Design, Manufacture and Testing of a Robot System for the UWA EyeBot Program

Andrew Adamson

20276755

School of Mechanical and Chemical Engineering

University of Western Australia

Supervisor: Prof. Thomas Bräunl

School of Electrical, Electronic and Computer Engineering

University of Western Australia

Final Year Project Thesis

School of Mechanical and Chemical Engineering

University of Western Australia

Submitted: November 12th, 2012

Abstract

The EyeCon is a controller for mobile robots (EyeBots) which are designed as a small, low powered, embedded device with good image processing capabilities. EyeCons are a valuable tool for student learning in the laboratory component of several robotics units, as well as providing control and monitoring functionality on other research robots, where they have been attached to wheeled, walking and swimming devices. The EyeCons in general use were created prior to 2006 and as such, all of them are becoming impractical to maintain and the hardware is outdated.

This project investigated a completely new EyeCon architecture based on a BeagleBoard xM single board computer. A custom USB expansion board was then designed to add EyeBot specific functionality to the BeagleBoard. The redesigned architecture was required to satisfy the needs of a large number of stakeholders while achieving modularity and standardisation along with reduced construction and development time.

After manufacturing and assembly, testing was conducted on the expansion board to confirm the functionality of individual features and ensure that the expansion board was suitable for use by undergraduates. Testing showed that the architectural principle was sound, however a number of technical issues were found that require a second iteration of the board to resolve. Suggestions are provided on how each technical issue may be resolved, and recommendations are presented as a framework for future EyeCon development.

`

Acknowledgements

First and foremost I would like to thank The University Computer Club (UCC), without which I would not have had the practical skills and knowledge to do this project. The UCC was also a huge help in providing tools, advice and resources that I was unable to access through the Engineering Faculty. The members of UCC were also a huge help in keeping me sane throughout what was a very challenging project, and had to put up with me practically living in one corner of the room to work on this project. One such member, John Hodge, wrote the code for testing the SPI chip when I discovered it was well outside my field of knowledge. At short notice, he quickly and expertly wrote a test program, for which I am extremely grateful.

I would also like to thank Ivan Neubronner for his help and advice with the PCB; it is largely thanks to his input that the PCB had so few manufacturing issues. He also stopped me from making some questionable design decisions, like making the components half the size they ended up being.

Thanks go to my family, who listened and supported me when I needed somebody to talk at, and were very understanding of the time I was spending on the project.

Finally, thanks go to Prof. Dr. Thomas Bräunl, who let me throw myself in the deep end with this project even after I asked what a decoupling capacitor was - I have learned a tremendous amount because of it!

Contents

1 Nomenclature

ADC	Analogue to Digital Converter
BIOS	Basic Input Output System
CAD	Computer Aided Design
CPU	Central Processing Unit
COM	Computer On Module
DSP	Digital Signal Processor
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input/Output
HID	Human Interface Device
I^2C	Inter-Integrated Circuit
I/O	Input/Output
JTAG	Standard Test Access Port and Boundary-Scan Architecture
LCD	Liquid Crystal Display
LED	Light Emitting Diode
PCB	Printed Circuit Board
PDI	Programming and Debugging Interface
PID	Proportional-Integral-Derivative
PSD	Position Sensing Device
PWM	Pulse Width Modulation
RAM	Random Access Memory
REV	UWA Renewable Energy Vehicle
RoHS	Restriction of Hazardous Substances Directive
ROS	Robot Operating System
SBC	Single Board Computer
SDK	Software Development Kit
SPI	Serial Peripheral Interface Bus
TVS	Transient Voltage Suppression
USB	Universal Serial Bus
USB OTG	USB On The Go
UVC	USB Video Device Class

2 Introduction

EyeBots are a class of small autonomous robots which also have real-time image processing capabilities. The controller at the heart of all EyeBots is called an EyeCon, however the terms “EyeBot” and “EyeCon” are frequently interchanged. The UWA EyeBot program has been running for a number of years and EyeCons have been attached to a wide range of wheeled, walking, swimming and flying robots. The program started as a way to teach robotics principles; students program EyeBots in assembly and the C programming languages for tasks such as line following, wall following, sensor calibration, motor Proportional-Integral-Derivative (PID) control and object tracking. In addition to their use in robotics lab programs, EyeCons are also used as control and monitoring tools in projects such as UWA’s Renewable Energy Vehicle, stereo vision research using Field Programmable Gate Arrays (FPGAs), and swarm robotics research.

EyeCons have undergone several upgrades during the life of the program. The original EyeCons, versions M1 to M5 (hereafter series 1), were based on a Motorola 68k processor and ran a fully custom operating system. Series 1 EyeCons ran at 25MHz, and had a 128x64 pixel black and white screen. In 2006, the M6 EyeCon was developed to try and replace the ageing series 1 fleet of EyeBots. The M6 is based on an ARM-9 processor running at 400MHz and runs a customised version of the Linux operating system. The M6 also had an FPGA connected to the main processor which allowed it to offload the image processing and free up the Central Processing Unit (CPU) for other tasks such as controlling the robot. Alas, this approach had hardware and useability issues which made it inappropriate for larger scale deployment, and consequently only a few EyeCon M6s were ever made. At the time of writing, students completing robotics labs at UWA are still using series 1 EyeCons.

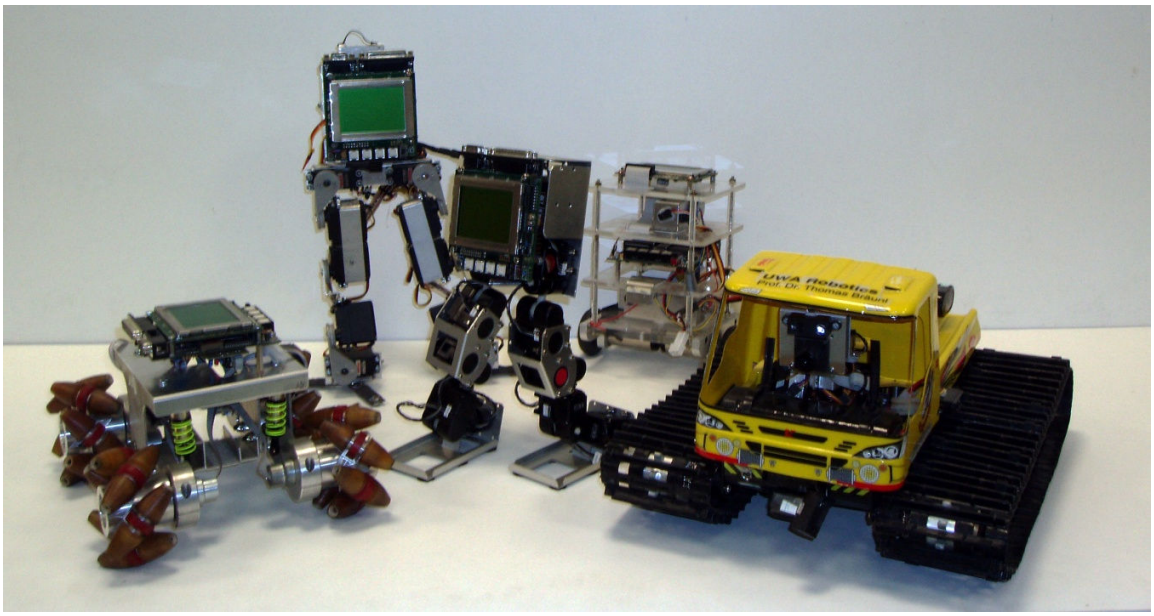


Figure 1: A selection of devices that the EyeCon has been attached to

When the EyeCon was first designed, it was the pioneering robot controller of its size which could do onboard image processing. Today, devices with high CPU speeds, hundreds of megabytes of Random Access Memory (RAM), and onboard image processing capabilities are ubiquitous in the form of mobile phones and tablet devices. The series 1 EyeCons, whilst still effective at analogue tasks such as driving motors and reading distance sensors, have failed to keep up with today's expectations in terms of image processing capabilities. It is now nearly impossible to update the series 1 EyeCons with new components; in particular the camera modules, for which camera sensors of a low enough resolution are difficult to source.

2.1 Design Specification

The design of the new system aims to be a general robotics platform that is able to take over the tasks of existing EyeBots whilst giving the system sufficient features and expandability that it can be adapted to other uses in the Engineering Faculty.

2.2 Project Scope

The main tasks in this project were:

- Research and definition of hardware, taking into consideration present and future needs for the EyeCon with respect to product usage and life,
- Acquisition of off-the-shelf hardware, then design and manufacture of any additional custom hardware that is required,
- Testing and documentation to allow others to continue working with the project.

Notable exclusions from the scope are the writing of software (beyond test code to ensure the hardware works), the definition of the communication protocol between the host board and the expansion board, and large scale deployment systems.

2.3 Major Contributions

The major contributions of this project are:

- The architectural design of the EyeCon M8,
- Sourcing of a host device to provide image processing capabilities,
- The schematic and Printed Circuit Board (PCB) layout of the EyeCon M8 expansion board,
- A bill of materials and sourcing of Restriction of Hazardous Substances Directive (RoHS) compliant components for the expansion board (supplied to supervisor),
- Outsourcing the manufacture of the expansion board PCB,
- Population of two expansion board PCBs,
- Testing results of the expansion board,
- Documentation and sample code to facilitate future programming and use of the expansion board.

3 Architectural Design

The architectural design of the EyeCon M8 had to take into account the needs of all stakeholders, whilst creating a product with sufficient life and cost effectiveness. The primary use scenario focused on was that of student labs, since that is where the largest number of EyeBots is currently deployed. Some consideration was also given to other uses, based on an assortment of past research projects which used the EyeBot.

The focus of the EyeCon M6 design was on "the ability to accelerate image processing through the use of an FPGA" (Blackham, 2006). The M8 does almost the opposite – it keeps the high speed tasks such as image processing on the main CPU, and delegates the slower robot control tasks to the microcontroller on the expansion board. This has the benefit of abstracting EyeBot specific tasks away from what is otherwise off-the-shelf hardware.

3.1 Product Lifetime Optimization

van Nes and Cramer (2005) outlined five design strategies which influence product lifecycle through design: design for reliability and robustness; design for repair and maintenance; design for upgradability; design for product attachment and design for variability.

Previous EyeCons have had a service life of approximately five years, and it is envisaged that the new design will match or exceed this. Making the EyeCon modular is consistent with all the strategies set out by van Nes and will help to maximize the EyeCon's lifecycle. Hardware modularity can be achieved by separating hardware roles where they cannot easily be replaced as a whole, for example, it is ideal to put most of the robot control hardware (motor drivers, servo drivers, distance sensors) onto an expansion module with a standard interface such as Universal Serial Bus (USB) so that, when it comes to upgrading the image processing capabilities, the expansion board can simply be plugged into the new host, have suitable drivers compiled and then used without further redevelopment. Modularising the robot controlling role potentially saves on having to build a fully custom EyeCon board every time a CPU upgrade is required, thus allowing the EyeCon to be

incrementally upgraded. Furthermore, abstracting the robot driving component out of the main board will make it easier to design for electrical protection.

3.2 Requirements

3.2.1 Undergraduate Lab User Requirements

A student completing labs with an EyeBot requires a tool that is reliable, consistent and easy to program. At present, the most complex of undergraduate lab tasks undertaken using EyeBots involves locating red cans using a camera, driving towards them, picking them up, and moving them to some location (Bräunl, 2012). For the EyeBot M8 to take over the task of existing labs it does not need stereo cameras or the ability to process high quality video from high resolution/high frame-rate cameras. As a result it must be asked why an FPGA is needed for these tasks at all. Indeed, all that is needed for student labs is a way to apply simple image processing to a stream from a single camera (e.g. find the lightest point), and the ability to control all peripherals on existing EyeBot bases (motors, servos and distance sensors).

Features implemented in previous EyeCons that should be re-implemented:

- Liquid Crystal Display (LCD),
- 4 DC motor drivers,
- 16 General Purpose I/O (GPIO),
- 6 analogue inputs,
- 6 position sensitive device (PSD) inputs
- Rotary encoder inputs to read the motors rotation,
- 14 servo motor drivers,
- RS-232 serial,
- Colour camera.

Prominent features upgraded from series 1 EyeBots:

- Move from a black and white LCD screen to a full colour screen,
- Remove mechanical buttons from below LCD screen in favour of touch screen buttons,

- Move to USB interface to allow for a greater range of peripheral hardware,
- Add options of programming via USB, Ethernet or SD card.

3.2.2 Requirements of Research Users

The needs of a research user are more difficult to predict than any other due to their ad-hoc nature. The previous research uses of the EyeCon to date can be split into three categories:

- A small vehicle/robot controller (e.g. soccerbots and other cooperative robotics (Du, 2003))
- Monitoring and recording (e.g. a blackbox for the REV project (Ewan MacLeod, 2008))
- Computer vision and stereo vision research (Chin, 2006)

Using the EyeCon as a vehicle/robot controller is something that requires it to be robust and tolerant of vibration. It also needs to have suitable I/O to be able to read sensors, control motors and servos, and have sufficiently long battery life. Ideally, the M8 should be compatible with any existing mounting constraints by matching mounting-hole locations and the physical dimensions as closely as possible.

Monitoring and recording using the EyeCon dictates that it has some sort of storage space available for data, in addition to having enough Input/Ouput (I/O) for sensors to monitor the object in question. For the EyeCon to be used in vehicles such as the Renewable Energy Vehicle (REV), it is required to be compatible with typical automotive voltages (5 to 15V depending on conditions).

The computer vision and stereo vision research requirements are the most difficult to quantify, since the standard of cameras, their interfaces, and the specifications of the FPGA required for some tasks are difficult to predict. It was decided that trying to satisfy the desire for an FPGA in every single EyeCon is both unnecessary and expensive; it increases development times, hardware maintenance time costs, forces the system to be unnecessarily complex, and is a resource that is not required to be on the EyeBot for the majority of its use. As such, the FPGA component of an EyeCon was excluded from the scope of this

project. Where there is a need for high powered image processing, it can be implemented in future as a USB expansion module (Bailey, 2011, pp. 380-383), which leaves the option open for a multitude of different camera interfaces; FireWire, CameraLink, SubMiniature, Ethernet, etc. It is essential that the interface between the EyeCon and such an expansion module be considered when the EyeBot M8 is being built. Due to bottlenecks in the I/O of the EyeCon M6, (Dietrich, 2009) calculated in his final year thesis that 25MB/s could be passed between the FPGA and the CPU, and this was judged to be sufficient. Considering Dietrich's finding and the speed of the newer USB 2.0 standard (which operates at a maximum rate of 50MB/s), a USB interface would be sufficient for communication between an FPGA and the CPU if it is ever implemented.

3.2.3 Faculty/University Requirements

The ultimate goal of the EyeCon M8 is for it to replace the collection of EyeBot M5's that are utilised at the university. Aside from the cost price of the hardware, deploying a large number of EyeBots at once requires personnel time to do any necessary assembly, modifications and programming. There is a high potential for this process to become expensive very quickly, therefore it is imperative that the hardware is not only cost effective, but easy to assemble, with fast programming and maintenance processes.

3.3 Constraints

- Power consumption:
    - EyeBots are generally battery operated, power consumption needs to be minimal for them to be able to function throughout a three-hour lab without having a battery replacement.
- EyeCon size:
    - To minimize the amount of existing hardware necessary to be changed, making the board the same size or smaller will enable the M8 to be used in almost every place the M5 is used,
    - The new EyeCon must have the same mounting hole locations as previous versions.

- Cost:
    - The design must be as cost effective as is practicable,
    - Be sufficiently robust to be minimal maintenance and hence lower cost.
- Usability:
    - The EyeCon M8 must be at least as simple to program as the EyeCon M5 (from a user perspective),
    - Bulk programming methods must be available.
- Easy to source and manufacture:
    - All parts should be replaceable with functionally similar parts,
    - Minimise complexity to allow for manufacturing by a range of board houses.
- The design must be fully RoHS compliant

3.4 Past Designs

In addition to showing us the requirements for a new design, examining previous EyeBot designs can help to better new designs by improving on past flaws and incorporating desirable characteristics from past designs. Previous EyeCon designs can be broken into series 1 EyeCons and series 2 EyeCons. This split of series represents a radical architectural change. Series 1 ran a custom Basic Input Output System (BIOS) on the Motorola 68332 processor and can be considered a 'traditional' embedded system in that it had very dedicated and fixed functions. Series 2 moved towards a more generic system running Linux on a much more powerful ARM-9 PXA255 processor (Blackham, 2006).

3.4.1 Series 1

Series 1, despite being relatively simple and having low processing power, was a device that was easy for beginners to learn to program; this made it ideal for student use (based on first-hand experience). This is partly due to its hardware design, which uses simple interfaces, and partly due to good documentation with plenty of examples. This high useability level is considered essential for the M8 to be successful for student use.

What is noteworthy of Series 1 is that the more modern modules (such as the Bluetooth module) have been implemented using RS-232 serial via a DE-9 connector (Bräunl, 2008). The standard serial interface on the EyeCon would have made these add-on modules simple to implement, and could be substituted with any off-the-shelf serial module, which in turn contributed to the longevity of the Series 1. While RS-232 is a reliable standard that is easy to work with, it is also extremely slow. USB has become a de facto standard in the embedded electronics industry, has very mature Linux support, and is what the M8 will use (Yaghmour, 2009). A further benefit of USB is that a USB-serial adapter can be used for backward compatibility with any old serial modules.

Despite having some standard interfaces, the most important interface of the series 1 EyeCons – the camera – was non-standard, making it difficult to replace old hardware as it reached end of life. Every new camera required a new board, had to be of a very specific type, and required the hardware description table to be updated in software. Given that available hardware is now sufficiently fast, an off the shelf camera using USB can be used instead of interfacing to a custom camera using parallel I/O. This permits the use of existing, standard software drivers such as Linux's USB Video Device Class (UVC) video drivers, which require no reconfiguration if the camera is changed.

3.4.2 Series 2

Series 2 made a radical change towards a Computer on Module (COM) mounted on an I/O expansion board, which also contained an FPGA. Running Linux on the M6 meant that there was already an existing software base to build on which was both free and open source. As a result, programming work wasn't being repeated unnecessarily, and costs were drastically reduced compared to a proprietary operating system. Running Linux meant that there was already a multitude of programming tools available, and the system could be emulated easily.

The major downside to the M6 was that, though powerful, it was not suitable to replace the Series 1 EyeBots. At the time, attempting to satisfy the stereo vision goal meant using an FPGA.

- The FPGA took on too many functions of the old EyeBot by trying to do both the image processing and low speed I/O (e.g. servo) control. This meant that it could only do one of these tasks at a time,
- Changing any of the hardware attached to the FPGA meant reprogramming it, which represented an inordinate amount of programming for what should be a simple change,
- The FPGA used so many data lines on CPU that there was little room for anything else to be attached to the CPU,
- A timing issue between the CPU and the FPGA meant that the FPGA could not be used to its full potential (Geier, 2009),
- There were so many peripherals connected to the FPGA that the system was unusable when the FPGA was put into low power mode (Singh, 2011).

3.5 Architecture Options

There were a number of possible design approaches that could be undertaken to arrive at a finished product, however it was important to consider how much of the EyeCon should be off-the-shelf and how much of it should be custom designed. At first glance, one might choose the approach that involves the least work and the shortest development time. However, each approach has trade-offs that had to be considered to ensure the product lifecycle is optimized and all the requirements of the EyeCon are satisfied. The following three subsections detail some of the benefits and disadvantages of each approach, from which option 3 was selected for the EyeCon M8.

3.5.1 Option 1: Design a complete EyeCon from scratch

- The product can suit the requirements exactly, with higher efficiency and without any unnecessary components,
- Takes significantly longer to develop and needs to include longer testing time,
- High likelihood of having to go through several iterations of hardware to arrive at the final product, which then has to go for manufacturing – all of which needs someone to be managing the process,

- Potential to spend significant amounts of time and money reproducing parts that are available off-the-shelf,
- Some technologies difficult to work with given available resources (ball-grid array parts, package-on-package designs),
- Have to adapt all software drivers to suit the board,
- High dependence on individual parts availability (high externalities).

3.5.2 Option 2: Design a board which expands the I/O of a COM

Note: This was the EyeCon M6 approach, which expanded on a Gumstix board.

- Shorter development time than a fully custom build,
- More software already available for the COM and some documentation already written, though most external chip drivers still need to be implemented,
- Unless there is a standard interface between the COM and the I/O board, if the COM changes (as happened with Gumstix board connected to the M6), the I/O board requires a complete redesign,
- Can eliminate unnecessary ports and have more direct access to the CPU,
- The functionality offered by each COM is highly variable. For example, one Gumstix model may have USB onboard, while the next may expect it to be implemented externally,
- Guaranteed to have the right size board to suit mounting on existing EyeBots,
- Have to adapt all software drivers to suit the board,
- Has a high dependence on individual parts availability (high externalities).

3.5.3 Option 3: Buy a Single Board Computer (SBC) off the shelf and add any required functionality using add-on modules

- Much faster development time; can make most full-sized SBCs run Linux out of the box and all drivers are already installed or available,
- Once any custom modules and drivers have been developed, they can easily be moved to other platforms as long as they use a standard interface such as USB,
- High likelihood of having unnecessary components and features,

- Difficult to find SBCs and screens of the correct size for EyeBot mounting (will have to make adaptor brackets or modify existing EyeBots to suit),
- Positions of ports are out of our control which could cause mounting incompatibilities if positions change in future releases,
- Very few SBCs have built-in FPGAs, so that would have to be created as an add-on module.

3.6 Design

For the EyeCon M8 it was decided to take an off-the-shelf SBC approach and use a BeagleBoard-xM as a host device, with an expansion board custom made to suit EyeBot specific functionality. Vision capabilities can then be provided by any Linux-compatible USB camera, however preference should be given to cameras that support the Linux UVC video drivers. In addition to having an ARM Cortex A8 core running at 1GHz, which was suggested by Singh (2011) for future EyeCon designs, the BeagleBoard xM has the following desirable features (Coley, 2010):

- Low power Organic Light Emitting Diode (OLED) touch screens available,
- Provides 4 USB 2.0 host ports and 1 USB on-the-go port, as well as a DE-9 serial connector for backwards compatibility,
- The hardware design files are all fully open source and can be customised if the need arises (BeagleBoard.org, 2011),
- Slightly smaller dimensions than the Series 1 EyeCons (when no screen attached), and only 2cm wider when screen attached. This will help maximise compatibility with existing EyeBots,
- Powerful enough to implement Robot Operating System (ROS) if desired in the future,
- The TI DM3730 processor includes a TI C64x+ Digital Signal Processing (DSP) core, which can be taken advantage of for offloading fixed-point image processing tasks from the ARM Cortex A8 core,
- Ample documentation is available for porting OpenCV to this platform ,
- Good battery life (6.5 hours with a 4500 mAh battery),

- Free libraries and a complete video Software Development Kit (SDK) are available for the TI DM3730 processor (Texas Instruments Incorporated, 2012b).

A range of other SBCs were considered and found to be unsuitable:

Candidate	**Reasons for exclusion**
Pandaboard	Insufficient USB ports, too big for EyeBot mounts
Blizzard Interface Baseboard for TDM-3730	Too big for EyeBot mounts
BeagleBone	All non-standard ports, no off-the-shelf screens, underpowered
Raspberry Pi	Immature hardware, supply issues, no off-the-shelf screens, underpowered
CMUcam	Not modular, no screen capability, underpowered, would require a complete rework to make it suitable
OMAP Zoom	Too big for EyeBot mounts
IGEPv2	Insufficient USB ports, otherwise could be a suitable replacement if that can be overcome

3.7 EyeCon M8 Block Diagram

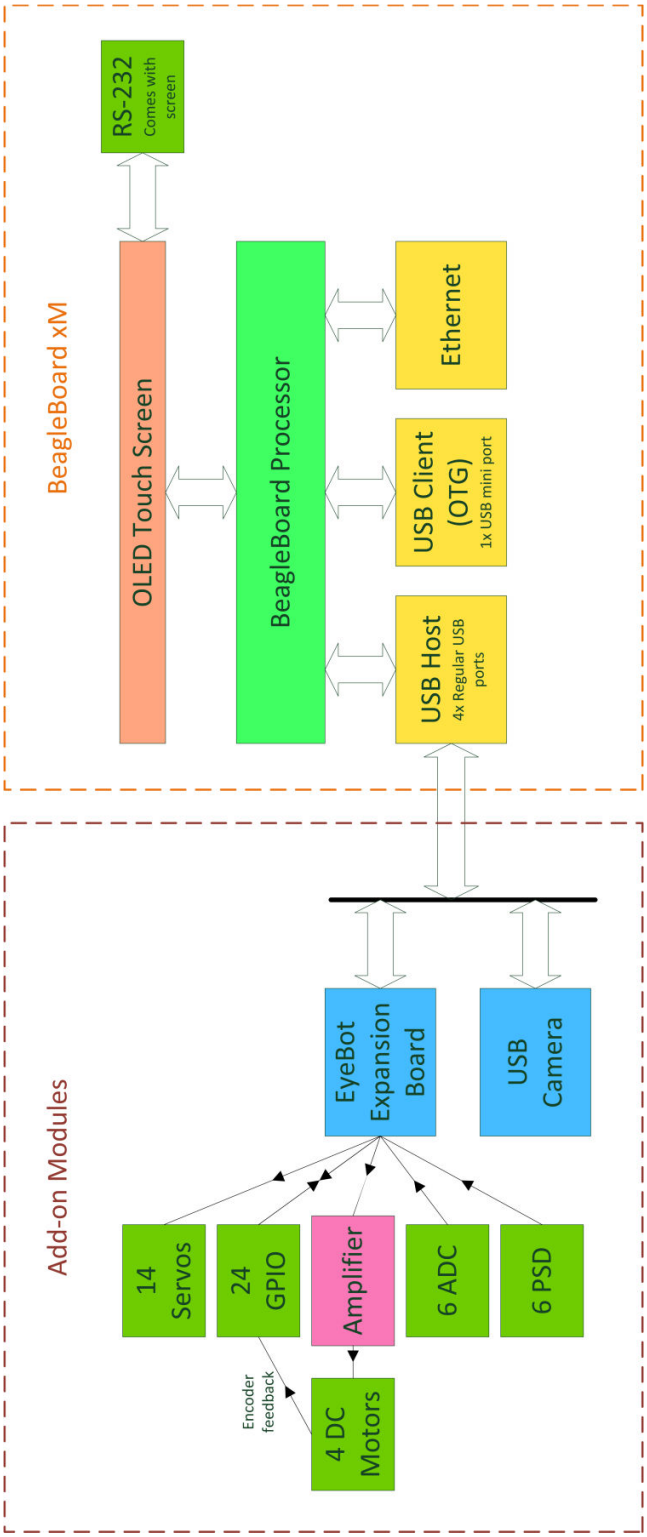


Figure 2: Block diagram of the EyeCon M8

4 PCB Design

Once the architecture of the EyeCon M8 had been agreed upon, the focus of the project moved to the design and manufacture of the expansion board. This entailed drawing up the schematics in an electrical Computer Aided Design (CAD) program, and then converting those schematics to a PCB layout. The schematic design and board layout took several months, as it had to be checked that every component could operate at the supplied voltages and could communicate with all the other components. Once the schematic was drafted, each component had to be sourced, and the schematic modified where necessary. Components were purchased as soon as the schematic was finalised as physical parts had to be checked against their PCB footprint before the board was sent for manufacturing. While the components were on order, the board was laid out and routed. This presented a whole new set of challenges as a number of extra constraints came into play; namely the PCB manufacturer's capabilities, the availability of tools to populate the board, mounting points, testing points, and size constraints. Figure 3 and Figure 4 show a 3D model of the expansion board.

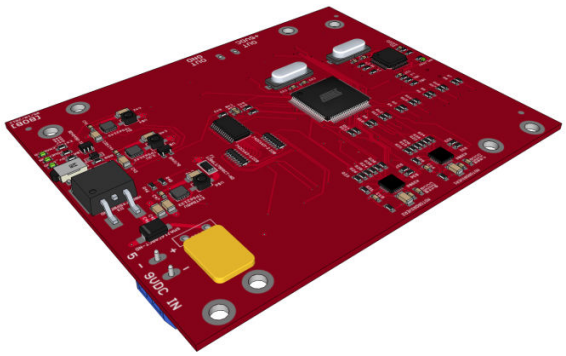


Figure 3: 3D model of the expansion board, top perspective view

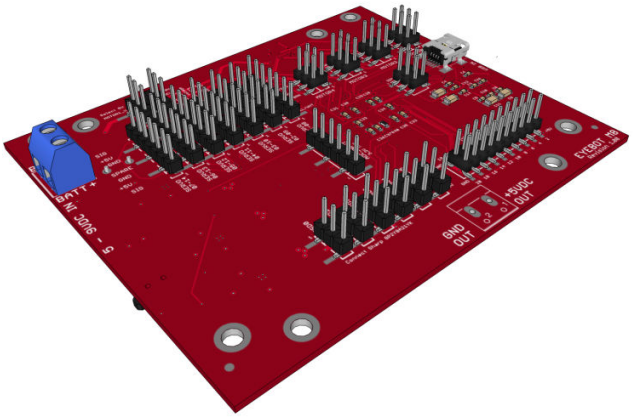


Figure 4: 3D model of the expansion board, bottom perspective view

4.1 Design Tools

4.1.1 Eagle PCB

The entire schematic and board design was done in Eagle PCB. This software was chosen for its large existing parts libraries, as well as third party libraries provided by large online electronics companies such as Sparkfun and Adafruit Industries.

4.1.2 Viewplot

Viewplot is a free gerber file viewing program, and was used to check the final PCB files before they were sent to the manufacturer. It was particularly useful for ensuring the PCB layers were in the right order, aligned correctly and weren't mirrored. It is also standard practice to tell anybody who has to work with your drill files what the number format of the files is. Since Eagle CAD does not allow you to set the number format, or even display the number format, Viewplot was essential for working out the number format of the generated Excellon drill file.

4.2 Expansion Board Features

4.2.1 Microcontroller

The microcontroller chosen for the expansion board is an Atmel ATxMega128A1. Atmel was the preferred brand of microcontroller because the author has worked with it before, and students are more likely to have worked with Atmel after working with hobbyist boards such as the Arduino. The ATxMega128A1 was chosen as it was the smallest and cheapest microcontroller which had the required functionality; that being:

- 16 interrupt enabled pins for GPIO,
- 14 interrupt enabled I/O pins for motor encoder feedback, stall flags and chip enable lines,
- 13 Analogue to Digital Converter (ADC) pins for battery level monitoring, PSD reading and general purpose ADC,
- 8 timer output enabled pins to provide the Pulse Width Modulation (PWM) for the motor controllers,
- A Serial Peripheral Interface (SPI) to be able to interface with the USB to SPI chip,
- An Inter-Integrated Circuit (I^2C, also known as Two Wire Interface, or TWI) interface to communicate with the PWM controller which drives the servos.

Initially the requirement was for 22 timer output compare pins, with the microcontroller also providing the PWM for the servos. This proved nearly impossible to satisfy, so the 14 servo PWM lines were offloaded to a PCA9685 chip. Unfortunately it was not possible to use a 5V microcontroller as no chip in the 5V range had the required features.

4.2.2 USB

USB is the common interface to most of the peripherals of the EyeCon, so it is important that it be used on the expansion board. This means that if ever the host device needs to be replaced - assuming it has USB - it will have hardware level compatibility with the expansion board. The chip selected for use on the expansion board is an FT232H chip made by FTDI. This is one of the few existing chips that supports Hi-Speed USB (up to 480Mbps), which ensures that the USB interface does not become a communications bottleneck. Many USB-SPI chips and USB-enabled microcontrollers present on the host as Human Interface Devices (HIDs) in order to function without further driver installations. They, however, have a maximum speed of 600Kbaud, and are unsuitable for controlling all the features of the expansion board in real time. The FT232H does not present as an HID device, which allows it to go much faster, however this requires additional drivers. Royalty free drivers for Windows, Mac and Linux are available from the FTDI website.

It was a deliberate decision not to select a microcontroller with an inbuilt USB port, as this would mean a complete change of host device drivers if ever the microcontroller is changed. On the other hand, converting from USB to SPI means that a new microcontroller

only has to have an SPI port, allowing a consistent software interface to be available on the host.

4.2.3 ISP

In System Programming is a feature of many programmable chips which allows them to be programmed after installation into the complete system. This eliminates the need for expensive, chip specific programming equipment which needs to be used before the chip is installed. The ATxMega128A1 supports in system programming via the Programming and Debug Interfaces (PDI) and Joint Test and Action Group (JTAG) interfaces.

4.2.4 Servos

The expansion board allows 14 servos to be connected and driven concurrently via a dedicated PWM chip. The chip selected is a PCA9685, which is also used on a 16-channel servo controller produced by Adafruit Industries. The PCA9685 is controlled via I^2C from the microcontroller and provides up to 16 PWM channels with 12-bit resolution. 12 bits of resolution allows for servo positioning in 0.8 degree increments (assuming a 50Hz frequency, 1-2ms duty-cycle, and a servo with 180 degree range). According to (Pololu Robotics and Electronics, 2011), "the frequency of the pulse train does not affect the servo position if the pulse width stays the same". Taking this into account, the frequency could be increased to 180Hz (most hobby grade servos permit this) and the positional increments can be reduced to approximately 0.25 degrees.

The PCA9685 runs at 3.3V to maintain compatibility with the ATxMega, however "standard" hobby grade servos are rated at 5V. To ensure the 3.3V signal would not cause issues with the servos, tests were conducted on a range of servos beforehand using a 3.3V Texas Instruments "mbed" board to ensure the behaviour was correct with the lower voltage.

4.2.5 Motors

Four motors are able to be driven by the expansion board using two A3906 motor drivers. These chips have a number of advantages over the L293 driver used in the EyeCon M6; they have configurable internal current limiting for each motor, and provide a stall flag for each motor which is convenient to check if the robot gets stuck or if the motor is pulling too much current.

Each A3906 chip is capable of supplying two motors with 1A each, and is controlled using 4 I/O lines from the microcontroller. For a regular DC motor, any state out of disabled/forward/reverse/braking can be selected, and the speed of the motor can be adjusted by varying the duty cycle of the I/O lines using PWM. Headers are provided which allow for connecting standard 6-pin Faulhaber motors with encoders. Each driver chip can also drive a stepper motor in half or full step mode by connecting a stepper motor to two pairs of the DC motor outputs and setting the 4 I/O lines appropriately.

The power source of each motor driver is jumper selectable; battery voltage or regulated 5V. The latter voltage should be selected if the supply battery voltage is over 9V, as the maximum motor voltage allowed by the A3906 is 9V. To this end, the battery input terminal on the board has been labeled with a smaller range than what is available; "5 – 9VDC IN", which is to try and prevent people unknowingly blowing up the motor controller. The idea behind this is that once the instructions for the board have been properly read, a user will find out that the allowable range is actually "5 – 16VDC" as long as the correct jumper is set for the motor controller chips. To conserve battery power, each A3906 chip can be put into a sleep mode when not in use, putting its power usage down to around 750nW.

4.2.6 Motor Encoders

Each Faulhaber motor used in the EyeBot program has a rotary encoder with up to 1024 pulses per revolution. These pulses are counted using two I/O lines for each encoder on the microcontroller, saving the host BeagleBoard from a continuous stream of interrupts which would otherwise completely occupy its time. On the EyeCon M6, considerable time was spent dealing with potentially noisy encoder signals to mechanical jitter on encoder switch contacts. This should no longer be necessary, as all existing Faulhaber motors use either magnetic or optical encoding, which provide a much cleaner signal. It will be necessary to make a small electrical adapter for UWA's existing Faulhaber motors, as they use either an older 6 pin connector with a different pin order, or in some cases an even older 10 pin connector.

4.2.7 PSDs

Position Sensitive Devices are used on the front and sides of some EyeBots to detect the distance and angle of some wall or object. The EyeCon provides 6 PSD ports, which have historically been connected to Sharp GP2D02 infrared distance sensors. These sensors required a specific waveform to be input before they could be read, which made programming for them quite complex. Sharp no longer manufactures the GP2D02 and they have been replaced with the Sharp GP2Y0A21YK, which has nearly identical distance sensing capabilities but uses an analogue signal to reflect the distance sensed. Determining a distance value is now as simple as reading an ADC value and converting this value to a distance based on a lookup table. The move to an ADC based reading has the added benefit of eliminating the previous timing constraint of 70ms between successive PSD reads; the distance is now calculated approximately every 38.3ms, and can be read as frequently as the user wishes. Additionally, the standard ADC interface allows not just the Sharp GP2Y0A21YK, but a range of ultrasonic range finders.

4.2.8 ADC and GPIO

The expansion board provides 16 GPIO ports on a 20 pin header. The 20 pin header adds a 5V pin, a 3.3V pin, and two ground pins to assist with powering add-on peripherals. Six ADC inputs are also provided on a 6 pin header for general use. All the GPIO pins and ADC pins are connected directly to the microcontroller and subsequently are only 3.3V tolerant. It was decided to leave circuit protections off the ADC and GPIO pins because it added a significant number of parts, increased the cost, and the use of these pins is relatively infrequent. This means that the ADC and GPIO pins are unprotected from overvoltage conditions, shorting to ground, and shorting to other I/O pins (Rugged Circuits LLC, 2011). If this proves to be a problem, it may be necessary to protect each pin by adding resettable fuses (with built-in current limiting resistors), and zener diodes to each pin, as per Figure 5.

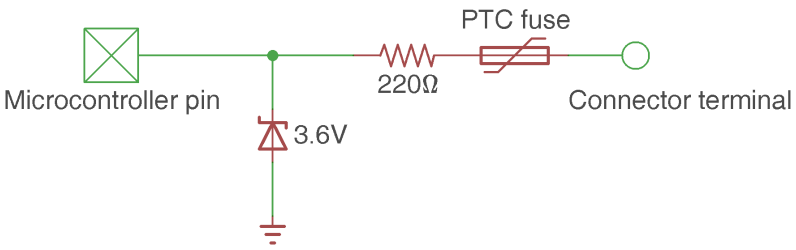


Figure 5: Possible pin protection means. Adapted from (Rugged Circuits LLC, 2011)

4.2.9 Power Supply

In addition to supplying power to all the expansion board components, the expansion board provides power to the BeagleBoard. The expansion board power is split into three switchmode supplies:

- A 2A/3.3V supply for chips such as the microcontroller,
- A 2A/5V supply for the low-noise peripherals such as the PSD sensors and BeagleBoard,
- A 3A/5V supply for the high-noise-high-current peripherals such as the motors and servo motors.

By default, only the 2A supplies are enabled, and the 3A supply is enabled only when the motors or servos are required. This allows the user to take advantage of the fact that the motors and servos are off for the majority of the time, and conserve power.

Step-down DC-DC supplies typically require an input voltage several volts above their output voltage. The TPS62142, TPS62143 and TPS62133 chips that were chosen do not have this requirement, and will actually allow voltages identical to (and even below) their specified output voltages thanks to a "100% duty cycle" mode which passes the input directly to the output. In contrast, the maximum duty cycle of the LM2678 supply on the EyeCon M6 was 91%, which implies a minimum input voltage of 5.49V (Texas Instruments Incorporated, 2012a). The benefit of a 100% duty cycle is that batteries can be drained to a lower voltage before the EyeBot shuts off, leading to a longer use time. The downside to this feature is that there is practically no voltage regulation once the battery drops below 5V, and additional parts are required to prevent the 5V power plane dropping below 4.9V and damaging the BeagleBoard.

An assortment of supply protections have been implemented to account for incorrect power supplies and misconnected power supplies, as these are errors which are most likely to be made by new users. If left unchecked, power supply problems have the potential to damage multiple components on the expansion board and BeagleBoard concurrently. Table 1 presents a list of potential power issues and how they have been mitigated.

Problem	**Solution**
Over voltage – over 16V	Transient Voltage Suppressing (TVS) diode and 4A resettable fuse
Undervoltage – under 4.9V	4.9V undervoltage lockout chip and microcontroller monitoring of battery voltage
AC voltage input	Schottky diode and 4A resettable fuse
Reverse voltage (up to 45V) input	Schottky diode and 4A resettable fuse
Motor or servo stall	3.5A resettable fuse, isolated power supply
Short circuit on the board	4A resettable fuse

Table 1: Power input protections

4.2.10 Power Switch

The EyeCon M8 uses a similar power switch setup to the M6, where the enable pin of the 3.3V supply can be enabled by either an IO line or a momentary switch. A single-pole-single-throw (SPST) momentary switch temporarily enables the switchmode supplies for long enough for the ATxMega to boot and permanently enable the supply with a dedicated I/O line. This allows the host device to power itself down by sending a command to the microcontroller to set that pin low. The implication of this method is that until the microcontroller has been programmed to set the I/O line high, the expansion board can't power itself without the power button being held down. To assist with initial programming and to account for situations where the board should boot as soon as it is supplied with power, an initially-closed solder jumper is used to short the power switch which can simply be cut after initial programming if required.

The power switch will not work if the battery does not supply at least 4.9V. A BD4949G under-voltage lockout chip with a cutoff of 4.9V is used in series with the power button, so that the power button only works when the supply voltage is at least 4.9V. This ensures that

an under-voltage condition cannot occur before the microcontroller is enabled and monitoring the battery voltage.

4.2.11 Power Control

Nearly every component of the expansion board can be individually enabled and disabled as it is needed. Where it isn't possible to completely disable a component (such as the microcontroller), it is possible to put it in a low power sleep mode to reduce power consumption. Idle power consumption of the board was measured at 59mW.

To prevent damage to the batteries from over-discharge, the battery level is monitored with an ADC line on the microcontroller, and the system can be shut down at a programmed voltage. Monitoring the battery voltage with an ADC line required mapping the 5 – 16V input range to a range with a maximum of 3.6V to avoid applying too high a voltage to the ADC. A 15KΩ/3.3KΩ resistive voltage divider was used, which maps 5 -16V to 1.1 - 3.52V.

4.2.12 Battery

The battery that the EyeCon M8 is designed for is a 7.2V nickel-metal hydride (NiMH) or 7.4V lithium-ion polymer (LiPo) battery. These two types of battery are the most suitable because they have a package size that suits most existing EyeBots, and they approach 5V when nearly 100% discharged. Both battery types have a relatively flat discharge curve which allows the battery to be completely drained before reaching under-voltage cutout. The only difficulty presented by a flat discharge curve is that it becomes difficult to predict what charge the battery has remaining if only the voltage is known. Figure 6 shows a typical 7.2V NiMH battery discharge curve. Note the equal voltage values at 0.5 and 2.0 AmpHrs on the 30A curve which make it difficult to calculate remaining charge. Despite this, it is still possible to know when the battery is close to discharged because the voltage drops sharply towards the right end of the discharge curve.

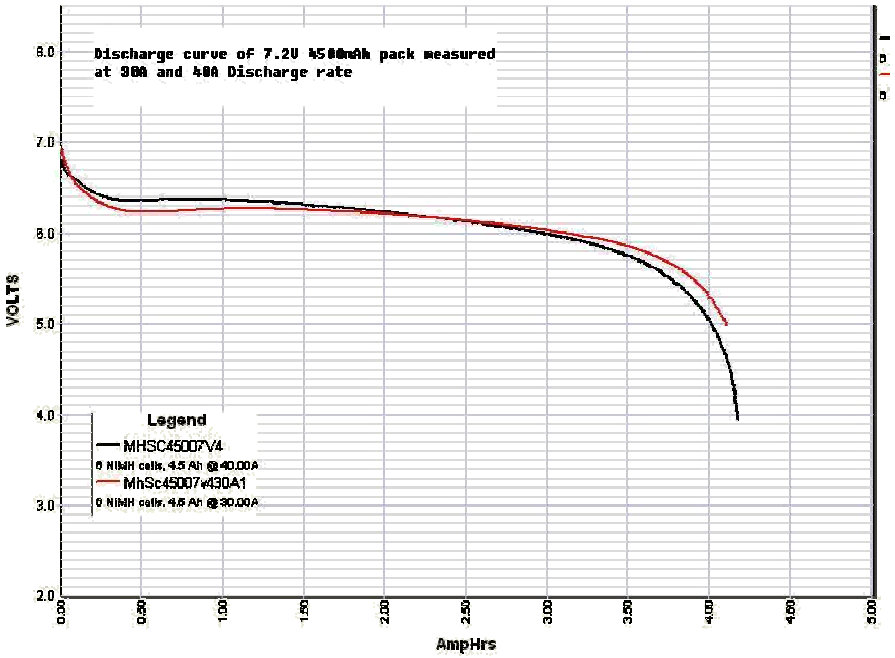


Figure 6: The discharge curve of a typical 7.2V NiMH battery. Source (AA Portable Power Corp, 2012)

5 Manufacturing and Assembly

5.1 Manufacturing

The expansion board PCBs were manufactured in China by PCB Cart. After a one-off $210 tooling fee, the boards cost approximately $30 each (depending on the USD-AUD exchange rate). Relevant manufacturing options are detailed in Table 2, and have been included because they directly affect the board layout and design.

Option	**Value**
Minimum spacing	6 mil
Minimum annular ring	4 mil
Smallest holes	12 mil
Maximum number of holes	300
Blind vias	No
Number of layers	4
Dimensions	4.094 inch x 3.346 inch

Table 2: PCB Cart manufacturing options

A minimum 6 mil spacing had to be specified because the pins on the ATxMega chip are closer than the typical 8 mil spacing. Furthermore, the copper thickness had to be reduced from 2oz to 1.5oz because the manufacturer could not do 6mil spacing at 2oz. Though this reduces the current carrying capacity of the traces, it is still well within the range of currents to be carried, as all trace widths were selected assuming a 1oz minimum copper thickness. The time taken from file submission to receiving the PCBs was 19 days, so at least three weeks should be allowed for PCB manufacturing when the next iteration of the board is made.

5.2 Assembly

Once the PCBs were delivered, assembly was done by hand using a hot-air rework station with solder paste applied by hand using a syringe. This process was particularly slow, but allowed any mistakes to be quickly found and dealt with. Despite following a place-and-test approach, a design issue with the power supplies destroyed several important components during assembly through over-voltage, which made programming and debugging a challenge until all the affected chips could be isolated.

6 Testing

With the PCBs only arriving two weeks before the project end date, testing was not as comprehensive as it otherwise would have been. Due to time constraints, testing had to be limited to:

- Ensuring the board could be programmed,
- Checking integrity of inter-chip communications such as the I^2C bus, SPI bus and motor control lines,
- Checking the power supply regulation and control signals worked,
- Testing the power supply protections,
- Testing GPIO lines,
- Testing the PSD inputs with a real sensor,
- Testing the servo outputs with real servos,

Testing was performed using a Link Instruments MSO-19 logic analyser to check communication lines and a digital oscilloscope to analyse signal integrity. Simple test code was written for the ATxMega to allow testing of relevant signals (see Appendix B:).

6.1 In System Programming

Programming the expansion board was somewhat difficult; due to a misunderstanding when reading the documentation, the programming header was incorrectly connected to the SPI pins in the prototype boards (which most ATmega chips use) instead of the PDI pins. The first attempted solution to this was to cut two traces and rewire the ISP header to the PDI pins. This was successful, however it was subsequently discovered that the AVR Dragon programmer that was being used could not program the ATxMega128A1 over PDI – a fact that is both poorly documented and contradicted in several places in the Atmel documentation (Atmel Corporation, n.d.; Atmel Corporation, 2009a). The ultimate solution was to break out the JTAG pins from the ATxMega and program the board using JTAG. Two of the pins were already connected to the ADC header, and the other changes required were:

- An additional two wires had to be soldered from ATxMega pins 12 to 7, and 11 to 6,

- Removing resistors R16 and R18 to prevent the battery voltage monitoring from interfering with the JTAG signals.

Using extra pins for JTAG meant sacrificing most of the ADC lines as well as the battery voltage monitoring, however all of these pins can be recovered in the next iteration of the board by adding a dedicated JTAG header and moving the battery voltage monitoring to a spare ADC pin.

6.2 SPI

The SPI bus is used for communication between the FT232H and ATxMega microcontroller to control the expansion board via USB. There are five settings which must be correctly set on both devices before communication can occur; endianness, clock polarity, clock phase, chip select polarity, and transmission direction. Furthermore, the clock frequency must be set such that the high and low times of the signal are longer than 2 clock cycles of the ATxMega (Atmel Corporation, 2009b, p. 230). The relevant settings are detailed in Appendix C.2:.

There are four different combinations of clock phase and clock polarity which define the basic SPI modes (see Appendix C.1:). The FT232H was initially configured for Mode 1, however this caused a short voltage spike to appear before the clock pulse and corrupted the clock pulses of the second byte, making it impossible to read the data (Figure 7). Further reading of the FT232H MPSEE command reference revealed that the design of the FT232H clock pin precludes it from being set in either Mode 1 or 3 (Future Technology Devices International Ltd., 2011, p. 6). Changing to Mode 0 eliminated the issue and the signal was as expected (Figure 8).

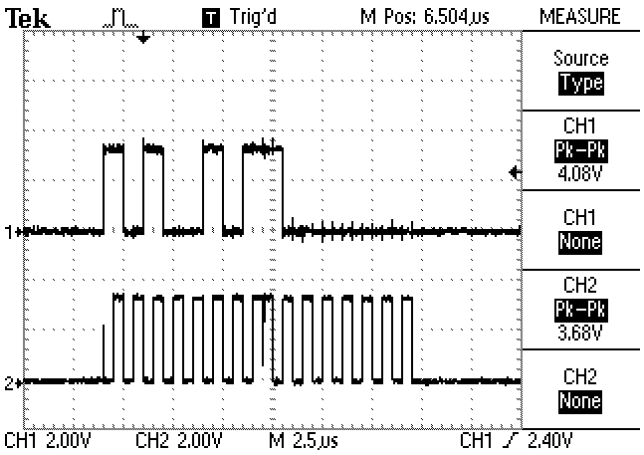


Figure 7: The MOSI (top) and CLK (bottom) lines with two bytes being send in SPI Mode 1

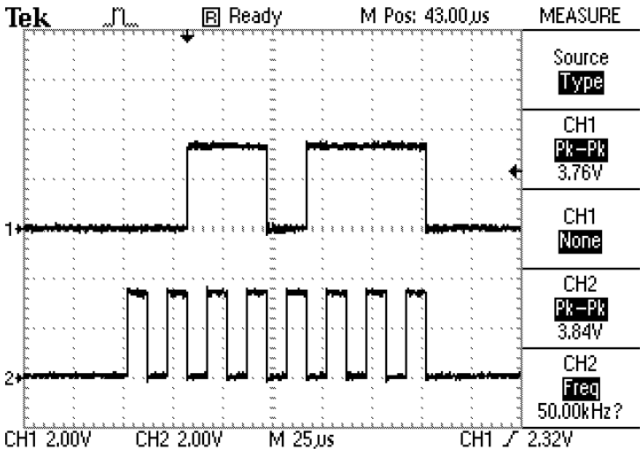


Figure 8: The SPI MOSI (top) and CLK (bottom) lines with 0x37 being sent in Mode 0

A small Linux test program for the FT232H chip was written by John Hodge, which served to verify that communications were possible between a host device and the expansion board (see Appendix B.4:). This test program does two things; it turns on a Light Emitting Diode (LED) connected to an I/O line on the FT232H, and it sends the byte 0x37 over SPI at a regular interval, as shown in Figure 8. The 0x37 value is somewhat arbitrary however it is convenient for testing because it does not read as the same value if reversed in binary. The test program was successfully compiled and run on both an x86-64 architecture PC and the BeagleBoard.

To avoid compound errors, the test program was first run against an FT232H development board (an FTDI UM232H) and the signal output was matched with the output from the chip on the expansion board. The signals matched, and code was subsequently written for the ATxMega chip to read the SPI input register and switch on an LED when the 0x37 byte is received (see Appendix B.1:).

There was one other minor mistake regarding the FT232H chip; the VREGIN line was not connected to the 3.3V plane. This was easily fixed by creating a solder bridge between pins 39 and 40.

6.3 I²C

The I²C bus is used for communication between the PCA9685 PWM chip and the ATxMega. Initial verification of the I²C bus involved sending out a call on the "all-call" address for the PCA9685 and waiting for an acknowledgement. The PCA9685 acknowledges the call by holding the SDA line low during the 9th clock pulse on the SCL line (NXP, 2010), as shown in Figure 9.

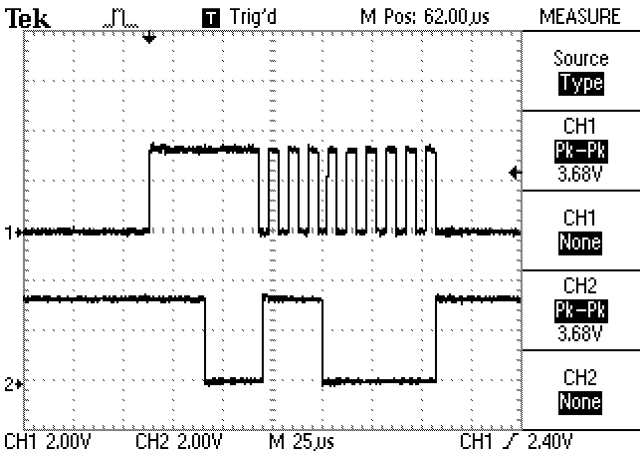


Figure 9: The I²C SCL (clock, top) and SDA (data, bottom) lines with the master address being sent and the SDA line being held low for the 9th clock pulse

The rise time of the I^2C lines is shown in Figure 10. The 30% to 70% rise time of the I^2C lines were measured as 50ns, which is well within the 80ns specification for High Speed I^2C devices (NXP, 2012).

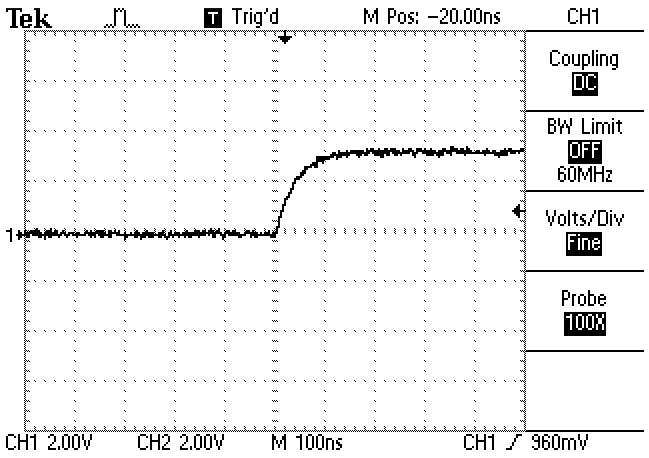


Figure 10: The rise time of the I^2C SCL line was measured at 50ns

6.4 Motors

The motors were tested with a single Faulhaber motor attached to the board, identical to those used on existing EyeBots. Unfortunately it was not possible to test with four motors connected because the available motors had older 10-pin connectors and insufficient adapters were available for the new 6-pin header. Comprehensively testing the motor controllers would require attaching four motors at once and checking performance is as expected under a range of loads and speeds.

The single motor with encoder was attached to each motor header, and then set to full speed in both directions. No problems with motor speed or power were found when applying a load to the motor by pinching the axle by hand and bringing it close to stopping, which is a reasonable approximation of the loads such a motor would be under when used in the student lab EyeBots. A single motor drew approximately 0.45A at 5V, so the operation of four motors concurrently is well within the capabilities of the power supply.

6.4.1 Motor Encoders

All Faulhaber motors have either magnetic or optical encoders running at 5V. A resistive voltage divider was used to step the voltage down to 3.3V to permit reading by the ATxMega. Testing revealed that the values of resistor chosen had the unintended side-

effect of pulling the encoder line to ground. This was quickly remedied by increasing the resistor values by an order of magnitude and the encoder signal could then be detected.

6.4.2 Motor Stall Flags

A total motor stall condition was emulated by connecting a high wattage 10Ω resistor across the motor pins, which is sufficiently close to the 11Ω winding resistance of the Faulhaber motor. This revealed two problems; the logic type of the stall-flag connection was wrong, and the over-current trip value had been set too high.

The board was designed expecting a push-pull output from the A3906 chip on the FLx pins, when the output is actually open-collector and required pulling up to 3.3V. To fix this, resistors R21, R22, R23 and R24 were removed, while R19, R20, R25 and R26 were replaced with 0Ω resistors. The line was then pulled up to 3.3V by configuring the relevant pins on the ATxMega as pull-ups in software.

The over-current value is set per-motor with a resistor, Rs, connected between ground and a sense pin on the A3906. The value is calculated as follows:

$$R_S = \frac{0.2}{I_{TRIP}(max)}$$

The trip current was initially set to 1A with a 0.2Ω resistor to prevent damage to the A3906. While this worked, it would not work to prevent damage to the Faulhaber motors used in labs. A more appropriate value would be 0.5Ω, which limits the current to 0.4A – just under the stall current of the motors.

6.5 Power Supply

During assembly, it was discovered that the output voltage sense pin of all three switchmode supplies had been connected to the wrong side of the inductor, which meant that the switchmode chips were not regulating the voltage at all. The fix for this was to cut the offending trace and solder a wire to the correct side of the inductor. The power supplies subsequently regulated correctly to within 0.03V of the expected voltage with a maximum ripple of 300mV.

6.5.1 *Over-voltage protection*

The over-voltage protection on the board is provided by a TVS diode with a 16V peak reverse voltage in conjunction with a resettable 4A poly fuse. This protection was tested by applying 17V to the board. It was subsequently discovered that the resettable fuse could not cut the circuit fast enough, forcing the TVS diode to dissipate more energy than it was rated for. The TVS diode promptly burned out, but failed to a closed position, allowing the poly fuse to cut the circuit. While it is good that the TVS diode failed in order to save the rest of the board, it is less than ideal to have to replace it every time too high a voltage is connected to the EyeCon. A better alternative to the TVS diode would be to use a crowbar circuit, such as a Linear Technologies LTC1696 overvoltage protection controller, which can cut out at a specified voltage and is specifically designed to handle the high current generated by shorting a power supply.

6.5.2 *Under-voltage protection*

Testing the under-voltage protection was done with a variable power supply and an oscilloscope to measure the cut-off voltage. The under-voltage protection relies on a BD4949G under-voltage lockout chip, with a simple zener regulator used to ensure the battery voltage did not exceed the permitted 10V input voltage of the chip. The target cut-off voltage was 4.9V, as this is the lowest voltage permitted by the BeagleBoard. The measured cut-off voltage was 5.67V, which was much higher than expected. This was because the design did not account for the voltage drop across resistor R58 (see Figure 11). Since the current at the cut-off voltage can be measured and is constant, a solution is to replace the BD4949G with a chip from the same line that has lower cut-off voltage (approximately 4.2V).

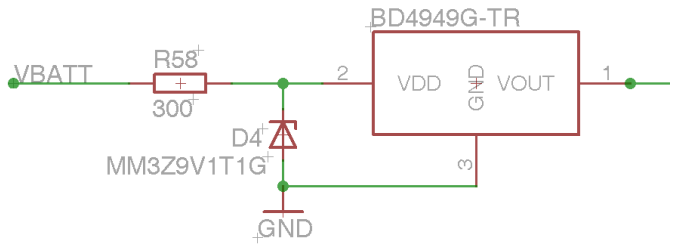


Figure 11: The under-voltage lockout portion of the circuit showing the resistor-zener voltage regulator and BD4949G lockout chip

6.6 PSDs

The PSDs were tested by connecting a Sharp GP2D12 infra-red distance sensor to a PSD input. The GP2D12 very similar to the GP2Y0A21Y, however is older and has a lower output voltage (2.6V versus 3.1V respectively). Code was written that reads a value from the ADC pin and outputs a byte value on 8 GPIO pins. It was subsequently discovered that the ADC can only read a voltage that is smaller than the 1V reference voltage. This limitation is not indicated anywhere in the xMega A Manual (Atmel Corporation, 2009b), and could only be confirmed by looking at the ADC specific manual (Atmel Corporation, 2010). To read the IR sensors, their output voltage needs to be reduced to a measurable range using a voltage divider, or the ADC reference voltage needs to be increased. Given the reference voltage cannot exceed 2.7V, the output voltage from the IR sensors has to be dropped regardless. Provision for this was not built into the board, so external adapters will have to be used to divide the voltage. In the next iteration of the board, an external precision voltage reference should be added to allow for a greater detection range, as well as provisioning for voltage dividers on the input. A suitable voltage reference chip would be the Texas Instruments LM4040-N Precision Micropower 2.048V Shunt Voltage Reference.

6.7 Servos

The servo testing was relatively straight forward, since the I^2C bus had already been confirmed working, and testing during the design stage had ensured that servos would work with 3.3V. The only remaining task was a load test, and to ensure the servos would work at the predicted 180Hz. Headers for initialising the PCA9685 and writing to the I^2C bus were written for the test program, which were sufficient to configure and enable all of the servos. Only three servos were available for testing, so it cannot be concluded that 14 servos will work at once, however the three that were tested were smoothly moving through their entire 180 degree range. Figure 12 confirms that the servos could function with a shortened period, as long as the on-time was within the normal 1-2ms limits.

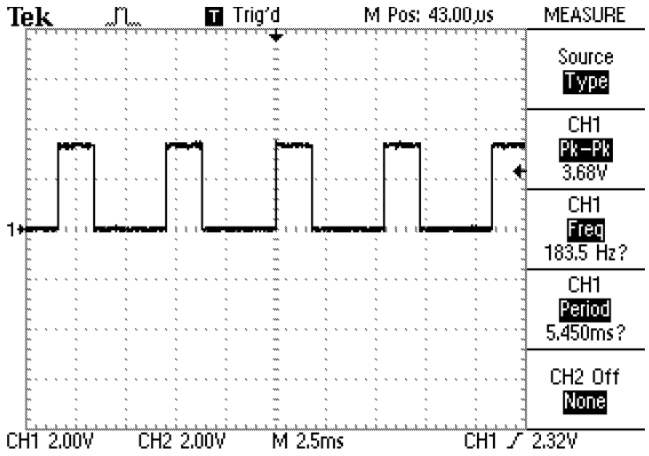


Figure 12: The servo PWM signal operating at 183.5Hz with a duty cycle of approximately 1.5ms

7 Recommendations for Future Work

In addition to fixing the design issues that were raised in the in the Testing section, the following items are recommended for future EyeCons and expansion board designs.

7.1 PCB layout

It was found during hand population of the board that the 0603 components did not align themselves with solder surface tension because the pads were wider than the components. It is recommended that all 0603 pads be made the minimum width for the next board revision, whilst maintaining the existing length. Furthermore:

- The font size should be made larger, particularly on the numbered pins, to allow the pin descriptions to be read at a greater distance.
- The GPIO header should be renumbered so that "1" starts on the first GPIO pin, not the 3.3V pin.
- Two additional mounting holes should be put on the expansion board in line with the BeagleBoard xM mounting holes. This will assist in supporting the middle board of the EyeCon and stop it touching the expansion board underneath.

7.2 Debugging

Debugging the expansion board was incredibly difficult because insufficient headers were put on the board to allow easy probing of signal lines. More feedback mechanisms such as LEDs were also needed. It is recommended that future iterations of the expansion board have at least:

- A full SPI header to allow easy SPI probing (this can replace the existing ISP header)
- An I^2C header between the ATxMega and PCA9685 to allow for easy data probing
- A JTAG header to allow for complete in-system debugging of the ATxMega
- More diagnostic LEDs, particularly on motor output lines and chip enable lines.

7.3 USB connector

A custom USB cable of the correct length was made to connect the expansion board to the BeagleBoard. When the expansion board was mounted underneath the BeagleBoard it was discovered that the USB connector sticks out in an unattractive way, and is quite exposed to physical knocks. The cable was also under considerable stress due to the tight curvature. A way this may be improved is to rotate the socket 90 degrees clockwise/ccw and cut a slot out of the edge of the board for the plug and cable. Another solution may be to add a board-to-wire connector such as a JST SSR connector to the USB lines, and do away with the mini USB plug altogether.

7.4 Through hole headers vs. surface mount headers

During the design phase, the author was strongly encouraged to use surface-mount headers instead of through-hole headers on the expansion board. The reasoning behind this was fourfold; it saves PCB space, mounting-angle is guaranteed with surface-mount headers, it is impossible to mount the headers at differing heights, and it "looks better". For future board revisions, it is strongly recommended that surface-mount headers be abandoned in favour of through-hole headers for the following reasons:

- Through-hole headers are much easier to source; finding the correct sized surface-mount headers necessitated buying from multiple suppliers and cost more than the same sized through-hole parts,
- Round pads are easier to route traces around than the long rectangular pads of surface mount components, which put lines of copper across the PCB - essentially forming a barrier through which no traces can be routed,
- The height of the expansion board can be reduced by approximately 2mm because the pins are sitting in the board, not on it,
- There is enough room on the PCB to allow through hole headers, and traces can be routed to pads on both sides of the PCB instead of just one side, potentially reducing the number of vias and making the board neater,
- Having the plastic of the header on the opposite side of the board to where the solder is being placed reduces the risk of melting the plastic during hand assembly.

The issue of mounting-angle variances and consistent mounting-height can be addressed by using the “slightly offset holes” approach, where the holes are arranged in a slight zigzag pattern to hold the header in place during soldering (Figure 13).

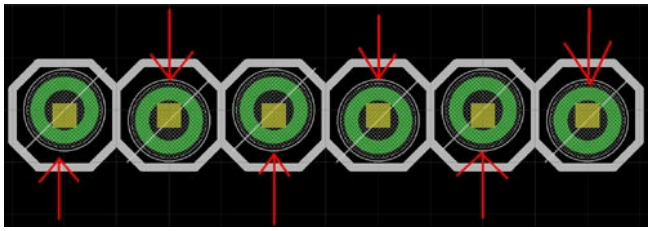


Figure 13: The offset holes approach to through-hole headers. Source (Lewis, 2008)

7.5 Power control

As designed, the expansion board has an initially-closed solder jumper across the power switch to keep it powered on until initial programming is performed, which is then cut after programming. This should be replaced in favour of a permanent two pin header, which will allow for both initial programming and situations where the EyeCon has to power on as soon as power is supplied (e.g. in the REV vehicles) or in situations where mounting prevents easy access to the power button. Removing the solder jumper also eliminates the risk of damage to the board and surrounding components when cutting the trace.

In the initial design, the power button is only used to switch the board on – it cannot act as a system shutdown or reset button. This is a problem because there is no way to hard reset the EyeCon without disconnecting the battery. The best way to solve this problem is to place an SPST on-off selectable switch inline with the battery leads, with the switch mounted in a convenient location on the relevant EyeBot. This ensures that the switch is in an accessible location, and eliminates slow discharge of the battery through the EyeCon.

It would save a small amount of power to move the power switch to the input side of the under-voltage lockout chip so the chip is only powered on when the switch is pressed. The tradeoff here is that the “battery good” LED display will only work when the power button is pressed, however it would reduce the power usage where the device is “off” but the battery has been left connected.

7.6 Assembly technique

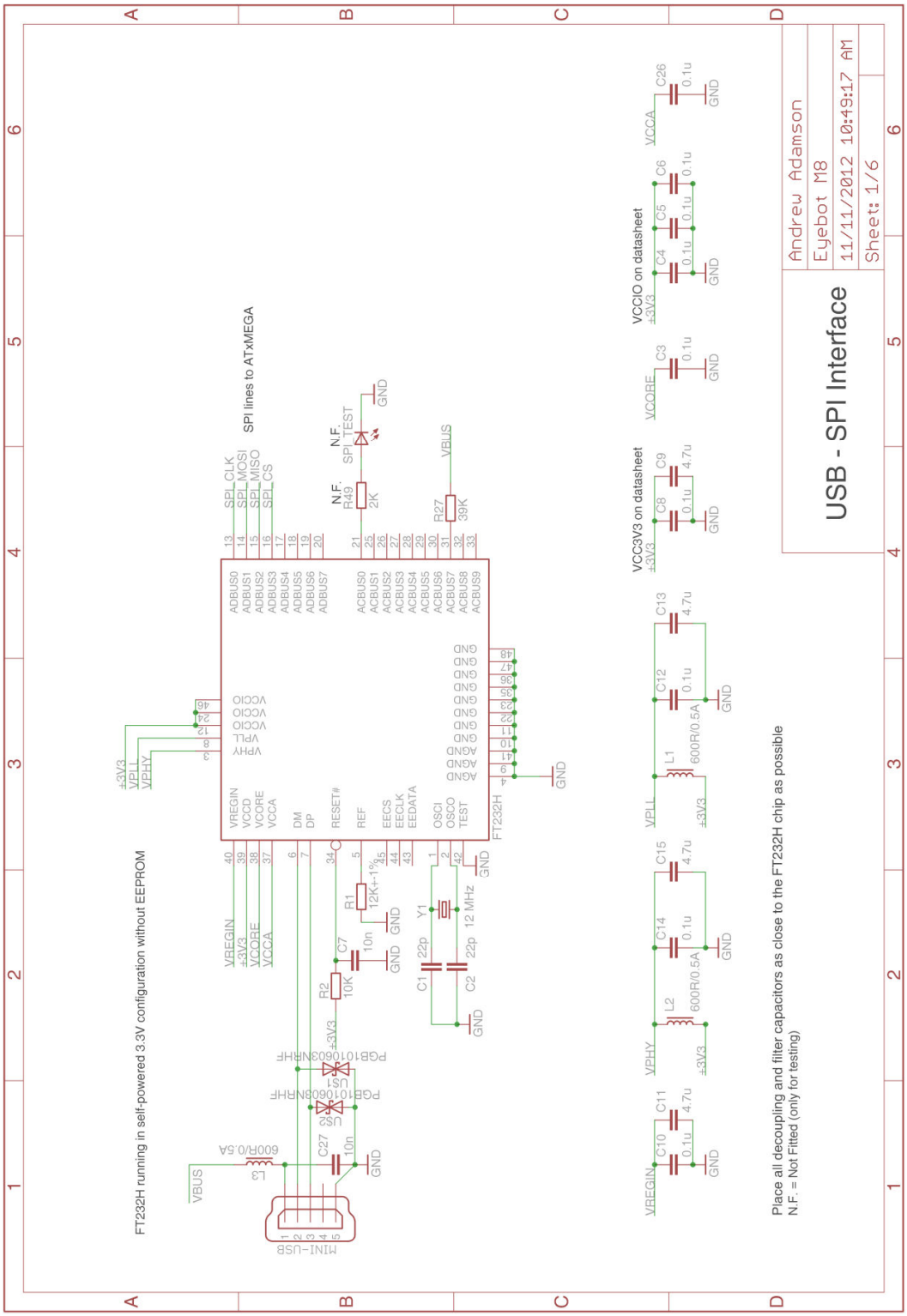
Assembling the expansion boards by hand is an arduous task – solder paste has to be applied to individual pads, and then a hot air gun is used on individual components. Aside from being slow, this assembly technique has the potential to thermally damage some components through incorrect temperatures and the creation of hot-spots by the assembler. It is highly recommended that a solder stencil is ordered with the next iteration of the expansion board, and that the electrical engineering workshop sources a reflow oven to allow bulk soldering.

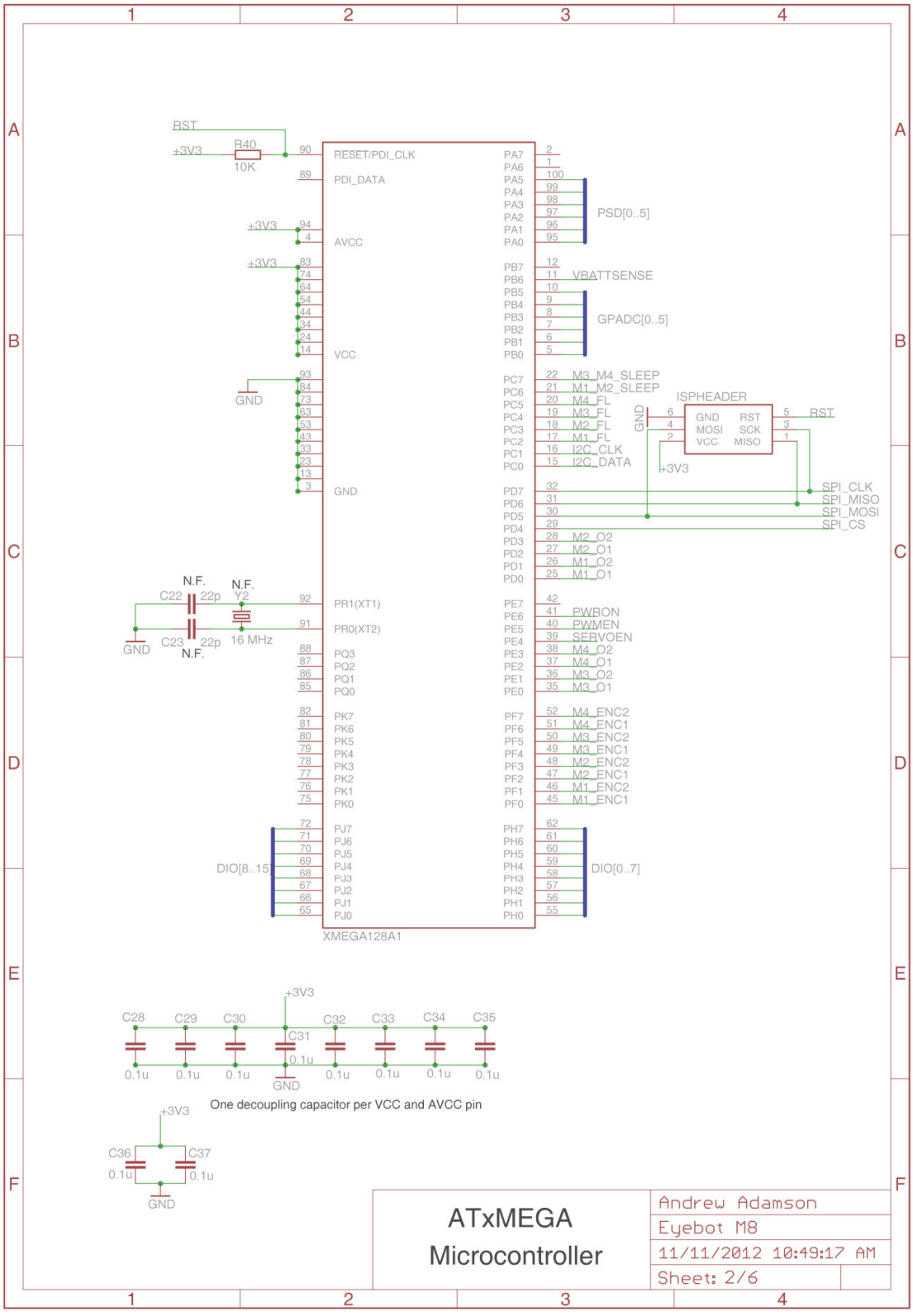
7.7 Software

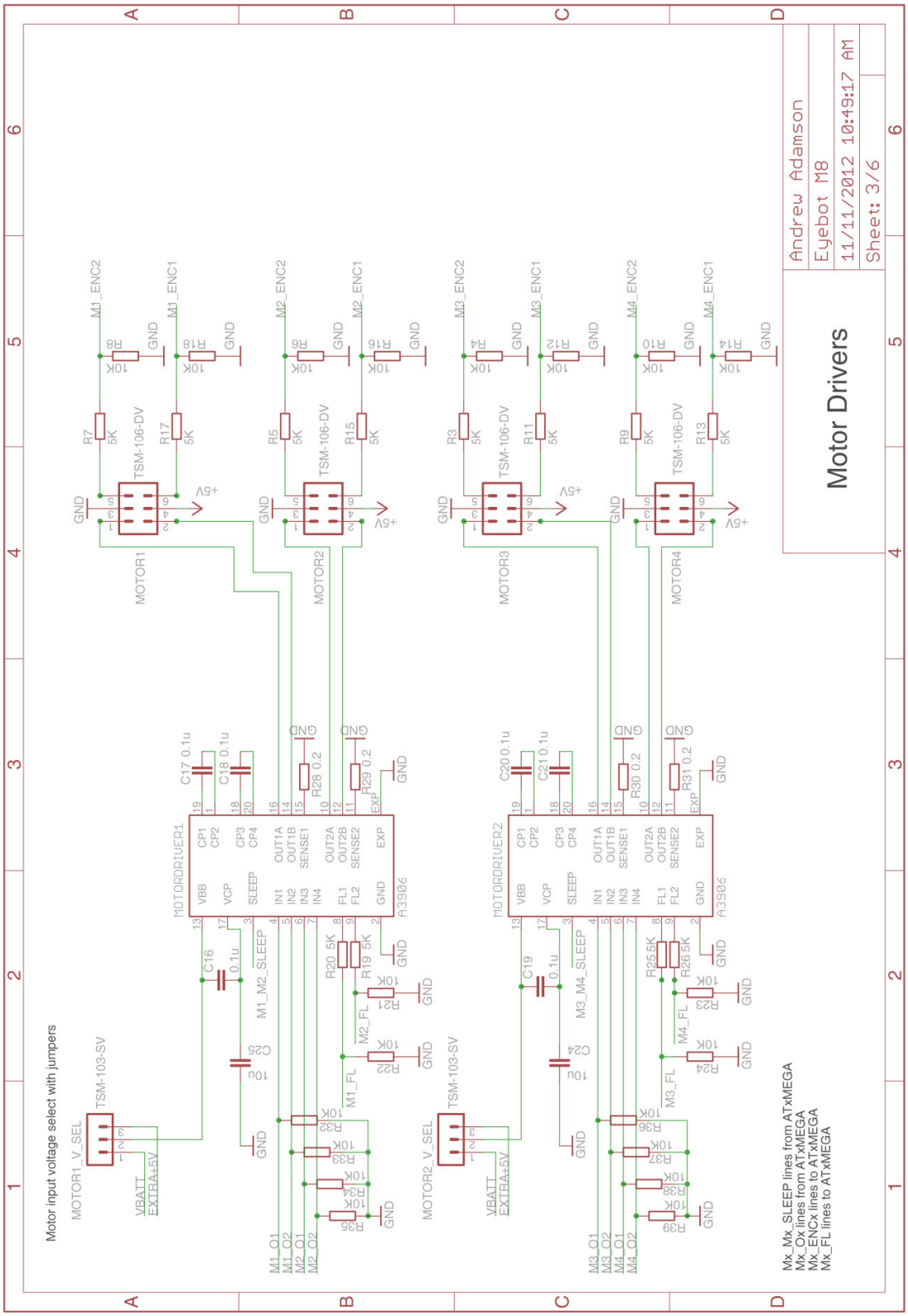
A minimal boot loader should be written and programmed onto the expansion board which will allow it to be programmed over USB. This will eliminate the slow task of connecting the JTAG multiple times, and allow the expansion board to be programmed by the BeagleBoard xM. The boot loader should have the battery level monitoring and power-enable functions built in so that the functionality is independent of general programming.

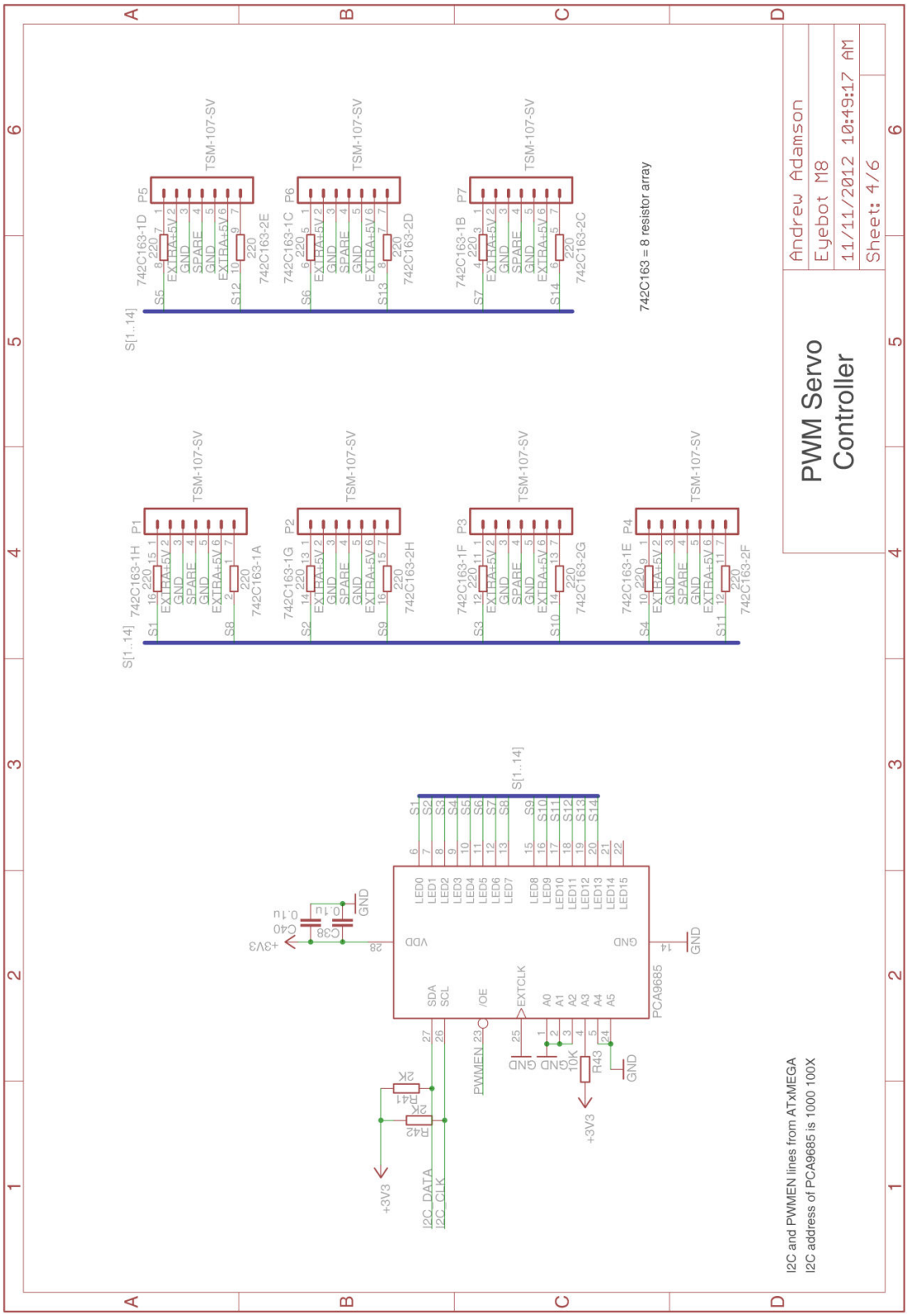
Appendix A: Expansion Board Detail

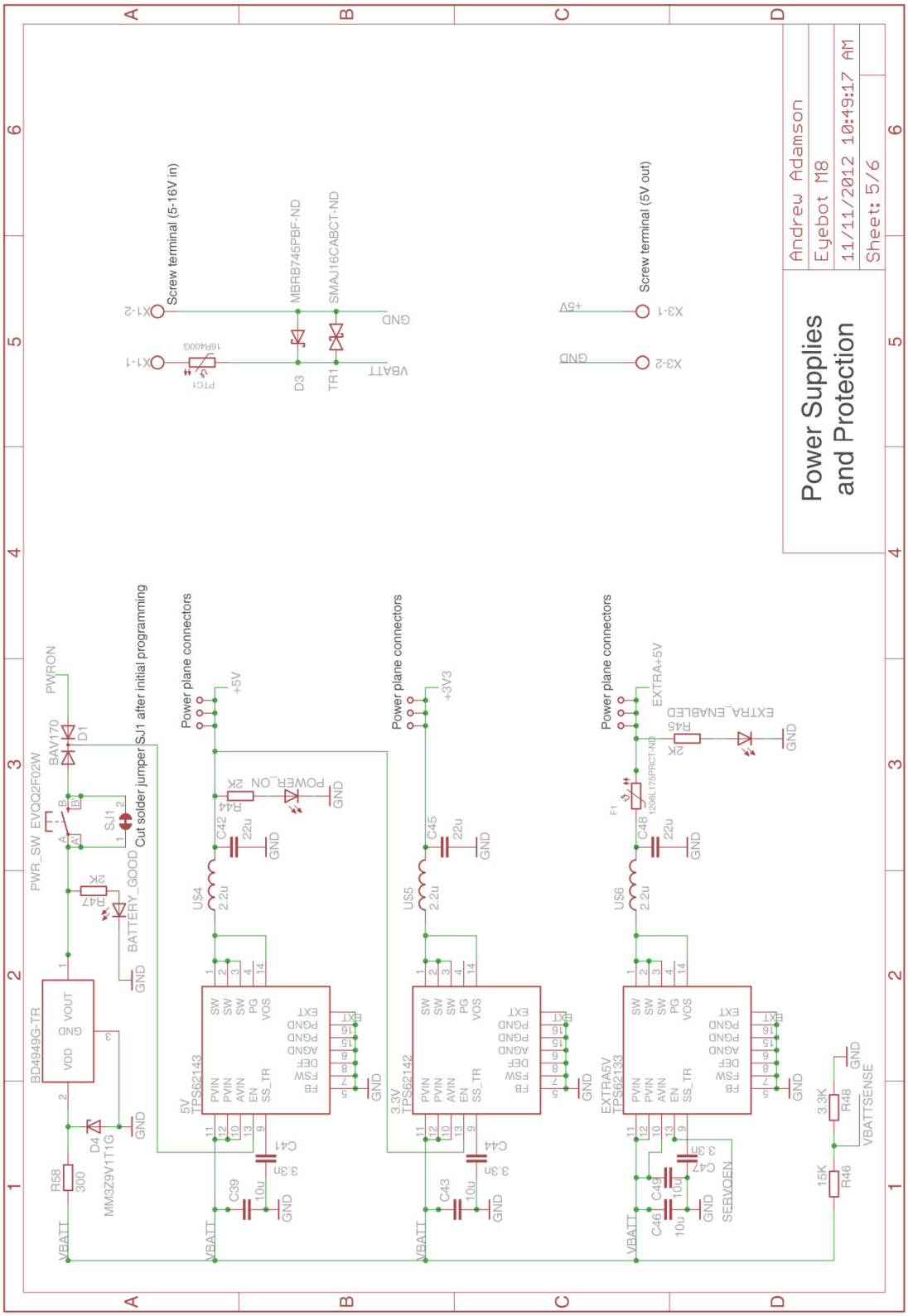
Appendix A.1: Schematics

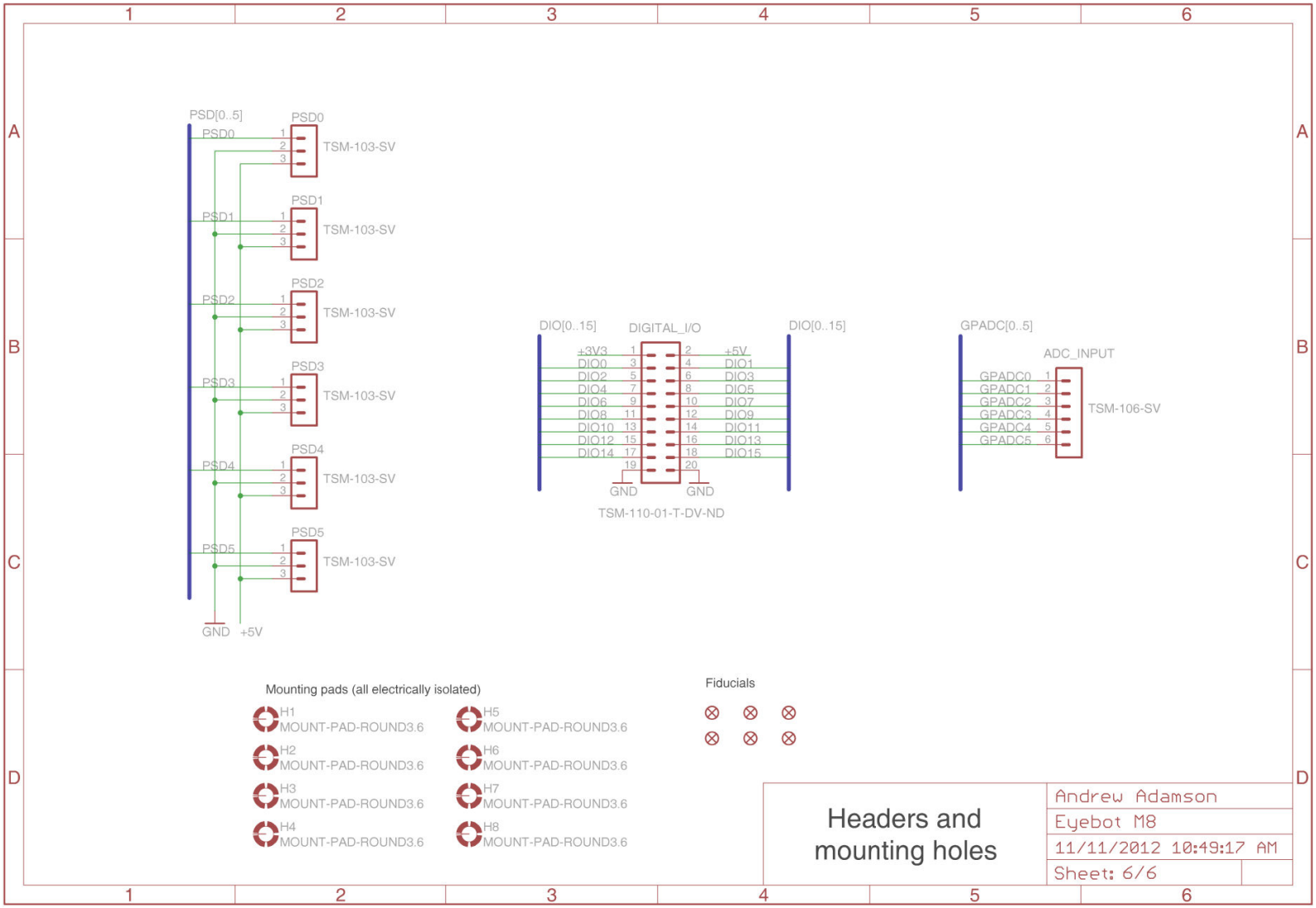



RST
+3V3
R40
10K
RESET/PDI_CLK
PDI_DATA
AVCC
VCC
GND
PR1(XT1)
PR0(XT2)
N.F.
C22
22p
Y2
16 MHz
C23
PQ3
PQ2
PQ1
PQ0
PK7
PK6
PK5
PK4
PK3
PK2
PK1
PK0
PJ7
PJ6
PJ5
PJ4
PJ3
PJ2
PJ1
PJ0
DIO[8..15]
XMEGA128A1
PA7
PA6
PA5
PA4
PA3
PA2
PA1
PA0
PSD[0..5]
PB7
PB6
PB5
PB4
PB3
PB2
PB1
PB0
VBATTSENSE
GPADC[0..5]
PC7
PC6
PC5
PC4
PC3
PC2
PC1
PC0
M3_M4_SLEEP
M1_M2_SLEEP
M4_FL
M3_FL
M2_FL
M1_FL
I2C_CLK
I2C_DATA
ISPHEADER
GND
MOSI
VCC
RST
SCK
MISO
PD7
PD6
PD5
PD4
PD3
PD2
PD1
PD0
SPI_CLK
SPI_MISO
SPI_MOSI
SPI_CS
M2_O2
M2_O1
M1_O2
M1_O1
PE7
PE6
PE5
PE4
PE3
PE2
PE1
PE0
PWRON
PWMEN
SERVOEN
M4_O2
M4_O1
M3_O2
M3_O1
PF7
PF6
PF5
PF4
PF3
PF2
PF1
PF0
M4_ENC2
M4_ENC1
M3_ENC2
M3_ENC1
M2_ENC2
M2_ENC1
M1_ENC2
M1_ENC1
PH7
PH6
PH5
PH4
PH3
PH2
PH1
PH0
DIO[0..7]
C28
C29
C30
C31
C32
C33
C34
C35
0.1u
One decoupling capacitor per VCC and AVCC pin
C36
C37
ATxMEGA
Microcontroller
Andrew Adamson
Eyebot M8
11/11/2012 10:49:17 AM
Sheet: 2/6


Motor Drivers
Andrew Adamson
Eyebot M8
11/11/2012 10:49:17 AM
Sheet: 3/6
MOTOR1
MOTOR2
MOTOR3
MOTOR4
TSM-106-DV
MOTORDRIVER1
MOTORDRIVER2
A3906
TSM-103-SV
MOTOR1_V_SEL
MOTOR2_V_SEL
Motor input voltage select with jumpers
VBATT
EXTRA+5V
M1_ENC1
M1_ENC2
M2_ENC1
M2_ENC2
M3_ENC1
M3_ENC2
M4_ENC1
M4_ENC2
M1_M2_SLEEP
M3_M4_SLEEP
M1_FL
M2_FL
M3_FL
M4_FL
M1_O1
M1_O2
M2_O1
M2_O2
M3_O1
M3_O2
M4_O1
M4_O2
Mx_Mx_SLEEP lines from ATxMEGA
Mx_Ox lines from ATxMEGA
Mx_ENCx lines to ATxMEGA
Mx_FL lines to ATxMEGA

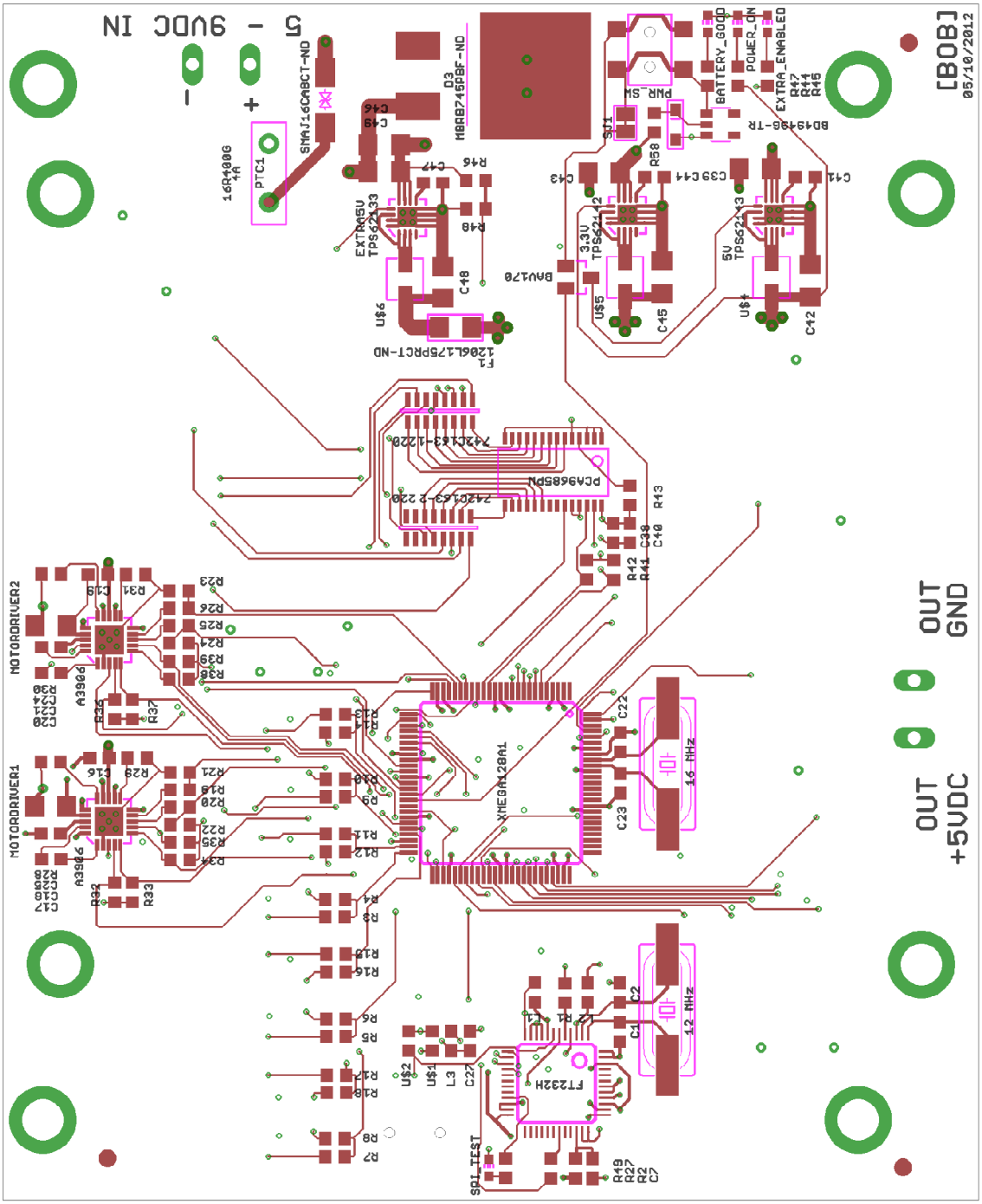

PWM Servo Controller
Andrew Adamson
Eyebot M8
11/11/2012 10:49:17 AM
Sheet: 4/6
742C163 = 8 resistor array
I2C and PWMEN lines from ATxMEGA
I2C address of PCA9685 is 1000 100X
TSM-107-SV
742C163-1A
742C163-1B
742C163-1C
742C163-1D
742C163-1E
742C163-1F
742C163-1G
742C163-1H
742C163-2C
742C163-2D
742C163-2E
742C163-2F
742C163-2G
742C163-2H
PCA9685
S[1..14]
+3V3
GND
I2C_DATA
I2C_CLK
PWMEN
EXTCLK


Power Supplies and Protection
Andrew Adamson
Eyebot M8
11/11/2012 10:49:17 AM
Sheet: 5/6
Screw terminal (5-16V in)
Screw terminal (5V out)
MBRB745PBF-ND
SMAJ16CABCT-ND
Power plane connectors
Cut solder jumper
SJ1 after initial programming
PWRON
PWR_SW EVQQ2F02W
BAV170
BD4949G-TR
MM3Z9V1T1G
BATTERY_GOOD
POWER_ON
EXTRA_ENABLED
EXTRA+5V
TPS62143
TPS62142
TPS62133
VBATT
GND
SERVOEN
VBATTSENSE

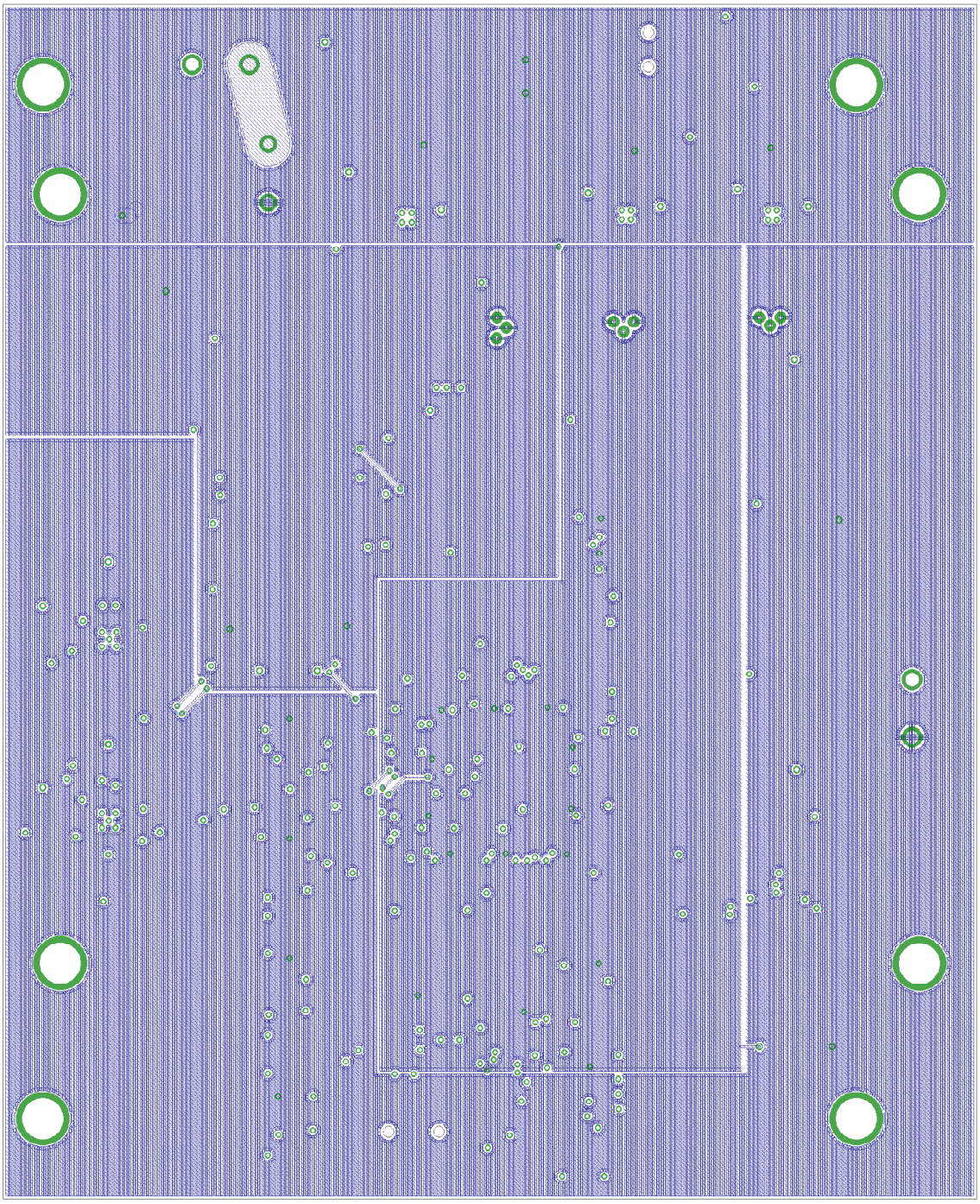

PSD[0..5]
PSD0
PSD1
PSD2
PSD3
PSD4
PSD5
TSM-103-SV
GND
+5V
DIO[0..15]
DIGITAL_I/O
+3V3
DIO0
DIO1
DIO2
DIO3
DIO4
DIO5
DIO6
DIO7
DIO8
DIO9
DIO10
DIO11
DIO12
DIO13
DIO14
DIO15
TSM-110-01-T-DV-ND
GPADC[0..5]
ADC_INPUT
GPADC0
GPADC1
GPADC2
GPADC3
GPADC4
GPADC5
TSM-106-SV
Mounting pads (all electrically isolated)
H1 MOUNT-PAD-ROUND3.6
H2 MOUNT-PAD-ROUND3.6
H3 MOUNT-PAD-ROUND3.6
H4 MOUNT-PAD-ROUND3.6
H5 MOUNT-PAD-ROUND3.6
H6 MOUNT-PAD-ROUND3.6
H7 MOUNT-PAD-ROUND3.6
H8 MOUNT-PAD-ROUND3.6
Fiducials
Headers and mounting holes
Andrew Adamson
Eyebot M8
11/11/2012 10:49:17 AM
Sheet: 6/6

Appendix A.2: PCB Layers

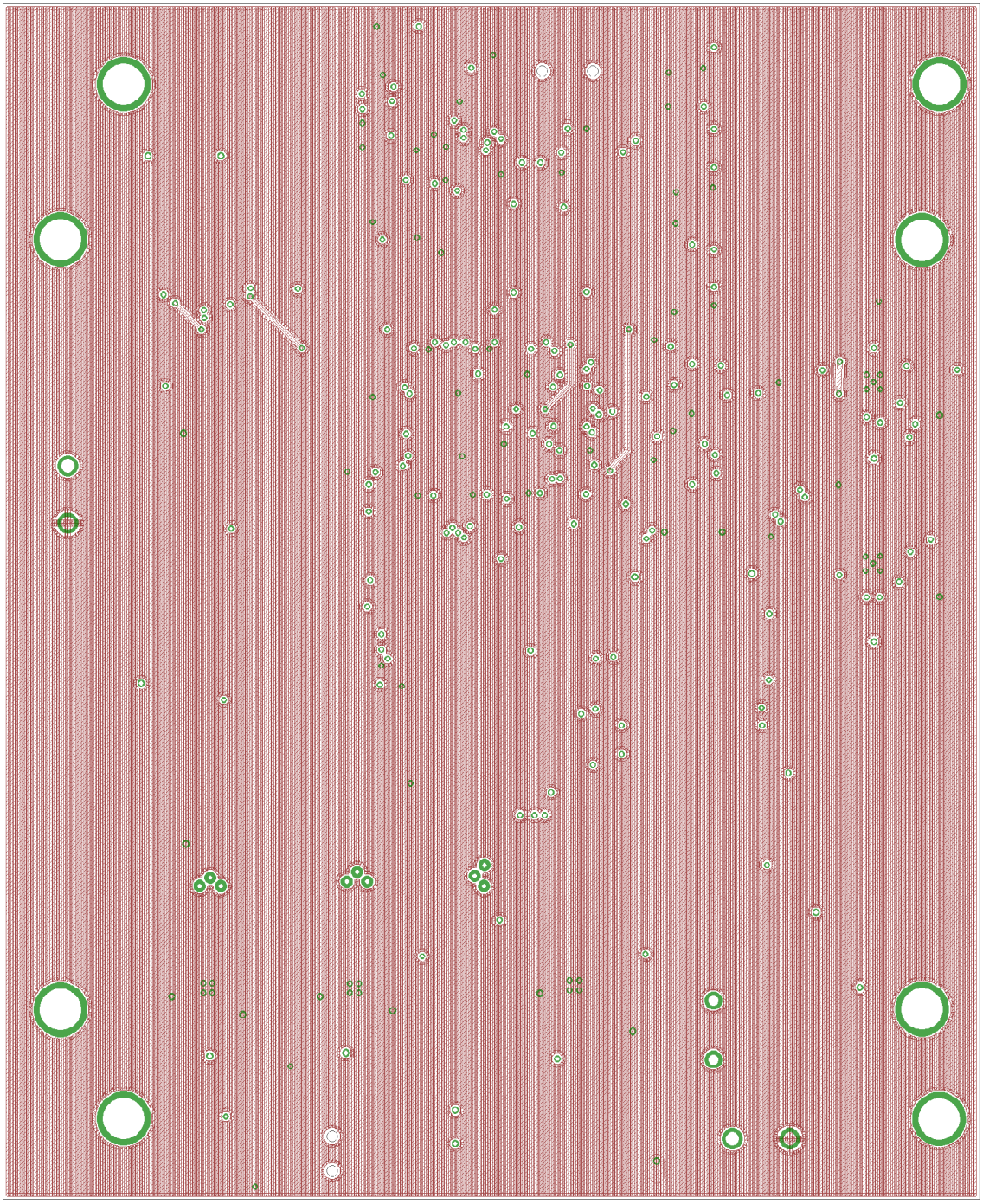
i. Top layer



ii. Power plane layer

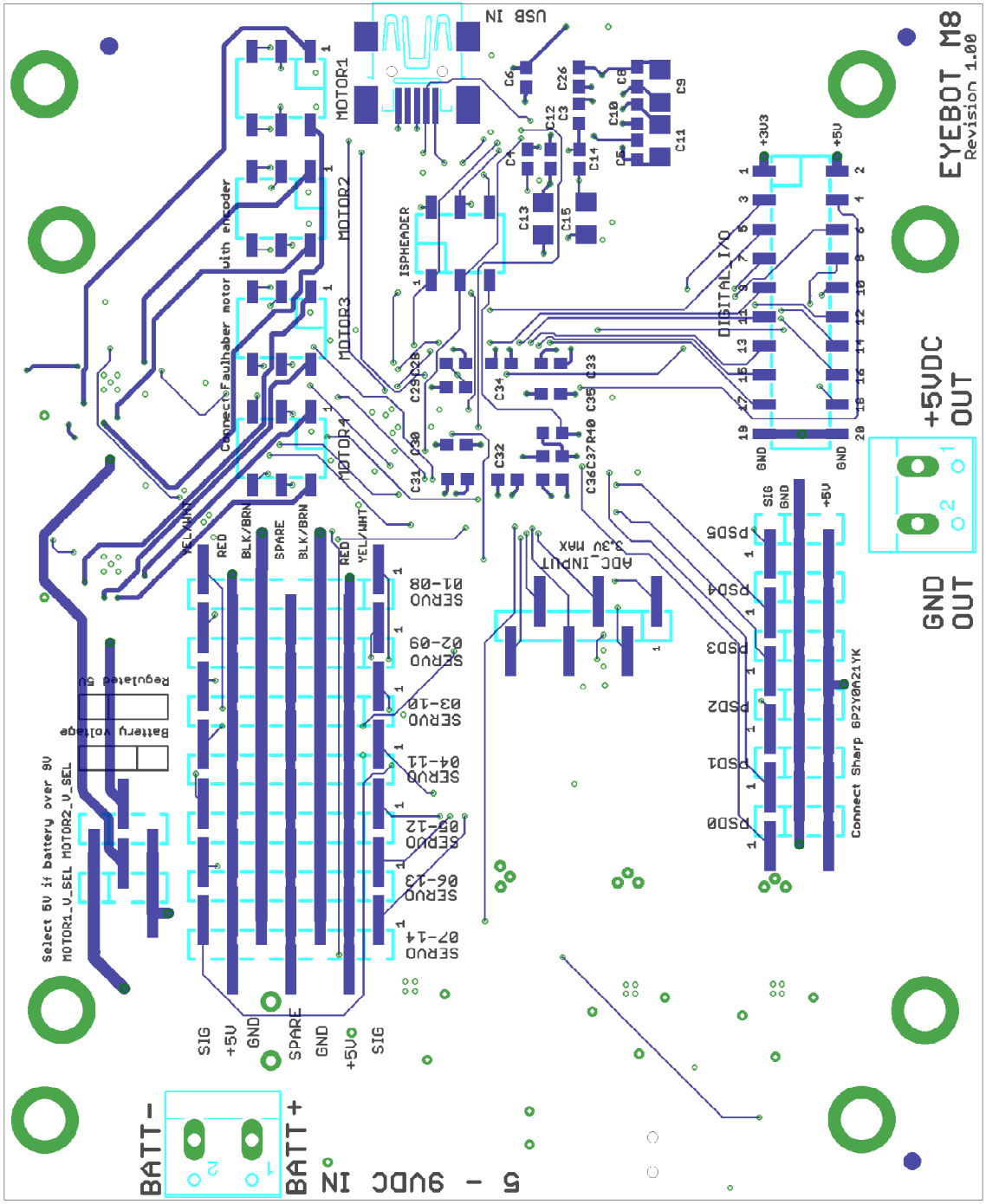


iii. Ground layer



iv. Bottom layer

Note: (mirrored for readability)



Appendix B: Test Software

Appendix B.1: xMega test program

```
/*
 * EyebotM8test.c
 *
 * Created: 31/10/2012
 * Author: Andrew Adamson
 */

#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>

//ADC calibration library
#include <stddef.h>
#include <avr/pgmspace.h>

//PCA9685 libraries
#include "pca9685.h"

//Bit positions
//Port A
#define PSD0_bp 0
#define PSD1_bp 1
#define PSD2_bp 2
#define PSD3_bp 3
#define PSD4_bp 4
#define PSD5_bp 5

//Port B
#define ADC0_bp 0
#define ADC1_bp 1
#define ADC2_bp 2 //Last four ADC's currently used for JTAG
#define ADC3_bp 3
#define ADC4_bp 4
#define ADC5_bp 5
#define VBATTSENSE_bp //Disconnected for JTAG at the moment

//Port C
//First two pins allocated to TWIC
#define M1_FL_bp 2 //Motor stall flags
#define M2_FL_bp 3
#define M3_FL_bp 4
#define M4_FL_bp 5
#define M12SLEEP_bp 6 //Sleep lines for each motor driver chip
#define M34SLEEP_bp 7

//Port D
#define M1O1_bp 0 //Motor 1 and 2 control lines
#define M1O2_bp 1
#define M2O1_bp 2
#define M2O2_bp 3
#define SPI_CS_bp 4 //SPI serial lines
```

```
#define SPI_MOSI_bp 5
#define SPI MISO bp 6
#define SPI_CLK_bp 7

//Port E
#define M3O1_bp 0 //Motor 3 and 4 control lines
#define M3O2 bp 1
#define M4O1_bp 2
#define M4O2 bp 3
#define SERVOEN_bp 4 //Extra power supply enable line
#define PWMEN bp 5 //PCA9685 PWM chip output enable line
#define  PWRON_bp  6  //Power  on  line  (not  needed  until  solder  jumper  is
cut)

//Port F
#define M1 ENC1 bp 0 //Two encoders per motor
#define M1_ENC2_bp 1
#define M2_ENC1_bp 2
#define M2 ENC2 bp 3
#define M3_ENC1_bp 4
#define M3 ENC2 bp 5
#define M4_ENC1_bp 6
#define M4_ENC2_bp 7

//Port H
//Caution: "GPIO" already has a #define
#define IO0_bp 0 //16 GPIO lines
#define IO1_bp 1
#define IO2_bp 2
#define IO3_bp 3
#define IO4 bp 4
#define IO5_bp 5
#define IO6_bp 6
#define IO7_bp 7

//Port J
#define IO8_bp 0
#define IO9 bp 1
#define IO10_bp 2
#define IO11 bp 3
#define IO12 bp 4
#define IO13_bp 5
#define IO14 bp 6
#define IO15_bp 7


//Turns on the high current 5V supply that drives the servos and motors
void extra_enable(void) {
      //Set PE4 as output
      PORTE.DIR |= (1<<SERVOEN_bp);
      //Turn on the servo+motor power supply.
      //There is an LED on the board to confirm the supply is on.
      PORTE.OUT |= (1<<SERVOEN_bp);
}

//Sets up the SPI ports and registers in slave mode
void spi_init(void) {
      /*
```

```
	* Configure MISO pin on Port D as output. Chip select is ignored
	* for now because we're only writing and there is only one slave
	*/
	PORTD.DIR |= 0b00100000;
	/*
	* Most significant bit transmitted first by omitting DORD flag,
	* enable spi, SPI mode 0, leave in slave mode by omitting master
	* enable flag
	*/
	SPID.CTRL = SPI_ENABLE_bm;
}

//Reads a byte out of the SPI data register
char spi_read(void) {
	//Wait for a write to the data register to complete
	while(!(SPID.STATUS&SPI_IF_bm));
	//Read the byte out of the register
	return SPID.DATA;
}

//Receives a byte from the test program by John Hodge, reads it,
//and turns on an LED if it matches some value
void spi_test(void) {
		spi_init();
	while(1) {
		uint8_t val = spi_read();
		/*
		* 0x37 is the value sent by John Hodge's spi_test program
		*/
		if ( val == 0x37) {
			//Use the LED for the extra power supply
			//as a diagnostic light
			extra_enable();
		}
		_delay_us(100);
	}
}

//Turns on motors 1 and 2. Reads one encoder line of motor 1 and
// makes IO0 match the input level
void motor_test(void) {
	//Port F is all motor encoder inputs
	PORTF.DIR = 0x00;

	//Enable the motor driver chips
	PORTC.DIR |= 0b11000000;
	PORTC.OUT |= (1<<M12SLEEP_bp);
	PORTC.OUT |= (1<<M34SLEEP_bp);

	//initialise output ports for the 4 motors
	PORTE.DIR |= 0x0F;
	PORTD.DIR |= 0x0F;

	//Enable the extra 5V power supply (assumes the motors are
	//selected to run off 5V with the jumper)
	extra_enable();

	/*
```

```
	* Can verify motor signals either with a faulhaber motor (be
	* sure to check the pinout, the old motors are different!)
	* or by connecting a CRO to the motor output pins. The lines
	* below will make the motors spin at full speed in one direction.
	* Speed control can be done using the timers on the xMega
	* to generate a PWM signal. Change or add to these lines to
	* control motors 3 and 4
	*/
	PORTD.OUT &= ~(1<<M1O1_bp);
	PORTD.OUT |= (1<<M1O2_bp);
	PORTD.OUT &= ~(1<<M2O1_bp);
	PORTD.OUT |= (1<<M2O2_bp);

	//Polling based code to check the encoder inputs are
	//working properly
	//Reads encoder from motor 1 and sets a GPIO pin high if
	//the encoder is outputting high
	//Can be verified with oscilloscope
	PORTH.DIR = 0xFF;
	while(1) {
		if (PORTF.IN > 0) {
			PORTH.OUT = 0x01;
		}
		else {
			PORTH.OUT = 0x00;
		}
	}
}

//Turns on motor 1. Reads the stall flag for motor 1 and
//makes IO0 match the input level
void motor_stall_test() {
	//Enable the motor driver chips
	PORTC.DIR |= (1<<M12SLEEP_bp) | (1<<M34SLEEP_bp);
	PORTC.OUT |= (1<<M12SLEEP_bp);
	PORTC.OUT |= (1<<M34SLEEP_bp);

	//initialise output ports for the 4 motors (two outputs per motor)
	PORTE.DIR |= 0x0F;
	PORTD.DIR |= 0x0F;

	//set the motor flag pins as pull-up (the A3908 chip pulls the
	//line low when the motor is stalled)
	PORTC.PIN2CTRL |= PORT_OPC_PULLUP_gc;
	PORTC.PIN3CTRL |= PORT_OPC_PULLUP_gc;
	PORTC.PIN4CTRL |= PORT_OPC_PULLUP_gc;
	PORTC.PIN5CTRL |= PORT_OPC_PULLUP_gc;

	//Enable the extra 5V power supply (we assume the motors
	//are selected to run off 5V with the jumper during testing)
	extra_enable();

	//turn on motor 1
	PORTD.OUT &= ~(1<<M1O1_bp);
	PORTD.OUT |= (1<<M1O2_bp);

	/*
	* polling based code to check the motor-stall flags do get set
```

```
	* when the motor is pulling more than 1A
	* Test by putting a high wattage 5ohm resistor across
	* the motor contacts (for a SHORT time) and checking any of the
	* first eight GPIO pins goes high with an oscilloscope
	*/
	//set all outputs on port H so we can use the GPIO pins
	PORTH.DIR = 0xFF; while(1) {
		if (!(PORTC.IN & (1<<M1_FL_bp))) {
			PORTH.OUT = 0x01;
		}
		else {
			PORTH.OUT = 0x00;
		}
	}
}

//Switches all GPIO pins on and off at 1Hz
void gpio_test(void) {
	//Initialise ports
	//Make all the GPIO ports outputs
	PORTH.DIR=0xFF;
	PORTJ.DIR=0xFF;
	//Set all the pins high then low. Verify signals with CRO.
	//For more comprehensive testing, set every alternate pin high
	//and also check with CRO to make sure there are no shorts
	//For super-comprehensive testing, change all the pin
	//states as fast as possible and check signal integrity
	while (1) {
		PORTH.OUT = 0xFF;
		PORTJ.OUT = 0xFF;
		_delay_ms(500);
		PORTH.OUT = 0x00;
		PORTJ.OUT = 0x00;
		_delay_ms(500);
	}
}

//Enables the eyebot specific features required to drive servos
void servo_enable(void) {
	//Enable outputs on the PCA9685
	//Set port direction
	PORTE.DIR |= (1<<PWMEN_bp);
	//Active low output enable. This only controls the output, and does
	//not disable I2C communication with the PCA9685
	PORTE.OUT &= ~(1<<PWMEN_bp);
	//Turn on the servo power
	extra_enable();
}

void servo_test(void) {
	//Set up eyebot specific things like enabling power
	servo_enable();
	//Initialise the PCA9685 on address 0xE0
	pca9685_init(0xE0);
	//Loop to make servos 1 to 3 swing backwards and forwards
	//to their extremes (1ms to 2ms duty cycle)
	//Pass a value between 0 and 4095 to set duty cycle
	//Signal frequency is set in pca9685.h (currently 184Hz)
```

```
	uint16_t i;
	while (1) {
		//Count up
		for(i = 800; i <= 1550; i+=20) {
			pca9685_send(0xE0, i, 0);
			pca9685_send(0xE0, i, 1);
			pca9685_send(0xE0, i, 2);
			_delay_ms(4);
		}
		//Count down
		for(i = 1550; i >= 800; i-=20) {
			pca9685_send(0xE0, i, 0);
			pca9685_send(0xE0, i, 1);
			pca9685_send(0xE0, i, 2);
			_delay_ms(4);
		}
	}
}

//Helper function to calibrate the ADC according to factory-set
//calibration value
uint8_t ReadCalibrationByte( uint8_t index )
{
	uint8_t result;

	//Load the NVM Command register to read the calibration row
	NVM_CMD = NVM_CMD_READ_CALIB_ROW_gc;
	result = pgm_read_byte(index);

	//Clean up NVM Command register
	NVM_CMD = NVM_CMD_NO_OPERATION_gc;

	return( result );
}

//Reads value from IR distance sensor and outputs it as 8 bits on the
// first 8 IO pins
void adc_test(void) {
	//Most of this function is based on an example from
	//http://www.bostonandroid.com/manuals/xmega-precision-adc-
	//howto.html

	//Calibrate the ADC from the production signature row
	ADCA.CALL  =  ReadCalibrationByte(  offsetof(NVM_PROD_SIGNATURES_t,
ADCACAL0) );
	ADCA.CALH  =  ReadCalibrationByte(  offsetof(NVM_PROD_SIGNATURES_t,
ADCACAL1) );

	//Set all of Port A as input
	PORTA.DIR = 0x00;
	//Set Port H as output to display the result
	PORTH.DIR = 0xFF;
	//12 bit conversion
	ADCA.CTRLB = ADC_RESOLUTION_12BIT_gc;
	/*
	* Use internal 1V bandgap reference
	* This means that only voltages between ground and 1V can be
	* measured, so the voltage must be dropped to a suitable level in
```

```
	* hardware
	*/
	ADCA.REFCTRL = ADC_REFSEL_INT1V_gc | 0x02;
	//Measure at 250kHz (divide system clock by 8)
	ADCA.PRESCALER = ADC_PRESCALER_DIV8_gc;
	//Select single ended measurement
	ADCA.CH0.CTRL = ADC_CH_INPUTMODE_SINGLEENDED_gc;
	//Choose which pin to use
	ADCA.CH0.MUXCTRL = ADC_CH_MUXPOS_PIN0_gc;
	//Set the enable bit to enable the ADC module
	ADCA.CTRLA |= 0x01;
	//Delay a little
	_delay_ms(3);
	//Go into a measurement loop
	while(1) {
		//Set the start-conversion bit to start a single conversion
		ADCA.CH0.CTRL |= ADC_CH_START_bm;
		//Wait for the conversion complete flag to be set
		while(!ADCA.CH0.INTFLAGS);
		//Read the value from the results register to
		// a local variable
		int result = ADCA.CH0RES;
		//Divide down to a range that can be represented with 8 bits
		char psd = result/16;
		//Set the pins on port H so the value can be read
		// with logic analyzer
		PORTH.OUT = psd;
		//Delay a little bit to make reading the byte easier
		// with logic analyzer
		_delay_ms(10);
	}
}

int main (void)
{
	/*
	* Uncomment the test you want to perform. Do these tests
	* individually since they were not written to work together (most
	* of the functions rely on polling rather than interrupts)
	*/
	//spi_test();
	//motor_test();
	//motor_stall_test();
	servo_test();
	//gpio_test();
	//extra_enable();
	//adc_test();
}
```

Appendix B.2: Two Wire Interface (I^2C) header

```
/*
 * xmegatwi.h
 * Author: Andrew Adamson
 * Provides polling based write routines to control a Two Wire Interface
 * (I2C) port on the xMega
 */

#ifndef XMEGATWI_H
#define XMEGATWI_H

#define CPU_SPEED 2000000
#define BAUDRATE 400000
#define TWI_BAUD(F_SYS, F_TWI) ((F_SYS / (2 * F_TWI)) - 5)
#define TWI_BAUDSETTING TWI_BAUD(CPU_SPEED, BAUDRATE)

/*
* Configures and initialises the given TWIx controller
* Input: pointer to a TWI struct
*/
void twi_init(TWI_t * twiname) {
	//enable smartmode to send the ack immediately after the data
	twiname->MASTER.CTRLB = TWI_MASTER_SMEN_bm;
	//Set the Baud
	twiname->MASTER.BAUD = TWI_BAUDSETTING;
	//Enable the TWI master
	twiname->MASTER.CTRLA = TWI_MASTER_ENABLE_bm;
	//Force the bus into idle mode
	twiname->MASTER.STATUS = TWI_MASTER_BUSSTATE_IDLE_gc;

	return;
}

/*
* Sets the given TWI controller to write mode for given slave address
* Input: pointer to a TWI struct, 8-bit slave address
*/
//Can probably refactor this to make a generic "start" function
void i2c_startWrite(TWI_t * twiname, uint8_t address) {
	//Write the address (which should end in a
	// zero for a write address)
	twiname->MASTER.ADDR = address & 0b11111110;
	//Poll for the write interrupt flag to be set
	while(!(twiname->MASTER.STATUS&TWI_MASTER_WIF_bm));
}

/*
* Puts a byte into the data register of the given TWI controller
* and waits for it to be sent to the slave
* Input: pointer to a TWI struct, 8-bit data value
*/
void i2c_write(TWI_t * twiname, uint8_t writeData) {
	//Write the data
	twiname->MASTER.DATA = writeData;
	//Poll for the write interrupt flag to be set
	while(!(twiname->MASTER.STATUS&TWI_MASTER_WIF_bm));
```

```
	return;
}

/*
* Puts a stop condition on the TWI bus to indicate
* communication has finished
* Input: pointer to a TWI struct
*/
void i2c_stop(TWI_t * twiname) {
	twiname->MASTER.CTRLC = TWI_MASTER_CMD_STOP_gc;
}
#endif /* XMEGATWI_H */
```

Appendix B.3: PCA9685 header

```
/*
 * pca9685.h
 * Author: Andrew Adamson
 * Based on PIC code from:
 * http://www.ccsinfo.com/forum/viewtopic.php?p=166816
 * Provides routines to initialise a PCA9685 and set servo PWM values
 */

#include "xmegatwi.h"
#include <util/delay.h>

#ifndef PCA9685 H
#define PCA9685_H

// Useful PCA9685 registers
#define MODE1 0x00      // 0x00 location of Mode1 register address
#define MODE2 0x01      // 0x01 location of Mode2 register address
#define SERVO0 0x06     // location for start of LED registers

/*
* Init code for the PCA9685
* Input: 8 bit write address of the PCA9685 to init
*/
void pca9685_init(uint8_t address) {
      /*
      * How to communicate with the PCA9685: send slave address,
      * a pointer to a register, and then the value of the register
      */
      twi_init(&TWIC);
      // Start
      i2c_startWrite(&TWIC, address);
      // Mode 1 address
      i2c_write(&TWIC, MODE1);
      // Setting mode to sleep so we can change the default PWM frequency
      i2c_write(&TWIC, 0b00110001);
      // Stop
      i2c_stop(&TWIC);
      // Required 50 us delay
      _delay_us(50);
      // Start
      i2c_startWrite(&TWIC, address);
      // PWM frequency PRE_SCALE address
      i2c_write(&TWIC, 0xfe);
      //The PWM frequency is set here. Set to 0x79 for 20ms period,
      //or 0x23 for 5.4ms period
      i2c_write(&TWIC, 0x23); // Value = 25000000/(4096*frequency)-1
      // Stop
      i2c_stop(&TWIC);
      // Delay at least 500 us
      _delay_us(500);
      // Start
      i2c_startWrite(&TWIC, address);
      // Mode 1 register address
      i2c_write(&TWIC, MODE1);
      // Set to our preferred mode[ Reset, INT_CLK, Auto-Increment,
```

```
	// Normal Mode]
	i2c_write(&TWIC, 0xa1);
	// Stop
	i2c_stop(&TWIC);
	// Delay at least 500 us
	_delay_us(500);
	// Start
	i2c_startWrite(&TWIC, address);
	// Mode2 register address
	i2c_write(&TWIC, MODE2);
	// Set to our preferred mode: output logic state not
	// inverted, outputs change on STOP,
	// totem pole structure, when OE = 1 (output drivers not
	// enabled), SERVOn = 0
	i2c_write(&TWIC, 0b00000100);
	// Stop
	i2c_stop(&TWIC);
}

/*
* Sends the 12 bit PWM data to the register
* Input: write address of the PCA9685, 0 to 4095 pwm value, 0 to 15 for
* servo channel
*/
void pca9685_send(uint8_t address, uint16_t value, uint8_t servo)
{
	// temp variable for PWM
	uint8_t pwm;
	// fully on if larger than 4095
	if(value > 4095) {
		value = 4095;
	}
	// Start
	i2c_startWrite(&TWIC, address);
	// Select the correct servo address
	i2c_write(&TWIC, SERVO0 + 4 * servo);
	// Servo on-time low byte
	i2c_write(&TWIC, 0x00);
	// Servo on-time high byte
	i2c_write(&TWIC, 0x00);
	// Take a copy of value
	pwm = value;
	// Servo off-time low byte
	i2c_write(&TWIC, pwm);
	// pwm is 16 bits long; shifts upper 8 to lower 8
	pwm = value>>8;
	// Servo off-time high byte
	i2c_write(&TWIC, pwm);
	// Stop
	i2c_stop(&TWIC);
}
#endif /* PCA9685_H */
```

Appendix B.4: FT232H Test Program

```
/*
 * FT232H SPI Test Program
 * - Created for the Eyebot M8 Board
 *
 * Author: John Hodge (20518201)
 * - Eyebot M8 by Andrew Adamson
 */
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <unistd.h>
#include <string.h>
#include <stdbool.h>
#include <ftd2xx.h>

// Buffer sizes, might want to tweak for production
#define OUTBUF_SIZE     512
#define INBUF_SIZE      512

// Some helpful macros
#define MIN(a,b)  ((a) < (b) ? (a) : (b))
#define Sleep(ms) usleep(ms*1000)

// === GLOBALS ===
FT_HANDLE   gDeviceHandle;
size_t      giOutBytes = 0;
uint8_t     gsOutBuffer[OUTBUF_SIZE];
size_t      giInBytes = 0;
uint8_t     gsInBuffer[INBUF_SIZE];

// === CODE ===
//
// Append data to the outbound queue
//
void AppendBuf(const void *Data, size_t Length)
{
      if( Length > OUTBUF_SIZE - giOutBytes ) {
            fprintf(stderr, "Out of space in output buffer\n");
            exit(EXIT_FAILURE);
      }
      Length = MIN(OUTBUF_SIZE - giOutBytes, Length);
      memcpy(gsOutBuffer + giOutBytes, Data, Length);
      giOutBytes += Length;
}

void AppendByte(uint8_t Byte)
{
      AppendBuf(&Byte, 1);
}

//
// Send the outbound queue to the device
//
FT_STATUS SendBuf(FT_HANDLE Handle)
{
```

```
	FT_STATUS	fts;
	DWORD bytes_sent;
	fts = FT_Write(Handle, gsOutBuffer, giOutBytes, &bytes_sent);
	if( fts != FT_OK ) {
		fprintf(stderr, "FT_Write failed (%i)\n", fts);
		giOutBytes = 0;
		return fts;
	}

	if( giOutBytes != bytes_sent ) {
		fprintf(stderr, "%i bytes queued, but only %i were sent.",
			(int)giOutBytes, (int)bytes_sent);
	}

	//printf("Send %i bytes\n", giOutBytes);
	giOutBytes = 0;
	return FT_OK;
}

//
// Wait for input and then read
//
FT_STATUS ReadInput(FT_HANDLE Handle)
{
	FT_STATUS	fts;
	 int  timeout = 1000;
	DWORD queue_bytes, bytes_read;
	giInBytes = 0;
	// Wait for data
	do {
		fts = FT_GetQueueStatus(Handle, &queue_bytes);
		if( !queue_bytes )
			usleep(1000);
	} while( fts == FT_OK && queue_bytes == 0 && --timeout );
	// - error (timeout included)
	if( fts != FT_OK )
		return fts;

	// Read as much as possible
	giInBytes = MIN(queue_bytes, INBUF_SIZE);
	fts = FT_Read(Handle, gsInBuffer, giInBytes, &bytes_read);
	if( fts != FT_OK )
		return fts;
	if( giInBytes != bytes_read ) {
		fprintf(stderr, "%i bytes in read queue, but only %i were
read\n",
			(int)giInBytes, (int)bytes_read);
	}
	giInBytes = bytes_read;
	return FT_OK;
}

//
// (Internal)
// Sends a byte to the MPSSE and checks for an error response.
//
FT_STATUS InitSPI_BadCmd(FT_HANDLE Handle, uint8_t cmdbyte)
{
```

```
	FT_STATUS	fts;

	// Ensure MPSSE is synchronized (send a bad command)
	AppendByte(cmdbyte);
	fts = SendBuf(Handle);
	if( fts != FT_OK )	return fts;
	fts = ReadInput(Handle);
	if( fts != FT_OK )	return fts;

	// Make sure the "Bad Command" response was recieved
	bool bad_command_found = false;
	if( giInBytes > 0 )
		printf("%02X ", gsInBuffer[0]);
	for( size_t i = 0; i < giInBytes - 1; i ++ )
	{
		printf("%02X ", gsInBuffer[i+1]);
		if( gsInBuffer[i] == 0xFA && gsInBuffer[i+1] == cmdbyte ) {
			bad_command_found = true;
			break ;
		}
	}
	printf("\n");
	if( !bad_command_found ) {
		fprintf(stderr, "MPSSE sync (0x%02X) failed. No 'Bad Command'
resp in %i bytes.\n",
			cmdbyte, (int)giInBytes);
		return FT_OTHER_ERROR;  // Mind if I use this?
	}

	return FT_OK;
}

//
// Initialise the FT232H for MPSSE mode and configure for SPI
// - Also turns on an LED, just for debugging
//
FT_STATUS InitSPI(FT_HANDLE Handle)
{
	FT_STATUS	fts;

	fts = FT_ResetDevice(Handle);
	if(fts != FT_OK ) return fts;

	// Clean out stale data in the buffer
	DWORD stale_buf_size;
	fts = FT_GetQueueStatus(Handle, &stale_buf_size);
	if( fts != FT_OK )	return fts;
	if(stale_buf_size > 0) {
		char  tmpbuf[stale_buf_size];
		fts	=	FT_Read(Handle,	tmpbuf,	stale_buf_size,
&stale_buf_size);
		if( fts != FT_OK )	return fts;
	}

	// Set some useful values
	fts = FT_SetUSBParameters(Handle, 0xFFFF, 0xFFFF);	//	set	USB
Req Transfer Size to 64k
	if( fts != FT_OK )	return fts;
```

```
	fts = FT_SetChars(Handle, false, 0, false, 0);  //           Disable
event/error chars
	if( fts != FT_OK )	return fts;
	fts = FT_SetTimeouts(Handle, 3000, 3000); //     set     read/write
timeouts to 3s
	if( fts != FT_OK )	return fts;
	fts = FT_SetLatencyTimer(Handle, 1);	//  Set  latency  timer  to
1ms (default is 16ms)
	if( fts != FT_OK )	return fts;
	fts   =   FT_SetFlowControl(Handle,   FT_FLOW_RTS_CTS,   0x00,   0x00);
	//Turn on flow control to synchronize IN requests (added by Andrew)
	  if( fts != FT_OK )	return fts;
	fts = FT_SetBitMode(Handle, 0, 0x00);	// Reset.
	if( fts != FT_OK )	return fts;
	Sleep(50);  // quick nap
	fts = FT_SetBitMode(Handle, 0, 0x02);	// Enable MPSSE mode
	if( fts != FT_OK )	return fts;

	// Sleep for a bit for USB to catch up
	Sleep(50);

	// Sync (send 0xAA - bad command)
	fts = InitSPI_BadCmd(Handle, 0xAA);
	if( fts != FT_OK )	return fts;
	// Second sync (0xAB)
	fts = InitSPI_BadCmd(Handle, 0xAB);
	if( fts != FT_OK )	return fts;

	// Base clock is 60MHz
	// Caution: high and low periods must be longer than 2 CPU clock
cycles
	uint16_t    divisor = 1200;   // 50KHz
	//uint16_t  divisor = 60;     //  1MHz  (cannot  run  at  this  speed
with a 2MHz system clock)
	AppendByte(0x8A); // Disable clock front stage divisor (/5)
	AppendByte(0x97); // Disable adaptive clocking
	AppendByte(0x8D); // Disable 3-phase data clock
	SendBuf(Handle);
	AppendByte(0x80); // "Set Data Bits Low Byte"
	AppendByte(0x00);
	AppendByte(0x0B);
	// NOTE: The below is for the debug LED
	AppendByte(0x82); // "Set Data Bits High Byte (AC Bus)"
	AppendByte(0x01); // (ACBUS0 High - Status LED)
	AppendByte(0x01);
	// Clock divisor
	AppendByte(0x86);
	AppendByte((divisor/2-1) & 0xFF);   // Low
	AppendByte((divisor/2-1) >> 8 );    // High
	SendBuf(Handle);
	// Wait a bit to ensure the chip is ready
	Sleep(30);

	// Turn off loopback on TDI/TDO connection
	AppendByte(0x85);
	SendBuf(Handle);

	// All done!
```

```
	return FT_OK;
}

//
// Raise the CS line
//
FT_STATUS SPI_CSRaise(FT_HANDLE Handle)
{
	// 5 repeats for 1us
	// - each command takes 0.2us to perform, so ensures that the
	//   line is high for at least 1us before data is sent
	for( int i = 0; i < 5; i ++ )
	{
		AppendByte(0x80); // GPIO ADBUS
		AppendByte(0x08); // CS line High
		AppendByte(0x0B); // (output mode, shouldn't change)
	}
	return FT_OK;
}

// Lower the CS line
FT_STATUS SPI_CSLower(FT_HANDLE Handle)
{
	// - 5 repeats for 1us (each command aparently raises for 0.2us)?
	for( int i = 0; i < 5; i ++ )
	{
		AppendByte(0x80); // GPIO ADBUS
		AppendByte(0x00); // CS line low
		AppendByte(0x0B); // (output mode, shoudn't change)
	}
	return FT_OK;
}

//
// Write a sequence of bytes to the SPI bus
//
FT_STATUS SPIWriteBytes(FT_HANDLE Handle, size_t Len, const void *Data)
{
	const uint8_t	*bdata = Data;

	if( Len == 0 )
		return FT_OK;

	// 0 = 1 byte, 0xFFFF = 2^16 bytes
	// - So, subtract one
	Len -= 1;

	// NOTE: CS is active-low
	SPI_CSLower(Handle);
	AppendByte(0x11); // MSB Falling Edge Change Clock - out on -ve
clock edge
	AppendByte(Len & 0xFF); // LSB
	AppendByte(Len >> 8);   // MSB
	// Append data from input buffer
	for( size_t i = 0; i < Len+1; i ++ )
		AppendByte(bdata[i]);
	SPI_CSRaise(Handle);
```

```
	return SendBuf(Handle);
}

int main(int argc, char *argv[])
{
	FT_STATUS fts;

	// Check for correct usage
	if( argc != 2 )
	{
		fprintf(stderr, "Usage: %s ID\n", argv[0]);

		DWORD ndev;
		fts = FT_CreateDeviceInfoList(&ndev);
		if( fts != FT_OK ) {
			fprintf(stderr, "Enum failed, FT_CreateDeviceInfoList,
fts = %i\n", fts);
			return 1;
		}

		printf("%i devices\n", ndev);

		return -1;
	}

	// Open the device
	int port = atoi(argv[1]);
	fts = FT_Open(port, &gDeviceHandle);
	if( fts != FT_OK ) {
		fprintf(stderr, "FT_Open(%i) failed, fts = %i\n", port, fts);
		return 1;
	}

	// Initialise SPI
	fts = InitSPI(gDeviceHandle);
	if( fts != FT_OK ) {
		fprintf(stderr, "InitSPI failed, fts = %i\n", fts);
		return 1;
	}

	// Continuously send test data to the device
	while( 1 )
	{
		uint8_t	data[] = {0x37};
		fts = SPIWriteBytes(gDeviceHandle, sizeof(data), data);
		sleep(10);
	}

	// Clean up, like a good program
	FT_Close(gDeviceHandle);

	return 0;
}
```

Appendix C: SPI configuration

Appendix C.1: SPI Modes

For the following table,

- CPOL=0 means the base value of the clock is zero
- CPOL=1 means the base value of the clock in one
- CPHA=0 means sample on the leading (first) clock edge
- CPHA=1 means sample on the trailing (second) clock edge

Mode	**CPOL**	**CPHA**
0	0	0
1	0	1
2	1	0
3	1	1

Appendix C.2: SPI Settings to Communicate with FT232H

Setting	Value
Clock frequency	50kHz. This can be increased as long as the high and low times are longer than two of the microcontroller clock periods.
Endianness	Most significant bit first
Clock polarity (CPOL)	0
Clock phase (CPHA)	0
Chip select polarity	Low enable
FT232H transmission direction	Currently set to write only with opcode 0x11. Can be set to bidirection with opcode 0x34.

8 References

AA Portable Power Corp, 2012, *NiMH Battery Pack: 7.2V 4500mAh for RC-10 Cars and Sumo Robots*. Available from: <http://www.batteryspace.com/nimhbatterypack72v4500mahforrc-10carsandsumorobots.aspx>. [accessed 5 July 2012].

Atmel Corporation, 2009, *AVR1005: Getting started with XMEGA*. Available from: <www.atmel.com/Images/doc8169.pdf>. [accessed 24 October 2012].

Atmel Corporation, 2009, *xMega A Manual*. Available from: <www.atmel.com/Images/doc8077.pdf>. [accessed 18 July 2012].

Atmel Corporation, 2012, *AVR1300: Using the Atmel AVR XMEGA ADC*. Available from: <http://www.atmel.com/Images/doc8032.pdf>. [accessed 2 November 2012].

Atmel Corporation, n.d., *AVR Dragon PDI Programming*. Available from: <http://www.atmel.no/webdoc/avrdragon/avrdragon.pdi_description.html>. [accessed 30 October 2012].

Bailey, DG 2011, *Design for Embedded Image Processing on FPGAs*, John Wiley & Sons.

BeagleBoard.org, *BeagleBoard Hardware Design*. Available from: <http://beagleboard.org/hardware/design/>. [accessed 20 March 2012].

Blackham, B 2006, The Development of a Hardware Platform for Real-time Image Processing, Honours thesis, University of Western Australia.

Bräunl, T, *EyeBot - Online Documentation*, The University of Western Australia. Available from: <http://robotics.ee.uwa.edu.au/eyebot/>. [accessed 30 March 2012].

Bräunl, T 2012, *ELEC2303 Lab Manual*, The University of Western Australia.

Chin, L 2006, FPGA Based Embedded Vision Systems, Honours thesis, University of Western Australia.

Coley, G 2010, *BeagleBoard-xM Rev C System Reference Manual*, C.1.0 vols, BeagleBoard.org.

Dietrich, B 2009, Design and Implementation of an FPGA-based Stereo Vision System for the EyeBot M6, Masters thesis, Technische Universität München and The University of Western Australia.

Du, JL 2003, Swarm Clustering System with Local Image Processing and Communication, Masters thesis, University of Western Australia and University of Stuttgart.

Ewan MacLeod 2008, *Eyebot M6 Controlled Sensor Package in a Renewable Energy Vehicle - Hyundai Getz*, The University of Western Australia.

Future Technology Devices International Ltd., 2011, *Application Note AN_108: Command Processor for MPSSE and MCU Host Bus Emulation Modes*. Available from: <http://www.ftdichip.com/Documents/AppNotes/AN_108_Command_Processor_for_MPSSE_and_MCU_Host_Bus_Emulation_Modes.pdf>. [accessed 8 November 2012].

Geier, M 2009, Design and Implementation of an FPGA-based Image Processing Framework for the EyeBot M6, Masters thesis, Technische Universität München and The University of Western Australia.

Lewis, P, 2008, *Sneaky Footprints: The Quest for a Right-Angle:*, Sparkfun Electronics. Available from: <http://www.sparkfun.com/tutorials/114>. [accessed 12 August 2012].

NXP, 2010, *PCA9685 Datasheet*. Available from: <www.nxp.com/documents/data_sheet/PCA9685.pdf>. [accessed 11 July 2012].

NXP, 2012, *I2C-bus specification and user manual*. Available from: <http://www.nxp.com/documents/user_manual/UM10204.pdf>. [accessed 1 August 2012].

Pololu Robotics and Electronics, 2011, *Servo control interface in detail*. Available from: <http://www.pololu.com/blog/17/servo-control-interface-in-detail>. [accessed 13 August 2012].

Rugged Circuits LLC, 2011, *10 Ways to Destroy an Arduino*. Available from: <http://ruggedcircuits.com/html/ancp01.html>. [accessed 4 October 2012].

Singh, J 2011, Hardware Redesign of an Experimental Embedded Platform, Honours thesis, University of Western Australia.

Texas Instruments Incorporated, 2012, *LM2678 SIMPLE SWITCHER High Efficiency 5A Step-Down Voltage Regulator (Rev. H)*. Available from: <http://www.ti.com/product/lm2678#technicaldocuments>. [accessed 10 October 2012].

Texas Instruments Incorporated, 2012, *TMS320C6000 DSP Library*. Available from: <http://www.ti.com/tool/sprc265>. [accessed 27 May 2012].

van Nes, N & Cramer, J 2005, 'Influencing product lifetime through product design', *Business Strategy and the Environment*, vol. 14, no. 5, pp. 286-299.

Yaghmour, K 2009, *Building Embedded Linux Systems*, Sebastopol, p. 232.