CODEN:LUTEDX/(TEIE-5409)/1-57(2019)

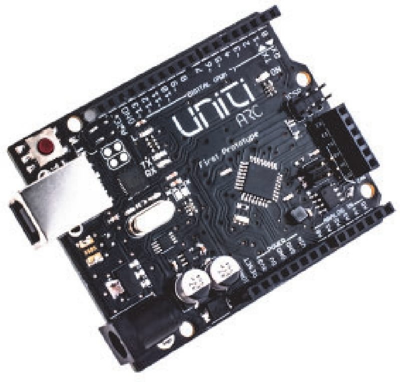
Industrial Electrical Engineering and Automation

Genuino/Arduino Compatible Board Using the ATmega M1-family of Microcontrollers

Simon Wrafter

Division of Industrial Electrical Engineering and Automation Faculty of Engineering, Lund University

Genuino/Arduino compatible board using the ATmega M1-family of microcontrollers



Simon Wrafter

Industrial Electrical Engineering and Automation
Lund University, Sweden

10th June 2018

Abstract

Arduino is a suite of software and hardware designed for the Atmel ATmega family of microcontrollers. In this project the Arduino suite has been adapted for the ATmega32M1 with the intent of utilising the built in three phase motor controller and CAN bus interface, making it easier to quickly prototype electric vehicles or other new applications.

There are a number of subtle differences between the ATmega32M1 and the ATmega328P used in the Arduino UNO. The hardware and software has therefore had to adapted and extended to fit the new microcontroller.

Furthermore a Motor Shield was designed and produced for the three phase motor controller of the ATmega32M1. On the Motor Shield there are three high/low-side gate drivers and six power MOSFETs capable of working at up to $60\,\mathrm{V}$ producing an estimated maximum $2\,\mathrm{kW}$ of power.

The overall design has proven successful with some minor bugs remaining. By relaxing the design requirements somewhat, much more powerful motors could be supported further extending the range of applications of the design.

Sammanfattning

Arduino är en hårdvaru- och mjukvarusvit designad för Atmels ATmega familj av mikrokontroller. I detta projekt har Arduinosviten anpassats för ATmega32M1 med avsikt att nyttja den inbyggda trefasmotor-kontrollern och CAN bus kommunikationen. Målet är att göra det lättare att utvekla prototyper till elektriska fordon eller andra applikationer.

Det finns ett antal mindre skillnader mellan ATmega32M1 och ATmega328P från Arduino UNO. Både hårdvaran och mjukvaran har därför anpassats för att passa med den nya mikrokontrollern.

Därtill har en Motor Shielddesignats och tillverkats för trefasmotor-kontrollern i ATmega32M1. På denna finns det tre hög/låg-sida gate-drivare och sex effekt MOSFETar som kan arbeta vid upp till $60\,\mathrm{V}$ och producerar uppskattningsvis maximalt $2\,\mathrm{kW}$.

Den övergripande designen har visats vara framgångsrik, även om det finns vissa kvarstående förbättringspunkter. Genom mer tillåtande designrestriktioner kan kraftfullare motorer stödjas och applikationsområdet för designen breddas.

Foreword

The idea for this thesis project has been with me for some time, so to get the opportunity to execute it according to my own ideas and adaptations has been an exciting experience.

However, I would never have succeeded without the help and support of those around me. Throughout I have relied on the helping hand of Freddie Olsson who supported me with his knowledge of power electronics and electric motor systems. His input has greatly influenced component selection, system design, and software implementation for which i am greatly thankful.

A big thank you also to David Bengtsson for helping with the software implementation of the CAN bus interface, allowing me to further concentrate on the hardware design and core software implementation.

Gunnar Lindstedt and Bengt Simonsson of the Industrial Electrical Engineering and Automation department at Lund University have been supportive and allowed me to work freely and find my own way. A project like this would not be otherwise possible. Thank you for giving me that freedom.

When work has been difficult I have had the unfaltering support and encouragement to push on from my family. Thank you for keeping me going.

Lastly I extend a thank you to the Uniti team for this opportunity, deviating from the main focus of their project.

Acronyms

CAN Control Area Network. 1, 6, 9–12, 20, 23, 26, *Glossary:* CAN

DAC Digital to Analogue Converter. 10, 19, 20, 26

ECAD Electronic Computer-Aided Design. 2, 7, 8

EEPROM Electrically Erasable Programmable Read-Only Memory. 20

EMS Electronics Manufacturing Services. 27

GPL GNU Public License. 8, *Glossary:* GPL

I2C Inter-Integrated Circuit. 10, 20, 22

IDE Integrated Development Environment. 2, 7, 13, 18, 19, 24, 26

JSON JavaScript Object Notation. 19

LIN Local Interconnect Network. 9, 10, 18, 19, 22, 25

LUFA Lightweight USB Framework for AVRs. 18, 24

MCU Microcontroller Unit. 1–4, 7, 9–13, 18–20, 23, 24, 26–28, 30, 33

MOSFET Metal–Oxide–Semiconductor Field-Effect Transistor. 13–17, 38

NRWW No Read While Write. 9, 22

PCB Printed Circuit Board. 6–8, 12–14, 24

PSC Power Stage Controller. 1, 3, 9–11, 16, 20, 26

PWM Pulse Width Modulated. 1, 3, 10, 20, 21, 24

RWW Read While Write. 9

SPI	Serial Peripheral Interface Bus. 10, 20, 21
UART	Universal Asynchronous Receiver/Transmitter. 9, 18–20, 22, 25
USB-IF	USB Implementers Forum. 18, 25

Glossary

Bootloader

In this case, a program residing in protected memory that runs before the main code. Definitions may vary.. 2, 7–9, 18, 22, 24, 25

CAN

A serial communications bus used primarily in automotive applications.. ii, 1, 8, 13

Creative Commons

A set of licenses freely available with the purpose of promoting shared cultural and artistic work.. 8

GPL

A common license used for free software.. ii, 8

Mil

Unit of measurement, 1/1000th of an inch.. 16, 17

Peripheral

An ancillary device used to put information into and get information out of the computer [1].. 1, 3, 9, 11, 13, 18–20, 25, 26

Shield

An add-on board for Arduino that clips in to the pinheaders.. 1–5, 7, 8, 13, 23, 24, 28

Table of Contents

1. | Introduction

This project has been a back-of-the-head idea for some time. In this section the background of the project is described as well as the basic implementation idea.

1.1 | Background

Hobbyist or Maker projects seldom use motors or components that draw more power than a few Watts at most. Arduino who are at the centre of hobbyist electronics, have a now retired product called Arduino Motor Shield [2] designed for running DC motors at up to $4\,\mathrm{A}$ and $5\,\mathrm{V}$ to $12\,\mathrm{V}$. For higher power applications the builder is required to design custom driver stages.

Motors running on alternating current, however, are not possible to control directly using an Arduino UNO or any of its most common clones. Commercially available AC motor drivers are proprietary and expensive, non of which are geared towards the maker community or designed for use with a Arduino.

During the course Formula Student (MVKN05) [3, 4] at Lund University, it became apparent that the Atmel ATmega32M1 [5] Microcontroller Unit (MCU) used was capable of running the Arduino software. For the purposes of Formula Student the ATmega32M1 was used due to its integrated Control Area Network (CAN) [6] controller. It also possesses a peripheral branded as Power Stage Controller (PSC) which is a single timer connected to three pairs of complementary output pins. The complementary output pairs each produce Pulse Width Modulated (PWM) signals such that while one pin is driving a high signal the other pin in the pair is held low, and vice versa. These three output pairs can be used to control three phase motors, syncronous or asyncronous alike, as well as for that matter two- or single-phase motors. At the time it was not possible to further investigate this possibility, and it was not picked up until a year later at Uniti Sweden AB [7].

1.2 | Uniti ARC

The decision was made at Uniti to develop an Arduino UNO clone using the ATmega64M1, which is in all respects identical to the aforementioned ATmega32M1 apart from available memory. However, due to limited component availability and cost the ATmega32M1 was selected for use in the first prototype design to be replaced later. This thesis represents the first design iteration.

The name Uniti ARC was decided upon for this board, and will be used in this report in reference to it.

1.2.1 | Hardware

To maximise the use cases of the ARC the choice was made to make it identical in size, shape, and pinout to the Arduino UNO. Thereby ensuring compatibility with existing example projects, shields, and other product extensions. All design files for the UNO are freely available under a Creative Commons Attribution Share-Alike [8] licence from the official product website [9]. The available design files are for Eagle [10], a common low-cost Electronic Computer-Aided Design (ECAD) tool popular with hobbyists. By basing the new board on the UNO the design effort is lowered by only needing to alter parts of it rather than redesigning everything.

The process is broken down into subtasks as follows:

- Pin-mapping for Arduino compatibility.
- Board design, adding ATmega32/64M1 and other additional components while retaining the board outline.
- Manufacturing, PCB sourcing and hand soldering.

Design files for the Uniti ARC are available on GitLab [11].

1.2.2 | Software

All code related to the Arduino products are released under the GNU General Public Licence, Version 2 [12] meaning that everyone is free to access and modify the code. This code is available on GitHub [13].

For the Uniti ARC to perform as an Arduino clone there are a number of steps to take to achieve this:

- Port code for the ATmega16u2 used as the boards USB interface.
- Port bootloader to ATmega32/64M1 MCU.
- Method of adding entry to Arduino Integrated Development Environment (IDE) ‘Boards’ menu.
- Port Arduino libraries to ATmega32/64M1 MCU.
- Add code for new features not found in the UNO or other Arduino products.

- Create Windows driver for Uniti ARC.

The tasks vary greatly in complexity, where some tasks are merely a matter of changing a few lines of code to reflect the new setup, others are rather invasive with large rewrites of code sections.

All code for the Uniti ARC is available at GitLab [14].

1.3 | Uniti ARC Motor Shield

As mentioned in Section 1.1 ‘Background’ the ATmega M1-family of MCUs have a special peripheral called PSC. Intended for use in high power applications, it consists of a single timer that controls three pairs of output pins capable of generating high frequency PWM signals. There are several uses for a PSC, perhaps the most prominent is the control and running of three phase motors for which the Motor Shield is designed.

1.3.1 | Hardware

No previous design is available to modify for this part of the project, a complete new design is required. Inspiration has been taken from Shane Colton’s ‘Hexbridge’ design [15], a private project published on his blog. The ‘Hexbridge’ is intended for use with a Arduino UNO, but it does not have the PSC peripheral so the process of generating the three sinusoidal waves relies on a less robust control process and more advanced MOSFET gate drivers.

The design targets of the Motor Shield were:

- Fit onto the Uniti ARC pin headers.
- Board allowed to be larger than Uniti ARC.
- Features available in the PSC do not need to be present on the Motor Shield.
- Galvanic separation between Motor Shield and Uniti ARC.
- Voltage limit at 60V, current draw limit dependant on board design.
- Possibility to modify board for high voltage applications.
- Temperature sensing, protect transistors from overheating.
- Back-EMF voltage sensing, output only in logic levels.

- Bonus: further additions that enhance or expands the usability of the Motor Shield may be added later.

1.3.2 | Software

No component on the Motor Shield is programmable or provides a communication interface. There is therefore no additional specific code to write for it, other than a Uniti ARC library making features of the Motor Shield as easy as possible to use. Such a library should:

- Take speed as an input and run a motor independently.
- Have an encoder interface.
- Be able to start a motor from a complete standstill.
- Run a motor by encoder of back-EMF alone.

1.4 | Prior Work

This thesis project is a derivative work primarily based on that from Arduino/Arduino, but also on on prior efforts to port the software to the ATmega32/64M1.

1.4.1 | Arduino

The Arduino company website is the primary source for retrieving prior work with the Arduino UNO [9], as this is the product upon which the Uniti ARC is based. This is possible thanks to the open source licensing of the design files and software. Arduino do not support the ATmega32/64M1 MCUs, nor do they have any three phase motor controllers or higher power boards.

1.4.2 | Porting Efforts

Two Arduino ports for ATmega32/64M1 are know of and freely available, one by Stuart Cording [16] and a second by Al Thomason [17]. The effort needed achieve compatibility with the Uniti ARC is significantly lowered by using as much of their code as possible.

1.4.3 | Motor Shield

No three phase motor shield is commercially available for the Arduino products. However, hobbyist designs such as the Hexbridge [15] are available and the design files can be downloaded and opened using Eagle [10]. The Hexbridge uses a different method for controlling the motor output than the Uniti ARC, but inspiration can be drawn from looking at the methods used.

1.5 | Project Limitations

The main focus of the thesis project is to complete the design and manufacturing of the hardware for Uniti ARC and Motor Shield. The software is a side target, it is necessary to prove the functionality of the system and integral to the design process and helps with understanding the features; therefore it is included in this report. Testing the Motor Shield dynamically in an advanced setup is a low priority target.

2. | Methodology

This chapter takes a few steps back, the method and process needs to be properly analysed. A discussion on possibilities and opportunities of the project has been had in Chapter 1 ‘Introduction’ and a set of targets there laid out. In this section further details will be discussed and the working process defined.

2.1 | Process

To reach the targets mentioned in Section 1.2 ‘Uniti ARC’ and Section 1.3 ‘Uniti ARC Motor Shield’ a process or sequence of developmental steps need to be defined. The steps are discussed below.

2.1.1 | Feasibility

The very first thing that needed to be done is to verify that the project is at all possible. By finding prior art and gaining an understanding of the technical requirements feasibility can be proven.

2.1.2 | Concept/Target generation

Concept generation is split in two parts, core and additional features.

The core concept is:

1. a board as similar to the original Arduino as possible in terms of physical size, pinout, programming interface, and software, adding two key features: CAN and three phase motor control using the ATmega32/64M1.

2. a board containing the power stage needed to power a small three phase motor.

Additional targets are less obvious requiring proper analysis of benefits versus implementation cost to decide what is to be included and what to omit. Cost is assessed through (in no particular order) design effort, Printed Circuit Board (PCB) real estate, and component price. Benefits are assessed from a user perspective based on ease of use, value added, flexibility, and feature availability in comparable products.

2.1.3 | Design

Despite the similarities between the ATmega32/63M1 [5] and ATmega328P [18] of the Arduino UNO, the process of redesigning the board to fit the new MCU requires some thought. The pinout of the two MCUs mentioned are very different, despite this the pinout of the boards should remain unaltered with all functionality available in the places where it is expected.

After achieving a satisfactory pinout the next step is to modify the Arduino UNO [9] design files using Eagle ECAD [10]. Also work on designing the Motor Shield can be started using the pinout configuration.

A two layer PCB design is used for both boards, in keeping with the design of the majority of products by Arduino. A low layer count also helps any new users to independently analyse and understand the design.

2.1.4 | Hardware Manufacturing

A straight forward process of soldering components to the boards. Due to the large number of surface mounted components some care needs to be taken in soldering to avoid damage to components. In addition to a soldering pen, a hot air soldering station is used to heat larger areas or components to avoid cold soldering joints.

2.1.5 | Software creation

As mentioned in Section 1.2 ‘Uniti ARC’ To make the Uniti ARC programmable using the Arduino IDE the process is as follows.

1. Port code for the ATmega16u2 used as the boards USB interface.
2. Port bootloader to ATmega32/64M1 MCU.
3. Method of adding entry to Arduino IDE ‘Boards’ menu.
4. Port Arduino libraries to ATmega32/64M1 MCU.
5. Add code for new features not found in the UNO or other Arduino products.
6. Create Windows driver for Uniti ARC.

The original software for this available from Arduino’s GitHub [13], with the adapted work available on GitLab [14].

Development work is done on a Linux platform using avr-gcc, avr-libc, avrdude, make, and git to compile, program and manage the code. The physical programming tool used is the AVRISP mk2 [19].

Porting the USB interface first along with the bootloader is important, once those two components are functional the libraries can be ported one at a time and individually verified on hardware.

2.1.6 | Testing & Verification

By programming the ARC with the required bootloader and running simple applications switching the outputs, the hardware can quickly be considered sufficiently tested. The same goes for the CAN interface, running a test program bouncing messages back and forth between two units is sufficient to consider the interface functional.

Testing the Motor Shield is also kept on a functional level. Ensuring motors can be run, but no further analysis of the performance on a component level is to be done.

Finally a check will be performed to ensure that the design requirements listed in Section 1.2 and Section 1.3 have been fulfilled.

2.2 | Open Source

Open Source is an integral aspect of this project. If Arduino had not released their design files and code under Creative Commons [8] and GNU Public License (GPL) [12] respectively, there would be nothing to build upon for this thesis. Open Source is a great enabler and accelerator of new ideas and technological efforts.

The basic principle of Open Source is sharing. Sharing work files – CAD or software – and in return who ever uses the files in derivative work has to publish their work too.

2.3 | Tools

Using a PC running a Linux environment the software tools used in this project are:

Cadsoft Eagle ECAD, schematic and PCB layout tool [10]
avr-gcc AVR compiler [20]
avr-libc standard C library for AVR [21]
avrdude AVR programming software [22]
make build automation tool [23]
git version control tool [24]

Hardware tools needed are:

AVRISP mk2 AVR programmer [19]
soldering station for manufacturing
hot air soldering station for heating larger areas and components
oscilloscope & multimeter for measuring signals and voltages

3. | Results

This chapter will describe the results and findings of the project.

3.1 | Evaluating the Microcontroller

Beyond the most notable additional peripherals, there are a number of other changes to the design of the ATmega32/63M1 [5] compared to the ATmega328P [18] on the Arduino UNO [9].

3.1.1 | Feasibility

There are a few similar projects available on the internet where the Arduino software has been ported for ATmega32/64M1. The main source of inspiration for this project has been the work done by Al Thomason [17], who in turn bases his work on that of Stuart Cording [16]. Referring to these two, it is evident that it is feasible to build an Arduino clone around an ATmega32/64M1.

Specifically the requirement for running the Arduino bootloader is that the flash memory is divided into two parts, Read While Write (RWW) and No Read While Write (NRWW) [5]. NRWW is the section where the bootloader is located such that the MCU can download and reprogram itself. No further dissection of the technicalities of this feature is needed for this thesis, verifying the availability of the feature is sufficient.

3.1.2 | Peripherals

The ATmega32/64M1 [5] is used in this project for its CAN and PSC peripherals. However, it also has a number of other additions and changes. Listed below are the key features of the ATmega32/64M1 as listed in the datasheet [5].

- CAN 2.0A/B with 6 message objects - ISO 16845 certified
- Local Interconnect Network (LIN) 2.1 and 1.3 controller or 8-Bit Universal Asynchronous Receiver/Transmitter (UART)

- One 12-bit high-speed PSC
- One 8-bit, and one 16-bit general purpose Timer/Counter
- One master/slave Serial Peripheral Interface Bus (SPI) serial interface
- 10-bit ADC:
  - Up to 11 single ended channels and 3 fully differential ADC channel pairs
  - Programmable gain (5x, 10x, 20x, 40x) on differential channels
  - Internal reference voltage
  - Direct power supply voltage measurement
- 10-bit Digital to Analogue Converter (DAC) for variable voltage reference (comparators, ADC)
- Four analogue comparators with variable threshold detection

New features when compared to the ATmega328P [18] are: CAN, LIN, PSC, differential analogue inputs to ADC, analogue comparators, DAC, four interrupts. Missing from the list is Inter-Integrated Circuit (I2C) which can be rather easily implemented in software. Also one 8-bit Timer/Counter is missing, meaning there are only three PWM outputs available (a fourth output shares pin PE1 with XTAL1, the crystal oscillator input). The PSC, however, can be configured to run in a mode with independent outputs, thus allowing for an additional six PWM outputs for a total of nine. Using both the Timers and PSC all six PWM outputs available on the Arduino UNO are accounted for.

3.1.3 | Package & Pinout

No through hole package is available for ATmega32/64M1 as used on the Arduino UNO. Instead a TQFP-32 package was selected as the visible pins are easier to solder by hand and make the purpose of the component a bit more obvious to a novice user.

There are four more pins on the ATmega32/64M1 as it is available in a 32 pin package whereas the ATmega328P has 28 pins. To fit them to the board an additional pinheader is added.

Pinout changes a lot between the two MCUs, Where the ATmega328P follows a more common port-wise grouping of pins the ATmega32/64M1 has a rather random pinout. It should also be noted that functions are not assigned to the same output pins. The pinout therefore has to be completely reworked.

3.2 | Pin & Function Mapping

The process of mapping the pins of the ATmega32/64M1 to the required pins of the board is divided into three steps.

1. Assign trivial pins.
2. Assign pin groups.
3. Select pins from groups according to best fit.

Pins 0, 1, 3, 11, 12, and 13 on the board are trivially assigned. Groups are formed where the assignment of pins can be made in several ways, the selection is done to ease the board layout as much as possible as well as ordering new features in an as logical way as possible. The resulting pin map is available in Table A.2 of Appendix A. Peripherals bolded in the right half of the table correspond to the feature named to the left, those not bolded do not correspond to a feature on the Arduino UNO.

The CAN bus and analogue outputs have no counterpart on the Arduino UNO and are thus placed in a new pinheader. An additional two pins are also needed to output the high and low outputs from the required CAN transceiver. A total of six outputs are needed on the new pinheader.

Pin mapping is done with little regard to the Motor Shield, other than having all PSC pins on the digital side. Beyond that the resulting pinout will have very little influence on the Motor Shield schematics and layout.

3.3 | ARC Design

With the pin map completed, it remains to properly select components before the schematics and layout stage can be done.

3.3.1 | Component Selection

New components have to be selected and added, old components used in the original Arduino UNO design have to be checked. All resulting design files are available through GitLab [11] and Appendix B.

3.3.1.1 | Components Added

The new components added to the design are:

- ATmega32/64M1 MCU.
- Decoupling circuitry for AVCC and AGND.

- Microchip MCP2561 CAN transceiver.
- Decoupling capacitor for MCP2561.
- Termination resistors for CAN bus, engaged by bridging two pairs of pins.
- New pinheader.

Adding a decoupling circuitry to the ATmega32/64M1 analogue supply pins is not strictly needed. It was added as it is recommended in the datasheet and improves the quality of analogue signal handling. Also the perceived reduction in quality of the $5\,\mathrm{V}$ and ground paths is mitigated to a degree by the decoupling circuitry. A further analysis as to the necessity of the decoupling has not been done, but should be completed before building any further prototypes.

Microchip MCP2561 is needed to translate the CANRX and CANTX signals of the ATmega32/64M1 CAN controller to the proper high and low signal pair used on the CAN bus. The choice of component is down to its popularity and known quality.

A CAN bus has to be terminated at both ends by a $120\,\Omega$ resistor. For the ARC to be used at any point along the bus the termination resistors need to be selectable. Activation the termination is accomplished by connecting the two pin pairs with jumpers (JP1 in Figure B.1 of Appendix B).

3.3.1.2 | Components Changed

The design files provided by Arduino [9] use a custom Eagle [10] library containing most of the components on the board. To have proper control over the component selection these were replaced by components available in the standard libraries and available through suppliers. As an example the Reset button used on the ARC has a different pin configuration from the one used on the Arduino UNO, forcing a redraw of that area.

All changes were made in such a way that the original physical component is still used.

3.3.2 | Schematics

With the pin map done the schematics are fairly straight forward. Adding in the ATmega32/64M1, Microchip MCP2561 [25], and surrounding support components, it is a mere matter of connecting the correct corresponding MCU pins to their board pins. The result of this work can be found in Appendix B, Figure B.1.

3.3.3 | Layout

Achieving a good layout with in the restrictions of a two layer PCB is difficult. The random nature of the ATmega32/64M1 pinout means that signal paths crossing each other is more common than not. A lot of time was therefore devoted to making the design and tuning it to produce best possible ground and 5 V, and to draw short paths with as few vias between the two layers as can be achieved.

See Appendix B, Figures B.2 and B.3, for pictures of the two layers of the ARC PCB. Following the first manufactured version of the ARC some faults were detected, they have been corrected in the design shown in Figures B.4 and B.5.

3.3.4 | Verification

In Section 1.2 there are a number of design tasks listed. These tasks have been mostly achieved, with the compromise that the pinout of the ARC board includes some minor deviations from that of the Arduino UNO. These differences stem from differences in the MCU pinout or peripheral setup, or in the case of the programming pinheader to make place for the new CAN interface.

Also looking at the software tasks have these been completed. All old code from Arduino has been ported, and code for the new features brought up to a basic functional level. A windows driver (though unsigned) exists and the ARC can be made selectable in the Arduino IDE.

3.4 | Motor Shield Design

As opposed to the ARC, the Motor Shield is not a modification of a prior design. All components, schematics and layout are new to this shield.

3.4.1 | Component Selection

The selection process of each component will be described following the signal path from the pinheaders to the power transistors. Finally the support and monitoring functions on the board are discussed. There are a number of decoupling capacitors and similar on the board, these are of little interest to this discussion.

3.4.1.1 | Signal Path

After entering the board the signal passes through a three main components, optocouplers, gate drivers, and the Metal–Oxide–Semiconductor Field-Effect Transistor (MOSFET) transistors.

Optocouplers are used to galvanically separate and protect the MCU from the power electronics and its higher voltages. A requirement for the optocouplers are that they should be fast, short turn on and turn off times, in order to properly producing a copy of the signal at the switching frequency used. The optocoupler output must support the working voltage of $12\,\mathrm{V}$ on the Motor Shield. To reduce the number of support components needed an active output (not open collector) is sought for. For these reasons the Toshiba TLP2345 [26] is used.

To be able to supply the gate of the MOSFET transistor with enough charge to turn on, and stay on, a gate driver is needed. For the Motor Shield the IR2110 [27] from Infineon (formerly International Rectifier) is selected. The ATmega32/64 has built in dead time control, it is therefore not a necessary feature of the gate driver. The IR2110 gate driver can run at up to $500\,\mathrm{V}$ on the power side. IR2110 is commonly used in inexpensive inverters with much information and design assistance available on the internet. After comparing to other options it was found to be the best option for the job.

Lastly the MOSFET used is an Infineon IRFS3207Z [28] (formerly International Rectifier). A D2PAK (TO-263) surface mount package is used, making it easier to design for and mount on a PCB. Since the transistor is mounted to the PCB with no direct physical contact to a heatsink other than through said PCB, the thermal performance is reduced. However, by selecting a transistor with a suitably low $R_{DS(on)}$ the power loses through heating are lowered. According to the manufacturer $R_{DS(on)} = 4.1\,\mathrm{m\Omega}$ maximum for the IRFS3207Z. The low resistance paired with maximum ratings of $75\,\mathrm{V}$ and $120\,\mathrm{A}$, makes it a good fit.

3.4.1.1.1 | Support Components

Dissipating the heat produced in the MOSFET transistors is helped by the cooling fins mounted over it. Fischer Elektronik FK244 13 D2PAK [29] is a good option for cost, size, and thermal resistance given the form factor. Because the heatsinks are soldered to the board they are live components and have to be kept apart to avoid a short circuit.

A diode across VCC and VB, and a ceramic and electrolytic capacitor across VB and VS form the bootstrap setup necessary for the IR2110. This circuitry holds the charge needed to keep the high side transistor open for the duration needed.

3.4.1.2 | Sensing

Temperature is measured at the three high side transistors due to space constraints. Close to the transistor there is a $22\,\mathrm{k\Omega}$ NTC thermistor (Vishay NTCS0603E3223JMT [30]), as the temperature changes so does the resistance of the thermistor. Thus, by dividing $5\,\mathrm{V}$ across a normal $2.2\,\mathrm{k\Omega}$ resistor and the thermistor the measured voltage relates to temperature. Using the data points given by the component datasheet the voltages for different temperatures can be calculated and a second degree polynomial estimated as per Equation 3.1. An RC-filter is applied to avoid picking up to much of the switching frequencies. Since the heating of the components is a rather slow process compared to the frequencies in the signals a low cut-off frequency is chosen at roughly $700\,\mathrm{Hz}$.

$$temp = -1.75 \cdot 10^{-4} \cdot v_{in}^2 - 9.63 \cdot 10^{-3} \cdot v_{in} + 4.87 \tag{3.1}$$

Keeping track of the motor position is important to correctly run a motor. For this purpose a voltage sensing system is implemented based on back-EMF. The circuitry is pictured in the upper right corner of Figure C.1 of Appendix C. A virtual ground is created by connecting the three phases through resistors. This virtual ground is then compared to each phase individually detecting the zero point crossings. The comparator output identifies one of six position ranges of the rotor [31]. An RC filter removes the switching frequency from the measured voltages, the capacitor is chosen so to induce as low a timing delay as possible. The Motor Shield is separated from the ARC by an optocoupler.

No further sensing is available for measuring the actual voltages or any of the currents on the board. However, these can be added externally by the user of the Motor Shield. This is due to lack of space on the board and cost of components to pass through analogue signals without breaking the galvanic separation.

3.4.1.3 | Power Supply

A standard $5.08\,\mathrm{mm}$ pitch screw terminal block is where the motor battery connects to the board. The voltage is applied directly across the three electrolytic capacitors by the MOSFET transistors with no protection if the voltage is applied in reverse. The $12\,\mathrm{V}$ regulator is protected by a diode for this purpose.

Reading the MOSFET gate driver datasheet, a $10\,\mathrm{V}$ to $20\,\mathrm{V}$ regulator is needed to power them. Texas Instruments TL2575HV-12IKTTR [32] outputs $12\,\mathrm{V}$ and $1\,\mathrm{A}$ (maximum), and can be supplied with up to $60\,\mathrm{V}$. Four additional components are required for the TL2575HV, the datasheet describes the process for selecting the correct values. Assuming a 48V supply the resulting values are: $330\,\mu\mathrm{F}$ input capacitor, $470\,\mu\mathrm{F}$ output capacitor, $330\,\mu\mathrm{H}$ output inductor, and a Schottky diode.

3.4.1.4 | Special Features

Two modifications of the Motor Shield are possible. Special design features enables modifying the board to run higher power transistors, or work as an MPPT to charge a battery from a solar panel. Though the features are made available, no further implementation details or system design has been prepared.

3.4.1.4.1 | High Voltage Cut Off

Should the $60\,\mathrm{V}$ limit of the system not be high enough, or the power output to low, the board can be modified to drive external transistors. After the three IR2110 [27] MOSFET gate drivers (along the signal path) there is a row of nine unused drill holes. By cutting the board in half just beyond these more powerful transistors can be connected working at voltages of up to $500\,\mathrm{V}$. However, higher voltages than $60\,\mathrm{V}$ should not be applied to the MOSFET gate driver circuitry.

3.4.1.4.2 | MPPT

Maximum Power Point Tracking is commonly used to maximise power extraction from solar panels. A section of traces near the third pair of transistor can be cut, thus isolating the high side transistor of that pair. Two drill holes are available to solder in a $5.08\,\mathrm{mm}$ pitch screw terminal next to the heatsink for connecting a solar panel. The floating transistor pair can thus be used to regulate the impedance seen by the panel, with the remaining two pairs used in a h-bridge configuration to do the charging. Some external components are necessary to implement an MPPT system.

3.4.2 | Schematics

The schematics are available in Figure C.1 of Appendix C. Starting with the signal path (left half of figure), the schematics is divided in to sections with optocouplers, gate drivers, and MOSFETs. All decoupling capacitors and similar support components are drawn in at their respective position. On the right hand side are (from top to bottom) the boost capacitors for the transistors, the power supply, temperature sensing and pinheaders, and lastly the back-EMF sensing circuitry.

3.4.3 | Layout

Appendix C, Figure C.2 shows the top layer design and component placement, Figure C.3 shows the bottom layer of the board as manufactured. Further improvements have been made and the latest design is pictured in Figures C.4 and C.5.

All $12\,\mathrm{V}$ electronics is fitted to the left of the design between the two rows of pinheaders. Signals coming from the PSC are kept together and have their own separated area with $5\,\mathrm{V}$ and ground by the optocouplers.

Due to higher voltages and greater currents the clearances between wires, pads, and vias are elevated to $16\,\mathrm{mil}$ for different signals and $12\,\mathrm{mil}$ for same signal.

The four mounting holes of the Uniti ARC are included with an additional two to the right of the MOSFET transitors.

3.4.4 | Verification

In Section 1.3 there are a number of hardware requirements to design by. These have been adhered to, the larger Motor Shield fits onto the ARC, providing galvanic separation between the two through optocouplers. The shield has temperature sensors by the MOSFETs, and back-EMF voltage sensing to determine the position of the rotor when the motor is running.

However, the softweare targets have not been completed to the same level. Sufficient code has been created to get a motor running, though not as a complete and stand alone library or with as simple an interface as initially wanted.

3.5 | Manufacture

Running conditions for the two boards are different, especially for the Motor Shield care had to be taken to setting the manufacturing parameters correctly. The parameters used are shown in Table 3.1.

Parameter	Uniti ARC	Motor Shield
Layers	2	2
Component load	Single sided	Single sided
Board thickness	1.6 mm FR4	1.6 mm FR4
Copper thickness	35 µm	105 µm
Clearance	10 mil	16 mil
Vias	Covered	Exposed
Soldering pad plating	Tin	Tin

Table 3.1: Manufacturing parameters for the two boards.

The thicker layer of copper on the Motor Shield is for it to be able to handle the higher currents it is purposed for. Leaving the vias without solder mask comes in handy when evaluating new designs, vias should be covered in a final design.

All prototype boards were completely hand soldered using a hot air and a standard soldering station.

3.6 | Software

Software development is divided into a series of steps. The software is fully based on the previous work done by Arduino, similar to the ARC hardware design. All code is available through GitLab [14].

To differentiate between the prototype based on the ATmega32M1, and the proposed final version based on ATmega64M1, two separate boards are introduced in the software under the names of ”Uniti ARC beta” and ”Uniti ARC” respectively.

3.6.1 | USB Interface

Programming the ARC using nothing more than a USB type B cable is an important feature. All USB communications are handled through a second MCU, the ATmega16U2 [33], embedded in the design specifically for this purpose.

The software for the ATmega16U2 is made up of two parts, the bootloader (uniti_arc-usbdfu) and application (uniti_arc-usbserial). Both parts have to be compiled against LUFA (Lightweight USB Framework for AVRs) [34] version 100807. Instructions for compiling the code and programming the MCU are available in the GitLab repository.

The amount of code to alter is small, both parts of the software contain a pair of files called Descriptors.c/.h. In these files information regarding the Uniti ARC is added. Files and folders are renamed to reflect the new product they are intended for. The makefiles [23] are altered to properly compile for the Uniti ARC, also erroneous references to the MCUs AT90USB82 and ATmega8u2 are removed.

3.6.1.1 | USB Vendor and Product ID

A unique pair consisting of a VID and PID are needed for each USB enabled product. The VID signifies the company behind the product, and the PID the product itself. Both IDs are made up of four hexadecimal digits for a total number of 65536 possibilities. VIDs are sold by the USB Implementers Forum (USB-IF) [35] at a price of $5000. The budget for this project was insufficient to carry such a cost, to be able to properly identify the Uniti ARC when connected to a computer a VID and PID pair was acquired through the pid.codes project [36].

3.6.2 | Bootloader

Similarily to the USB interface, only minor edits are needed to make use of the Optiboot bootloader on the ATmega32/64M1. UART functions differently on the ATmega32/64M1 as it shares peripheral with LIN, therefore the functions for sending and receiving data have to be completely rewritten. An effort to port the bootloader has previously been done by Stuart Cording [16], his edits lay the foundation for this version of the bootloader. Further edits include removing some conditionals where the specific circumstances of this project are known, such as soft_UART which is only used when there is no UART peripheral built in to the MCU.

3.6.3 | IDE Integration

With the USB interface and bootloader ported the Uniti ARC has almost reached a usable state. But to program it using the Arduino IDE a few additional files need to be created.

First is the pins_arduino.h file, this file is used to define all board specific variables. Through the arrays and functions defined in this file the pins of the board is converted to pins of the MCU. The file is located in the variants folder.

Secondly three further files are needed. Boards.txt contains information about available MCU memory, how to program it and what core and variant to compile against. Platform.txt specifies in detail the compilation and programming procedure to the IDE. ATmega32/64M1 is not included in the version of avrdude [22] included with the IDE, so to be able to program the ARC the avrdude.conf configuration file is included.

Finally, the third step is to create an installable package and to distribute it. The code files are compressed into a single tar.gz file and placed in a separate folder of the project and pushed to GitLab [14], a JavaScript Object Notation (JSON) file is also available in the same folder with information about each of the compressed archives. By including the url (`https://gitlab.com/uniti-arc/Arc/raw/master/packages/package_uniti_index.json`) to the mentioned JSON file in the ”Additional Boards Manager URLs” text field of the Properties window, the Uniti ARC boards are installable and selectable through the Boards menu.

3.6.4 | Porting Arduino

The bulk of the edits to the core section of the Arduino code base concern either Serial communications, signal handling, or the new peripherals. Much of the new code needed is imported from the work done by Al Thomason [17], his work is in turn based on that of Stuart Cording [16].

3.6.4.1 | Communications

As previously mentioned in Subsection 3.6.2 ‘Bootloader’, the UART peripheral on the ATmega32/63M1 is combined with the LIN peripheral. All registers for setting up the LIN/UART system are completely different and therefore all functions relating to it need to be amended. Also since there is only one serial port available, the files for the three further serial ports on the Arduino Mega [37] are removed.

USB communications are possible on some boards (most notably the Arduino Leonardo [38] and Micro [39]) using an ATmega32U4 [40]. This is not possible with the ATmega32/64M1 and the seven files that contain code for the USB peripheral are removed along with any references to them in the remaining files.

3.6.4.2 | Signal Handling

Configuring pins to be inputs or outputs remains identical with the addition of code to handle the Analogue Differential Amplifiers, Analogue Comparators, and DAC. Configuring said peripherals is done using the pinMode() function, the chosen unit is set up and the physical pins associated are configured as required. When configuring the comparators or amplifiers their input pins will be reconfigured regardless of their previous state as this is impossible to check. Similarly, should one of the input pins in use be reconfigured to be used in standard I/O mode the comparator or amplifier will be shut down.

Once configured the outputs from the comparators are read by the digitalRead() function, amplifier output is read using analogRead(), and writing to the DAC is done with the analogWrite() function.

The only changes needed to get all external interrupts working is to add sections of code that handle the fourth available interrupt. The additions are simplistic as they are identical in form to the first three interrupts.

3.6.4.3 | Libraries

A few peripherals of the MCU are handled through separate libraries not part of the immediate core code. These are Electrically Erasable Programmable Read-Only Memory (EEPROM), SPI, I2C, and SoftwareSerial (a software implementation of UART). They have not been worked on, though (with the exclusion of I2C) should be a simple task of reviewing the code as the peripherals are unlikely to differ much from the standard implementation of other ATmega MCUs. I2C needs to be replaced by a software implementation as the hardware for it is not included in the ATmega32/64M1.

3.6.5 | New Libraries

Two new libraries are required to use CAN and the PSC on the Uniti ARC.

The CAN library is written by David Bengtsson based on his previous work with the same MCU at Lund Formula Student. It is written in C++ and C with an API similar in style to that for UART, SPI, and I2C.

The PSC can be run in two modes, one where the six PWM outputs are run together in pairs, and one where they are configured individually. The first case is used for three phase motors, where as the second mode offers PWM to those pins that do not have a timer output on the ARC but do so on the Arduino UNO through the usual analogWrite() function.

When running the PSC in the first mode described above, three sinusoidal signals phase shifted in software by $120\,°$ from each other are generated. A look-up table containing the values of $sin(x), x = [0..\pi/2]$ is declared, and by mirroring the quarter period to the sought after intervals all three PWM duty-cycles can be calculated.

3.6.5.1 | Motor Shield

No Library specifically for the Motor Shield has been written or designed. To run a motor the PSC library can be used directly. The PSC library provides functions to set and update the PWM signals, effectively setting the angle of the rotor. A Motor Shield library should run autonomously, updating the rotor position automatically to achieve a requested rotational speed.

3.6.6 | Windows Driver

Connecting the Uniti ARC to a Linux or Apple OSX computer works at first attempt without additional software; Windows, on the other hand, requires a driver to be able to communicate with it.

Writing the driver is done by creating a .inf file containing information of the USB VID and PID, driver type, and more. Using the inf2cat command window program, a .cat file is generated from the .inf file. The .cat file is what is installed on computers to use the Uniti ARC. For proper distribution the .cat file needs to be signed using an SSL-key supplied by one of the major cryptography companies. As with the USB VID, the cost of doing so at this stage is out of scope for the project. The driver can, however, be installed if the computer is booted in an insecure mode.

3.7 | Testing

By continuously programming the ARC with new versions of the software and running simple application the board design has proven functional. A minor design error was detected early on where the SPI pins were not correctly connected to the programming header. The fault was easily corrected by cutting the erroneous trace and connecting a thin wire to the correct points.

Testing of the Motor Shield was done by powering a motor designed for an electric bicycle. The motor was successfully run and the hardware design can therefore be considered functional. An issue was found with the diode protecting the 12 V regulator against application of reverse polarity; it is severely under-dimensioned to handle the initial current draw as the voltage is applied to the board. A larger diode should be used in its stead or, preferably, a redesign of the reverse polarity protective circuitry be done to include also the transistors and gate drivers.

4. | Discussion & Conclusion

When designing any embedded system there are a number of design assumptions (known or unknown) and choices (willing or unwilling) made. Even more assumptions are forced to be made to achieve a satisfactory ”one size fits all” result for a general purpose product like Uniti ARC.

4.1 | Feasibility

Despite some differences in the basic feature set (no I2C, combined LIN/UART, random pinout), the fundamental similarities (AVR core, NRWW memory, etc.) are the important parts to consider. This proves the software can be made to run on the ATmega32/64M1 and properly interface with a computer. However, whether fitting the components on to the area given on a two layer board was possible could not be tested prior to doing the design work.

Assessing the amount of software work that would be required was also difficult. No work had previously been done on the underlying Arduino structures, nor any work with bootloaders or USB interfaces.

4.2 | Hardware Design

The two boards are discussed separately below.

4.2.1 | ARC

Deciding to base the Uniti ARC fully on the Arduino UNO ECAD files lowered the bar for getting started significantly. The circuitry for USB, power, etc. was already there along with perhaps most importantly the board size and pinheader locations.

The datasheet for the ATmega32/64M1 [5] specifies an inductance and decoupling capacitor be connected to the analogue power supply (AVCC, AGND). This recommendation can also be found in the ATmega328P [18] datasheet. The circuitry is not included in the Geniuino UNO design but added to the ARC as a compensation for the poorer ground plane. To test whether the inclusion is necessary the digital ports can be set to switch at high frequencies while measuring the stability of the analogue voltage supply.

Mapping the MCU pins to the board required proper planning. Certain functions have to be on specific pins for the ARC to be compatible with the UNO, but to achieve that there are many crossings and conflicts that have to be resolved in the layout. With so many intertwined traces the risk for ground to be cut of at some point and to not reach everywhere grows. A lot of time was spent redesigning sections of the layout to allow for the ground plane to fit in. Following the first prototype further improvements have been made by remapping some of the pins. The ground planes are now much better and some signal paths shortened.

Including a CAN transceiver on the board makes the functionality easily accessible, and was therefore an obvious decision. The jumper pins were a necessity if the ARC should be able to connect at any point of a CAN bus. Initial thoughts was to use a surface mount switch but the availability and size of such a solution made it difficult to implement.

ATmega32/64M1 has four more pins than an ATmega328P, being able to use the additional pins was considered important. An early thought was to replace the original power connector with a CAN connector, but routing space and the need for special cables to use the feature made for the decision to place the high and low CAN signals in a standard pinheader too. So, where should the signals be routed too? One possibility would be to extend the board length and add the new pins next to the original ones, this would add a lot of board area with only little added value. The chosen solution is thus to place the new pinheader orthogonally to the standard pins on the short side of the board. The first prototype as manufactured had a standing pinheader, a problem with this option was soon discovered as the pins can not be used if a shield is mounted on top. A $90\,^{\circ}$ angled pinheader has replaced it in the updated design. The programming header for the ATmega32/64M1 had to be moved to make space for the new pinheader, even though this may cause some compatibility issues with third-party shields that rely on it.

Over all getting a sufficiently high quality layout has proven the most challenging part of working with the ARC. Eagle as a design tool is easy to use, however, the limited functionality can make simple edits slow and the overall work flow tedious.

4.2.2 | Motor Shield

With all logic and control electronics embedded in the ATmega32/64M1 the purpose of the Motor Shield is to simply amplify the signals and provide sufficient power to the connected motor.

The available area for component placement is very limited and further restricted by the Arduino footprint. Perhaps by rethinking some of the design options a better layout would be possible. For instance a different 12V supply design could save space, what components this design would consist of has not been looked into. Replacing the large electrolytic capacitors used with other lower options stacking further shields on top of the Motor Shield would be possible.

To remove the reliance on a very specific optocoupler a signal transformer such as those used in Ethernet applications may improve the part availability while reducing cost.

IR2110 proved easy to work with. It does not add additional features to the circuitry other than simply provide sufficient power to switch the gates of the transistors according to the incoming PWM signal. It is common for other similar gate drivers to impose a dead-time between turning on and off the high and low side transistors respectively. But as this is done by the software adding on further margins would be unnecessary.

Surface mount D2PAK transistors are a good option form a mechanical perspective; they are secured to the PCB and do not require fastening to a cooling block to ensure they do not break off. One major drawback is the additional PCB area needed to fit the transistors, another is the reduced cooling as all heat is transferred through the PCB to the heatsinks.

The PCB is a two layer design with $105\ \mu m$ of copper, adding two internal layers for routing and placing components also on the back of the PCB would improve the layout both with regards to signal paths and to power and ground planes.

Only a very rudimentary type of voltage sensing is available on the Motor Shield. It would be preferable to have analogue voltage and current sensing on each of the three outgoing phases. Properly implementing this while retaining galvanic separation and low design complexity could prove to be quite difficult.

4.3 | Software

The Arduino software is not without its limitations, and writing in more functionality poses some questions to be answered. The software development can be divided into five parts, USB interface, bootloader, Arduino core, new code, IDE integration; to avoid mixing things up they are discussed individually.

4.3.1 | ATmega16U2 and USB

There are a few errors to the ATmega16U2 [33] makefiles that needed to be corrected before anything could be compiled. The MCUs declared to the compiler and avrdude is wrong, as is the memory position to write the bootloader to. Correcting the MCU declaration was promptly done, however the memory position required more investigation to correct.

Instructions for compiling the software is included in the readme files. It is mentioned that the version of LUFA should be 100807, this was discovered to be a requirement for the code to compile at all.

The second thing is the $5000 charged by the USB-IF [35] to provide a Vendor ID. An exorbitant amount of money for a small project. USB-IF are the ruling body behind the USB standard, they license the use of USB to all who make products that use it. It is not possible to purchase a smaller subset of PIDs, and it is not permitted according to the current license agreement for VID holders to further distribute sets of PIDs. An attempt by Arachnid Labs [41] to provide PIDs to open source and hobbyist projects was closed down after receiving a cease and desist letter. Trough the project pid.codes [36], however, PIDs can be freely acquired. This is because the VID used was issued before PID redistribution was prohibited by the USB-IF license agreement.

As all the development work was initially performed in a Linux environment, the issue of needing a driver for use with Windows was missed for some time. The process of creating and signing a driver is very poorly documented by Microsoft [42]. After finding and reading a blog post by David Grey [43] and other resources, the necessary steps of creating a .inf file and generating a .cat file could be taken to test out an unsigned driver. Installing an unsigned driver requires one to reboot the computer in an unsafe mode where no security keys are checked. To sign the driver a certificate has to be purchased from Globalsign, Verisign, GoDaddy or others, the cost is approximately 200 €/year. The command line program Inf2Cat [44] creates a .cat file from the .inf file, to install it Visual Studio 2015, the Windows SDK, and Windows Driver Kit [45] all have to be downloaded and installed, totalling multiple gigabytes of unnecessary and unused binaries.

All in all the major hurdles of a project such as this lies in the USB interface. High costs and poor documentation from USB-IF and Microsoft creates a lock-out effect for small or inexperienced projects. The actual work necessary to get the ATmega16U2 properly programmed, and a (unsigned) Windows driver is small. Time is easily lost figuring out details.

4.3.2 | Bootloader

Code changes necessary to get the bootloader are picked from Stuart Cording's [16] previous work. Due to the combined LIN/UART peripheral the UART code has to be replaced. The code was simplified to an extent by removing unused options for soft_UART and other special cases. Further cleaning up of the code could be possible, however as it is functional there is little need to do so at this point.

4.3.3 | Arduino Core

Relying on Stuart Cording [16] and Al Thomason [17] the work effort to port the Arduino core libraries was reduced to a rather simple task. Almost complete functionality could be achieved very fast by simply copying their code. In most places the new code is placed within pre-compiler conditional statements, retaining the original Arduino code for other MCUs. As the code is meant specifically for the ATmega32/64M1 it can be vastly simplified by removing those conditionals that do not apply to that MCU. For instance all USB code has been removed as there is no USB peripheral in the ATmega32/64M1.

4.3.4 | New Code

Adding functionality can be done in two ways, as part of the core or as new libraries. The DAC, differential amplifiers, and analogue comparators are added to the core, whereas CAN and PSC are written as libraries.

Including the DAC, differential amplifiers, and analogue comparators to the core is because of the low amount of code needed. Only a setup and read function is needed, pinMode() and analogWrite(), analogRead(), or digitalRead() respectively can serve that purpose.

Compiling a sketch against the Uniti ARC core with the additions mentioned above adds considerable size to the produced hex file. Most added size comes from the additions to pinMode(). The additional size caused could be kept down by reducing the configurability of the peripherals or moving their code to separate library files. Considering that the memory of the ATmega64M1 is double the size of that in the ATmega328P the added code size might be acceptable. Further testing and evaluation of different software implementations is needed but although it does not fall within the scope of this thesis.

CAN and PSC are larger independent peripherals that require more complex code to setup and use. Implemeting the necessary code in libraries is therefore an obvious solution. The PSC implementation is specifically for controlling the MCU peripheral and does not implement functionalities of the Motor Shield.

4.3.5 | Distribution and IDE Integration

No complete guide for developing a custom core and distributing it is available in the Arduino documentation. Most of the process is documented in a slightly fragmented fashion, and the rest can be extrapolated by looking at existing packages. Once figured out, the process is easy to follow and the distribution not very difficult to do. Perhaps the way new packages are added to the IDE is a little complicated, but that is only a minor issue.

To make it easier to create new release packages a Linux shell script was written called packscript.sh, it is available on GitLab with the code proper.

4.4 | Manufacture and Testing

A disproportionate number of ARC boards were malfunctioning post manufacturing. The large number of components and the package format of some of them (ATmega16u2 being QFN-32 etc.) made the soldering of components difficult. Using an Electronics Manufacturing Services (EMS) to complete the manufacturing would have saved time and reduced the number of faulty boards but at a much higher cost.

With the ARC being based on a previously widespread and well used design the testing effort once operational was minor. By ensuring that the ported Arduino code was functional the overall design could be approved.

Manufacturing the Motor Shield was much easier that the ARC due to the larger and less sensitive components. Running tests on the Motor Shield depended heavily on there being at the very least some rudimentary code available to run it. Because of this the testing was not done until very late in the process. Also to guarantee that it is fully functional further testing would be necessary.

4.5 | Future Work

The design of the ARC and Motor Shield can be further improved by breaking some of the imposed constraints. While some possible changes are minor, others would affect the product at the core.

4.5.1 | MCU Re-Selection & ARC Layout

For more computational power an ARM Cortex series MCU could be used, such as the STM32F303RE [46] and many more from ST. As the name indicates the STM32 is a 32-bit MCU, clocked at $72\,\mathrm{MHz}$ such a device would heavily out-perform the ATmega32/64M1. As an added advantage said MCU has an integrated USB controller, consequently lowering the component count and therefore also cost. Moving to a completely new hardware architecture would require a complete rework of all code as well as a complete redesign of the ARC schematic and layout.

By not adhering to the Arduino form factor and pinout a small ”embeddable” version of the ARC could be made, moving the USB circuitry off-board. The board could be made in four layers for improved routing and EMI performance. Higher grade contacts for CAN, power, and IO-pins would further improve the ruggedness of the product. Such a redesign would require a new Motor Shield.

4.5.2 | Motor Shield

The Motor Shield design is limited by the Arduino pinheader footprint. Fitting all components to one side, and routing on two, proved difficult. Introducing additional routing layers and possibly moving some components to the bottom layer should not only improve the trace lengths and ease the design effort, but also improve the stability of the 12 V, battery voltage, and grounding layers. As a consequence of the improved paths and traces, the board could be used for higher power applications.

Stacking multiple shields onto one ARC is impaired by the vertical space taken by the capacitors used by the 12 V supply. Moving components to the back of the board would free up space to lay down the capacitors, making it possible to place further boards atop the Motor Shield.

Because the heat has to travel through the board to the heatsink thermal performance is lacking when using D2PAK (TO-263) transistors. Better performance could be gained from standing TO-220 with a larger cooling block attached.

To accommodate a wider set of differing power applications a split board approach might prove beneficial. The Motor Shield manufactured has 105 µm copper thickness with rather large clearances, even for the low voltage, low current traces. Though thick copper and large clearances are needed for the power transistors, it is redundant for the gate driver stage. The driver stage would be on one board with a series of connectors for attaching either one of a selection of dedicated transistor boards, or individual transistors with custom cooling. With such a solution the usability of the ARC and Motor Shield is improved. However, the many configuration options might be confusing and large number of differing boards affect the total cost for customers.

4.5.3 | Code improvements

Though all new functions have been implemented, it is not certain that their current placements are optimal. Re-evaluation of whether the amplifiers and comparators should be part of the immediate core or moved to separate libraries should be done.

A cohesive Motor Shield library is still missing, the difficulty in writing one should not be very high.

All references to MCUs in the Arduino core should be removed from the code base, leaving only specific ATmega32/64 code. This is a simple, though perhaps time consuming, task.

Proper documentation of the code and the new functionalities is missing. The main README file is still unchanged and written by Arduino. Also this task is easy but time consuming.

Appendix A. | Pin Mapping

MCU		Board	
Port	pin	Function	pin
VCC	4	VCC	5 V
GND	5	GND	GND
AVCC	19	AVCC	5 V
AGND	20	AGND	GND
AREF	21	AREF	AREF
PB0	8	PSCOUT2A / MISO	12
PB1	9	PSCOUT2B / MOSI	11
PB2	16	ADC5 / INT1 / ACMPN0	A1 (19)
PB3	23	AMP0-	4
PB4	24	AMP0+	E1 (15)
PB5	26	ADC6 / INT2 / ACMPN1 / AMP2-	2
PB6	27	PSCOUT1B	5
PB7	28	PSCOUT0B / SCK	13
PC0	30	PSCOUT1A / INT3	3
PC1	3	PCSIN1 / OC1B / SS_A	9
PC2	6	TXCAN	E3 (17)
PC3	7	RXCAN	E2 (16)
PC4	17	ADC8 / AMP1- / ACMPN3 / (SCL)	A5 (23)
PC5	18	ADC9 / AMP1+ / ACMP3 / (SDA)	A4 (22)
PC6	22	ADC10 / ACMP1	A0 (18)
PC7	25	D2A / AMP2+	E0 (14)
PD0	29	PSCOUT0A	10
PD1	32	PSCIN0	8
PD2	1	OC1A / PSCIN2 / MISO_A	6
PD3	2	TXD / TXLIN / SS / OC0A / MOSI_A	1
PD4	12	RXD / RXLIN / SCK_A	0
PD5	13	ADC2 / ACMP2	A2 (20)
PD6	14	ADC3 / ACMPN2 / INT0	A3 (21)
PD7	15	ACMP0	7
PE0	31	RESET	Reset
PE1	10	XTAL1 / OC0B	
PE2	11	XTAL2 / ADC0	

Table A.1: Table showing the connections of the MCU to the board pinout.

Pin	Port	Feature	Communication	Interrupt	Analog	Comparator	Diff. Amplifier	PSC
13	PB7	led/sck	**SCK**		ADC4			PSCOUT0B
12	PB0	miso	**MISO**					PSCOUT2A
11	PB1	pwm/mosi	**MOSI**					PSCOUT2B
10	PD0	pwm/(ss)						**PSCOUT0A**
9	PC1	pwm			**OC1B**			PSCIN1
8	PD1							PSCIN0
7	PD7					ACMP0		
6	PD2	pwm			**OC1A**			PSCIN2
5	PB6	pwm			ADC7			**PSCOUT1B**
4	PB3						AMP0-	
3	PC0	pwm/int		**INT3**				**PSCOUT1A**
2	PB5	int		**INT2**	ADC6	ACMPN1	AMP2-	
1	PD3	tx	**TXD**/SS		OC0A			
0	PD4	rx	**RXD**		ADC1			
E3	PC2	extra	TXCAN					
E2	PC3	extra	RXCAN					
E1	PB4	extra					AMP0+	
E0	PC7	extra			D2A		AMP2+	
A5	PC4	SCL	**–**		**ADC8**	ACMPN3	AMP1-	
A4	PC5	SDA	**–**		**ADC9**	ACMP3	AMP1+	
A3	PD6			INT0	**ADC3**	ACMPN2		
A2	PD5				**ADC2**	ACMP2		
A1	PB2			INT1	**ADC5**	ACMPN0		
A0	PC6				**ADC10**	ACMP1		

Table A.2: Table showing the functions of each pin on the board. Items bolded match up to the pin function.

Appendix B. | Uniti ARC Design Files

Quantity	Value / Name	Package	Description
2	1 MΩ	0603	Resistor
1	300 Ω	0603	Resistor
2	60 Ω	0603	Resistor
1	10 kΩ	CAY16	Resistor
2	1 kΩ	CAY16	Resistor
1	22 Ω	CAY16	Resistor
8	100 nF	0603	Capacitor
2	1 µF	0603	Capacitor
2	22 pF	0603	Capacitor
1	4700 pF	0603	Capacitor
2	47 µF	Panasonic	Capacitor
1	10 µH	2012	Inductor
1	BLM21	0805	Ferite
2	CG0603MLC-05E	0603	Varistor
1	MF-MSMF050-2	L1812	Fuse
1	M7	SMB	Diode
2	CD1206-S01575	MINIMELF	Diode
3	Yellow LED	0805	LED
1	Green LED	0805	LED
2	3x2	Male	Pinheader
1	2x2	Male	Pinheader
1	10x1	Female	Pinheader
2	8x1	Female	Pinheader
2	6x1	Female	Pinheader
1	USB B Port	USB B	USB Connector
1	Power Jack		Power Connector
1	SKHMPSE010		Button
1	ATMEGA32/64M1	TQFP-32	Main MCU
1	ATMEGA16U2	QFN-32	USB Interface MCU
1	MCP2561-E/SN	SOIC-8N	CAN Tranceiver
1	LMV358IDGKR	MSOP-08	Operational Amplifier
1	FDN340P	SOT-23	PMOS Transistor
1	NCP1117ST50T3G	SOT-223	5 V Regulator
1	LP2985-33DBVR	SOT-23	3.3 V Regulator
1	16MHz	QS	Crystal
1	CSTCE16M0V53-R0	Resonator	

Table B.1: Bill of Materials for Uniti ARC

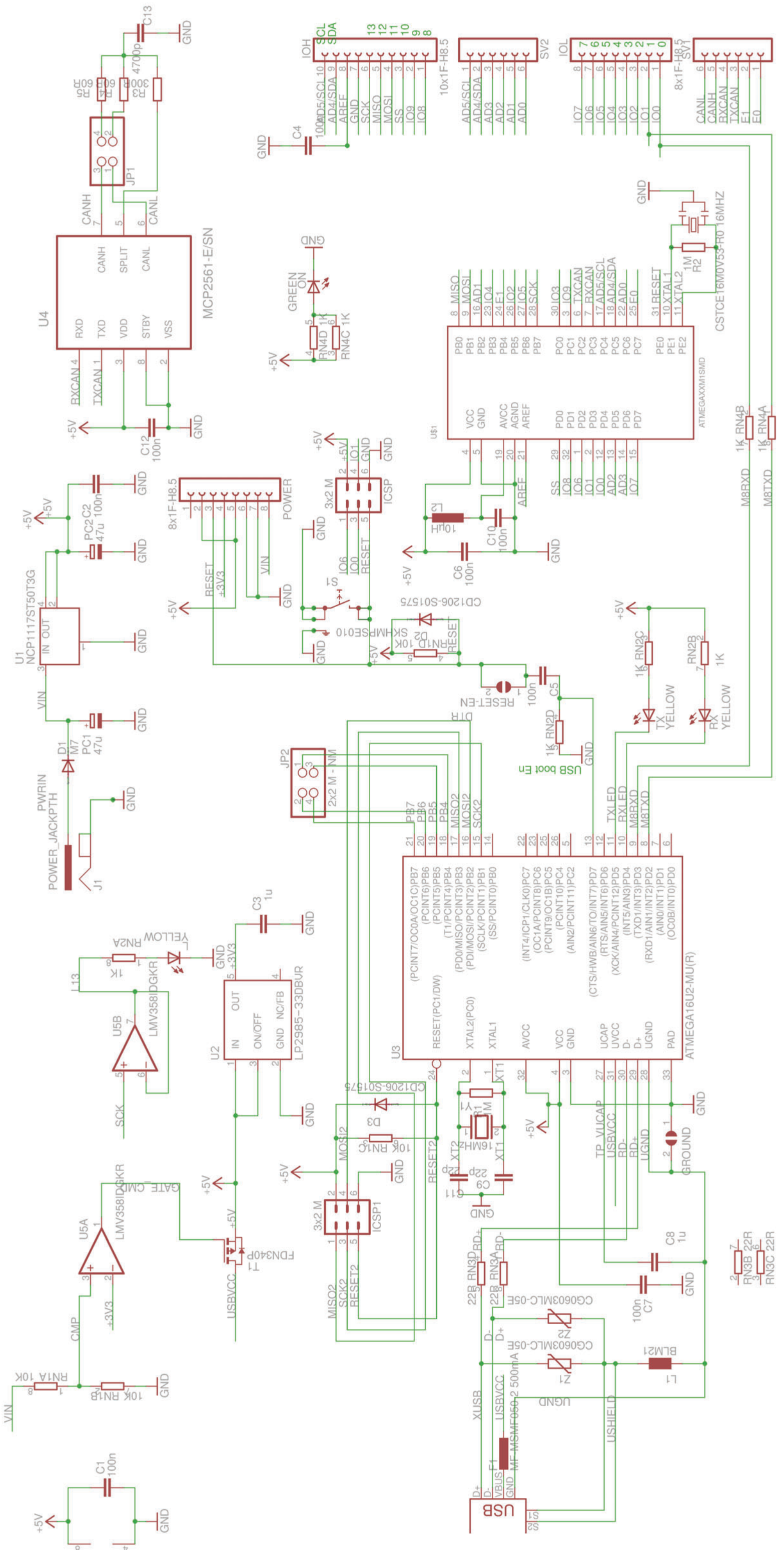


Figure B.1: Schematic of Uniti ARC, First Prototype

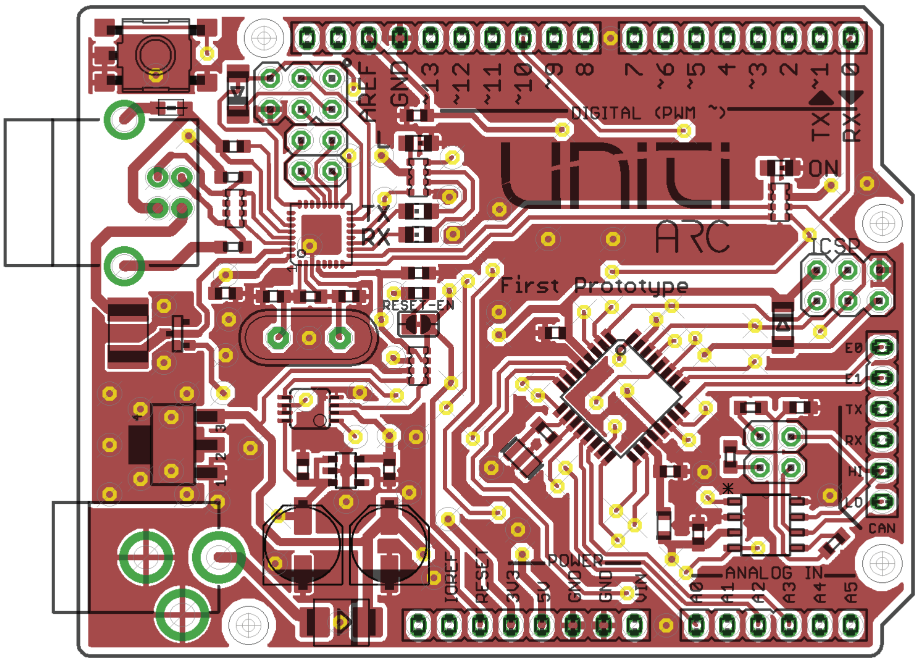


Figure B.2: Top layer layout of Uniti ARC, First Prototype

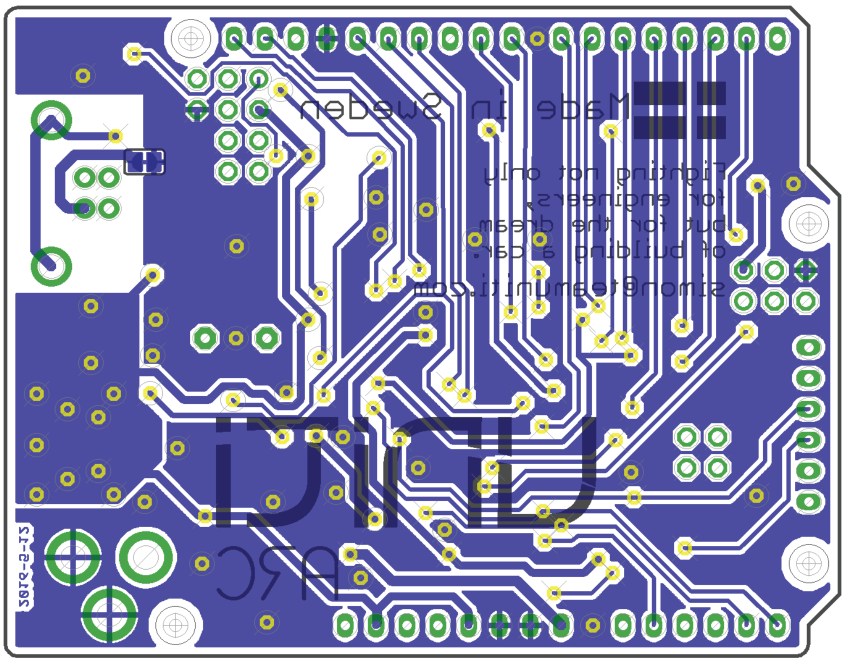


Figure B.3: Bottom layer layout of Uniti ARC, First Prototype

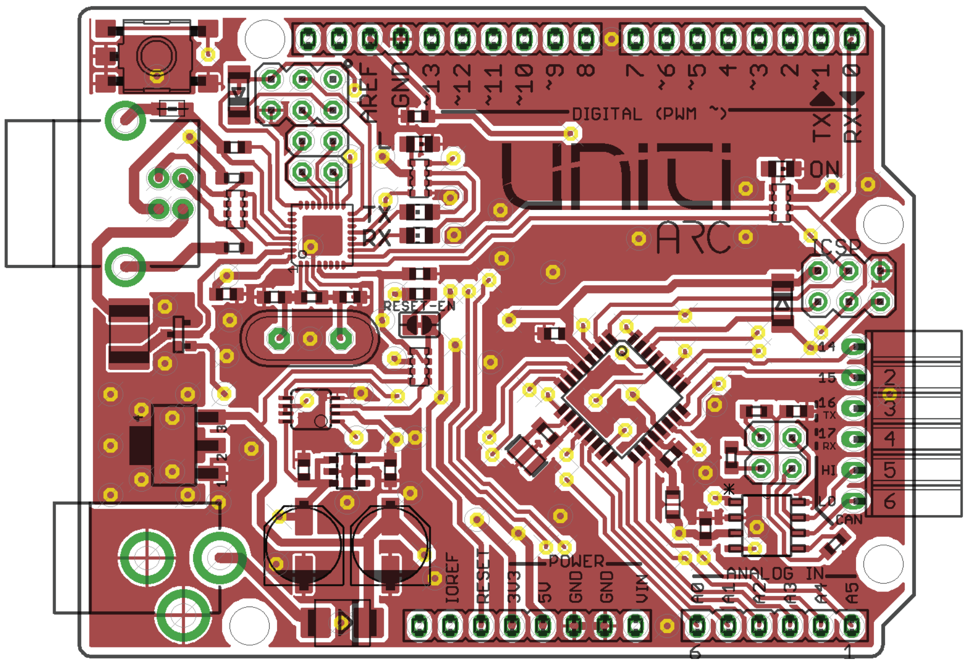


Figure B.4: Top layer layout of Uniti ARC, Second Prototype

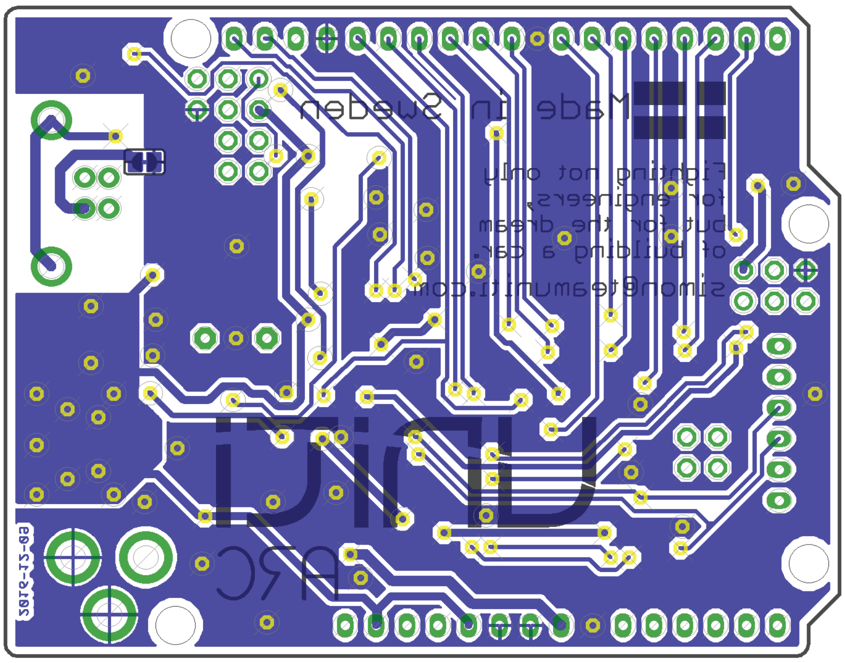


Figure B.5: Bottom layer layout of Uniti ARC, Second Prototype

Appendix C. | Motor Shield Design Files

Quantity	Value / Name	Package	Description
6	$10\,\Omega$	0603	Resistor
11	$560\,\Omega$	0603	Resistor
9	$1\,\mathrm{k\Omega}$	0603	Resistor
3	$2.2\,\mathrm{k\Omega}$	0603	Resistor
4	$20\,\mathrm{k\Omega}$	0603	Resistor
7	$100\,\mathrm{k\Omega}$	0603	Resistor
1	$47\,\mathrm{k\Omega}$	1206	Power dissipater
3	$22\,\mathrm{k\Omega}$	0805	NTC Varistor
4	$22\,\mathrm{nF}$	0603	Ceramic Capacitor
19	$100\,\mathrm{nF}$	0603	Ceramic Capacitor
3	$22\,\mathrm{\mu F}$	E2-5	Electrolytic Capacitor
1	$330\,\mathrm{\mu F}$	E5-13	Electrolytic Capacitor
3	$470\,\mathrm{\mu F}$	E7,5-16	Electrolytic Capacitor
1	$470\,\mathrm{\mu F}$	E3,5-8	Electrolytic Capacitor
1	$330\,\mathrm{\mu H}$		Bourns SRR1260A, Inductor
5	NXP PMEG6002EJ	SOD323F	Schottky Diode
1	LED	0805	Power Indicator
7	Toshiba TLP2345	SOP-5	Optocoupler
1	Broadcom ACPL-247-500E	SOIC-16	Optocoupler
1	Texas Instruments LM339D	SOIC-14N	Differential Comparator
1	Texas Instruments TL2575HV	TO263-5	$12\,\mathrm{V}$ Regulator
3	Infineon IR2110S	SOIC-16W	MOSFET Gate Driver
6	Infineon IRFS3207Z	D2PAK	MOSFET Transistor
6	Fischer Elektronik FK244-13	For D2PAK etc.	Heatsink
1	6 pin, single row		Female Header
2	8 pin, single row		Female Header
1	10 pin, single row		Female Header
1	Phoenix Contact SMKDS3/3-5,08	$5.08\,\mathrm{mm}$ pitch	Screw Terminal Block
1	Phoenix Contact SMKDS3/2-5,08	$5.08\,\mathrm{mm}$ pitch	Screw Terminal Block

Table C.1: Bill of Materials for Uniti ARC Motor Shield

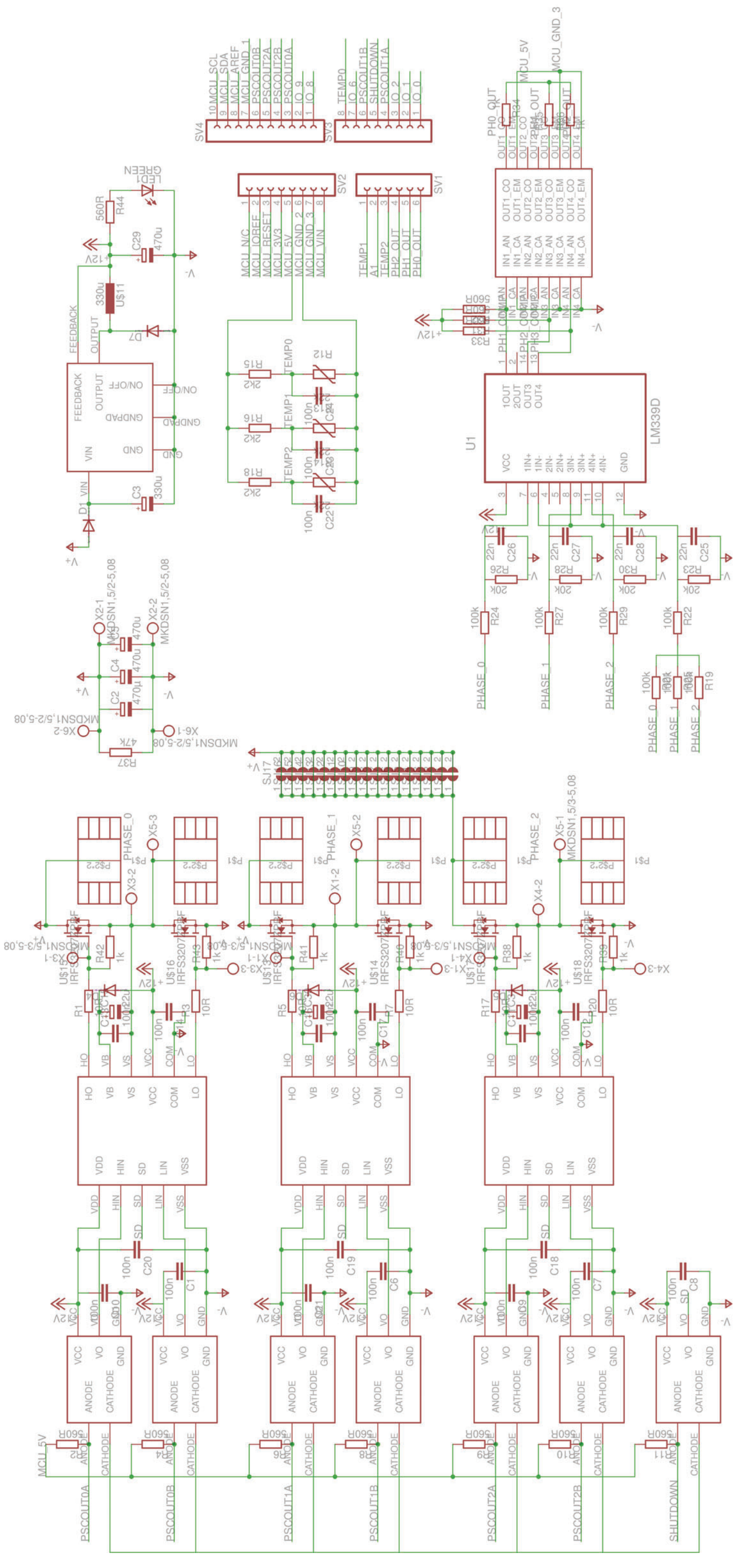


Figure C.1: Schematic of Uniti ARC Motor Shield

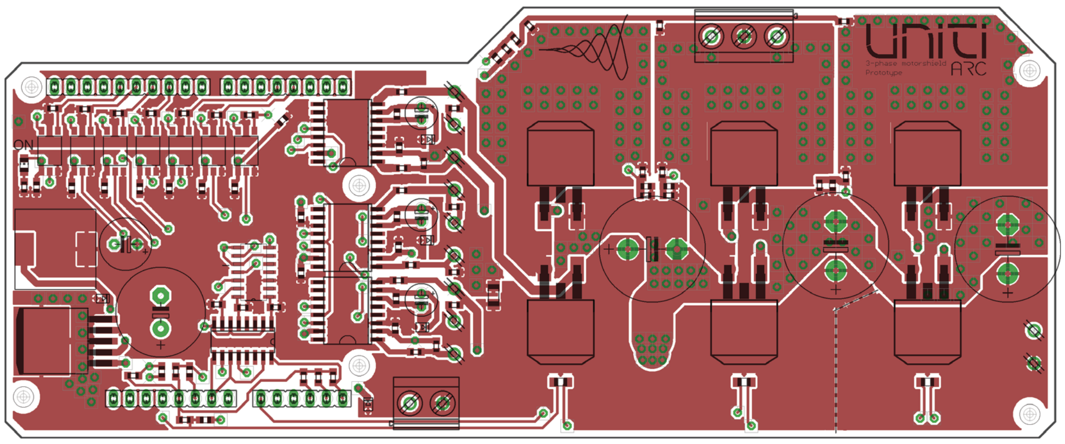


Figure C.2: Top layer layout of Uniti ARC Motor Shield

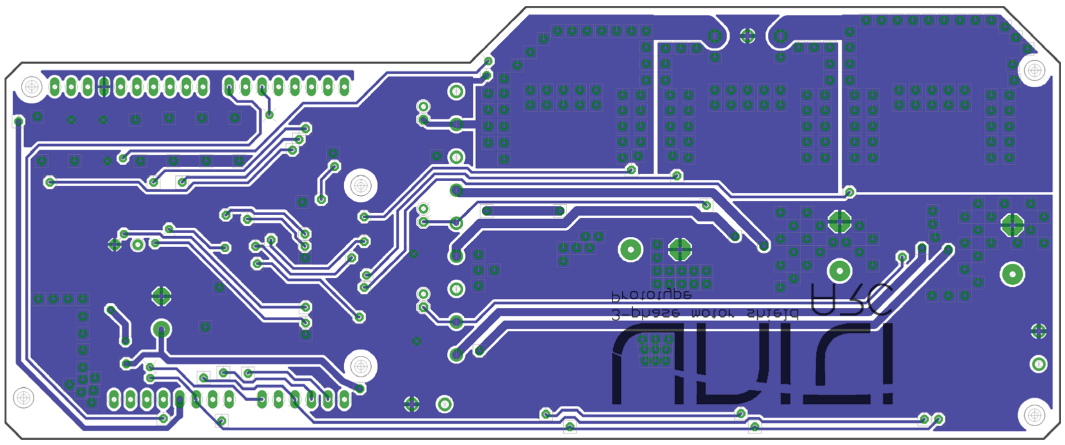


Figure C.3: Bottom layer layout of Uniti ARC Motor Shield

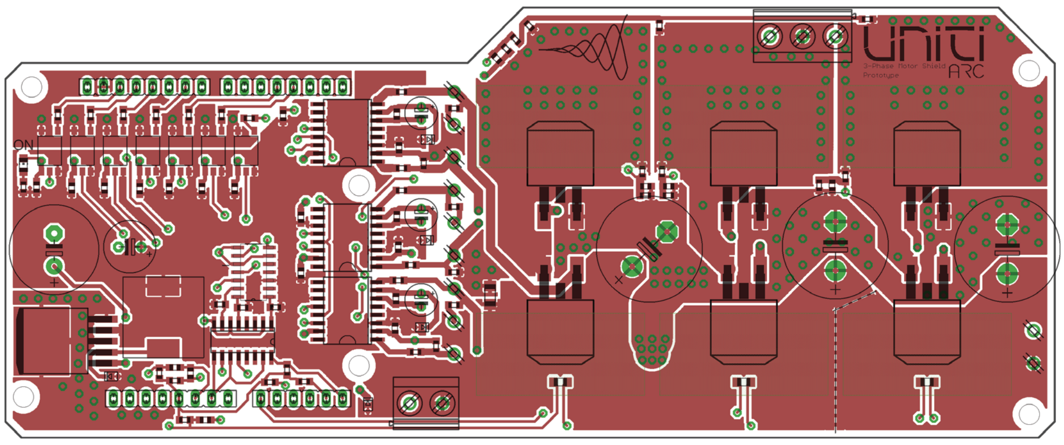


Figure C.4: Top layer layout of Uniti ARC Motor Shield

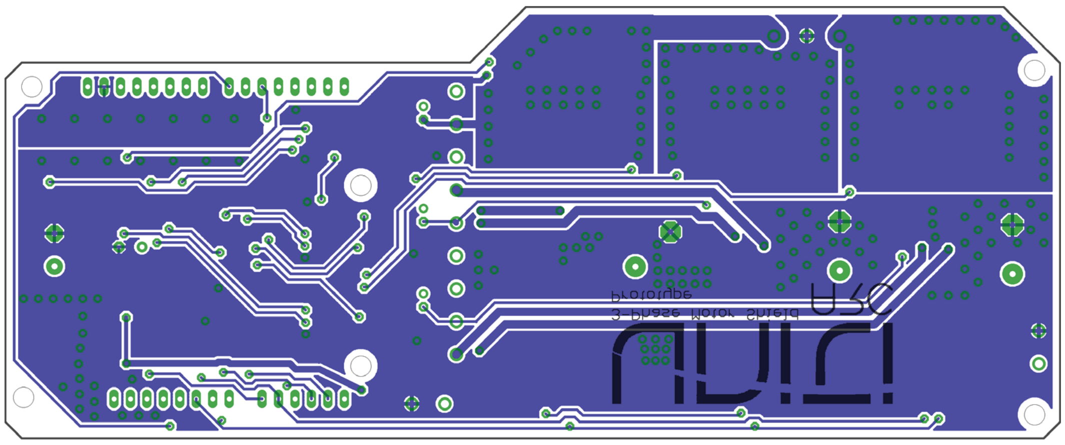


Figure C.5: Bottom layer layout of Uniti ARC Motor Shield

References

[1] P. A. Laplante, *Dictionary of Computer Science, Engineering and Technology*. CRC Press, Dec. 2000, google-Books-ID: U1M3clUwCfEC.

[2] "Arduino - Motor Shield R3." [Online]. Available: https://www.arduino.cc/en/Main/ArduinoMotorShieldR3

[3] "Formula Student | Bringing Engineering to Life." [Online]. Available: http://www.formulastudent.lu.se/

[4] "MVKN05 Projekt - Formula Student." [Online]. Available: http://www.ce.energy.lth.se/kurser/mvkn05_projekt_formula_student/

[5] "ATmega32M1 Automotive." [Online]. Available: http://www.atmel.com/devices/atmega32m1-automotive.aspx

[6] "ISO 11898-1:2015 - Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling." [Online]. Available: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=63648

[7] "Uniti ingenious electric car." [Online]. Available: http://www.unitisweden.com/

[8] "Creative Commons." [Online]. Available: https://creativecommons.org/

[9] "Arduino - UNO." [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardUno

[10] "Cadsoft - Eagle." [Online]. Available: https://cadsoft.io/

[11] "Uniti ARC Hardware · GitLab." [Online]. Available: https://gitlab.com/uniti-arc/Arc-eagle

[12] "GNU General Public Licence, v2." [Online]. Available: https://www.gnu.org/licenses/gpl-2.0.html

[13] "Arduino - GitHub.com." [Online]. Available: https://github.com/arduino/Arduino

[14] “Uniti ARC Software · GitLab.” [Online]. Available: https://gitlab.com/wrafter/Arc

[15] S. Colton, “Arduino HexBridge Shield v2.0.” [Online]. Available: http://scolton.blogspot.com/2010/06/arduino-hexbridge-shield-v20.html

[16] S. Cording, “codinghead/Arduino.” [Online]. Available: https://github.com/codinghead/Arduino/tree/allegro

[17] A. Thomason, “thomasonw/ATmegaxxM1-C1.” [Online]. Available: https://github.com/thomasonw/ATmegaxxM1-C1

[18] “ATmega328P.” [Online]. Available: http://www.atmel.com/devices/atmega328p.aspx

[19] “AVRISP mkII.” [Online]. Available: http://www.atmel.com/tools/AVRISPMKII.aspx

[20] “avr-gcc - GCC Wiki.” [Online]. Available: https://gcc.gnu.org/wiki/avr-gcc

[21] “AVR Libc Home Page.” [Online]. Available: http://www.nongnu.org/avr-libc/

[22] “AVRDUDE - AVR Downloader/UploaDEr.” [Online]. Available: http://www.nongnu.org/avrdude/

[23] “Gnu make.” [Online]. Available: https://www.gnu.org/software/make/

[24] “Git.” [Online]. Available: https://git-scm.com/

[25] “MCP2561 - Interface- Controller Area Network (CAN).” [Online]. Available: http://www.microchip.com/wwwproducts/en/MCP2561

[26] “TLP2345 | Photocouplers / Photorelays | TOSHIBA Storage & Electronic Devices Solutions Company.” [Online]. Available: https://toshiba.semicon-storage.com/eu/product/opto/photocoupler/detail.html

[27] “Products - Infineon Technologies - IR2110.” [Online]. Available: http://www.infineon.com/cms/en/product/power/motor-control-and-gate-driver-ics/non-isolated-gate-driver-ics-and-controllers/general-purpose-gate-driver-ics-industrial/IR2110/productType.html?productType=5546d462533600a401533d22a6185782

[28] “Products - Infineon Technologies - IRFS3207Z.” [Online]. Available: http://www.infineon.com/cms/en/product/power/power-mosfet/20v-300v-n-channel-power-mosfet/80v-100v-n-channel-power-mosfet/IRFS3207Z/productType.html?productType=5546d462533600a401533d3c64b33f17

[29] "FK244_13_d2_pak_, Heatsinks for D PAK and others, Heatsinks f cool, Fischer Elektronik." [Online]. Available: http://www.fischerelektronik.de/web_fischer/en_GB/heatsinks/C04/Heatsinks%20for%20D%20PAK%20and%20others/PR/FK244_13_D2_PAK_/$productCard/parameters/index.xhtml

[30] "Vishay - Glass Protected NTC Thermistors." [Online]. Available: http://www.vishay.com/product?docid=29056

[31] "Controlling Sensorless BLDC Motors via Back EMF | DigiKey." [Online]. Available: http://www.digikey.com/en/articles/techzone/2013/jun/controlling-sensorless-bldc-motors-via-back-emf

[32] "TL2575hv-12 | Buck Converter (Integrated Switch) | Step-Down (Buck) | Description & parametrics." [Online]. Available: http://www.ti.com/product/TL2575HV-12?keyMatch=tl2575hv-12ikttr&tisearch=Search-EN-Everything

[33] "ATmega16U2." [Online]. Available: http://www.atmel.com/devices/atmega16u2.aspx

[34] "Four Walled Cubicle - LUFA (Formerly MyUSB)." [Online]. Available: http://www.fourwalledcubicle.com/LUFA.php

[35] "USB.org - Getting a Vendor ID." [Online]. Available: http://www.usb.org/developers/vendor/

[36] "pid.codes." [Online]. Available: http://pid.codes/

[37] "Arduino - Mega." [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardMega2560

[38] "Arduino - Leonardo." [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardLeonardo

[39] "Arduino - Micro." [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardMicro

[40] "ATmega32U4." [Online]. Available: http://www.atmel.com/devices/atmega32u4.aspx

[41] "Arachnid Labs." [Online]. Available: http://www.arachnidlabs.com/blog/2013/10/18/usb-if-no-vid-for-open-source/

[42] "Developing, Testing, and Deploying Drivers." [Online]. Available: https://msdn.microsoft.com/en-us/windows/hardware/drivers/develop/index

[43] "Practical Windows Code and Driver Signing." [Online]. Available: http://www.davidegrayson.com/signing/

[44] "Inf2cat." [Online]. Available: https://msdn.microsoft.com/en-us/windows/hardware/drivers/devtest/inf2cat

[45] "WDK and WinDbg downloads - Windows Hardware Dev Center." [Online]. Available: https://developer.microsoft.com/en-us/windows/hardware/windows-driver-kit

[46] "STM32f303re - Mainstream Mixed signals MCUs ARM Cortex-M4 core with DSP and FPU, 512 Kbytes Flash, 72 MHz CPU, MPU, CCM, 12-bit ADC 5 MSPS, PGA, comparators - STMicroelectronics." [Online]. Available: http://www.st.com/content/st_com/en/products/microcontrollers/stm32-32-bit-arm-cortex-mcus/stm32f3-series/stm32f303/stm32f303re.html