 17, &a);
179             //get image 1
180             f_open(&Files[1], "image_l.bmp", FA_WRITE);
181             while (2 == GetImageIfAvailable(&Files[1], 1)) ;
182             f_close(&Files[1]);
183             f_write(&Files[0], "Captured image1.\n", 17, &a);
184             StatusReg |= STATUS_READY;
185             StatusReg &= ~STATUS_CAPTURING;
186             UI_LEDs(StatusReg);
187             break; //break case(1<<ButtonCapture)
188
189         case (1<<Button_Exit):
190             f_write(&Files[0], "\nSystem Exiting...\n", 19, &a);
191             f_close(&Files[0]); //close log file
192
193             StatusReg = 0x41;
194             UI_LEDs(StatusReg);
195             return 0; //Q
196     } //End switch
197 } //End if(Input)
198 else
199 {
200     StatusReg |= STATUS_READY;
201     UI_LEDs(StatusReg);

```

```
202     _delay_ms(250); //wait
203 } //end else (Input)
204 } //End while(1)
205 } //End Main
```

### C.1.1.2 Bitmap.h

..../Code/DualOV7670/Bitmap.h

```
1  /*
2   * Bitmap.h
3   *
4   * Created: 29/10/2012 11:31:11
5   * Author: hslovett
6   */
7
8
9 #ifndef BITMAP_H_
10 #define BITMAP_H_
11
12 #define BMPHEADERSIZE 14
13 #define DIBHEADERSIZE 124 //v5
14 #define FILESIZE 153738
15
16 #include "ff.h"
17 #include "Config.h"
18
19
20 FRESULT WriteBMPHeader(FIL *File)
21 FRESULT WriteDIBHeader(FIL *File)
22
23
24 #endif /* BITMAP_H_ */
```

### C.1.1.3 Bitmap.c

..../Code/DualOV7670/Bitmap.c

```

14     0x03, 0x00, 0x00, 0x00, //Compression
15     0x00, 0x86, 0x25, 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}; //Reserved

42 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 };

50
51 FRESULT WriteBMPHeader(FIL *File)
52 {
53     uint32_t p;
54     FRESULT f;

55     f_lseek(File, 0);
56     f = f_write(File, BMPHeader, BMPHEADERSIZE, &p);

58     return f;
59 }

61 FRESULT WriteDIBHeader(FIL *File)
62 {
63     uint32_t p;
64     FRESULT f;

66     f_lseek(File, BMPHEADERSIZE); //place just after the bitmap header
67     f = f_write(File, DIBHead, DIBHEADERSIZE, &p);
68     return f;
69 }

```

#### C.1.1.4 Config.h

..../Code/DualOV7670/Config.h

```

55 #define TWI_SCL      PC0
56 #define TWI_SDA      PC1
57 #define FIFO_RCLK_1   PC2
58 #define FIFO_nRRST_1  PC3
59 #define FIFO_WEN_1    PC4
60 #define FIFO_WRST_1   PC5
61 #define FIFO_nOE_0    PC6
62 #define FIFO_nOE_1    PC7
63 ///////////////////////////////////////////////////////////////////
64 //  Port D
65 ///////////////////////////////////////////////////////////////////
66 #define USART0_RX    PD0
67 #define USART0_TX    PD1
68 #define OV7670_VSYNC_0 PD2 //MUST BE AN INTERRUPT PIN
69 #define OV7670_VSYNC_1 PD3 //MUST BE AN INTERRUPT PIN
70 #define FIFO_RCLK_0   PD4
71 #define FIFO_nRRST_0  PD5
72 #define FIFO_WEN_0    PD6
73 #define FIFO_WRST_0   PD7

77 #endif /* CONFIG_H_ */

```

### C.1.1.5 Config.c

..../Code/DualOV7670/Config.c

```

1  /*
2  * Config.c
3  *
4  * Contains Global Methods and initialisations
5  *
6  * Created: 25/10/2012 21:59:06
7  * Author: hslovett
8  */

10 #include "Config.h"
11 #include <avr/io.h>
12 void IO_Init(void)
13 {
14     //initialise timer 0 to interrupt every 10 ms
15     TIMSK0 |= (1 << OCIE0A);
16     TCCROA |= (1 << WGM01);
17     OCROA = 117; //10ms interrupt at 12MHz
18     TCCROB |= (1 << CS02) | (1 << CS00);

21     DDRA = 0x00;
22     //PORTB = 0xBF;
23     DDRC = 0xFC;
24     DDRD = 0xF2;

28     //set int0 and int1 to trigger on falling edge

```

```

29     EIMSK = (1 << INTO) | (1 << INT1);           //Enable INTO and INT1
30     EICRA = (1 << ISC01) | (1 << ISC11);           //Trigger INTO and INT1 on the
31         falling edge
32 }
```

### C.1.1.6 DualCameras.h

..../Code/DualOV7670/DualCameras.h

```

1  /*
2   * DualCameras.h
3   *
4   * Created: 10/11/2012 15:19:52
5   * Author: hslovett
6   */
7
8 #ifndef DUALCAMERAS_H_
9 #define DUALCAMERAS_H_
10
11 #include "Config.h"
12
13 ///////////////////////////////////////////////////////////////////
14 // Constants
15 ///////////////////////////////////////////////////////////////////
16 #define HEIGHT          240
17 #define WIDTH           320
18 #define PIXELSIZE        2
19 #define SETTINGS_LENGTH 167
20 #define OV7670_ADDR      0x21
21
22 ///////////////////////////////////////////////////////////////////
23 // Globals
24 ///////////////////////////////////////////////////////////////////
25 const char default_settings[SETTINGS_LENGTH][2];
26 volatile uint8_t VSYNC_0_Count;
27 volatile uint8_t VSYNC_1_Count;
28
29 // Methods
30 ///////////////////////////////////////////////////////////////////
31 unsigned char OV7670_init(void);           //Initialises Camera
32 void FIFO_init(void);                    //Initialises Buffer
33 uint8_t GetImageIfAvailable(FIL *File, uint8_t CameraID);
34 void LoadImagesToBuffer(void);
35 unsigned char rdOV7670Reg(unsigned char regID, unsigned char *regData);
36 unsigned char OV7670_SCCB_init(void);
37 void FIFO_Reset(uint8_t CameraID);
38
39 // Pins & Macros
40 ///////////////////////////////////////////////////////////////////
41 #define FIFO_RCLK_1    PC2
42 #define FIFO_nRRST_1   PC3
43 #define FIFO_WEN_1     PC4
44 #define FIFO_WRST_1    PC5
45 #define FIFO_nOE_0     PC6
46 #define FIFO_nOE_1     PC7
```

```

48 #define FIFO_RCLK_1_SET { PORTC |= (1 << FIFO_RCLK_1); }
49 #define FIFO_RCLK_1_CLR { PORTC &= ~(1 << FIFO_RCLK_1); }
50 #define FIFO_nRRST_1_SET { PORTC |= (1 << FIFO_nRRST_1); }
51 #define FIFO_nRRST_1_CLR { PORTC &= ~(1 << FIFO_nRRST_1); }
52 #define FIFO_WEN_1_SET { PORTC |= (1 << FIFO_WEN_1); }
53 #define FIFO_WEN_1_CLR { PORTC &= ~(1 << FIFO_WEN_1); }
54 #define FIFO_WRST_1_SET { PORTC |= (1 << FIFO_WRST_1); }
55 #define FIFO_WRST_1_CLR { PORTC &= ~(1 << FIFO_WRST_1); }
56 #define FIFO_nOE_0_SET { PORTC |= (1 << FIFO_nOE_0); }
57 #define FIFO_nOE_0_CLR { PORTC &= ~(1 << FIFO_nOE_0); }
58 #define FIFO_nOE_1_SET { PORTC |= (1 << FIFO_nOE_1); }
59 #define FIFO_nOE_1_CLR { PORTC &= ~(1 << FIFO_nOE_1); }

62 #define FIFO_RCLK_0 PD4
63 #define FIFO_nRRST_0 PD5
64 #define FIFO_WEN_0 PD6
65 #define FIFO_WRST_0 PD7

67 #define FIFO_RCLK_0_SET { PORTD |= (1 << FIFO_RCLK_0); }
68 #define FIFO_RCLK_0_CLR { PORTD &= ~(1 << FIFO_RCLK_0); }
69 #define FIFO_nRRST_0_SET { PORTD |= (1 << FIFO_nRRST_0); }
70 #define FIFO_nRRST_0_CLR { PORTD &= ~(1 << FIFO_nRRST_0); }
71 #define FIFO_WEN_0_SET { PORTD |= (1 << FIFO_WEN_0); }
72 #define FIFO_WEN_0_CLR { PORTD &= ~(1 << FIFO_WEN_0); }
73 #define FIFO_WRST_0_SET { PORTD |= (1 << FIFO_WRST_0); }
74 #define FIFO_WRST_0_CLR { PORTD &= ~(1 << FIFO_WRST_0); }

76 ///////////////////////////////////////////////////////////////////
77 //Camera Register Address definitions
78 ///////////////////////////////////////////////////////////////////
79 #define OV_GAIN 0x00 //Gain Control Setting - ACG[7:0]
80 #define OV_BLUE 0x01 //Blue Channel Gain
81 #define OV_RED 0x02 //Red Channel Gain
82 #define OV_VREF 0x03 //Vertical Frame Control & ACG[9:8]
83 #define OV_COM1 0x04 //CCIR656 enable, AEC low bits (AECHH, AECH)
84 #define OV_BAVE 0x05 //U/B Average level - AUTO UPDATED
85 #define OV_GbAVE 0x06 //Y/Gb Average Level - AUTO UPDATED
86 #define OV_AECHH 0x07 //Exposure value [15:10] (AECH, COM1)
87 #define OV_RAVE 0x08 //V/R Average level - AUTO UPDATED
88 #define OV_COM2 0x09 //Soft Sleep, Output drive capability
89 #define OV_PID 0x0A //Product ID MSB Read only
90 #define OV_VER 0x0B //Product ID LSB Read Only
91 #define OV_COM3 0x0C //Output data MSB/LSB swap + other stuff
92 #define OV_COM4 0x0D //Average values - MUST BE SAME AS COM17
93 #define OV_COM5 0x0E //RESERVED
94 #define OV_COM6 0x0F //COM6
95 #define OV_AECH 0x10 //Exposure value [9:2] (see AECHH, COM1)
96 #define OV_CLKRC 0x11 //Internal Clock options
97 #define OV_COM7 0x12 //RESET, Output format
98 #define OV_COM8 0x13 //Common control 8
99 #define OV_COM9 0x14 //Automatic Gain Ceiling
100 #define OV_COM10 0x15 //PCLK, HREF and VSYNC options
101 #define OV_RSVD 0x16 //RESERVED
102 #define OV_HSTART 0x17 //Output format Horizontal Frame start
103 #define OV_HSTOP 0x18 //Output format Horizontal Frame end
104 #define OV_VSTRT 0x19 //Output format Vertical Frame start
105 #define OV_VSTOP 0x1A //Output format Vertical Frame Stop
106 #define OV_PSHFT 0x1B //Pixel Delay Select

```

```

107 #define OV_MIDH 0x1C //Manufacturer ID MSB - READ ONLY
108 #define OV_MIDL 0x1D //Manufacturer ID LSB - READ ONLY
109 #define OV_MVFP 0x1E //Mirror / VFlip Enable
110 #define OV_LAEC 0x1F //RESERVED
111 #define OV_ADCCTR0 0x20 //ADC Control
112 #define OV_ADCCTR1 0x21 //RESERVED
113 #define OV_ADCCTR2 0x22 //RESERVED
114 #define OV_ADCCTR3 0x23 //RESERVED
115 #define OV_AEW 0x24 //ACG/AEC Stable Operating Region Upper Limit
116 #define OV_AEB 0x25 //ACG/AEC Stable Operation Region Lower Limit
117 #define OV_VPT 0x26 //ACG/AEC Fast Mode Operation Region
118 #define OV_BBIAS 0x27 //B Channel Signal Output Bias
119 #define OV_GbBIAS 0x28 //Gb Channel Output Bias
120 #define OV_RSVD1 0x29 //RESERVED
121 #define OV_EXHCH 0x2A //Dummy Pixel Insert MSB
122 #define OV_EXHCL 0x2B //Dummy Pixel Insert LSB
123 #define OV_RBIAS 0x2C //R Channel Signal Output Bias
124 #define OV_ADVFL 0x2D //LSB of insert dummy line in vertical direction
125 #define OV_AdVFH 0x2E //MSB of insert dummy line in vertical direction
126 #define OV_YAVE 0x2F //Y/G Channel Average Value
127 #define OV_HSYST 0x30 //HSYNC Rising Edge Delay (low 8 bits)
128 #define OV_HSYEN 0x31 //HSYNCE Falling Edge Delay (low 8 bits)
129 #define OV_HREF 0x32 //HREF Control
130 #define OV_CHLF 0x33 //Array Current Control - RESERVED
131 #define OV_ARBLM 0x34 //Array Reference Control - RESERVED
132 #define OV_RSVD2 0x35 //RESERVED
133 #define OV_RSVD3 0x36 //RESERVED
134 #define OV_ADCCTRL 0x37 //ADC Control - RESERVED
135 #define OV_ACOM 0x38 //ADC and Analog Common Mode Control - RESERVED
136 #define OV_OFON 0x39 //ADC Offset Control
137 #define OV_TSLB 0x3A //Line Buffer Test Option
138 #define OV_COM11 0x3B //COM11
139 #define OV_COM12 0x3C //COM12
140 #define OV_COM13 0x3D //COM13
141 #define OV_COM14 0x3E //COM14
142 #define OV_EDGE 0x3F //Edge Detection Adjustment
143 #define OV_COM15 0x40 //COM15
144 #define OV_COM16 0x41 //COM16
145 #define OV_COM17 0x42 //COM17
146 #define OV_AWBC1 0x43
147 #define OV_AWBC2 0x44
148 #define OV_AWBC3 0x45
149 #define OV_AWBC4 0x46
150 #define OV_AWBC5 0x47
151 #define OV_AWBC6 0x48
152 #define OV_RSVD4 0x49
153 #define OV_RSVD5 0x40
154 #define OV_RSVD6 0x4A
155 #define OV_REG4B 0x4B
156 #define OV_DNSTH 0x4C
157 #define OV_RSVD7 0x4D
158 #define OV_RSVD8 0x4E
159 #define OV_MTX1 0x4F
160 #define OV_MTX2 0x50
161 #define OV_MTX3 0x51
162 #define OV_MTX4 0x52
163 #define OV_MTX5 0x53
164 #define OV_MTX6 0x54
165 #define OV_BRIGHT 0x55

```

```
166 #define OV_CONTRAS      0x56
167 #define OV CONTRASCNTR  0x57
168 #define OV_MTXS        0x58
169 #define OV_RSVD9       0x59
170 #define OV_RSVD9_1     0x5A
171 #define OV_RSVD9_2     0x5B
172 #define OV_RSVD9_3     0x5C
173 #define OV_RSVD9_4     0x5D
174 #define OV_RSVD9_5     0x5E
175 #define OV_RSVD9_6     0x5F
176 #define OV_RSVD10      0x60
177 #define OV_RSVD11      0x61
178 #define OV_LCC1         0x62
179 #define OV_LCC2         0x63
180 #define OV_LCC3         0x64
181 #define OV_LCC4         0x65
182 #define OV_LCC5         0x66
183 #define OV_MANU        0x67
184 #define OV_MANV        0x68
185 #define OV_GFIX        0x69
186 #define OV_GGAIN        0x6A
187 #define OV_DBLV         0x6B
188 #define OV_AWBCTR3      0x6C
189 #define OV_AWBCTR2      0x6D
190 #define OV_AWBCTR1      0x6E
191 #define OV_AWBCTR0      0x6F
192 #define OV_SCALING_XSC  0x70
193 #define OV_SCALING_YSC  0x71
194 #define OV_SCALING_DCWCTR 0x72
195 #define OV_SCALING_PCLK_DIV 0x73
196 #define OV_REG74        0x74
197 #define OV_REG75        0x75
198 #define OV_REG76        0x76
199 #define OV_REG77        0x77
200 #define OV_RSVD12       0x78
201 #define OV_RSVD13       0x79
202 #define OV_GAM1         0x7A
203 #define OV_GAM2         0x7B
204 #define OV_GAM3         0x7C
205 #define OV_GAM4         0x7D
206 #define OV_GAM5         0x7E
207 #define OV_GAM6         0x7F
208 #define OV_GAM7         0x80
209 #define OV_GAM8         0x81
210 #define OV_GAM9         0x82
211 #define OV_GAM10        0x83
212 #define OV_GAM11        0x84
213 #define OV_GAM12        0x85
214 #define OV_GAM13        0x86
215 #define OV_GAM14        0x87
216 #define OV_GAM15        0x88
217 #define OV_GAM16        0x89
218 #define OV_RSVD14       0x8A
219 #define OV_RSVD15       0x8B
220 #define OV_RSVD16       0x8C
221 #define OV_RSVD17       0x8D
222 #define OV_RSVD18       0x8E
223 #define OV_RSVD19       0x8F
224 #define OV_RSVD20       0x90
```

```
225 #define OV_RSVD21 0x91
226 #define OV_DM_LNL 0x92
227 #define OV_DM_LNH 0x93
228 #define OV_LCC6 0x94
229 #define OV_LCC7 0x95
230 #define OV_RSVD22 0x96
231 #define OV_RSVD23 0x97
232 #define OV_RSVD24 0x98
233 #define OV_RSVD25 0x99
234 #define OV_RSVD26 0x9A
235 #define OV_RSVD27 0x9B
236 #define OV_RSVD28 0x9C
237 #define OV_BD50ST 0x9D
238 #define OV_BD60ST 0x9E
239 #define OV_HIST0 0x9F
240 #define OV_HIST1 0xA0
241 #define OV_HIST2 0xA1
242 #define OV_HIST3 0xA2
243 #define OV_HIST4 0xA3
244 #define OV_HIST5 0xA4
245 #define OV_HIST6 0xA5
246 #define OV_HIST7 0xA6
247 #define OV_HIST8 0xA7
248 #define OV_HIST9 0xA8
249 #define OV_HIST10 0xA9
250 #define OV_HIST11 0xAA
251 #define OV_HIST12 0xAB
252 #define OV_STR_OPT 0xAC
253 #define OV_STR_R 0xAD
254 #define OV_STR_G 0xAE
255 #define OV_STR_B 0xAF
256 #define OV_RSVD28_1 0xB0
257 #define OV_RSVD29 0xB1
258 #define OV_RSVD30 0xB2
259 #define OV_THL_ST 0xB3
260 #define OV_RSVD31 0xB4
261 #define OV_THL_DLT 0xB5
262 #define OV_RSVD32 0xB6
263 #define OV_RSVD33 0xB7
264 #define OV_RSVD34 0xB8
265 #define OV_RSVD35 0xB9
266 #define OV_RSVD36 0xBA
267 #define OV_RSVD37 0xBB
268 #define OV_RSVD38 0xBC
269 #define OV_RSVD39 0xBD
270 #define OV_AD_CHB 0xBE
271 #define OV_AD_CHR 0xBF
272 #define OV_AD_CHGb 0xC0
273 #define OV_AD_CHGr 0xC1
274 #define OV_RSVD40 0xC2
275 #define OV_RSVD41 0xC3
276 #define OV_RSVD42 0xC4
277 #define OV_RSVD43 0xC5
278 #define OV_RSVD44 0xC6
279 #define OV_RSVD45 0xC7
280 #define OV_RSVD46 0xC8
281 #define OV_SATCTR 0xC9
283 #endif /* DUALCAMERAS_H */
```

### C.1.1.7 DualCameras.c

..../Code/DualOV7670/DualCameras.c

```
1  /*
2   *  DualCameras.c
3   *
4   *  * Created: 10/11/2012 15:20:03
5   *  * Author: hl13g10
6   */
7
8 #include "DualCameras.h"
9
10 const char default_settings[SETTINGS_LENGTH][2] =
11 {
12 {OV_TSLB, 0x04},
13 {OV_COM15, 0xd0}, //RGB565 / RGB555
14 {OV_COM7, 0x14},
15 {OV_HREF, 0x80},
16 {OV_HSTART, 0x16},
17 {OV_HSTOP, 0x04},
18 {OV_VSTART, 0x02},
19 {OV_VSTOP, 0x7b}, //0x7a,
20 {OV_VREF, 0x06}, //0xa,
21 {OV_COM3, 0x00},
22 {OV_COM14, 0x00}, //
23 {OV_SCALING_XSC, 0x00},
24 {OV_SCALING_YSC, 0x00},
25 {OV_SCALING_DCWCTR, 0x11},
26 {OV_SCALING_PCLK_DIV, 0x00}, //
27 {0xa2, 0x02},
28 {OV_CLKRC, 0x01},
29 {OV_GAM1, 0x20},
30 {OV_GAM2, 0x1c},
31 {OV_GAM3, 0x28},
32 {OV_GAM4, 0x3c},
33 {OV_GAM5, 0x55},
34 {OV_GAM6, 0x68},
35 {OV_GAM7, 0x76},
36 {OV_GAM8, 0x80},
37 {OV_GAM9, 0x88},
38 {OV_GAM10, 0x8f},
39 {OV_GAM11, 0x96},
40 {OV_GAM12, 0xa3},
41 {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_MTXS, 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 };  
  
184 //ISR for controlling WEN.  
185 ISR(INT0_vect)  
{  
187     //printf("ISR INT0 Entered\n");  
188     if (VSYNC_0_Count==1)//start a frame read  
189     {  
190         FIFO_WEN_0_SET;  
191         VSYNC_0_Count++;  
192     }  
193     else if (VSYNC_0_Count==2)//end a frame read  
194     {  
195         FIFO_WEN_0_CLR;  
196         VSYNC_0_Count++;  
197     }  
198     else if(VSYNC_0_Count == 3)  
199     {  
200         FIFO_WEN_0_CLR;  
201     }  
202     else  
203     {  
204         FIFO_WEN_0_CLR  
205         VSYNC_0_Count = 0;//wait for a read to be started  
206     }  
207 }  
208 //ISR for controlling WEN.  
209 ISR(INT1_vect)  
{  
211     //printf("ISR INT1 Entered\n");  
212     if (VSYNC_1_Count==1)//start a frame read  
213     {  
214         FIFO_WEN_1_SET;  
215         VSYNC_1_Count++;  
216     }  
217     else if (VSYNC_1_Count==2)//end a frame read  
218     {  
219         FIFO_WEN_1_CLR;  
220         VSYNC_1_Count++;  
221     }  
222     else if(VSYNC_1_Count == 3)  
223     {  
224         FIFO_WEN_1_CLR;  
225     }  
226     else  
227     {
```

```

228     FIFO_WEN_1_CLR
229     VSYNC_1_Count = 0; //wait for a read to be started
230 }
231 }

233 //Write Register Method
234 unsigned char wrOV7670Reg(unsigned char regID, unsigned char regDat)
235 {
236     /* I2C Traffic Generated:
237      * S | OV_7670 + W | A | RegID | A | Data | A | P |
238      */
239 //I2C Interface
240 unsigned char messageBuf[TWI_BUFFER_SIZE];
241 messageBuf[0] = (OV7670_ADDR << TWI_ADR_BITS) | (FALSE << TWI_READ_BIT); //
242     The first byte must always consist of General Call code or the TWI slave
243     address.
244 messageBuf[1] = regID; // The first byte is used for commands.
245 messageBuf[2] = regDat; // The second byte is used
246     for the data.
247 TWI_Start_Transceiver_With_Data( messageBuf, 3 );

248 while(TWI_Transceiver_Busy()); //Wait for transceiver to clear

249 return TWI_statusReg.lastTransOK;
250 }

251 //Read Register Method
252 unsigned char rdOV7670Reg(unsigned char regID, unsigned char *regDat)
253 {
254     /* I2C Traffic Generated:
255      * S | OV_ADDR + W | A | RegID | A | P |
256      * S | OV_ADDR + R | A | Data | ~A | P |
257      */
258 //I2C Interface
259 unsigned char messageBuf[TWI_BUFFER_SIZE]; //Initialise a buffer
260 messageBuf[0] = (OV7670_ADDR << TWI_ADR_BITS) | (FALSE << TWI_READ_BIT); // The
261     first byte must always consist of General Call code or the TWI slave
262     address.
263 messageBuf[1] = regID; // The first byte is used for Address
264     Pointer.
265 TWI_Start_Transceiver_With_Data( messageBuf, 2 );

266 // Request/collect the data from the Slave
267 messageBuf[0] = (OV7670_ADDR << TWI_ADR_BITS) | (TRUE << TWI_READ_BIT); // The
268     first byte must always consist of General Call code or the TWI slave
269     address.
270 TWI_Start_Transceiver_With_Data( messageBuf, 2 );

271 // Get the received data from the transceiver buffer
272 TWI_Get_Data_From_Transceiver( messageBuf, 2 );
273 *regDat = messageBuf[1];
274 return TWI_statusReg.lastTransOK;
275 }

276 unsigned char OV7670_init()
277 {
278     uint8_t i = 0;
279     if(0==wrOV7670Reg(OV_COM7, 0x80)) //Reset Camera

```

```
279     {
280         return 1;
281     }
282     _delay_ms(10);
283     for(i=0; i<SETTINGS_LENGTH; i++)
284     {
285         if( 0==wrOV7670Reg(default_settings[i][0], default_settings[i][1] ))
286         {
287             return 1;
288         }
289         _delay_ms(1);
290     }
291
292     return 0;
293 }
294
295 void FIFO_init( void )
296 {
297     //disable both outputs
298     FIFO_nOE_0_SET;
299     FIFO_nOE_1_SET;
300     //Reset Buffer 0
301     FIFO_WRST_0_CLR;
302     FIFO_RCLK_0_CLR;
303     //FIFO_nOE_0_CLR;
304     FIFO_nRRST_0_SET;
305     FIFO_WEN_0_CLR;
306     _delay_us(10);
307     FIFO_RCLK_0_SET;
308     _delay_us(10);
309     FIFO_RCLK_0_CLR;
310     FIFO_nRRST_0_CLR;
311     _delay_us(10);
312     FIFO_RCLK_0_SET;
313     _delay_us(10);
314     FIFO_RCLK_0_CLR;
315     FIFO_nRRST_0_SET;
316     _delay_us(10);
317     FIFO_WRST_0_SET;
318
319     //Reset Buffer 1
320     FIFO_WRST_1_CLR;
321     FIFO_RCLK_1_CLR;
322     //FIFO_nOE_1_CLR;
323     FIFO_nRRST_1_SET;
324     FIFO_WEN_1_CLR;
325     _delay_us(10);
326     FIFO_RCLK_1_SET;
327     _delay_us(10);
328     FIFO_RCLK_1_CLR;
329     FIFO_nRRST_1_CLR;
330     _delay_us(10);
331     FIFO_RCLK_1_SET;
332     _delay_us(10);
333     FIFO_RCLK_1_CLR;
334     FIFO_nRRST_1_SET;
335     _delay_us(10);
336     FIFO_WRST_1_SET;
```

```
338 }
340 //Write one pixel in AVR
341 uint16_t FIFO_TO_AVR(uint8_t ID)
342 {
343     uint16_t data = 0;
345     DDRA = 0;
346     if(ID == 1)
347     {
348         FIFO_RCLK_1_SET;
349         data = PINA;
350         FIFO_RCLK_1_CLR;
351         data <= 8;
352         FIFO_RCLK_1_SET;
353         data |= PINA;
354         FIFO_RCLK_1_CLR;
355     }
356     else
357     {
358         FIFO_RCLK_0_SET;
359         data = PINA;
360         FIFO_RCLK_0_CLR;
361         data <= 8;
362         FIFO_RCLK_0_SET;
363         data |= PINA;
364         FIFO_RCLK_0_CLR;
365     }
366     return(data);
367 }

370 //Resets both pointers
371 void FIFO_Reset(uint8_t CameraID)
372 {
373     FIFO_nOE_0_SET;
374     FIFO_nOE_1_SET;
375     if(CameraID == 0)
376     {
377         FIFO_WRST_0_CLR;
378         FIFO_nRRST_0_CLR;
379         FIFO_RCLK_0_SET;
380         FIFO_RCLK_0_CLR;
381         FIFO_nRRST_0_SET;
382         FIFO_WRST_0_SET;
383     }
384     else
385     {
386         FIFO_WRST_1_CLR;
387         FIFO_nRRST_1_CLR;
388         FIFO_RCLK_1_SET;
389         FIFO_RCLK_1_CLR;
390         FIFO_nRRST_1_SET;
391         FIFO_WRST_1_SET;
392     }
394 }
395 void LoadImagesToBuffer()
396 {
```

```
397     VSYNC_0_Count = 0;
398     VSYNC_1_Count = 0;
399     FIFO_Reset(0);
400     FIFO_Reset(1);
401     VSYNC_0_Count = 1;
402     VSYNC_1_Count = 1;

404 }
405 uint8_t GetImageIfAvailable(FIL *File, uint8_t CameraID)
406 {
407
408     if( ((CameraID == 0) && (VSYNC_0_Count == 3)) ||
409         ((CameraID == 1) && (VSYNC_1_Count == 3)) )
410     {
411
412         //Write Bitmap Headers
413         WriteBMPHeader(File);
414         WriteDIBHeader(File);
415         if (CameraID == 0)
416         {
417             //Enable output of Camera 0
418             FIFO_nOE_0_CLR;
419             //Reset Read Pointer
420             FIFO_nRRST_0_CLR;
421             FIFO_RCLK_0_SET;
422             FIFO_RCLK_0_CLR;
423             FIFO_nRRST_0_SET;
424         }
425         else
426         {
427             //Enable output of Camera 0
428             FIFO_nOE_1_CLR;
429             //Reset Read Pointer
430             FIFO_nRRST_1_CLR;
431             FIFO_RCLK_1_SET;
432             FIFO_RCLK_1_CLR;
433             FIFO_nRRST_1_SET;
434         }
435         int i, j;
436         uint32_t pointer;
437         uint16_t Temp;
438         uint32_t p;
439         FRESULT fr;
440         //for(j = HEIGHT; j>0; j--)
441         for(j = 0; j < HEIGHT; j++)
442         {
443             pointer = 0;
444             for(i = 0; i < WIDTH; i++)
445             {
446                 Temp = FIFO_TO_AVR(CameraID);
447                 //USART0_Senduint16(Temp);

448                 Buff[pointer++] = (uint8_t)(Temp >> 8);
449                 Buff[pointer++] = (uint8_t)Temp;
450             }
451             pointer = (uint32_t)j * (uint32_t)WIDTH * 2 + BMPHEADERSIZE +
452             DIBHEADERSIZE;
453             f_lseek(File, pointer);
454             fr = f_write(File, Buff, WIDTH * 2, &p);
```

```

455     if(fr != FR_OK)
456     {
457         //printf("Write Fail.\n");
458         VSYNC_0_Count = 0;
459         VSYNC_1_Count = 0;
460         FIFO_Reset(CameraID);
461         FIFO_nOE_0_SET;
462         FIFO_nOE_1_SET;
463         return 1;
464     }
465 }
466 FIFO_Reset(CameraID);
467 //fr = f_close(File);
468 FIFO_nOE_0_SET;
469 FIFO_nOE_1_SET;
470 return 0;
471 }
472 else
473 {
474     return 2;
475 }
476 }
```

### C.1.1.8 PCA9542A.h

..../Code/DualOV7670/PCA9542A.h

```

1  /*
2   * PCA9542A.h
3   *
4   * Created: 13/11/2012 23:24:48
5   * Author: hslovett
6   */
7
8 #ifndef PCA9542A_H_
9 #define PCA9542A_H_
10 #include "Config.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 CHO 0x04
19 #define CH1 0x05
20
21 unsigned char PCA9542A_Init();
22 unsigned char PCA9542A_SetChannel(uint8_t Channel);
23
24#endif /* PCA9542A_H_ */
```

## C.1.1.9 PCA9542A.c

..//Code/DualOV7670/PCA9542A.c

```

1  /*
2   *  PCA9542A.c
3   *
4   *  Created: 13/11/2012 23:24:40
5   *  Author: hslovett
6   */
7  #include "PCA9542A.h"

10 unsigned char PCA9542A_Init()
11 {
12     unsigned char messageBuf[TWI_BUFFER_SIZE];
13     messageBuf[0] = (PCA9542A_ADDR <<TWI_ADR_BITS) | (FALSE<<TWI_READ_BIT); // 
14         The first byte must always consist of General Call code or the TWI slave
15         address.
16     messageBuf[1] = NO_SELECT;           // The first byte is used for
17         commands.
18     // The second byte is used for the data.
19     TWI_Start_Transceiver_With_Data( messageBuf, 2 );

20     while(TWI_Transceiver_Busy()) ; //Wait for transceiver to clear
21
22     return TWI_statusReg.lastTransOK;
23 }

24 unsigned char PCA9542A_SetChannel( uint8_t Channel )
25 {
26     unsigned char messageBuf[TWI_BUFFER_SIZE];
27     messageBuf[0] = (PCA9542A_ADDR <<TWI_ADR_BITS) | (FALSE<<TWI_READ_BIT); // 
28         The first byte must always consist of General Call code or the TWI slave
29         address.
30     messageBuf[1] = Channel;           // The first byte is used for commands.
31         // The second byte is used for the data.
32     TWI_Start_Transceiver_With_Data( messageBuf, 2 );

33     while(TWI_Transceiver_Busy()) ; //Wait for transceiver to clear
34
35     return TWI_statusReg.lastTransOK;
36 }

37 unsigned char PCA9542A_ReadChannel()
38 {
39     unsigned char messageBuf[TWI_BUFFER_SIZE];
40     messageBuf[0] = (PCA9542A_ADDR <<TWI_ADR_BITS) | (TRUE<<TWI_READ_BIT); // 
41         The first byte must always consist of General Call code or the TWI slave
42         address.
43
44     TWI_Start_Transceiver_With_Data( messageBuf, 1 );
45
46     while(TWI_Transceiver_Busy()) ; //Wait for transceiver to clear
47     // Get the received data from the transceiver buffer
48     TWI_Get_Data_From_Transceiver( messageBuf, 2 );
49
50     return TWI_statusReg.lastTransOK;
51 }
```

### C.1.1.10 TWI\_Master.h

..../Code/DualOV7670/TWI\_Master.h

```

1  ****
2  *
3  * Atmel Corporation
4  *
5  * File : TWI_Master.h
6  * Compiler : IAR EWAAVR 2.28a/3.10c
7  * Revision : Revision: 1.13
8  * Date : Date: 24. mai 2004 11:31:22
9  * Updated by : Author: ltwa
10 *
11 * Support mail : avr@atmel.com
12 *
13 * Supported devices : All devices with a TWI module can be used.
14 *                      The example is written for the ATmega16
15 *
16 * AppNote : AVR315 - TWI Master Implementation
17 *
18 * Description : Header file for TWI_Master.c
19 *                 Include this file in the application.
20 *
21 ****
22
23 /* Modified by Henry Lovett (hl13g10@ecs.soton.ac.uk) to allow SCL frequency
   to be specified and TWBR calculated
   * Also allows AVR internal pull up resistors to be used.
   */
24
25 #ifndef _TWI_MASTER_H
26 #define _TWI_MASTER_H
27 #include <avr/io.h>
28 #include <avr/interrupt.h>
29 #include "Config.h"
30
31 ****
32 TWI Status/Control register definitions
33 ****
34
35 #define INTERNAL_PULLUPS 0
36
37 #define TWI_BUFFER_SIZE 4 // Set this to the largest message size that will
   be sent including address byte.
38
39 #define SCL_Freq 100000 //SCL Frequency in Hertz
40 #define TWI_TWBR (char)(F_CPU / 2 / SCL_Freq - 8) //Equation to calculate
   TWBR Based on SCL Frequency and Clock Frequency
41
42 ##define TWI_TWBR 0x0C //400KHz // TWI Bit rate Register
   setting.
43 ##define TWI_TWBR 0x34 //100KHz // Application note for detailed
   // information on setting this value.
44 // Not used defines!
45 //define TWI_TWPS 0x00 // This driver presumes prescaler = 00
46
47 ****
48 Global definitions
49

```

```

50 *****/
51
52 union TWI_statusReg // Status byte holding flags.
53 {
54     unsigned char all;
55     struct
56     {
57         unsigned char lastTransOK:1;
58         unsigned char unusedBits:7;
59     };
60 };
61
62 extern union TWI_statusReg TWI_statusReg;
63
64 /**
65     Function definitions
66 */
67 void TWI_Master_Initialise( void );
68 unsigned char TWI_Transceiver_Busy( void );
69 unsigned char TWI_Get_State_Info( void );
70 void TWI_Start_Transceiver_With_Data( unsigned char * , unsigned char );
71 void TWI_Start_Transceiver( void );
72 unsigned char TWI_Get_Data_From_Transceiver( unsigned char * , unsigned char );
73
74 /**
75     Bit and byte definitions
76 */
77 #define TWI_READ_BIT 0 // Bit position for R/W bit in "address byte".
78 #define TWI_ADR_BITS 1 // Bit position for LSB of the slave address
    bits in the init byte.
79
80 #define TRUE 1
81 #define FALSE 0
82
83 /**
84     TWI State codes
85 */
86 // General TWI Master status codes
87 #define TWI_START 0x08 // START has been transmitted
88 #define TWI_REPEAT_START 0x10 // Repeated START has been
    transmitted
89 #define TWI_ARB_LOST 0x38 // Arbitration lost
90
91 // TWI Master Transmitter status codes
92 #define TWI_MTX_ADR_ACK 0x18 // SLA+W has been transmitted and ACK
    received
93 #define TWI_MTX_ADR_NACK 0x20 // SLA+W has been transmitted and
    NACK received
94 #define TWI_MTX_DATA_ACK 0x28 // Data byte has been transmitted and
    ACK received
95 #define TWI_MTX_DATA_NACK 0x30 // Data byte has been transmitted and
    NACK received
96
97 // TWI Master Receiver status codes
98 #define TWI_MRX_ADR_ACK 0x40 // SLA+R has been transmitted and ACK
    received
99 #define TWI_MRX_ADR_NACK 0x48 // SLA+R has been transmitted and
    NACK received

```

```

100 #define TWI_MRX_DATA_ACK          0x50 // Data byte has been received and
101   ACK transmitted
102 #define TWI_MRX_DATA_NACK         0x58 // Data byte has been received and
103   NACK transmitted

104 // TWI Slave Transmitter status codes
105 #define TWI_STX_ADR_ACK           0xA8 // Own SLA+R has been received; ACK
106   has been returned
107 #define TWI_STX_ADR_ACK_M_ARB_LOST 0xB0 // Arbitration lost in SLA+R/W as
108   Master; own SLA+R has been received; ACK has been returned
109 #define TWI_STX_DATA_ACK           0xB8 // Data byte in TWDR has been
110   transmitted; ACK has been received
111 #define TWI_STX_DATA_NACK          0xC0 // Data byte in TWDR has been
112   transmitted; NOT ACK has been received
113 #define TWI_STX_DATA_ACK_LAST_BYTE 0xC8 // Last data byte in TWDR has been
114   transmitted (TWEA = 0); ACK has been received

115 // TWI Slave Receiver status codes
116 #define TWI_SRX_ADR_ACK           0x60 // Own SLA+W has been received ACK
117   has been returned
118 #define TWI_SRX_ADR_ACK_M_ARB_LOST 0x68 // Arbitration lost in SLA+R/W as
119   Master; own SLA+W has been received; ACK has been returned
120 #define TWI_SRX_GEN_ACK            0x70 // General call address has been
121   received; ACK has been returned
122 #define TWI_SRX_GEN_ACK_M_ARB_LOST 0x78 // Arbitration lost in SLA+R/W as
123   Master; General call address has been received; ACK has been returned
124 #define TWI_SRX_ADR_DATA_ACK       0x80 // Previously addressed with own SLA+
125   W; data has been received; ACK has been returned
126 #define TWI_SRX_ADR_DATA_NACK      0x88 // Previously addressed with own SLA+
127   W; data has been received; NOT ACK has been returned
128 #define TWI_SRX_GEN_DATA_ACK       0x90 // Previously addressed with general
129   call; data has been received; ACK has been returned
130 #define TWI_SRX_GEN_DATA_NACK      0x98 // Previously addressed with general
131   call; data has been received; NOT ACK has been returned
132 #define TWI_SRX_STOP_RESTART        0xA0 // A STOP condition or repeated START
133   condition has been received while still addressed as Slave

134 // TWI Miscellaneous status codes
135 #define TWI_NO_STATE              0xF8 // No relevant state information
136   available; TWINT = 0
137 #define TWI_BUS_ERROR              0x00 // Bus error due to an illegal START
138   or STOP condition

139 #endif

```

### C.1.11 TWIMaster.c

..../Code/DualOV7670/TWIMaster.c

```

1 ****
2 *
3 * Atmel Corporation
4 *
5 * File           : TWI_Master.c
6 * Compiler       : IAR EWAAVR 2.28a/3.10c
7 * Revision       : Revision: 1.13

```

```

8 * Date : Date: 24. mai 2004 11:31:20
9 * Updated by : Author: ltwa
10 *
11 * Support mail : avr@atmel.com
12 *
13 * Supported devices : All devices with a TWI module can be used.
14 * The example is written for the ATmega16
15 *
16 * AppNote : AVR315 - TWI Master Implementation
17 *
18 * Description : This is a sample driver for the TWI hardware modules.
19 * It is interrupt driven. All functionality is
20 * controlled through
21 * passing information to and from functions. See main.c for
22 * samples
23 * of how to use the driver.
24 ****
25
26
27 #include "TWI_Master.h"
28
29 static unsigned char TWI_buf[ TWI_BUFFER_SIZE ]; // Transceiver buffer
30 static unsigned char TWI_msgSize; // Number of bytes to be
31 transmitted.
32 static unsigned char TWI_state = TWI_NO_STATE; // State byte. Default set
33 to TWI_NO_STATE.
34
35 union TWI_statusReg TWI_statusReg = {0}; // TWI_statusReg is
36 defined in TWI_Master.h
37
38 ****
39 Call this function to set up the TWI master to its initial standby state.
40 Remember to enable interrupts from the main application after initializing the
41 TWI.
42 ****
43 void TWI_Master_Initialise(void)
44 {
45     #if INTERNAL_PULLUPS == 1 //enable built in pullups for I2C Lines
46     DDRC = 0x00;
47     PORTC = (1 << PC0) | (1 << PC1);
48     #else
49     #pragma message("External I2C Pull Ups Required.")
50     #endif
51     TWBR = TWI_TWBR; // Set bit rate register (
52     // Baudrate). Defined in header file.
53     // TWSR = TWI_TWPS; // Not used. Driver
54     // presumes prescaler to be 00.
55     TWDR = 0xFF; // Default content = SDA
56     // released.
57     TWCR = (1<<TWEN) | // Enable TWI-interface
58     and release TWI pins.
59     (0<<TWIE)|(0<<TWINT)| // Disable Interrupt.
60     (0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)| // No Signal requests.
61     (0<<TWWC); // /
62 }
63
64 ****

```

```

57 Call this function to test if the TWI_ISR is busy transmitting.
58 ****
59 unsigned char TWI_Transceiver_Busy( void )
60 {
61     return ( TWCR & (1<<TWIE) ); // IF TWI Interrupt is enabled
62     then the Transceiver is busy
63 }
64 ****
65 Call this function to fetch the state information of the previous operation.
66 The function will hold execution (loop)
67 until the TWI_ISR has completed with the previous operation. If there was an
68 error, then the function
69 will return the TWI State code.
70 ****
71 unsigned char TWI_Get_State_Info( void )
72 {
73     while ( TWI_Transceiver_Busy() ); // Wait until TWI has
74         completed the transmission.
75     return ( TWI_state ); // Return error state.
76 }
77 ****
78 Call this function to send a prepared message. The first byte must contain the
79 slave address and the
80 read/write bit. Consecutive bytes contain the data to be sent, or empty
81 locations for data to be read
82 from the slave. Also include how many bytes that should be sent/read including
83 the address byte.
84 The function will hold execution (loop) until the TWI_ISR has completed with
85 the previous operation,
86 then initialize the next operation and return.
87 ****
88 void TWI_Start_Transceiver_With_Data( unsigned char *msg, unsigned char
89     msgSize )
90 {
91     unsigned char temp;
92
93     while ( TWI_Transceiver_Busy() ); // Wait until TWI is ready for
94         next transmission.
95
96     TWI_msgSize = msgSize; // Number of data to transmit.
97     TWI_buf[0] = msg[0]; // Store slave address with R/
98     W setting.
99     if ( !( msg[0] & (TRUE<<TWI_READ_BIT) ) ) // If it is a write operation,
100        then also copy data.
101    {
102        for ( temp = 1; temp < msgSize; temp++ )
103            TWI_buf[ temp ] = msg[ temp ];
104    }
105    TWI_statusReg.all = 0;
106    TWI_state = TWI_NO_STATE ;
107    TWCR = (1<<TWEN)| // TWI Interface enabled.
108        (1<<TWIE)|(1<<TWINT)| // Enable TWI Interrupt and
109        clear the flag.
110        (0<<TWEA)|(1<<TWSTA)|(0<<TWSTO)| // Initiate a START condition.
111        (0<<TWWC); //
```

```

103  ****
104  Call this function to resend the last message. The driver will reuse the data
105  previously put in the transceiver buffers.
106  The function will hold execution (loop) until the TWI_ISR has completed with
107  the previous operation,
108  then initialize the next operation and return.
109  ****
110 void TWI_Start_Transceiver( void )
111 {
112     while ( TWI_Transceiver_Busy() ); // Wait until TWI is ready for
113         next transmission.
114     TWI_statusReg.all = 0;
115     TWI_state         = TWI_NO_STATE ;
116     TWCR = (1<<TWEN)| // TWI Interface enabled.
117         (1<<TWIE)|(1<<TWINT)| // Enable TWI Interrupt and
118         clear the flag.
119     (0<<TWEA)|(1<<TWSTA)|(0<<TWSTO)| // Initiate a START condition.
120         (0<<TWWC); // 
121 }
122 ****
123 Call this function to read out the requested data from the TWI transceiver
124 buffer. I.e. first call
125 TWI_Start_Transceiver to send a request for data to the slave. Then Run this
126 function to collect the
127 data when they have arrived. Include a pointer to where to place the data and
128 the number of bytes
129 requested (including the address field) in the function call. The function
130 will hold execution (loop)
131 until the TWI_ISR has completed with the previous operation, before reading
132 out the data and returning.
133 If there was an error in the previous transmission the function will return
134 the TWI error code.
135 ****
136 unsigned char TWI_Get_Data_From_Transceiver( unsigned char *msg, unsigned char
137     msgSize )
138 {
139     unsigned char i;
140
141     while ( TWI_Transceiver_Busy() ); // Wait until TWI is ready for
142         next transmission.
143
144     if( TWI_statusReg.lastTransOK ) // Last transmission competed
145         successfully.
146     {
147         for ( i=0; i<msgSize; i++ ) // Copy data from Transceiver
148             buffer.
149         {
150             msg[ i ] = TWI_buf[ i ];
151         }
152     }
153     return( TWI_statusReg.lastTransOK );
154 }
155
156 // ***** Interrupt Handlers **** //
157 ****
158 This function is the Interrupt Service Routine (ISR), and called when the TWI
159 interrupt is triggered;

```

```

146 that is whenever a TWI event has occurred. This function should not be called
147 directly from the main
148 application.
149 ****
150 ISR(TWI_vect)
151 {
152     static unsigned char TWI_bufPtr;
153
154     switch (TWSR)
155     {
156         case TWI_START:           // START has been transmitted
157         case TWI_REP_START:      // Repeated START has been transmitted
158             TWI_bufPtr = 0;          // Set buffer
159             // pointer to the TWI Address location
160         case TWI_MTX_ADDR_ACK:   // SLA+W has been transmitted and ACK received
161         case TWI_MTX_DATA_ACK:   // Data byte has been transmitted and ACK
162             received
163             if (TWI_bufPtr < TWI_msgSize)
164             {
165                 TWDR = TWI_buf[TWI_bufPtr++];
166                 TWCR = (1<<TWEN) | // TWI Interface
167                     enabled
168                     (1<<TWIE)|(1<<TWINT)| // Enable TWI
169                     Interrupt and clear the flag to send byte
170                     (0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)| // 
171                     (0<<TWWC); // 
172             } else // Send STOP after last byte
173             {
174                 TWI_statusReg.lastTransOK = TRUE; // Set status bits
175                 to completed successfully.
176                 TWCR = (1<<TWEN) | // TWI Interface
177                     enabled
178                     (0<<TWIE)|(1<<TWINT)| // Disable TWI
179                     Interrupt and clear the flag
180                     (0<<TWEA)|(0<<TWSTA)|(1<<TWSTO)| // Initiate a STOP
181                     condition.
182                     (0<<TWWC); // 
183             }
184             break;
185         case TWI_MRX_DATA_ACK:    // Data byte has been received and ACK
186             transmitted
187             TWI_buf[TWI_bufPtr++] = TWDR;
188         case TWI_MRX_ADDR_ACK:    // SLA+R has been transmitted and ACK received
189             if (TWI_bufPtr < (TWI_msgSize-1) ) // Detect the last
190             byte to NACK it.
191             {
192                 TWCR = (1<<TWEN) | // TWI Interface
193                     enabled
194                     (1<<TWIE)|(1<<TWINT)| // Enable TWI
195                     Interrupt and clear the flag to read next byte
196                     (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)| // Send ACK after
197                     reception
198                     (0<<TWWC); // 
199             } else // Send NACK after next reception
200             {
201                 TWCR = (1<<TWEN) | // TWI Interface
202                     enabled

```

```

189         (1<<TWIE)|(1<<TWINT)|                                // Enable TWI
190         Interrupt and clear the flag to read next byte
191         (0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|                  // Send NACK after
192         reception
193         (0<<TWWC);                                         //
194     }
195     break;
196 case TWI_MRX_DATA_NACK:      // Data byte has been received and NACK
197     transmitted
198     TWI_buf[TWI_bufPtr] = TWDR;
199     TWI_statusReg.lastTransOK = TRUE;                      // Set status bits to
200     completed successfully.
201     TWCR = (1<<TWEN)|                                     // TWI Interface
202     enabled
203     (0<<TWIE)|(1<<TWINT)|                                // Disable TWI
204     Interrupt and clear the flag
205     (0<<TWEA)|(0<<TWSTA)|(1<<TWSTO)|                  // Initiate a STOP
206     condition.
207     (0<<TWWC);                                         //
208     break;
209 case TWI_ARB_LOST:          // Arbitration lost
210     TWCR = (1<<TWEN)|                                     // TWI Interface
211     enabled
212     (1<<TWIE)|(1<<TWINT)|                                // Enable TWI Interrupt
213     and clear the flag
214     (0<<TWEA)|(1<<TWSTA)|(0<<TWSTO)|                  // Initiate a (RE)
215     START condition.
216     (0<<TWWC);                                         //
217     break;
218 case TWI_MTX_ADDR_NACK:     // SLA+W has been transmitted and NACK
219     received
220 case TWI_MRX_ADDR_NACK:     // SLA+R has been transmitted and NACK
221     received
222 case TWI_MTX_DATA_NACK:    // Data byte has been transmitted and NACK
223     received
224 //     case TWI_NO_STATE                                // No relevant state information
225     available; TWINT = 0
226 case TWI_BUS_ERROR:         // Bus error due to an illegal START or STOP
227     condition
228 default:
229     TWI_state = TWSR;                                    // Store TWSR and
230     automatically sets clears noErrors bit.
231
232     TWCR = (1<<TWEN)|                                // Reset TWI Interface
233     interface and release TWI pins
234     (0<<TWIE)|(0<<TWINT)|                            // Disable Interrupt
235     (0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|                  // No Signal requests
236     (0<<TWWC);                                         //
237 }
238 }
```

### C.1.1.12 Usart.h

..../Code/DualOV7670/Usart.h

1 /\*

```

2  * Usart.h
3  *
4  * Created: 25/10/2012 22:25:14
5  * Author: hslovett
6  */

9 #ifndef USART_H_
10 #define USART_H_

12 #include "Config.h"
13 #include <stdio.h>
14 #include <avr/io.h>
15 #define USART0_BITRATE 57600
16 #define UBBR F_CPU/16/USART0_BITRATE -1

18 void USART0_Init ();
19 void Usart_SendChar(char data);
20 unsigned char Usart_Receive( void );
21 int Usart_printf(char var, FILE *stream);
22 void Usart_get_line (char *buff, int len);
23 void USART0_Senduint16 (uint16_t Data);
24 // void USART0_SendChar( unsigned char data );
25 // unsigned char USART0_Receive( void );
26 // void USART0_SendString(char str[]);

28 #endif /* USART_H_ */

```

### C.1.1.13 Usart.c

..../Code/DualOV7670/Usart.c

```

1  /*
2  * Usart.c
3  *
4  * Created: 25/10/2012 22:25:04
5  * Author: hl13g10@ecs.soton.ac.uk
6  */

9 #include "Usart.h"

12 void USART0_Init()
13 {
14     uint16_t ubrr = UBBR;
15     //Set baud rate
16     UBRROH = (unsigned char)(ubrr >>8);
17     UBRROL = (unsigned char)ubrr ;
18     //Enable receiver and transmitter
19     UCSROB = (1<<RXENO)|(1<<TXENO);

21     UCSROC = 0x06; //set asynchronous, no parity, one stop bit, 8 bit transfer.

23     //UCSROB |= (1 << RXCIE0) | (1 << TXCIE0); //set RX and TX interrupt on
24 }

```

```
25 void Usart_SendChar(char data)
26 {
27     // Wait for empty transmit buffer
28     while ( !(UCSROA & (1 << UDRE0)) );
29     // Start transmission
30     UDRO = data;
31 }
32 unsigned char Usart_Receive( void )
33 {
34     /* Wait for data to be received */
35     while ( !(UCSROA & (1<<RXC0)) )
36     ;
37     /* Get and return received data from buffer */
38     //Usart_SendChar(UDRO);
39     return UDRO;
40 }

42 //to use this copy the following as a global-
43 //    static FILE mystdout = FDEV_SETUP_STREAM(Usart_printf, NULL,
44 //        _FDEV_SETUP_WRITE);
44 // and add this line at the beginning of main:
45 //    stdout = &mystdout;
46 // stdio.h must be used.
47 int Usart_printf(char var, FILE *stream) {
48     // translate \n to \r for br@y++ terminal
49     if (var == '\n') Usart_SendChar('\r');
50     Usart_SendChar(var);
51     return 0;
52 }

54 void Usart_get_line (char *buff, int len)
55 {
56     cli();
57     char c;
58     int i = 0;

61     for (;;) {
62         c = Usart_Receive();
63         if (c == '\r') break;
64         if ((c == '\b') && i) {
65             i--;
66             Usart_SendChar(c);
67             continue;
68         }
69         if (c >= ' ' && i < len - 1) { /* Visible chars */
70             buff[i++] = c;
71             Usart_SendChar(c);
72         }
73     }
74     buff[i] = 0;
75     Usart_SendChar('\n');
76     sei();
77 }
78 void USART0_Senduint16 (uint16_t Data)
79 {
80     Usart_SendChar(Data >> 8);
81     Usart_SendChar(Data & 0xFF);
82 }
```

## C.1.2 Dual Camera User Interface

### C.1.2.1 DualCamera\_UI.c

..../Code/DualCamera\_UI/DualCamera\_UI.c

```
1  /*
2   *  DualCamera_UI.c
3   *
4   *  Created: 12/11/2012 08:32:27
5   *  Author: hslovett
6   */
7
8
9 #include <avr/io.h>
10 #include <avr/interrupt.h>
11 #include "TWI_slave.h"
12
13 #define ButtonMask 0x0F
14
15 #define TWI_CMD_MASTER_WRITE 0x10
16 #define TWI_CMD_MASTER_READ 0x20
17
18 // When there has been an error, this function is run and takes care of it
19 unsigned char TWI_Act_On_Failure_In_Last_Transmission ( unsigned char
20             TWIErrorMsg );
21
22 int main(void)
23 {
24
25     char ButtonStatus = 0xFF;
26     unsigned char TWI_slaveAddress;
27     unsigned char messageBuff[TWI_BUFFER_SIZE];
28     DDRD = 0xFF; // Port D is the LED output
29     DDRC = 0x00; //PortC is the switch input
30     //PORTC = 0xFF;
31     TWI_Slave_Initialise( (unsigned char)((TWI_slaveAddress<<TWI_ADR_BITS) | (
32         TRUE<<TWI_GEN_BIT) ) );
33     sei();
34     TWI_Start_Transceiver();
35     while(1)
36     {
37
38         ButtonStatus = (ButtonStatus & PINC) & ButtonMask;
39         //PORTD = ButtonStatus;
40         // Check if the TWI Transceiver has completed an operation.
41         if ( ! TWI_Transceiver_Busy() )
42         {
43             // Check if the last operation was successful
44             if ( TWI_statusReg.lastTransOK )
45             {
46                 // Check if the last operation was a reception
```

```

46     if ( TWI_statusReg.RxDataInBuf )
47     {
48         TWI_Get_Data_From_Transceiver(messageBuff, 2);
49         // Check if the last operation was a reception as General Call
50         if ( TWI_statusReg.genAddressCall )
51         {
52             // Put data received out to PORTB as an example.
53             PORTB = messageBuff[0];
54         }
55         else // Ends up here if the last operation was a reception as
Slave Address Match
56         {
57             // Example of how to interpret a command and respond.

58             // TWI_CMD_MASTER_WRITE stores the data to PORTB
59             if (messageBuff[0] == TWI_CMD_MASTER_WRITE)
60             {
61                 PORTD = messageBuff[1];
62             }
63             // TWI_CMD_MASTER_READ prepares the data from PINB in the
transceiver buffer for the TWI master to fetch.
64             if (messageBuff[0] == TWI_CMD_MASTER_READ)
65             {
66                 messageBuff[0] = ButtonStatus;
67                 TWI_Start_Transceiver_With_Data( messageBuff, 1 );
68                 ButtonStatus = ButtonMask; //clear all logged button presses
69             }
70         }
71     }
72     else // Ends up here if the last operation was a transmission
73     {
74         //__no_operation(); // Put own code here.
75     }
76     // Check if the TWI Transceiver has already been started.
77     // If not then restart it to prepare it for new receptions.
78     if ( ! TWI_Transceiver_Busy() )
79     {
80         TWI_Start_Transceiver();
81     }
82     else // Ends up here if the last operation completed unsuccessfully
83     {
84         //TWI_Act_On_Failure_In_Last_Transmission( TWI_Get_State_Info() );
85     }
86 }
87 }
88 }
89 }
90 }

91 unsigned char TWI_Act_On_Failure_In_Last_Transmission ( unsigned char
TWIerrorMsg )
92 {
93     // A failure has occurred, use TWIerrorMsg to determine the nature of the
failure
94     // and take appropriate actions.
95     // See header file for a list of possible failures messages.

96     // This very simple example puts the error code on PORTB and restarts the
transceiver with
97     // all the same data in the transmission buffers.
98
99 }
```

```

100 //PORTB = TWIerrorMsg;
101 TWI_Start_Transceiver();
102
103 return TWIerrorMsg;
104 }
```

### C.1.2.2 TWI\_slave.h

..../Code/DualCamera\_UI/TWI\_slave.h

```

1 ****
2 *
3 * Atmel Corporation
4 *
5 * File : TWI_Slave.h
6 * Compiler : IAR EWAAVR 2.28a/3.10c
7 * Revision : Revision: 2475
8 * Date : Date: 2007-09-20 12:00:43 +0200 (to, 20 sep 2007)
9 * Updated by : Author: mlarsson
10
11 * Support mail : avr@atmel.com
12 *
13 * Supported devices : All devices with a TWI module can be used.
14 * The example is written for the ATmega16
15 *
16 * AppNote : AVR311 - TWI Slave Implementation
17 *
18 * Description : Header file for TWI_slave.c
19 * Include this file in the application.
20 *
21 ****
22 /*! \page MISRA
23 *
24 * General disabling of MISRA rules:
25 * * (MISRA C rule 1) compiler is configured to allow extensions
26 * * (MISRA C rule 111) bit fields shall only be defined to be of type
27 * unsigned int or signed int
28 * * (MISRA C rule 37) bitwise operations shall not be performed on signed
29 * integer types
30 * As it does not work well with 8bit architecture and/or IAR
31
32 * Other disabled MISRA rules
33 * * (MISRA C rule 109) use of union - overlapping storage shall not be used
34 * * (MISRA C rule 61) every non-empty case clause in a switch statement shall
35 * be terminated with a break statement
36 */
37
38 ****
39 TWI_Status/Control register definitions
40 ****
41
42 #define TWI_BUFFER_SIZE 4 // Reserves memory for the drivers transceiver
43 // buffer.
44 // Set this to the largest message size that
45 // will be sent including address byte.
```

```

42  /*******************************************************************************
43   * Global definitions
44  ******************************************************************************/
45
46 union TWI_statusReg_t                                // Status byte holding flags.
47 {
48     unsigned char all;
49     struct
50     {
51         unsigned char lastTransOK:1;
52         unsigned char RxDataInBuf:1;
53         unsigned char genAddressCall:1;                                // TRUE =
54         General call, FALSE = TWI Address;
55         unsigned char unusedBits:5;
56     };
57 };
58
59
60 extern union TWI_statusReg_t TWI_statusReg;
61
62 //static unsigned char dont_sleep = 0;
63
64 /*******************************************************************************
65 * Function definitions
66 ******************************************************************************/
67 void TWI_Slave_Initialise( unsigned char );
68 unsigned char TWI_Transceiver_Busy( void );
69 unsigned char TWI_Get_State_Info( void );
70 void TWI_Start_Transceiver_With_Data( unsigned char * , unsigned char );
71 void TWI_Start_Transceiver( void );
72 unsigned char TWI_Get_Data_From_Transceiver( unsigned char * , unsigned char );
73
74 ISR( TWI_vect );
75
76 /*******************************************************************************
77 * Bit and byte definitions
78 ******************************************************************************/
79 #define TWI_READ_BIT    0 // Bit position for R/W bit in "address byte".
80 #define TWI_ADR_BITS   1 // Bit position for LSB of the slave address bits in
81             // the init byte.
82 #define TWI_GEN_BIT    0 // Bit position for LSB of the general call bit in
83             // the init byte.
84
85 #define TRUE          1
86 #define FALSE         0
87
88 /*******************************************************************************
89 * TWI State codes
90 ******************************************************************************/
91 // General TWI Master status codes
92 #define TWI_START        0x08 // START has been transmitted
93 #define TWI_REP_START    0x10 // Repeated START has been
94             // transmitted
95 #define TWI_ARB_LOST    0x38 // Arbitration lost
96
97 // TWI Master Transmitter status codes
98 #define TWI_MTX_ADR_ACK 0x18 // SLA+W has been transmitted and ACK
99             // received
100 #define TWI_MTX_ADR_NACK 0x20 // SLA+W has been transmitted and
101             // NACK received

```

```

95 #define TWI_MTX_DATA_ACK          0x28 // Data byte has been transmitted and
96   ACK received
96 #define TWI_MTX_DATA_NACK         0x30 // Data byte has been transmitted and
97   NACK received

98 // TWI Master Receiver status codes
99 #define TWI_MRX_ADR_ACK           0x40 // SLA+R has been transmitted and ACK
100  received
100 #define TWI_MRX_ADR_NACK          0x48 // SLA+R has been transmitted and
101  NACK received
101 #define TWI_MRX_DATA_ACK           0x50 // Data byte has been received and
102  ACK transmitted
102 #define TWI_MRX_DATA_NACK          0x58 // Data byte has been received and
103  NACK transmitted

104 // TWI Slave Transmitter status codes
105 #define TWI_STX_ADR_ACK            0xA8 // Own SLA+R has been received; ACK
106  has been returned
106 #define TWI_STX_ADR_ACK_M_ARB_LOST 0xB0 // Arbitration lost in SLA+R/W as
107  Master; own SLA+R has been received; ACK has been returned
107 #define TWI_STX_DATA_ACK            0xB8 // Data byte in TWDR has been
108  transmitted; ACK has been received
108 #define TWI_STX_DATA_NACK           0xC0 // Data byte in TWDR has been
109  transmitted; NOT ACK has been received
109 #define TWI_STX_DATA_ACK_LAST_BYTE 0xC8 // Last data byte in TWDR has been
110  transmitted (TWEA = 0); ACK has been received

111 // TWI Slave Receiver status codes
112 #define TWI_SRX_ADR_ACK            0x60 // Own SLA+W has been received ACK
113  has been returned
113 #define TWI_SRX_ADR_ACK_M_ARB_LOST 0x68 // Arbitration lost in SLA+R/W as
114  Master; own SLA+W has been received; ACK has been returned
114 #define TWI_SRX_GEN_ACK             0x70 // General call address has been
115  received; ACK has been returned
115 #define TWI_SRX_GEN_ACK_M_ARB_LOST 0x78 // Arbitration lost in SLA+R/W as
116  Master; General call address has been received; ACK has been returned
116 #define TWI_SRX_ADR_DATA_ACK        0x80 // Previously addressed with own SLA+
117  W; data has been received; ACK has been returned
117 #define TWI_SRX_ADR_DATA_NACK       0x88 // Previously addressed with own SLA+
118  W; data has been received; NOT ACK has been returned
118 #define TWI_SRX_GEN_DATA_ACK        0x90 // Previously addressed with general
119  call; data has been received; ACK has been returned
119 #define TWI_SRX_GEN_DATA_NACK       0x98 // Previously addressed with general
120  call; data has been received; NOT ACK has been returned
120 #define TWI_SRX_STOP_RESTART        0xA0 // A STOP condition or repeated START
121  condition has been received while still addressed as Slave

122 // TWI Miscellaneous status codes
123 #define TWI_NO_STATE               0xF8 // No relevant state information
124   available; TWINT = 0
124 #define TWI_BUS_ERROR              0x00 // Bus error due to an illegal START
125   or STOP condition

```

### C.1.2.3 TWI\_slave.c

..../Code/DualCamera\_UI/TWI\_slave.c

```
1  ****
2  *
3  * Atmel Corporation
4  *
5  * File : TWI_Slave.c
6  * Compiler : IAR EWAAVR 2.28a/3.10c
7  * Revision : Revision: 2475
8  * Date : Date: 2007-09-20 12:00:43 +0200 (to, 20 sep 2007)
9  * Updated by : Author: mlarsson
10 *
11 * Support mail : avr@atmel.com
12 *
13 * Supported devices : All devices with a TWI module can be used.
14 *                      The example is written for the ATmega16
15 *
16 * AppNote : AVR311 - TWI Slave Implementation
17 *
18 * Description : This is sample driver to AVRs TWI module.
19 *                 It is interrupt driven. All functionality is controlled
20 *                 through
21 *                 passing information to and from functions. See main.c for
22 *                 samples
23 *                 of how to use the driver.
24 */
25 */
26 * General disabling of MISRA rules:
27 * * (MISRA C rule 1) compiler is configured to allow extensions
28 * * (MISRA C rule 111) bit fields shall only be defined to be of type
29 *     unsigned int or signed int
30 * * (MISRA C rule 37) bitwise operations shall not be performed on signed
31 *     integer types
32 * As it does not work well with 8bit architecture and/or IAR
33 *
34 * Other disabled MISRA rules
35 * * (MISRA C rule 109) use of union - overlapping storage shall not be used
36 * * (MISRA C rule 61) every non-empty case clause in a switch statement shall
37 *     be terminated with a break statement
38 */
39
40 #include <avr/io.h>
41 #include <avr/interrupt.h>
42 #include "TWI_slave.h"
43
44 static unsigned char TWI_buf[TWI_BUFFER_SIZE];      // Transceiver buffer. Set
45     the size in the header file
46 static unsigned char TWI_msgSize = 0;                // Number of bytes to be
47     transmitted.
48 static unsigned char TWI_state = TWI_NO_STATE; // State byte. Default set
49     to TWI_NO_STATE.
50
51 // This is true when the TWI is in the middle of a transfer
52 // and set to false when all bytes have been transmitted/received
53 // Also used to determine how deep we can sleep.
54 static unsigned char TWI_busy = 0;
```

```

50 union TWI_statusReg_t TWI_statusReg = {0};           // TWI_StatusReg is
51   defined in TWI_Slave.h
52
53 ****
54 Call this function to set up the TWI slave to its initial standby state.
55 Remember to enable interrupts from the main application after initializing the
56   TWI.
57 Pass both the slave address and the requirements for triggering on a general
58   call in the
59 same byte. Use e.g. this notation when calling this function:
60 TWI_Slave_Initialise( (TWI_slaveAddress<<TWI_ADR_BITS) | (TRUE<<TWI_GEN_BIT) )
61 ;
62 The TWI module is configured to NACK on any requests. Use a
63   TWI_Start_Transceiver function to
64 start the TWI.
65 ****
66 void TWI_Slave_Initialise( unsigned char TWI_ownAddress )
67 {
68   TWAR = TWI_ownAddress;                                // Set own TWI slave
69   address. Accept TWI General Calls.
70   TWCR = (1<<TWEN)|                                     // Enable TWI-interface
71   and release TWI pins.
72   (0<<TWIE)|(0<<TWINT)|                                // Disable TWI Interrupt.
73   (0<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|                  // Do not ACK on any
74   requests, yet.
75   (0<<TWWC);                                         //
76   TWI_busy = 0;
77 }
78
79 ****
80 Call this function to test if the TWI_ISR is busy transmitting.
81 ****
82 unsigned char TWI_Transceiver_Busy( void )
83 {
84   return TWI_busy;
85 }
86
87 ****
88 Call this function to fetch the state information of the previous operation.
89   The function will hold execution (loop)
90 until the TWI_ISR has completed with the previous operation. If there was an
91   error, then the function
92 will return the TWI State code.
93 ****
94 unsigned char TWI_Get_State_Info( void )
95 {
96   while ( TWI_Transceiver_Busy() ) {}                   // Wait until TWI has
97   completed the transmission.
98   return ( TWI_state );                                // Return error state.
99 }
100
101 ****
102 Call this function to send a prepared message, or start the Transceiver for
103   reception. Include
104 a pointer to the data to be sent if a SLA+W is received. The data will be
105   copied to the TWI buffer.
106 Also include how many bytes that should be sent. Note that unlike the similar
107   Master function, the
108 Address byte is not included in the message buffers.

```

```

95  The function will hold execution (loop) until the TWI_ISR has completed with
96  the previous operation,
97  then initialize the next operation and return.
98  ****
99  void TWI_Start_Transceiver_With_Data( unsigned char *msg, unsigned char
100    msgSize )
101  {
102    unsigned char temp;
103
104    while ( TWI_Transceiver_Busy() ) {}                                // Wait until TWI is ready
105    for next transmission.
106
107    TWI_msgSize = msgSize;                                              // Number of data to transmit.
108    for ( temp = 0; temp < msgSize; temp++ )                            // Copy data that may be
109      transmitted if the TWI Master requests data.
110    {
111      TWI_buf[ temp ] = msg[ temp ];
112    }
113    TWI_statusReg.all = 0;
114    TWI_state         = TWI_NO_STATE ;
115    TWCR = (1<<TWEN)|                                         // TWI Interface enabled.
116          (1<<TWIE)|(1<<TWINT)|                               // Enable TWI Interrupt and
117          clear the flag.
118          (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|                // Prepare to ACK next time
119          the Slave is addressed.
120          (0<<TWWC);                                         //
121    TWI_busy = 1;
122  }
123
124  ****
125  Call this function to start the Transceiver without specifying new transmission
126  data. Useful for restarting
127  a transmission, or just starting the transceiver for reception. The driver
128  will reuse the data previously put
129  in the transceiver buffers. The function will hold execution (loop) until the
130  TWI_ISR has completed with the
131  previous operation, then initialize the next operation and return.
132  ****
133  void TWI_Start_Transceiver( void )
134  {
135    while ( TWI_Transceiver_Busy() ) {}                                // Wait until TWI is ready
136    for next transmission.
137    TWI_statusReg.all = 0;
138    TWI_state         = TWI_NO_STATE ;
139    TWCR = (1<<TWEN)|                                         // TWI Interface enabled.
140          (1<<TWIE)|(1<<TWINT)|                               // Enable TWI Interrupt and
141          clear the flag.
142          (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|                // Prepare to ACK next time
143          the Slave is addressed.
144          (0<<TWWC);                                         //
145    TWI_busy = 0;
146  }
147
148  ****
149  Call this function to read out the received data from the TWI transceiver
150  buffer. I.e. first call
151  TWI_Start_Transceiver to get the TWI Transceiver to fetch data. Then Run this
152  function to collect the
153  data when they have arrived. Include a pointer to where to place the data and
154  the number of bytes

```

```

139 to fetch in the function call. The function will hold execution (loop) until
140 the TWI_ISR has completed
141 with the previous operation, before reading out the data and returning.
142 If there was an error in the previous transmission the function will return
143 the TWI State code.
144 ****
145 unsigned char TWI_Get_Data_From_Transceiver( unsigned char *msg, unsigned char
146 msgSize )
147 {
148     unsigned char i;
149
150     while ( TWI_Transceiver_Busy() ) {} // Wait until TWI is ready
151     for next transmission.
152
153     if ( TWI_StatusReg.lastTransOK ) // Last transmission completed
154         successfully.
155     {
156         for ( i=0; i<msgSize; i++ ) // Copy data from Transceiver
157             buffer.
158         {
159             msg[ i ] = TWI_buf[ i ];
160         }
161         TWI_StatusReg.RxDataInBuf = FALSE; // Slave Receive data has been
162         read from buffer.
163     }
164     return( TWI_StatusReg.lastTransOK );
165 }

166 // ***** Interrupt Handlers **** //
167 ****
168 This function is the Interrupt Service Routine (ISR), and called when the TWI
169 interrupt is triggered;
170 that is whenever a TWI event has occurred. This function should not be called
171 directly from the main
172 application.
173 ****
174 ISR(TWI_vect)
175 {
176     static unsigned char TWI_bufPtr;

177     switch (TWSR)
178     {
179         case TWI_STX_ADR_ACK: // Own SLA+R has been received; ACK has
180         been returned
181         // case TWI_STX_ADR_ACK_M_ARB_LOST: // Arbitration lost in SLA+R/W as
182         Master; own SLA+R has been received; ACK has been returned
183         TWI_bufPtr = 0; // Set buffer pointer
184         to first data location
185         case TWI_STX_DATA_ACK: // Data byte in TWDR has been transmitted
186         ; ACK has been received
187         TWDR = TWI_buf[TWI_bufPtr++];
188         TWCR = (1<<TWEN)| // TWI Interface
189         enabled
190         (1<<TWIE)|(1<<TWINT)| // Enable TWI Interrupt
191         and clear the flag to send byte
192         (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)| // 
193         (0<<TWWC); // 
194         TWI_busy = 1;

```

```

183     break;
184     case TWI_STX_DATA_NACK:           // Data byte in TWDR has been transmitted
185     ; NACK has been received.
186                                         // I.e. this could be the end of the
187                                         // transmission.
188     if (TWI_bufPtr == TWI_msgSize) // Have we transceivied all expected data?
189     {
190         TWI_statusReg.lastTransOK = TRUE;           // Set status bits to
191                                         completed successfully.
192     }
193     else                               // Master has sent a NACK before all data
194     where sent.
195     {
196         TWI_state = TWSR;                   // Store TWI State as
197                                         errormessage.
198     }

199     TWCR = (1<<TWEN)|                         // Enable TWI-
200                                         interface and release TWI pins
201                                         (1<<TWIE)|(1<<TWINT)|           // Keep interrupt
202                                         enabled and clear the flag
203                                         (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|   // Answer on next
204                                         address match
205                                         (0<<TWWC);                      //

206     TWI_busy = 0;    // Transmit is finished, we are not busy anymore
207     break;
208     case TWI_SRX_GEN_ACK:           // General call address has been received
209     ; ACK has been returned
210 //     case TWI_SRX_GEN_ACK_M_ARB_LOST: // Arbitration lost in SLA+R/W as
211 //     Master; General call address has been received; ACK has been returned
212 //     TWI_statusReg.genAddressCall = TRUE;
213 //     case TWI_SRX_ADR_ACK:           // Own SLA+W has been received ACK has
214 //     been returned
215 //     case TWI_SRX_ADR_ACK_M_ARB_LOST: // Arbitration lost in SLA+R/W as
216 //     Master; own SLA+W has been received; ACK has been returned
217                                         // Dont need to clear
218                                         TWI_S_StatusRegister.generalAddressCall due to that it is the default
219                                         state.
220     TWI_statusReg.RxDataInBuf = TRUE;
221     TWI_bufPtr = 0;                  // Set buffer pointer
222     to first data location

223                                         // Reset the TWI
224                                         Interrupt to wait for a new event.
225     TWCR = (1<<TWEN)|           // TWI Interface
226                                         enabled
227                                         (1<<TWIE)|(1<<TWINT)|           // Enable TWI Interrupt
228                                         and clear the flag to send byte
229                                         (1<<TWEA)|(0<<TWSTA)|(0<<TWSTO)|   // Expect ACK on this
230                                         transmission
231                                         (0<<TWWC);
232     TWI_busy = 1;

233     break;
234     case TWI_SRX_ADR_DATA_ACK:      // Previously addressed with own SLA+W;
235     data has been received; ACK has been returned
236     case TWI_SRX_GEN_DATA_ACK:      // Previously addressed with general call
237     ; data has been received; ACK has been returned

```

```

221     TWI_buf[TWI_bufPtr++] = TWDR;                                // Set flag
222     TWI_statusReg.lastTransOK = TRUE;                            // transmission successfull.
223                                         // Reset the TWI
224     TWCR = (1<<TWEN) |                                         // TWI Interface
225     enabled;                                                     // TWIE | (1<<TWINT) | // Enable TWI Interrupt
226     and clear the flag to send byte
227     (1<<TWEA) | (0<<TWSTA) | (0<<TWSTO) | // Send ACK after next
228     reception
229     (0<<TWWC);                                                 //
230     TWI_busy = 1;
231     break;
232 case TWI_SRX_STOP_RESTART: // A STOP condition or repeated START
233 condition has been received while still addressed as Slave
234                                         // Enter not addressed
235 mode and listen to address match
236     TWCR = (1<<TWEN) |                                         // Enable TWI-
237 interface and release TWI pins
238     (1<<TWIE) | (1<<TWINT) | // Enable interrupt
239 and clear the flag
240     (1<<TWEA) | (0<<TWSTA) | (0<<TWSTO) | // Wait for new
241 address match
242     (0<<TWWC);                                                 //
243
244     TWI_busy = 0; // We are waiting for a new address match, so we are not
245 busy
246
247     break;
248 case TWI_SRX_ADR_DATA_NACK: // Previously addressed with own SLA+W;
249 data has been received; NOT ACK has been returned
250 case TWI_SRX_GEN_DATA_NACK: // Previously addressed with general call
251 ; data has been received; NOT ACK has been returned
252 case TWI_STX_DATA_ACK_LAST_BYTE: // Last data byte in TWDR has been
253 transmitted (TWEA = 0); ACK has been received
254 // case TWI_NO_STATE // No relevant state information
255 available; TWINT = 0
256 case TWI_BUS_ERROR: // Bus error due to an illegal START or STOP
257 condition
258     TWI_state = TWSR; //Store TWI State as errormessage,
259 operation also clears noErrors bit
260     TWCR = (1<<TWSTO) | (1<<TWINT); //Recover from TWI_BUS_ERROR, this
261 will release the SDA and SCL pins thus enabling other devices to use the
262 bus
263     break;
264 default:
265     TWI_state = TWSR; // Store TWI State as
266 errormessage, operation also clears the Success bit.
267     TWCR = (1<<TWEN) | // Enable TWI-
268 interface and release TWI pins
269     (1<<TWIE) | (1<<TWINT) | // Keep interrupt
270 enabled and clear the flag
271     (1<<TWEA) | (0<<TWSTA) | (0<<TWSTO) | // Acknowledge on any
272 new requests.
273     (0<<TWWC); //
274
275     TWI_busy = 0; // Unknown status, so we wait for a new address match that
276 might be something we can handle

```

```
256     }
257 }
```

## C.2 MATLAB Code for Image Algorithm Prototyping

### C.2.1 Image Matching Algorithms

#### C.2.1.1 loadimages.m

..../MATLAB/loadimages.m

```
1 left = imread('viprectification_deskLeft.png');
2 right = imread('viprectification_deskRight.png');

4 % left = imread('battery_left.bmp');
5 % right = imread('battery_right.bmp');

7 % left = imread('square_left.bmp');
8 % right = imread('square_right.bmp');

10 % left = imread('fiftycm_left.bmp');
11 % right = imread('fiftycm_right.bmp');

13 % left = imread('2objs_left.bmp');
14 % right = imread('2objs_right.bmp');
```

#### C.2.1.2 GetSubImage.m

..../MATLAB/GetSubImage.m

```
1 function [ SubImage ] = GetSubImage( Image, BoxSize, StartCoordinates )
2 %GETSUBIMAGE Returns a sub section of the image according to the other
3 %inputs
4 % Image - The image of which a subimage is to be taken from
5 % BoxSize - A 2x1 matrix containing the size of the subImage
6 % StartCoordinates - A 2x1 matrix with the start point of the image
7 % Dimensions - How many planes - 3 for colour, 1 for grey scale

9 XLow = StartCoordinates(1)-(BoxSize(1)/2);
10 YLow = StartCoordinates(2)-(BoxSize(2)/2);
11 if(XLow<1)
12     XLow = 1;
13 end

15 if(YLow < 1)
16     YLow = 1;
17 end

19 XHigh = XLow + BoxSize(1);
20 YHigh = YLow + BoxSize(2);
```

```

21 [~, ~, LZ] = size(Image);
22
23 %SubImage = zeros(BoxSize);
24 for i = XLow:XHigh
25     for j = YLow:YHigh
26         if LZ == 3
27             for z = 1:3
28                 SubImage(i-XLow+1,j-YLow+1,z) = Image(i,j,z);
29             end
30         elseif LZ == 1
31             SubImage(i-XLow+1,j-YLow+1) = Image(i,j);
32         else
33             error('Number of Dimensions "%d" are not supported', LZ);
34         end
35     end
36 end
37
38 end

```

### C.2.1.3 SADAll.m

..../MATLAB/SADAll.m

```

1 %function [ Results ] = SADAll( Left, Right )
2 %SADALL Function to compute all SADs of an image
3 % The sum of absolute differences is calculated and returned on a mesh
4 % graph to show how well matched the sub image is to the image. A box out
5 % of the right image is taken and compared with the left image.
6 loadimages;
7 BoxSize = [50,50];
8 [~,~,C] = size(right);
9 [I,J,D] = size(left);
10 if C ~= D
11     error('Images have different number of colour planes');
12 end
13
14 RightSub = GetSubImage(right, BoxSize, [190,190]);
15
16 for i = 25:(I-25)
17     for j = 25:(J-25)
18         LeftSub = GetSubImage(left, BoxSize, [i, j]);
19         Diff = LeftSub - RightSub;
20
21         Results(i,j) = sum(Diff(:));
22     end
23 end
24
25 %Display
26 figure;
27 subplot(2,2,1);
28 imshow(left);
29 title('Left Image');
30
31 subplot(2,2,2);
32 imshow(right);
33 title('Right Image');

```

```

35 subplot(2,2,3);
36 imshow(RightSub);
37 title('Right Sub');

39 figure;
40 surf(Results);
41 shading flat;
42 %end

```

#### C.2.1.4 SSDAll.m

..//MATLAB/SSDAll.m

```

1 %function [ Results ] = SADALL( Left , Right )
2 %SADALL Function to compute all SADs of an image
3 % The sum of absolute differences is calculated and returned on a mesh
4 % graph to show how well matched the sub image is to the image. A box out
5 % of the right image is taken and compared with the left image.
6 loadimages;
7 BoxSize = [50,50];
8 [~,~,C] = size(right);
9 [I,J,D] = size(left);
10 if C ~= D
11     error('Images have different number of colour planes');
12 end

14 RightSub = GetSubImage(right, BoxSize, [190,190]);

16 for i = 25:(I-25)
17     for j = 25:(J-25)
18         LeftSub = GetSubImage(left, BoxSize, [i, j]);
19         Diff = LeftSub - RightSub;
20         Diff = Diff.^2;
21         Results(i,j) = sum(Diff(:));
22     end
23 end

25 %Display
26 figure;
27 subplot(2,2,1);
28 imshow(left);
29 title('Left Image');

31 subplot(2,2,2);
32 imshow(right);
33 title('Right Image');

35 subplot(2,2,3);
36 imshow(RightSub);
37 title('Right Sub');

39 figure;
40 surf(Results);
41 shading flat;
42 %end

```

### C.2.1.5 NCC.m

..//MATLAB/NCC.m

```

2 loadimages;
3 show;
4 BoxSize = [50,50];
5 MaxConfMatches = 20;
6 %SubCoord = [145, 300];
7 figure(1);
8 %[rightSub, rect_Sub] = imcrop(right);
9 figure(2);
10 imshow(right);
11 rSubCoord = ginput(1);
12 rSubCoord = [190,190]; [rSubCoord(2), rSubCoord(1)];
13 rSubCoord = round(rSubCoord);
14 close;
15 tic;
16 rightSub = GetSubImage(right, BoxSize, rSubCoord);
17 %imshow(rightSub);
18 rightSubGray = rgb2gray(rightSub);
19 leftGray = rgb2gray(left);
20 rightGray = rgb2gray(right);
21 cL = normxcorr2(rightSubGray(:,:,1), leftGray(:,:,1));
22 figure(2);
23 % subplot(1,2,1);
24 surf(cL), shading flat;
25 title('Normalised Cross Correlation of Right Sub and Left Image');
26 toc;
27 % cR = normxcorr2(rightSubGray(:,:,1), rightGray(:,:,1));
28 % subplot(1,2,2);
29 % surf(cR), shading flat;
30 % title('Normalised Cross Correlation of Right Sub and Right Image');

32 % cD = cL - cR;
33 %
34 % figure;
35 % surf(cD), shading flat;
36 % title('Differences of the Normalised Cross Correlation of Right and Left');

38 %Find coordinates of best match.
39 [Y,X] = size(cL);
40 maxValue = 0;
41 LeftMatchCoord = [0,0];
42 NumConfidentMatches = 0;

44 for i = 1:X
45     for j = 1:Y
46         Val = cL(j,i);
47         if Val > 0.9
48             NumConfidentMatches = NumConfidentMatches + 1;
49         end
50         if Val > maxValue

```

```
51         maxValue = Val;
52         LeftMatchCoord = [j-(BoxSize(1) / 2), i-(BoxSize(2) / 2)];
53     end
54 end
55 end

56 Result = [maxValue, LeftMatchCoord];
57 figure(1);
58 if NumConfidentMatches >= 1 && NumConfidentMatches < MaxConfMatches
59     left(LeftMatchCoord(1)-(BoxSize(1)/2):LeftMatchCoord(1)+(BoxSize(1)/2),
60          LeftMatchCoord(2)-(BoxSize(2)/2)=255;
61     left(LeftMatchCoord(1)-(BoxSize(1)/2):LeftMatchCoord(1)+(BoxSize(1)/2),
62          LeftMatchCoord(2)+(BoxSize(2)/2)=255;
63     left(LeftMatchCoord(1)-(BoxSize(1)/2),LeftMatchCoord(2)-(BoxSize(2)/2):
64          LeftMatchCoord(2)+(BoxSize(2)/2)=255;
65     left(LeftMatchCoord(1)+(BoxSize(1)/2),LeftMatchCoord(2)-(BoxSize(2)/2):
66          LeftMatchCoord(2)+(BoxSize(2)/2))=255;

67     right(rSubCoord(1)-(BoxSize(1)/2):rSubCoord(1)+(BoxSize(1)/2),rSubCoord(2)
68          -(BoxSize(2)/2))=255;
69     right(rSubCoord(1)-(BoxSize(1)/2):rSubCoord(1)+(BoxSize(1)/2),rSubCoord(2)
70          +(BoxSize(2)/2))=255;
71     right(rSubCoord(1)-(BoxSize(1)/2),rSubCoord(2)-(BoxSize(2)/2):rSubCoord(2)
72          +(BoxSize(2)/2))=255;
73     right(rSubCoord(1)+(BoxSize(1)/2),rSubCoord(2)-(BoxSize(2)/2):rSubCoord(2)
74          +(BoxSize(2)/2))=255;

75 subplot(1,2,1);
76 imshow(left);
77 subplot(1,2,2);
78 imshow(right);
79 % LeftMatchCoord
80 % rSubCoord
81 % NumConfidentMatches
82 Distance = Range(rSubCoord(2), LeftMatchCoord(2));
83 sprintf('Distance to Object = %d metres', Distance)
84 elseif NumConfidentMatches >= MaxConfMatches
85     title(sprintf('Too many matches found : %d', NumConfidentMatches));
86 else
87     title(sprintf('No Reliable Match Found'));
88 end
```



# References

- Atmel Corporation. *AVR311: TWI Slave*, 2007.
- Atmel Corporation. *AT32UC3C0512C Datasheet*, 2012a.
- Atmel Corporation. *ATMega644P Datasheet*, 2012b.
- Atmel Corporation. [Atmel software framewrok](#), 2012c.
- Atmel Corporation. *ATXMega256A3BU Datasheet*, 2012d.
- Electronic Lives Manufacturing. [Fatfs - generic fat file system module](#), 2012.
- Farnell. [Farnell online store](#), 2012.
- François Fleuret, Jérôme Berclaz, Richard Lengagne, and Pascal Fua. Multi-camera people tracking with a probabilistic occupancy map. Technical report, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 2007.
- 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 stereovision 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.
- Microsoft. [Xbox 360 kinect](#), 2012.
- 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.
- Phillips. *PCA9542A : 2-channel I2C-bus multiplexer and interrupt logic*, 2009.

- M. Ruhnke, R. Kummerle, G. Grisetti, and W. Burgard. Highly accurate maximum likelihood laser mapping by jointly optimizing laser points and robot poses. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2812–2817. IEEE, 2011.
- Ashutosh Saxena, Jamie Schulte, and Andrew Y. Ng. Depth estimation using monocular and stereo cues, 2007.
- Stephen Se, David Lowe, and Jim Little. Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks. *The International Journal of Robotics Research*, pages 735–758, 2002.
- Texas Instruments. *Stellaris LM3S9B96 Datasheet*, 2012.
- Sebastian Thrun and Michael Montemerlo. The graphslam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, pages 403–428, 2006.
- 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.
- K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, Anchorage, AK, USA, May 2010. Software available at <http://octomap.sf.net/>.