BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

CONTROLLING LEGO TECHNIC ACTIVE ELEMENTS FROM EV3

ŘÍZENÍ AKTIVNÍCH PRVKŮ LEGO TECHNIC Z PROSTŘEDÍ EV3

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **DANIEL BLAŠKO**
AUTOR PRÁCE

SUPERVISOR **Ing. RADEK HRANICKÝ, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2024

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

153618

Institut: Department of Information Systems (DIFS)
Student: **Blaško Daniel**
Programme: Information Technology
Title: **Controlling LEGO Technic Active Elements from EV3**
Category: Embedded Systems
Academic year: 2023/24

Assignment:

1. Compare motorization environments for LEGO models over the last decades (9V system, Mindstorms RIS, Power Functions, Control+/Powered-up, NXT, EV3, Robot Inventor). Analyze the differences in hardware, software, and communication protocols for controlling active elements.
2. Become familiar with the Mindstorms EV3 environment. Explore options for programming with EV3 MicroPython.
3. After a consultation with the supervisor, propose a solution to control officially incompatible elements (e.g., Power Functions motors) from the EV3 environment, e.g., in the form of a driver and a library for MicroPython or by desigining an external adapter.
4. Implement the proposed solution.
5. Build several LEGO models from officially incompatible elements and create the corresponding programs to operate them.
6. Demonstrate the usability of your solution on the created models.
7. Discuss the results.

Literature:

- Crnokić, Boris, et al. STEM Classroom: Creating a Python Application for an EV3 Brick Robotic System Used to Transport 3D Printed Boxes. *Annals of DAAAM & Proceedings* 7.1. 2020.
- Li, Yixiao, et al. A platform for LEGO mindstorms EV3 based on an RTOS with MMU support. *OSPERT:* 51. 2014.
- LEGO MindStorms EV3 Communication Developer Kit, Lego Group. 2013 [online, vis. 1. 9. 2023], available from: https://education.lego.com/en-us/product-resources/mindstorms-ev3/downloads/developer-kits/.
- Getting started with LEGO Mindstorms Education EV3 MicroPython. Lego Group. Version 2.0.0. 2020 [online, vis. 1. 9. 2023], available from: https://education.lego.com/en-us/product-resources/mindstorms-ev3/teacher-resources/python-for-ev3/.
- PyBricks Documentation. The Pybricks Authors. 2020. [online, vis. 1. 9. 2023], available from: https://docs.pybricks.com/en/stable/.

Requirements for the semestral defence:
Points 1 to 3

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Hranický Radek, Ing., Ph.D.**
Head of Department: Kolář Dušan, doc. Dr. Ing.
Beginning of work: 1.11.2023
Submission deadline: 9.5.2024
Approval date: 30.10.2023

Abstract

LEGO Mindstorms is a popular series of programmable bricks with an advanced set of functionality, which has been helping children learn the basics of robotics and computer science for 25 years and which enjoys wide community support. The goal of this bachelor's thesis is to provide hardware and software solutions to enhance the compatibility between the LEGO Mindstorms EV3 intelligent brick and other motorized LEGO parts, primarily focusing on the Power Functions product line. In this thesis, I will create motor drivers in several iterations that broaden the capabilities of the EV3 brick to control a large number of motors, using my designs for printed circuit boards and 3D-printed enclosures. Additionally, I will develop software code blocks for the EV3 that allow it to communicate with the aforementioned drivers, as well as other UART devices.

Abstrakt

LEGO Mindstorms je populárna rada programovateľných kociek s rozsiahlou sadou funkcií, ktorá už 25 rokov pomáha učiť deti základy robotiky a počítačových vied, a ktorá sa teší širokej komunitnej podpore. Cieľom tejto bakalárskej práce je vytvoriť hardvérové a softvérové riešenia, ktoré zlepšia kompatibilitu medzi kockou Mindstorms EV3 a ostatnými motorizovanými LEGO súčiastkami, s primárnym zameraním sa na produktovú radu Power Functions. V rámci tejto práce boli v rôznych iteráciách vytvorené radiče, ktoré rozširujú možnosti kocky EV3 ovládať motory, s použitím vlastných návrhov dizajnov plošných spojov a 3D tlačených puzdier. Okrem nich boli vyvinuté softvérové kódovacie bloky, ktoré umožňujú kocke komunikovať s týmito radičmi a ďalšími UART zariadeniami.

Keywords

LEGO, LEGO Mindstorms EV3, EV3-G Blocks, LEGO Power Functions, microcontroller, Arduino, ESP, motor driver, I2C, UART, printed circuit board

Klíčová slova

LEGO, LEGO Mindstorms EV3, EV3-G Bloky, LEGO Power Functions, mikrokontrolér, Arduino, ESP, motorový radič, I2C, UART, plošný spoj

Reference

BLAŠKO, Daniel. *Controlling LEGO Technic Active Elements from EV3.* Brno, 2024. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Radek Hranický, Ph.D.

Rozšířený abstrakt

V roku 1998 prišla na trh prvá generácia LEGO Mindstorms, počiatok revolučnej rady produktov, ktoré pomohli tisíckam detí pochopiť základy robotiky a programovania [12]. Išlo o programovateľnú LEGO kocku so vstavaným mikročipom a príslušenstvom v podobe motorov, senzorov a LEGO dielikov [43]. Programovateľnosť a modularita tohto setu zaručovala takmer neobmedzené možnosti. Táto sada sa dočkala mimoriadneho úspechu v komerčných aj edukačných kruhoch, vznikla taktiež komunita nadšencov rozširujúcich možnosti využitia tejto programovateľnej kocky. V roku 2013 spoločnosť LEGO vydala tretiu generáciu série, Mindstorms EV3. Išlo o vyspelý produkt s firmvérom založeným na operačnom systéme Linux. Sada disponovala veľkým množstvom komunikačných rozhraní, ako Wi-Fi, Bluetooth, či USB, a množstvom príslušenstva v podobe rozličných senzorov a motorov, ktoré s kockou komunikovali pomocou I2C a UART komunikačných protokolov [27]. Aj z tohto dôvodu sa generácia EV3 dočkala najširšej podpory zo strany nadšencov.

Napriek tomu, že sú možnosti komunikácie s Mindstorms EV3 kockou početné, je jej kompatibilita s inými LEGO zariadeniami neúplná. EV3 na komunikáciu a ovládanie motorov a senzorov používa konektory a protokoly nekompatibilné s motormi a senzormi predošlých generácii Mindstorms, či zariadeniami Power Functions alebo WeDo. Až na pár spôsobov, ktoré využívajú medzery originálneho EV3 systému, nie je možné s týmito zariadeniami interagovať. Nezávisle vyvinuté rozšírené softvéry ev3dev a leJOS umožnili veľký pokrok v zlepšovaní vzájomnej kompatibility LEGO zariadení. Tieto operačné systémy naplno využívajú hardvérové schopnosti kocky EV3 a umožňujú ovládanie DC motorov, no aj komunikáciu prostredníctvom I2C a UART rozhraní. Tieto operačné systémy však vyžadujú väčšie množstvo programovacích znalostí v porovnaní so štandardným systémom EV3, čo môže byť problematické pre mladších nadšencov. Z tohto dôvodu som sa rozhodol v mojej bakalárskej práci zamerať na vylepšovanie kompatibility v rámci pôvodného systému tohto zariadenia.

Po otestovaní funkcionality existujúcich riešení som sa rozhodol vytvoriť vlastnú implementáciu, ktorá by kompatibilitu EV3 a iných motorizovaných LEGO zariadení posunula na vyššiu úroveň. Keďže 9V a Power Functions motory na rozdiel od NXT, EV3 a Powered Up motorov nedisponujú rotačnými enkodérmi a nedokážu odosielať dáta o ich súčasnom stave, považoval som systém, kedy by bol jeden takýto motor pripojený na jeden EV3 port za nevhodný a nevyužívajúci plný potenciál komunikačných portov kocky. Riešenie tohto problému prišlo s návrhom prepojenia kocky s mikrokontrolérom, ktorý by pomocou motorových radičov ovládal viaceré motory, pričom komunikácia medzi EV3 a mikrokontrolérom by mohla prebiehať pomocou oboch dostupných komunikačných protokolov. Prvý prototyp ovládača som vytvoril z vývojovej dosky Arduino Uno, spojenej s radičovým štítom pre Arduino vyvinutým spoločnosťou Adafruit. V tejto verzii komunikoval mikrokontrolér s EV3 cez I2C protokol. Pre prenos dát som vytvoril protokol, ktorý určoval stav, polaritu a striedu pre konkrétny motor. Prototyp bol napájaný LEGO Power Functions battery boxom využívajúcim alkalické batérie. Týmto prototypom bolo možné jednoducho ovládať až štyri motory využitím jediného portu kocky. Tento prvotný prototyp však dosahoval rozmery príliš veľké pre využitie v pohyblivých LEGO modeloch a PF Battery Box poskytoval nedostatočný výkon pre ovládanie štyroch motorov, preto som sa v úprave implementácie zameral na elimináciu týchto dvoch slabín prototypu. S využitím softvéru KiCad som navrhol vlastnú kremíkovú dosku, inšpirovanú štítom Adafruit, ktorý je však veľkosťou prispôsobený doske Arduino Uno a ktorý obsahuje komponenty nepotrebné pre moju aplikáciu. Z tohto dôvodu som sa pri návrhu mojej dosky orientoval na efektívne využitie priestoru, napríklad nahradením THT komponentov SMD komponentami. Plocha týchto

dvoch dosiek je takmer totožná, no mnou navrhnutý dizajn obsahuje integrovaný ESP-12F mikrokontrolér a port pre pripojenie ku kocke. Vzhľadom na to, že MCU je integrované do dosky, nie je nutnosť štít pripájať na vývojovú dosku a celkový dizajn je výrazne kompaktnejší. Pôvodný box na batérie som nahradil trojčlánkovým akumulátorom, ktorý poskytuje vyšší výkon pri kompaktnejších rozmeroch. Z dôvodu problematickej implementácie slave módu I2C komunikácie pri mikrokontroléri ESP12-F som sa rozhodol v tejto verzii implementácie využiť LEGO UART Sensor Protocol na vzájomnú komunikáciu. Obal prototypu bol na mieru navrhnutý a vytlačený na 3D tlačiarni. Po objavení chyby v implementácii, kedy na logiku radičov bolo vyvedených až 12 voltov, pričom maximálna povolená hodnota je 7 voltov [46], som sa rozhodol navrhnúť a vytvoriť druhú, vylepšenú verziu implementácie. Mikrokontrolér som nahradil vyspelejším čipom ESP32, L293D radiče som vymenil za TC4427COA MOSFET radiče. Toto spojenie mi umožnilo vynechať zo schémy posuvný register, vďaka čomu je doska osadená týmito komponentami kompaktnejšia, čo sa odrazilo aj na dizajne 3D tlačeného obalu, ktorý je v tejto iterácii o 25% užší.

Programovanie kocky EV3 funguje štandardne na základe blokov, ktorých vnútorná implementácia je odvodená od LabView kódu firmy National Instruments. Rôzne organizácie a individuá vytvorili nové bloky, ktoré rozširujú základnú funkctionalitu kocky, avšak bloky umožňujúce UART komunikáciu nezávislú na pripojenom zariadení neexistujú. Štandardný operačný systém EV3 využíva proprietárny protokol LEGO UART Sensor Protocol, ktorý nad UART protokolom definuje komunikáciu medzi kockou a senzorom. Vytvoril som bloky pre všeobecnú komunikáciu s UART zariadeniami, rovnako aj bloky, ktoré generujú dáta ovládajúce motory v mojej implementácii, a to prostredníctov oboch dostupných protokolov. Okrem toho som vyriešil dlho dokumentovaný problém I2C blokov spoločnosti Dexter Industries, kedy nebolo možné zasielať hodnoty väčšie ako 127 v rámci jedného bajtu.

Pomocou mojej implementácie je možné ovládať až štyri motory pri využití jediného portu kocky EV3. Tento prototyp môže nájsť využitie vo vozidlách s nezávislým pohonom všetkých kolies, kde implementácia dovoľuje ovládať každý motor nezávisle a umožniť väčšiu mieru kontroly nad vozidlom. Uplatnenie môže prototyp taktiež nájsť vo veľkoškálových motorizovaných modeloch a diorámach, pričom pri plnom využití portov kocky je naraz možné ovládať až 16 motorov.

Controlling LEGO Technic Active Elements from EV3

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Ing. Radek Hranický, PhD. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Daniel Blaško
May 6, 2024

Acknowledgements

I would like to thank my supervisor, Ing. Radek Hranický, PhD., for his time, effort, and guidance throughout the creation process of this thesis, as well as his exceptional approach to his mentees. Additionally, I would like to thank my family and friends, whose moral support helped to see this thesis through to completion.

Contents

List of Figures

Chapter 1

Introduction

LEGO is one of the most well-known and beloved toy brands in the World. Ever since its dawn, LEGO Group has been creating fun and educative toys that teach children creativity, problem-solving skills, and basic concepts of science, technology, engineering, and mathematics, or STEM for short. Later on, the Technic line was introduced, opening even wider possibilities for creating models with functional elements, further complemented by the addition of motorized components and hydraulics. In 1980, the Dacta branch was created with the aim of helping children learn in a playful and fun way at school, not only at home [10]. A partnership was struck between The LEGO Group and the Massachusetts Institute of Technology in 1985 with the goal of exploring the possibilities of teaching computer science and robotics via kids' toys. This research resulted in several prototypes of programmable bricks and directly influenced the creation of LEGO control boards and other computer-enabled products. However, the most influential fruit this collaboration bore was Mindstorms, a series of programmable bricks with motorized parts and sensors that allowed for the creation of robots that could interact with their surroundings and perform tasks [38]. Every Mindstorms generation had a set of pre-configured creations. However, the true purpose of the Mindstorms was to build and program one's own creations.

Since its debut in 1998, the Mindstorms series has seen immense success, being widely popular among consumers and schools alike. For many young children, it was their first encounter with the world of robotics, and it inspired many to seek further knowledge in this field. The Mindstorms spanned 4 generations, the RIS, NXT, EV3, and the Robot Inventor. Though it was discontinued in 2022, its fan base keeps its legacy alive. With custom-made accessories, modifications, and alternatives to the original brick, new fans can still indulge in the world of LEGO robotics.

Due to rapid advancements in computer science and technology, the inter-compatibility between different LEGO product lines and even between Mindstorms generations is problematic. While the first Mindstorms generation made use of the existing technology within the 9V product lineup, its successors abandoned existing solutions and used new designs and technologies to allow for more advanced functionality. This, however, meant that Mindstorms bricks would not work with simple accessory components, such as DC motors, at least not officially. As such, the fanbase of Mindstorms took it upon themselves to create custom hardware and software that broadens the abilities of the Mindstorms bricks and deals with these compatibility issues.

The aim of my project is to contribute to this shared initiative. My goal is to make use of the Mindstorms EV3, the most advanced entry to the Mindstorms lineup, and create hardware and software solutions that aim to improve the compatibility of the EV3 brick

and other motorized LEGO parts, such as the Power Functions line of products. I will create custom motor drivers that allow the EV3 to control multiple motors while only utilizing a single connection port, increasing the maximum number of devices connected to the LEGO brick. This will firstly be achieved by using off-the-shelf products. Later, I will create fully bespoke prototypes using my own printed circuit board design with an integrated microcontroller in two iterations and custom 3D printed casings that allow these prototypes to be elegantly incorporated into LEGO models, examples of which will also be provided in this thesis.

These hardware solutions will be complemented by custom software for the microcontrollers, allowing them to communicate with the EV3 via both available wired communication protocols, I2C and LEGO UART Sensor Protocol. Moreover, I will create new code blocks for the EV3 software that allow the programmable brick to execute general UART communication with connected devices, as well as control motors by using my custom data protocol built on top of the available communication protocols. Lastly, I will make modifications to preexisting third-party blocks that will solve their long-standing issues.

Chapters 2 and 3 provide a historical context of the development of programmable and motorized products manufactured by LEGO. In Chapter 4 I will elaborate on the current state of compatibility between the EV3 brick and other products. Chapters 5, 6, and 7 closely describe the process of creating motor drivers with integrated microcontrollers and their accompanying software in several iterations, from using pre-existing implementations to designing and constructing fully custom solutions, also including LEGO creation examples that demonstrate the possibilities of my implementation. In Chapter 8, I conclude the work on this thesis by creating new code blocks that expand the communication capabilities of the EV3 brick.

Chapter 2

Development of Programmable LEGO Bricks

The LEGO Group and its educational branch Dacta (now LEGO Education) first delved into the world of programmable bricks during the 1980s, when they partnered with MIT scientists in order to create a programmable educational toy for children [36]. Said MIT research group, led by Seymour Papert, has been developing robotic educational kits since the late 1960s, primarily in the Logo language, which he was a co-creator of. This partnership would result in several educational product lines, most notably the Mindstorms lineup, which was named in honor of Papert's 1980 book of the same name.

2.1 LOGO Brick

The first product attempts of this partnership required a permanent connection to a desktop computer to execute programs, inconvenient for creating moving creations [38]. However, due to the rapid speed of computer technology advancements, by 1989 they managed to create a standalone programmable brick, called the MIT Logo Brick (shown in Figure 2.1). This brick was used to conduct experiments with elementary school students, with children creating programs such as light-seeking or obstacle-avoiding robots [36].

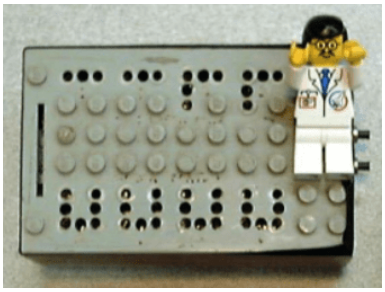

Figure 2.1: LEGO Logo Brick[1]

[1] https://www.cs.uml.edu/~fredm/papers/magical-machines.pdf

2.2 MIT Grey/Red Brick

In 1995, MIT's researchers came up with the Pocket Programmable Brick, dubbed the "grey brick" [38]. It was a fairly advanced device, with eight input sensor ports and four output ports for motors or lights from the 9V product series, an IR interface, an LCD screen, network connectors, as well as a microphone and a speaker. This, however, made it very difficult and expensive to manufacture, especially on a small, research-only scale. Trying to leverage processing power and power consumption low enough to be able to get sufficient power from batteries, the team landed at the Motorola 6811 CPU [42]. Its design scheme can be seen in Figure 2.2.

The next iteration of this project was created in 1996 in the form of the MIT Red Brick [38] (Figure 2.3). This version was more streamlined, with six input ports instead of eight. The sound and network devices were also removed. The LCD screen was kept, as it was found out that it was much easier for children to see the output directly on the brick rather than having to connect it to a computer beforehand. There was pressure to create an even more stripped-out version with even fewer input and output ports, but all of the brick's ports were often utilized at once during testing, so the I/O was kept unchanged. An emphasis was also put on making the brick easy to manufacture and robust enough to endure kids' use. More than 100 of these programmable bricks were made in total and they were put to use in research groups in classrooms in the United States and Thailand between 1996 and 1999.

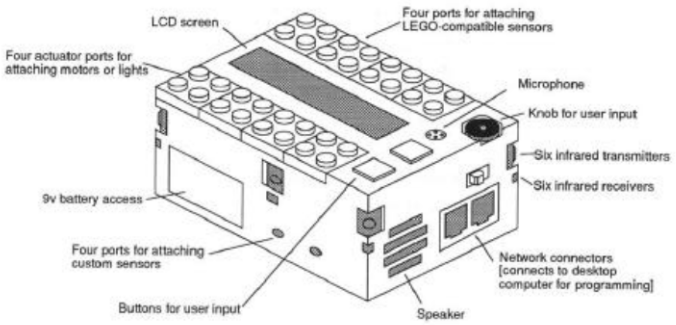

Figure 2.2: Schema of the MIT Grey Brick [2]

Figure 2.3: MIT Red Brick[3]

[2] https://dspace.mit.edu/bitstream/handle/1721.1/34694/32601552-MIT.pdf?sequence=2&isAllowed=y

2.3 RCX

MIT's research would eventually lead to the creation of the LEGO Mindstorms RCX, the first commercially available fully programmable LEGO brick. It featured a Hitachi/Renesas 8-bit CPU with an internal 16-bit architecture, operating at 10MHz, with 32KB of RAM and 16kB of ROM [9]. The I/O consisted of an LCD screen, an IR interface for communication between an RCX and a computer or between several RCXs, three input and three output ports [43]. This downgrade, together with a smaller battery pack compared to the Red Brick, resulted from an objective to reduce the weight of the programmable brick. The Red Brick tipped the scales at approximately 370 grams [12], which could make its application in small and mobile LEGO creations quite cumbersome. The RCX, weighing 192 grams, was considerably lighter.

The first version of the brick, the RCX 1.0 (depicted in Figure 2.4), featured a connector for an AC power adapter beside battery power, which was removed in the subsequent 1.5 and 2.0 revisions [43]. The RCX shipped with electric motors, touch and light sensors, an IR Tower, and LEGO Technic pieces to create various models. The programs for RCX to run were written on a PC using RCX Code software, which provided a block-based coding environment suited for children. These programs were then loaded onto the RCX via an IR Tower connected to the PC, utilizing the RCX's IR interface.

The RCX brick was released in 1998 and saw great commercial success [12]. A community quickly emerged, sharing reverse-engineered firmware and creating alternative operating systems, such as the Java-based leJOS, or programming languages such as NQC (Not Quite C) [53, 43].

Figure 2.4: LEGO Mindstorms RCX[4]

[3] https://web.archive.org/web/20221017071530/http://web.mit.edu/6.933/www/Fall2000/LegoMindstorms.pdf

[4] https://www.electricbricks.com/lego-lego-parts-electric-yellow-mindstorms-rcx-with-power-jack-complete-brick-p-1015.html?language=en

2.4 Scout

The Mindstorms Scout (shown in Figure 2.5) was a simplified version of the Mindstorms RCX brick, aimed at younger audiences [22]. It was released in 1999, a year after the release of the RCX. The I/O was reduced to two input and two output ports, though it had a built-in light sensor for input and a light for communication with other devices, as well as an IR transceiver and a piezo sound module [23]. Instead of creating programs on a computer, the programs primarily were built inside the brick's user interface, though it was possible to write them on a PC and send them to the brick via the RCX IR tower [22]. Over 70 commands are available in the LEGO Assembly language, with more than 30 command built-in command subroutines and an option for three user-defined subroutines. A user program can run up to six tasks in parallel with features such as access control commands, which let the user set a priority of the tasks, improving on the capabilities of the RCX.

Figure 2.5: LEGO Mindstorms Scout[5]

2.5 NXT

The next iteration of the Mindstorms robot, the NXT, debuted in 2006. It was powered by a much more capable 32-bit ARM7-based Atmel CPU running at 48MHz and an 8-bit Atmel AVR microcontroller for motor control, with 64KB of RAM and 256KB of Flash memory [41]. The NXT also featured a speaker, a USB Full Speed port, and a Bluetooth interface. The display was now significantly larger at 100x64 pixels, which enabled simple programs to be written directly in the NXT brick's graphical interface. More complex programs were written in the NTX-G PC software and then downloaded to the brick via USB or Bluetooth. Bluetooth connectivity also enabled up to three NXT bricks to communicate wirelessly. I/O was also upgraded, with four input ports and three output ports. The ports used proprietary 6-wire connectors based on and almost identical to the RJ12 connector, with the only difference being an offset latch on the NXT connector. The NXT set also

[5] https://www.worthpoint.com/worthopedia/lego-9735-robotic-discovery-set-lego-150133089

featured more advanced motors. Unlike the RCX's simple DC motors, the NXT motors' internals featured an optical encoder to enable position and velocity data feedback [16]. The set also included touch, light, sound, and ultrasonic sensors. The NXT brick can be seen in Figure 2.6.

A revised NXT 2.0 version was brought to market in 2009, with a more advanced color sensor replacing the light sensor and added RFID functionality for communication between multiple NXT bricks [51]. Due to the popularity of Mindstorms sets among enthusiasts and programmers, The LEGO Group decided to make the NXT's software open-source.

Figure 2.6: LEGO Mindstorms NXT[6]

2.6 EV3

The most advanced entry into the Mindstorms series, the EV3 (seen in Figure 2.7), was launched in 2013. With an ARM9 Texas Instruments AM1808 processor, 64MB of RAM, and 16MB of flash storage, this brick saw a massive step forward in computing performance [27]. The display has an increased resolution of 178x128 pixels. Improvements to the I/O include a microSD card slot, a mini-USB port, a USB 2.0 high-speed interface, and networking functionality via USB. The EV3 uses the same modified RJ12 interface to connect to its sensors and motors as the NXT, though its number of output ports was increased to four. This enables the EV3 to be backward compatible with older NXT sensors and motors, however, the EV3's devices are not compatible with the NXT brick [52].

The EV3 shipped with two large motors, one medium motor, a remote control, a touch sensor, a color sensor, and an infrared sensor. The educational variant of the EV3, the Education EV3 Core Set, included one additional touch sensor, a gyroscope, and ultrasonic sensors instead of the retail version's infrared sensor. It was also powered by a rechargeable battery pack instead of the six AA batteries needed to power the consumer variation.

The powerful internals of EV3 allow it to run a Linux distribution as its kernel [26]. A LEGO application Virtual Machine runs above the hardware interfaces, kernel and shared

[6] https://www.brick-shop.de/Electric-4951.html?language=en

libraries, this virtual machine is controlled by a built-in system and user-created byte-code programs. This firmware structure can be seen in detail in Figure 2.8.

Figure 2.7: LEGO Mindstorms EV3

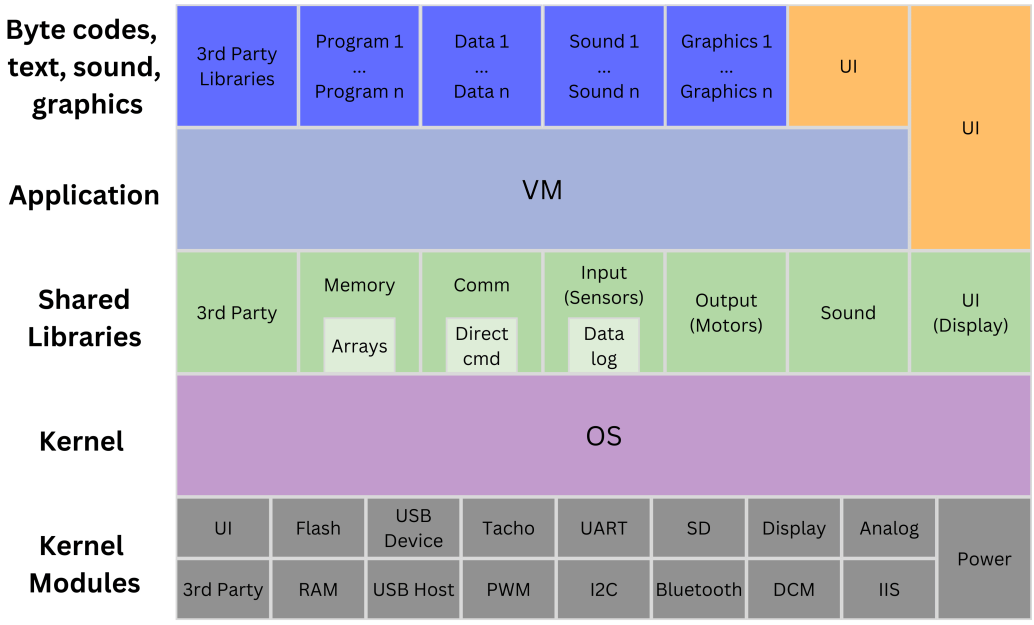

Figure 2.8: A graphical representation of EV3's firmware structure

2.6.1 Third-Party Extensions to EV3's Functionality

EV3's operating system was released as open source, and the presence of an SD card slot allowed third parties to create alternative operating systems running in dual boot-like mode, eliminating the need to flash the brick's built-in system, like it was in previous iterations of the Mindstorms brick.

The most popular third-party operating system made for the EV3 is ev3dev, an open-source Debian-based Linux distribution operating system. It allows for extended functionality of the EV3, supporting programs in numerous programming languages, such as Python, Java, Go, or C++ [49]. The most popular language for programming the EV3's behavior in this system is Python, with extensive `python-ev3dev` and `Pybricks` libraries that allow for extended functionality with devices otherwise not compatible with the default EV3 operating system. Another example of an alternative to the native EV3 system is leJOS, a Java-based virtual machine. In its third iteration, previously running on the RCX and NXT devices, it includes a Java EV3 API and a Java runtime system.

2.6.2 Third-Party Alternatives to the EV3

Third-party support does not end with alternative software, several companies built custom hardware as well. Mindsensors manufactured hardware extending the base EV3's functionality, such as RFID readers, additional sensors, or controller interfaces[7]. These devices can be controlled via the python-ev3dev library, alternatively, an `.ev3b` block file can be imported into the stock Mindstorms LabView programming application and allow block-based programming with these devices in the original EV3 firmware. Dexter Industries sold BrickPi, a Raspberry Pi-based alternative to the EV3 that can run EV3-compatible operating systems. This product made it easier for new enthusiasts to enter the world of Mindstorms since the EV3 has been discontinued for several years. In addition, this company also manufactured custom devices and sensors for the EV3[8]. However, it seems that these companies have ceased their manufacturing operations.

2.7 Robot Inventor

The latest and last member of the Mindstorms family due to the discontinuation of the series after the year 2022, Robot Inventor (Figure 2.9), is aimed at a younger audience compared to previous sets [31]. The Robot Inventor brick provides a limited interface, with only a 5x5 matrix display. Its CPU is an ARM-based Cortex-M4 processing unit running at 100MHz, with 320KB of memory, 1MB of flash, and 32MB of memory for programs, sound files, and other content [30]. This too is a downgrade compared to the EV3, but the simplified firmware results in significantly faster boot times. The brick is controlled by a mobile phone or a controller through a Bluetooth connection. Simple programs can be written in Scratch, a high-level block-based programming language. The Robot Inventor brick also supports programming in Python for more advanced projects. The brick has built-in three-axis gyroscope and accelerometer sensors, a speaker, and six ports, all of which can be used for either input or output, unlike the previous Mindstorms generations, which had separate ports for motors and sensors. Power was provided by a rechargeable battery pack, charged via USB Micro-B. Inside the set were four medium motors with rotational encoders and color and distance sensors.

[7] http://www.mindsensors.com/
[8] https://www.dexterindustries.com/
[9] https://www.amazon.co.uk/LEGO-Mindstorms-Large-Hub-88016/dp/B00B42IRBC

Figure 2.9: LEGO Mindstorms Robot Inventor[9]

2.8 Spike

LEGO Spike sets are educational variations of the Robot Inventor set. The Spike Prime set was released in 2020. Its Technic Large Hub is identical to the Robot Inventor Hub, but the Prime set includes a new Force sensor [30]. It replaced the touch sensor, expanding its functionality by being able to measure the force applied to it. Instead of four medium motors in the Robot Inventor set, the Spike Prime included two medium motors and a large motor.

In 2021, the Spike Essential set was released as a simplified counterpart to the Spike Prime set, oriented at younger children, featuring the LEGO Technic Small Hub (shown in Figure 2.10). Inside it was the same processing unit as in the Spike Prime set [32]. The main difference between the two was the Small Hub's reduced size and I/O, with just two motor/sensor ports, a lack of a speaker, and a single RGB status LED. This set shipped with two small motors with positional encoders, a color sensor, and a 3x3 Color light matrix brick.

Figure 2.10: LEGO Spike Essential[10]

[10] https://www.lego.com/en-gb/product/lego-technic-small-hub-45609

2.9 Other Programmable LEGO Products

Mindstorms sets were not the only LEGO Group products that were programmable, however, these other products could either not be considered 'bricks' due to their form factor, or their programming capabilities are limited compared to the Mindstorms series.

2.9.1 Technic Interface A

The LEGO Technic Interface A (Figure 2.11) was the first programmable control board for electronic LEGO devices [35]. It had two input ports for optosensors and six output ports for 4.5V motors or lights. Output ports were paired together, so two ports could be occupied either by two motors rotating uni-directionally, or one motor rotating bi-directionally. Optosensors could be paired with a black-and-white disk to act as rotational encoders. The control center was connected to a computer through a separate interface card [33, 3]. There were three choices of software for controlling the Technic interface, LEGO Lines, LEGO TC Logo, and LogoWriter Robotics [35]. The programs were mainly written using the Logo language, but there were other options available as well, depending on the computer to which the control center was connected. These languages include 6502 Assembly, BASIC, and COMAL-80 [21].

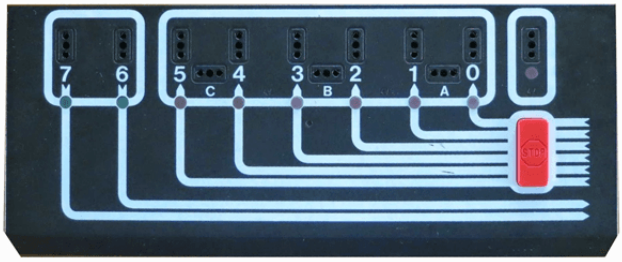

Figure 2.11: LEGO Technic Interface A[11]

2.9.2 Technic Control Center

This control center was released in 1990 and was a standalone unit, with no necessity to connect to a computer [11]. As such, the programming options were limited. The programming was done by recording the moves by controlling the station, which would then repeat the steps, and two separate programs could be recorded [5]. The control center could control motors, lights, and sound elements across three channels, though there were no input ports due to the simplicity of the control center. The control center was the first LEGO product to use the 9V line of devices. The first version was powered by six C-type batteries, a revision of the control center added an AC/DC transformer, allowing the unit to be powered by a wall socket [50]. It can be seen in Figure 2.12.

[11] https://www.bricklink.com/v2/catalog/catalogitem.page?P=70455#T=C&C=11

[12] https://brickscout.com/en/products/detail/4057296513314/part__2840c01/black-technic-control-center-i

Figure 2.12: LEGO Technic Control Center[12]

2.9.3 Control Lab Serial Interface

Released in 1993, this product was the successor to the Technic Interface A, this time fairly more complex [34]. As can be seen in Figure 2.13, it featured 8 input and 8 output ports and worked with the 9V line of LEGO motors and sensors. It connected to the computer via an RS-232 serial port and could be controlled by the LEGO RoboLab software or the LEGO Control Lab software using the Logo language.

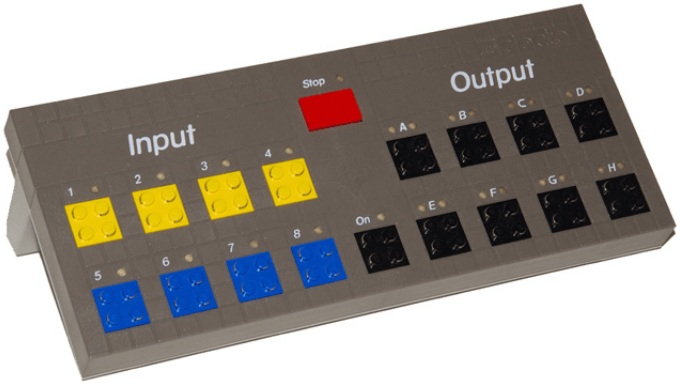

Figure 2.13: LEGO Control Lab Serial Interface[13]

2.9.4 Code Pilot

This brick used a barcode reader to write its programs. A sheet of 44 three-digit barcodes was included, and they could be scanned by the Code pilot Brick to assemble programs [6]. Alternatively, pre-made programs could be also scanned to program the predefined LEGO build with more ease. The brick featured a single input connector and one output connector and came with a motor and a touch sensor from the 9V line. The brick also had a built-in sound module that could also be programmed to play at specified times with several notes. The code powering this brick was proprietary and was never released publicly by the LEGO Group.

[13] https://www.bricklink.com/v2/catalog/catalogitem.page?P=2954#T=C&C=10

2.9.5 WeDo 1.0

The first generation of the WeDo was launched in 2009 as an educational kit [24]. Its main component was the WeDo USB Hub, which was used to connect to a computer to program and execute tasks. It featured two Power Functions-style ports to connect motors, sensors, or lights, and it shipped with a Power Functions Medium motor, a tilt sensor, and a motion sensor. Up to three Hubs could be connected within a single program to increase the number of motors or sensors working at once. The hub was not a standalone computer, so a permanent connection to a computer was needed to execute tasks and power electrical components, which hindered the ability to create moving designs. As the USB 2.0 connection only allows for a 5V output and the Power Functions work on a 9-volt basis, the power output was also decreased compared to standard Power Functions connections. WeDo programs were written in Scratch.

2.9.6 WeDo 2.0/Powered Up

An updated version of the WeDo line was brought to market in 2016 [2]. The new generation of the Hub was still not a fully independent device, it still worked only by communicating with a computer to control motors and sensors. The communication was now carried out via Bluetooth, which eliminated the need for a cable connection to a computer. A new, proprietary 6-pin connector was used which allowed for similar functionality to Mindstorms cables (power, device identification, and serial data communication) [13]. This connector was shared between the educational WeDo and Spike sets and the consumer-oriented Mindstorms Robot Inventor and Technic sets, which allows for greater interoperability between various sets. Several different versions of the hubs were released:

- **WeDo 2.0 Smart Hub** - A part of the WeDo educational set. It is powered by two AA batteries and has two connection ports for motors and sensors.
- **Powered Up Hub** - Similar to the WeDo Smart Hub, it features two motor/sensor ports. Six AAA batteries provide a maximum of 9 volts of voltage for the motors.
- **Technic Hub** - A more advanced version of the previous hubs, the Technic hub has four ports and a built-in tilt sensor. Power is provided via 6 AA batteries.
- **Powered Up Move Hub** - This hub features two built-in motors with positional encoders and a tilt sensor, with two I/O ports, powered by 6 AAA batteries (Figure 2.14).

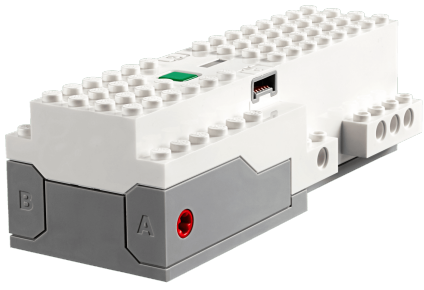

Figure 2.14: LEGO Powered Up Move Hub[14]

[14] https://www.lego.com/en-fr/product/move-hub-88006

Chapter 3

An Overview of LEGO Powered Bricks

LEGO first introduced motorized components to its lineups in 1966 in its train set, allowing for new ways to be played with. Since then, there have been several generations of motorized products, each new generation being more advanced and enabling more creative ways of use.

3.1 4.5V Motor Line

The first LEGO motors were released in train sets in 1966 and were based on a 4.5V architecture, powered by three C-type batteries [4]. The connector was a simple two-pin prong. These train motors came out in different iterations, as well as a 12-volt version. In 1977, a standalone motor, shown in Figure 3.1, was released [1]. This motor was used in the Technic Interface A control center. The 4.5V line also included a light module, a battery box, a touch sensor, and an optosensor [33].

Figure 3.1: LEGO 4.5V Motor[1]

[1] https://brickscout.com/en/products/detail/4057296837304/part__6216m2/light-gray-electric-motor-45v-type-2-for-2-prong-connectors-with-middle-pin

3.2 9V Motor Line

The first 9V products were motors that came out with the release of the Technic Control Center in 1990 [15], shown in Figure 3.2. Later, with the release of other sets, numerous other products from this line were released, such as other motors of different sizes and power outputs and several types of sensors, such as touch, light, rotation, and temperature sensors, as well as battery boxes and lights. The connector uses a two-line connection with four pins accessible from both the top and bottom of the connecting brick, allowing for stacking of the devices. These products were used in numerous sets, notably in the first generation of the Mindstorms brick. They were officially discontinued in 2007 after the release of the Power Functions line.

Figure 3.2: Technic Motor 9V

3.3 Power Functions

The Power Functions (PF for short) line was introduced in late 2007 as a new generation of motorized LEGO pieces, replacing the 9V line. It included several types of electric motors, battery boxes, lights, IR remote control devices and IR receivers, a servo motor, and a switch. Several of these devices can be seen in Figure 3.3, from the left to the right: PF M Motor, PF L Motor, PF Servo Motor, PF IR Receiver V2, PF Battery Box, PF IR Remote Control.

Power Functions devices utilized a 4-pin connector with pins on both the top and the bottom of the connector (seen in Figure 3.4). Like in the 9V connector, this allowed for mating multiple devices in a single serial connection. Pulse width modulation with a frequency of 1200Hz is used to regulate the speed or the angle of rotation of motors [8]. The duty cycle can be set in 7 non-zero stages (0%, 26.6%, 38.9%, 51.2%, 63.0%, 75.0%, 87.2% a 100%), providing seven speed values for the motors in either direction, for example, a duty cycle on pin C1 set at 51.2% would run a motor clockwise at half speed or turn the servo motor to 45° clockwise, setting this duty cycle on the C2 pin would rotate motors counter-clockwise.

In most usages, only two of the four pins were utilized, since the PF M, L, and XL motors were simple direct current motors connected by the C1 and C2 pins with no rotational encoders or other sensors that would send feedback data. The 88004 Servo Motor is the only Power Functions product to use all four pins [15]. The 9V and 0V pins power the motor while the C1 and C2 control pins set the angle of rotation between $-90°$ and $+90°$.

Figure 3.3: Power Functions products

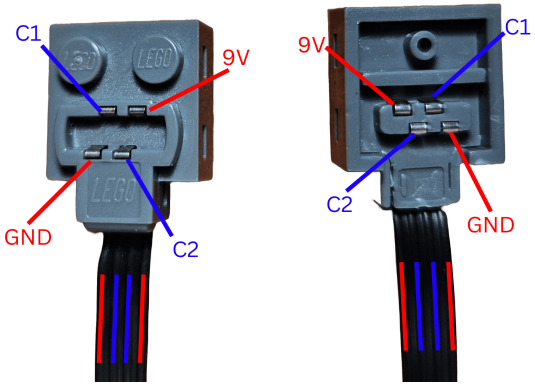

Figure 3.4: Power Functions connector pinout

3.4 Powered Up

First products from the Powered Up line were released in 2017. While the early components were released under several names, such as Boost or Control+, the naming scheme was later unified under the Powered Up moniker. A successor to Power Functions products, the Powered Up line consists of electric motors, control hubs, remote controls, and lights. Powered Up products use a proprietary 6-pin connector, which allows for similar functionality to Mindstorms NXT and EV3 cables, such as power delivery, device identification, and serial data transmission [13]. Pins 1 and 2 are PWM-controlled power leads for motors, pin 3 is a ground pin, pin 4 provides power for device electronics at 3.3 volts, and pins 5 and 6 are used for analog device identification via resistance and digital serial data transmission. Pin 5 sends data from the hub to a device and pin 6 transmits data the other way [14].

The Powered Up Medium motor and the Powered Up Train motor are the only motors in the series without a rotational encoder, providing no feedback. The rest of the range features rotational encoders, providing positional data as well as motor speed. One such member of the range is depicted in Figure 3.5. The encoder resolution is 360 counts per revolution, the sensor input accuracy is +/- 1 degree of rotation, and the motors can be controlled with an accuracy of +/- 3 degrees [30]. The sensor's refresh rate is 100Hz. While the Boost Medium Motor only provides relative rotational position data, other motors also include absolute encoders, containing data about the motor's true angular position. This

makes them suitable for use as servo motors where the use cases require a return-to-zero functionality, such as vehicle steering.

Most motors are made to work with 9 volts, but several hubs provide less voltage (WeDo Smart Hub provides 3 volts and Robot Inventor and Spike sets' battery packs provide 7.3 volts) [32]. The Technic Small Angular motor can only work with voltages between 3.3 and 6 volts, while the Medium and Large Angular motors are compatible with power sources of voltage between 5 and 9 volts. This means that not all motors might be compatible with all types of Powered Up hubs.

Figure 3.5: LEGO Powered UP Medium Angular Motor[2]

[2] https://www.lego.com/en-us/product/medium-angular-motor-88018

Chapter 4

The Current State of Compatibility Between EV3 and Motorized LEGO Products

As the history of motorized LEGO bricks spans decades, it is understandable that with advances in technology, newer generations of products might not be backward compatible with their older counterparts. Since the older motors, notably from the 9V and Power Functions line, are simple direct current motors (apart from a few exceptions, like the PF servo motor), it is not too difficult to convert them for use in different applications. Their connectors can be modified in various ways, the LEGO Group even released converter cables to these devices, though their use in the Mindstorms EV3 is problematic.

4.1 Stock EV3 Firmware and its Compatibility

The EV3 uses auto-identification of its devices by measuring the resistance on pins 5 and 6 of its connector. Since the 9V and PF motors do not have the elements necessary to carry out the identification functionality, the detection fails on EV3's part, and the output to the motors is stopped after roughly one second, even with official LEGO extension cables.

This problem can be overridden in various ways. Firstly, the fact that the motors run for a short time even without successful identification can be exploited. In EV3's block-based programming software, a function can be created that periodically switches the power output to the motor between different power levels that are close enough to each other that the change in output might not be noticeable. Since this resets the power output before identification failure can stop the output, the motor should be able to run [19]. A second option also exists, though this one requires working with the hardware itself. When a Power Functions cable and an EV3 cable are cut, they can be connected as depicted in Figure 4.1. The added resistors create a resistance that is equivalent to a Mindstorms NXT motor, which is compatible with the EV3 system. It can be then run in the LabView software as an NXT motor in its unregulated form with no issues. Another solution is to use third-party hardware. The Mindsensors group developed a controller that attaches to a Power Functions IR receiver and sends infrared signals to connected devices that can be programmed on the EV3 or NXT bricks; however, this solution requires the ownership of said IR receiver [1].

[1] http://www.mindsensors.com/ev3-and-nxt/123-pf-motor-controller-for-nxt-or-ev3-pfmate-

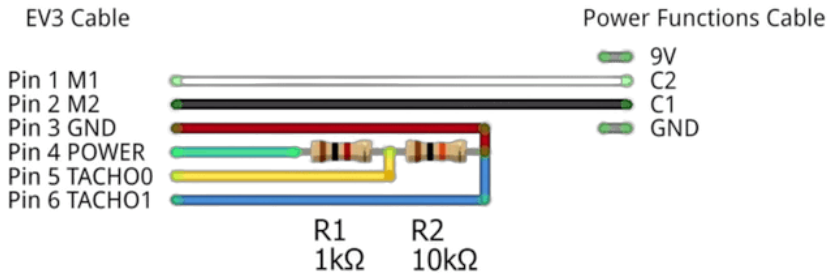

Figure 4.1: Connection between EV3 and a Power Functions motor[2]

4.2 Compatibility with the ev3dev Firmware

The open-source ev3dev and its python-ev3dev Python library make great strides in providing improved compatibility with other devices. The simple direct-current motors, which most of the 9V and PF range are, can be connected to the EV3's port with connections that do not require components for identification. The `dc-motor` class provides functions that can run motors, set their polarity, and set the duty cycle for pulse width modulated motors. This class is not only meant for simple LEGO motors, but it allows for connecting any 2-pin DC motor. This functionality is also present in the Pybricks MicroPython library, which also offers numerous other classes for a wide range of devices, including Powered Up hubs, third-party sensors, and generic UART and I2C devices [49]. This allows for compatibility with virtually any device that supports the aforementioned communication protocols.

4.3 Compatibility with the leJOS Firmware

leJOS is an another alternative firmware for the EV3, based on a Java runtime system [48]. In its third iteration for the EV3, it was previously developed for both the RCX and the NXT bricks. This firmware also improves on the stock firmware's functionality, with classes for DC motors, UART and I2C-compatible devices, and third-party devices. It also includes a fairly robust interface for communicating with RCX sensors and devices, a functionality that ev3dev and Pybricks lack.

4.4 Available Protocols for Communicating with the EV3

Since the EV3 bricks contain the necessary hardware to communicate via I2C and UART protocols and both the ev3dev and the leJOS firmwares implement classes for handling communication via these protocols, I can consider using these protocols during the design process of an implementation prototype.

4.4.1 I2C Protocol

Inter-Integrated Circuit (I2C for short) is a serial, synchronous communication bus [39]. It connects multiple devices, which are assigned as a master or as a slave. There can be multiple masters and slaves on a single bus, each device is assigned a unique 7-bit address,

[2] https://www.youtube.com/watch?v=Ns3TODgRlCE

meaning that up to 128 devices can share the bus. The I2C protocol also allows for 10-bit addresses, potentially increasing the number of devices connected to the bus, though this address format sees rare use. The bus only contains two wires, serial data (SDA) and serial clock (SCL), which can communicate bi-directionally. The clock signal is set by a master device, which a slave device adapts to. If a slave device cannot process requests fast enough and has the necessary controllers, it can make use of clock stretching, where it holds the SCL line at low value until it is ready to process further requests. If such a situation occurs, the master device cannot send more transactions. Both master and slave devices can send and receive messages, however, slave devices can only send bytes as a response after being addressed by a master device. Bus speeds vary between 100kbit/s and 3.4Mbit/s in bidirectional communication depending on the bus' mode. An ultra-fast mode with a capability of transferring 5Mbit/s also exists, though the communication is only unidirectional, the protocol is heavily modified compared to standard I2C operation and its use is rather niche.

4.4.2 UART Protocol

Universal asynchronous receiver/transmitter, also known as UART, is a serial communication protocol that works asynchronously [44]. Instead of a clock line that sets the clock signal of the communication, an identical baud rate has to be set on both devices, which ensures that both devices will communicate at the same clock speed. Apart from the baud rate, several other aspects of communication must be set to establish a reliable connection, such as start, stop, parity bits, and voltage levels. Both communicating devices contain RX (receive) and TX (transmit) ports. Due to the elimination of the clock line, the protocol allows for full-duplex communication (sending data both ways simultaneously), though half-duplex or simplex communication is also possible. The drawback is, however, that only two devices can share a single connection, unlike the I2C protocol. Moreover, both devices need to be equipped with their clock rate generators.

4.4.3 LEGO UART Sensor Protocol

The EV3 uses a proprietary UART-based protocol for communicating with its sensors. The brick uses voltage measurements on pins 1 and 2 of a port to determine the type of the connected device. For digital UART sensors, pin 1 must be set to low and pin 2 set to high [27]. After connecting, a handshake procedure is initiated by the sensor at a baud rate of 2400. The handshake consists of messages bearing information about sensor type, communication speed, the number of sensor modes, data types of sensor measurements, and more [28]. A single message consists of a message type byte, payload, and a checksum byte. Several messages have a set default value and can therefore be omitted from the handshake, hence the only required messages are for sensor type, its modes and their data format [18].

After the handshake data has been transmitted, an ACK (0x04) byte is sent by the sensor. If the EV3 returns the ACK byte within the timeout period of 80 milliseconds, the connection has been established and the communication speed changes to the value presented in the handshake data if specified, otherwise the baud rate stays at the default value [29]. Data is sent from the sensor at a period ranging from 1ms to 100ms, and after receiving a NACK (0x02) byte from the EV3. The sensor must reset and begin the handshake after not receiving a NACK message for longer than one second [18]. The length

of the data payload can be set at exponents of two, with the maximum value being 8 bytes. An example handshake captured by a logic analyzer can be seen in Figure 4.2.

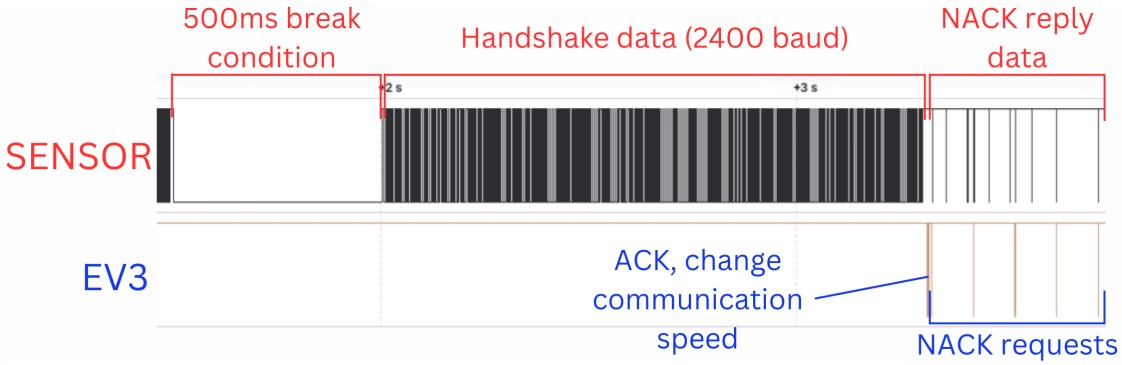

Figure 4.2: Handshake of the EV3 IR Sensor captured by a logic analyzer, shown in Saleae Logic software

The LEGO Powered Up UART Protocol used in the newer Mindstorms products shares the same base as the UART Sensor Protocol, and enhances the protocol with extended functionality and syntax [20].

4.4.4 LEGO EV3-G Blocks

The most common way of interacting with sensors is using LEGO's own software, which uses block-based programming that is user friendly even for younger enthusiasts. These blocks may have multiple modes of operation, every one of which is implemented by its own .vix file. These XML-based files are a modified version of National Instruments' LabVIEW Virtual Instruments files, or VIs for short. They consist of inputs and outputs, functions, their connections, also known as wires, and "grey blobs". These blobs or primitives, named such by LEGO in its Block Developer Kit, are references to byte codes implemented in EV3's firmware that are executed upon being called in a program [25]. They are used to handle I/O and file operations, display information on EV3's screen, and more. Depending on the hardware they make use of, these blocks can be backward compatible with the NXT generation of Mindstorms.

The block is made up of several files. The **blocks.xml** file located in the root of the directory contains information about the block, describes its inputs, outputs, the names of VIX files for its modes and default values for aforementioned inputs and outputs. A file of the same name, located in the *strings* directory, includes text descriptions of the block's modes and objects contained within. VIs directory includes VIX files for the block inside PBR and NXT directories, the former containing files for the EV3, while files for the NXT are located within the latter directory. Illustrations for the blocks are situated in the *images* directory, and an HTML file describing the block's functionality can be found in the *help* directory. These files are then added to a ZIP archive and the archive's extension is changed to **.ev3b**. This file can then be imported into the programming interface for the EV3.

Chapter 5

Improving the Compatibility Between EV3 and LEGO Motors

As stated in the previous chapter, there are attempts to improve on the inter-compatibility between various LEGO devices, with the Power Functions line receiving the most attention from the community. The release of the ev3dev firmware and its motor drivers meant a large increase in usability of PF motors with EV3, however, there are still areas where improvements could be made. Firstly, I explored and implemented existing options for connecting LEGO motors to the EV3, and later on I would explore the possibilities of creating new and improved ways of integration.

5.1 Controlling Power Function Motors via Arduino

After consulting possible options for improvement with my thesis supervisor, I have decided to implement a scheme where an Arduino Uno microcontroller would act as a middleman between the EV3 and Power Functions motors, receiving input data from EV3 and driving one or several motors accordingly. I have chosen to use the Arduino Uno for its ease of use, low price, and availability of various accessories and shields for the microcontroller. A Raspberry Pi, for example, might be too powerful for this usage, moreover, its price is also significantly higher. Another alternative would be an ESP microcontroller, which provides more computing power in a smaller package with similar costs in comparison with the Arduino. However, the lack of pre-built shields, hence the necessity of using bare components, would pose a more difficult challenge, should anyone replicate this work.

In my first draft, the Arduino MCU would be connected to an EV3's output port. A program on the EV3 would utilize the python-ev3dev motor driver libraries to set a pulse width modulation duty cycle, polarity, and other parameters that set how a motor should run, just like with a regular connection. The PWM signals run through pins 1 and 2 of the EV3's port. These signals would then be translated by the MCU, which would drive the motors with the same PWM duty cycle. However, while theoretically simple, this implementation would be problematic. The PWM pins of the EV3 can run up to 8+ volts of voltage and 1A of peak current (500mA continuously) [27]. This much voltage and current would likely damage the microcontroller. Furthermore, this implementation would do little to improve on the current state of the art, as similar functionality could be achieved by stacking multiple PF motor connectors and connecting them to an EV3 port. However,

it should be mentioned that this method might be harmful to the EV3, as running several motors from a single port could draw too much current and damage the EV3's components.

5.2 Connection via the I2C Protocol

Having realized the risks of connecting an Arduino microcontroller directly to EV3's PWM pins, I sought a different way to connect the two devices. Mindstorms EV3' AM1808 SoC includes controllers for I2C and UART communication [27]. For this implementation I have chosen to use the I2C protocol, a brief description of which can be found in Section 4.4.1. This protocol can be used via the `I2CDevice` class of the Pybricks library [49], the I2CPort interface of the leJOS EV3 API, or the I2C Block in LabView software, developed by Dexter Industries [47]. This is the reason I chose the I2C protocol, since at the time I had not yet implemented UART communication blocks in the EV3 software. Another reason for using I2C over UART is its simplicity. As it is a synchronous protocol, there is no need to set up matching communication speeds for both devices.

According to the AM1808 microprocessor datasheet, the EV3 utilizes 3.3V logic that is 5V tolerant [45]. Communicating via I2C with an Arduino Uno using 5V logic did not pose any problems, however, direct UART communication was not functional, likely due to the more nuanced nature of the UART Sensor Protocol, which includes identification processes. After I connected a 6N137A optocoupler into the circuit to separate the logic voltage levels, the successfully EV3 responded to the handshake data with an ACK byte. However, I was not able to complete the communication circuit since I am not in possession of an optocoupler with a 3.3V input voltage rating that would enable the Arduino to read the data coming from the EV3. An ESP8266 microcontroller with its 3.3V logic is compatible with the EV3 and direct UART communication works without any issues.

5.2.1 First Prototype

As my first prototype, I devised a connection between the EV3 and the Arduino, which would control a single DC motor. The Arduino would read I2C messages sent by the EV3 and control the motor connected directly to the MCU. Unlike the first draft, this would control the motor safely, without the risk of damaging the microcontroller's components. In this prototype, the Arduino and the motor would be powered either by pin 4 of the EV3's port, which provides 5 volts, or a more powerful external battery could be connected, such as the Power Functions battery box. Pins 3 (ground), 5 (Clock) and 6 (Data) would also be connected to allow for sending data to the MCU. The schema of the connection can be seen in Figure 5.1. This model, however, has its drawbacks. Due to the design of the Arduino MCU, only a maximum of 5 volts can be supplied to the motor, no matter the voltage of the power source. As Power Functions are made to work with up to 9 volts, this would severely limit the motor's performance. Consequently, just like with the very first design, this schema would be too complicated to control just one motor and would provide no real benefits compared to the simple direct connection between the EV3 and a motor. Despite its shortcomings, though, this prototype would prove to be the starting point that more advanced controllers would be based on.

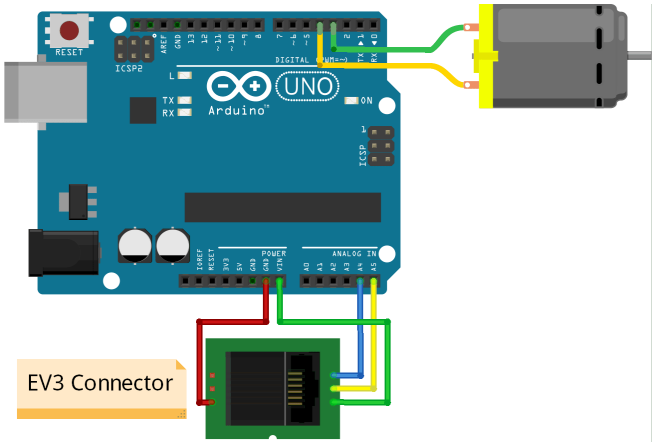

Figure 5.1: Connection schema of the first prototype

5.2.2 Prototype with a Motor Driver

This prototype allows for the control of up to four DC motors. They will be driven using an Arduino motor shield designed by Adafruit Industries, with Texas Instruments L293D drivers. This shield contains two of these drivers and can power up to four DC motors, each with up to 600mA of continuous power draw [46]. While any pulse width modulated motors are compatible with this prototype, I have used a set of Power Functions motors due to the lower price and higher availability of these devices compared to other LEGO motors. To provide sufficient power, I will connect a 9-volt Power Functions battery box as the power source, which will power the shield, the motors, and the Arduino itself. I connected the microcontroller to the EV3 by connecting pins 6 (Data), 5 (Clock), and 3 (Ground). It is unnecessary to connect pin 4 (Power), as the battery box provides power for the Arduino as well. The schema of the connection can be seen in Figure 5.2. As is standard with Arduino and I2C connections, the data pin is connected to the A4 pin, and the clock is connected to pin A5.

The Arduino is then configured as a slave in the I2C connection and receives bytes from the EV3. As I will only be using simple DC motors in this prototype, there is no need to send feedback data from the Arduino back to the EV3. Despite controlling only the output and not receiving any messages back, this connection is created on an input port, since EV3's output ports lack I2C and UART interfaces [27].

5.2.3 Prototype Communication Protocol

I have created a communication protocol (Figure 5.3), which can set up the motors' configurations in 2 bytes. After receiving a message, it sets the motors to run in the set mode indefinitely, until the next message is received, which is when the MCU changes the state of the motors accordingly. The first byte turns the whole set of motors off if the MSB is set to zero, the other 7 bits are used for setting the motors' PWM duty cycle. By setting the bits of the second byte, I can turn individual motors on or off and set their polarity. The advantage of this protocol is its simplicity, as there is no need for sending different bytes for every motor, the whole set can be controlled by just one message. The drawback, however, is that there is only one duty cycle for all the motors, which means that they can only be turned off or run at the same duty cycle as all the other motors. This implementation

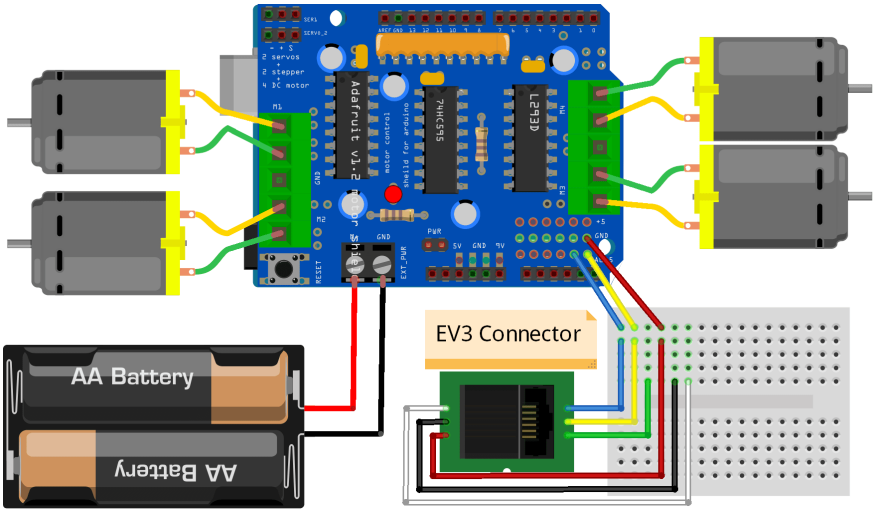

Figure 5.2: EV3, Arduino and motor shield connection diagram

is sufficient in applications like remote-controlled cars, but implementing more complex projects using this protocol might turn out problematic.

Byte 1							
On/Off	PWM Duty Cycle						
7	6	5	4	3	2	1	0

Byte 2							
Motor 4		Motor 3		Motor 2		Motor 1	
Polarity	On/Off	Polarity	On/Off	Polarity	On/Off	Polarity	On/Off
7	6	5	4	3	2	1	0

Figure 5.3: I2C Motor Control Protocol v1

5.2.4 Enhanced Communication Protocol

In the second version of the protocol, I aimed to mitigate the shortcomings of the first version, primarily the lack of ability to set a different duty cycle for each motor. I modified the protocol significantly. Instead of one message controlling all motors, a single message only controls one motor. The length of the message is one byte, the bits 6 and 7 determine, which motor will be set, bit 5 turns the motor on or off, bit 4 sets the polarity, and bits 0-3 set the duty cycle. If bit 5 is set to 0 (motor off), bits 0-4 will not influence the state of the motor. Having a 4-bit resolution of the duty cycle is a downgrade compared to the 7-bit resolution of the first version of the protocol, however, a resolution this high is unnecessary in this use case. I consider 16 duty cycle values plentiful, especially when taking into consideration the fact that the original LEGO Power Function controllers only had 8 values available. It is now not possible to run or stop all motors with one command,

each motor needs to be set with one message, though the broadened options of the motors' configuration should offset this drawback. The protocol can be seen in more detail in Figure 5.4.

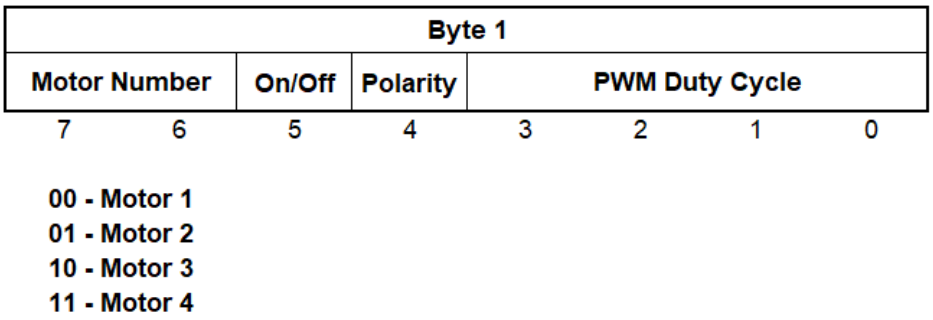

Figure 5.4: I2C Motor Control Protocol v2

5.2.5 EV3 and Arduino Code Implementation

Utilizing the Pybricks' I2CDevice class, I was able to set up the EV3 as a master device in the I2C communication and send data to Arduino. I have created several functions to make motor control more convenient for the end user.

- **setMotor(set_motor: int, set_on: bool, set_polarity: int, set_duty_cycle: int)** - A function that constructs a data byte for the MCU based on the input parameters, which include the controlled motor's index, value determining the motor's on/off state, the motor's rotation direction (polarity) and its duty cycle. The program will raise an exception if incorrect parameters are set, such as an invalid input for the motor duty cycle.
- **runAllMotors(set_duty_cycle: int)** - A function that runs all motors at a single set duty cycle value. This function mimics the functionality of the first version of the protocol.
- **stopAllMotors()** - A function that sends halt commands to all four motors.

When the program of the EV3 executes a motor control command, these functions construct a message based on the parameters provided. This message is then sent to the Arduino via I2C. In the code for the Arduino microcontroller, I have set up the device to listen for incoming messages. When it receives a byte, it deconstructs it using the `setMotor()` function to gather data and set a motor to its new state. To control the motors, I have used the *Adafruit Motor Shield Library V1*, from which I used the `AF_DCMotor` class to define the motors, as well as the `setSpeed()` and `run()` functions to set, run and stop motors. A sequence diagram depicting the communication between the EV3 and the microcontroller in a program example can be seen in Figure 5.5.

I have successfully managed to implement this version of the prototype. All four motors can be controlled either separately or together, using the Micropython functions I have written. It should be noted that with all four motors running simultaneously, their speed drops noticeably. This is a result of the Power Functions battery box not being able to provide sufficient power for all four motors and this problem can be easily fixed by

connecting a more powerful source to the prototype since the motor driver shield can be powered by a source with a voltage of up to 15 volts. An image of the working proof of concept is shown in Figure 5.6.

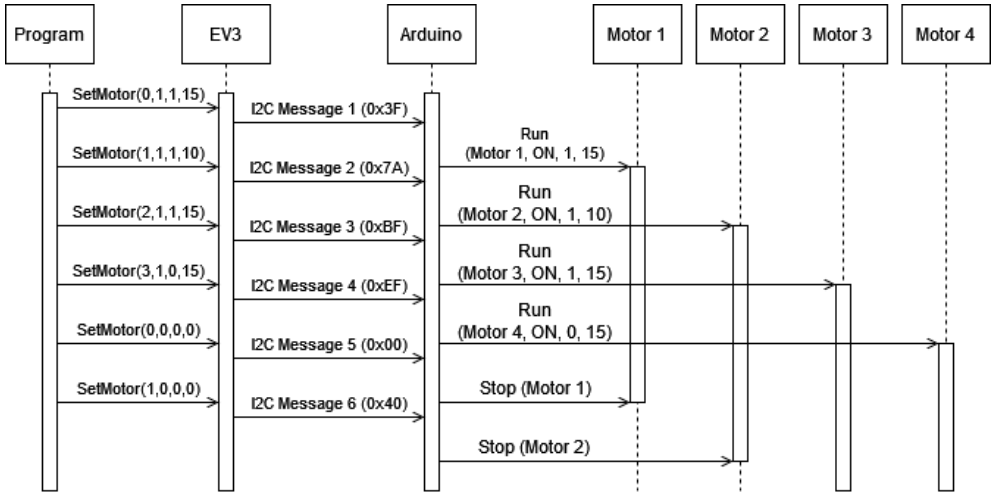

Figure 5.5: Sequence diagram of a program example

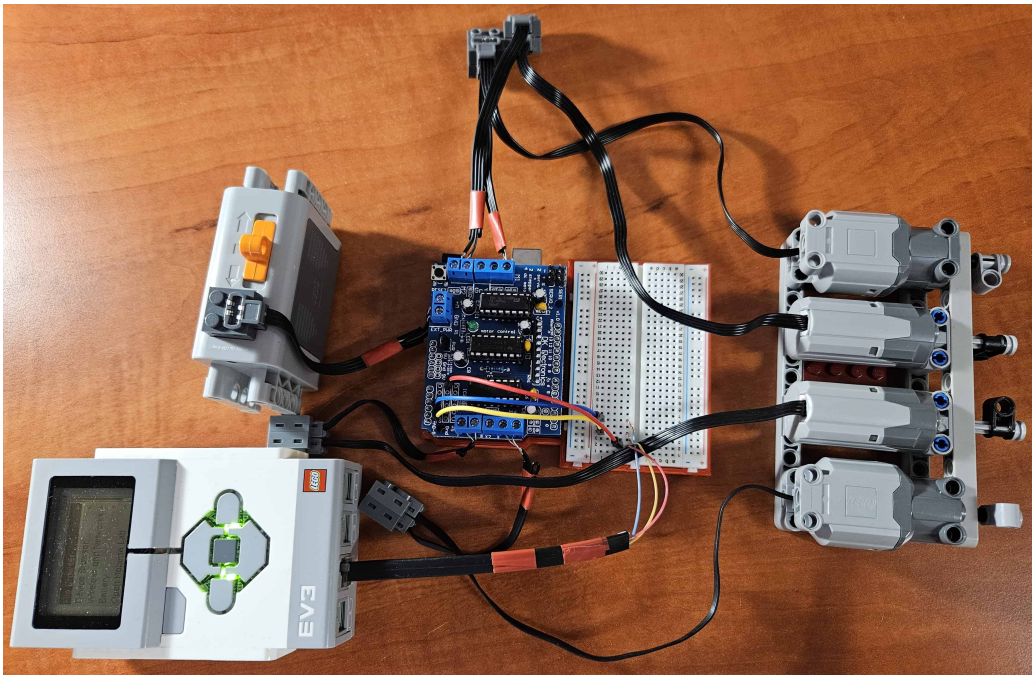

Figure 5.6: Implementation of the Arduino prototype

Chapter 6

Streamlining the Prototype

After creating a motor driver setup using pre-fabricated parts, I have set out to streamline the design of the prototype. While functionally sound, the concept is too large to be implemented into LEGO creations with ease. The Adafruit driver shield has to conform to the size of the rather large Arduino Uno development board, other than that, it contains a number of pin extensions and replications. To design my own board, I used the Adafruit shield datasheet as a reference, simplifying its design and removing any components unused in this project. Instead of connecting to a microcontroller development board, I will integrate an ESP12-F microcontroller directly onto the board, thus further reducing the overall size of the motor driver. The first step was to create a breadboard concept of the board, as seen in Figure 6.1. After testing and confirming its functionality, I began the design process of a PCB (printed circuit board) that integrates the driver components.

6.1 Schematic of the Board

I used KiCAD 8.0 software to lay out a schematic of the board, which can be seen in Figure 6.2. The ESP connects to the 74HC595 shift register with four GPIO pins, another four GPIO pins are connected to the channel enable pins of the two L293D drivers. The last two GPIO pins are connected to two pins of the integrated 6P6C connector and will be used to communicate with the EV3. The scheme also contains pin holes for every of the ESP12-F's pins, this has been done with the purpose of making the programming, debugging, or adjusting the connections easier, should such circumstances arise. An AMS1117 voltage regulator is in place to reduce the voltage of the battery down to 3.3 volts, powering the ESP and shift register safely. Both motor drivers use capacitors for input filtering, the voltage regulator's output also connects to a capacitor to ensure its stability. The motors and power source connect to the board via screw terminals. The schematic of the board can be seen in Figure B.1.

6.2 Designing the PCB

Using the KiCAD PCB Editor, I created the design of the PCB. The components were positioned with maximizing the use of space in mind. The design consists of two layers, with all components being top-mounted. For capacitors, resistors, and the status LED I chose to implement SMD footprints of size 1206 instead of designing the board for through-hole components. This too was done in pursuit of maximizing size efficiency. Components that

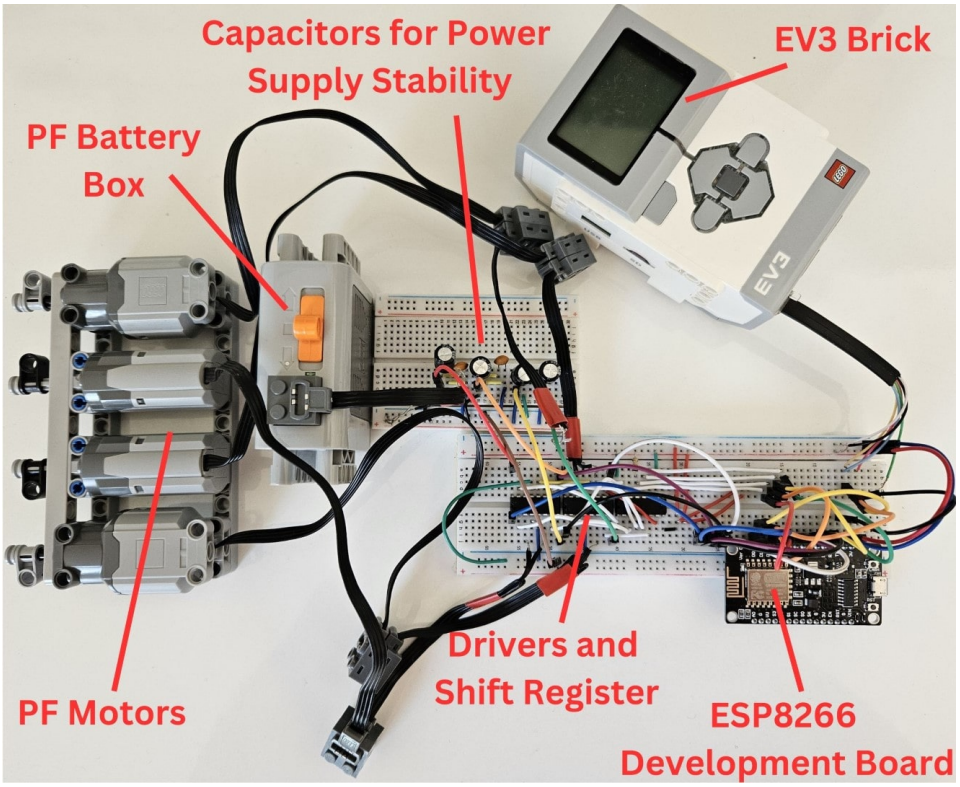

Figure 6.1: Breadboard scheme of the second prototype iteration

utilize 9 volts are connected via 0.5mm wide tracks, while 3.3V components use 0.2mm tracks.

To prevent signal interference and improve the heat dissipation capabilities of the PCB, a copper ground plane is present on both layers of the board. To complement this, I have adjusted the design in a way that minimizes the length and number of tracks on the bottom layer of the PCB. This way, the bottom ground plane is as large and consistent as possible, which further improves heat dissipation and eliminates ground plane islands, which need to be joined to the rest of the plane with a track connection. A similar plane connected to the output of the voltage regulator is present on the top layer, as energy transfer losses in the regulator generate heat as well. If we consider the maximum power draw from the microcontroller, shift register, and status LED, which are 170mA, 70mA, and 15mA respectively, and a 9-volt power source, the power loss comes in at approximately

$$P = (U_{IN} - U_{OUT}) * I_{OUT} = (9V - 3.3V) * 255mA = 1.4535W.$$

The area of this plane is $67.83mm^2$. The intricacies of the design can be seen in Figure 6.2. The size of the PCB is 71.5 millimeters in length and 51 millimeters in width. While it is slightly larger in length compared to the 69mm-long Adafruit driver shield, it is narrower by two millimeters and is a much more compact device vertically, since it already contains an integrated MCU, unlike the Adafruit product. A picture of the finished board with soldered components is present in Figure 6.3.

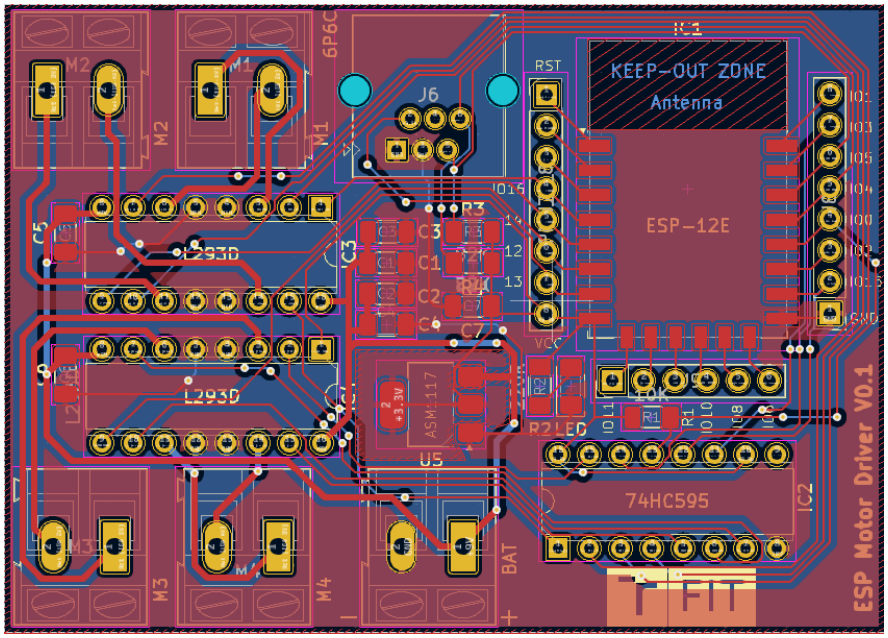

Figure 6.2: Design of the PCB in KiCAD 8.0

6.3 Functionality of the Driver Board

The prototype uses the shift register to access a total of 8 input pins between the two L293D drivers while using just four of the microcontroller's GPIO pins. The functionality is carried out via the *Adafruit Motor Shield Library V1* library. This library also generates PWM signals that are sent to the drivers' output-enabling pins. The library is originally made for the Arduino Uno and makes use of its three timer modules to generate square waves [17]. To make the library compatible with the ESP microcontroller, I slightly modified it, now using the **AnalogWrite()** function to generate the PWM signals.

While the prototype from Subsection 5.2.2 used the I2C communication protocol, where the EV3 acts as a master device and the Arduino as a slave, such setup is not possible in this configuration, as the ESP-12F lacks the implementation of a slave mode. As such, it is necessary for the microcontroller to communicate via the LEGO UART Sensor Protocol, described in Subsection 4.4.3. Using this protocol is a more demanding task compared to using the I2C protocol, as the MCU must send data to the EV3 to keep the connection alive. Though the communication protocol states that the connected device must send data in an interval of 1-100 milliseconds, as well as when prompted by the EV3 with a NACK byte, I found out that the connection works without any issues even when data is only sent after NACK prompt [28]. While it would be satisfactory to only send an arbitrary message of a single byte without any meaningful payload, I programmed the MCU to send back a message with a duty cycle of a single motor at a time, iterating over all motors in a loop. This way, their state can be checked using the **MotorStateCheck** EV3-G Block, a more detailed description of which can be found in Section 8.3. This can be useful in a situation where the duty cycles of motors change rapidly and their current state might not be clear.

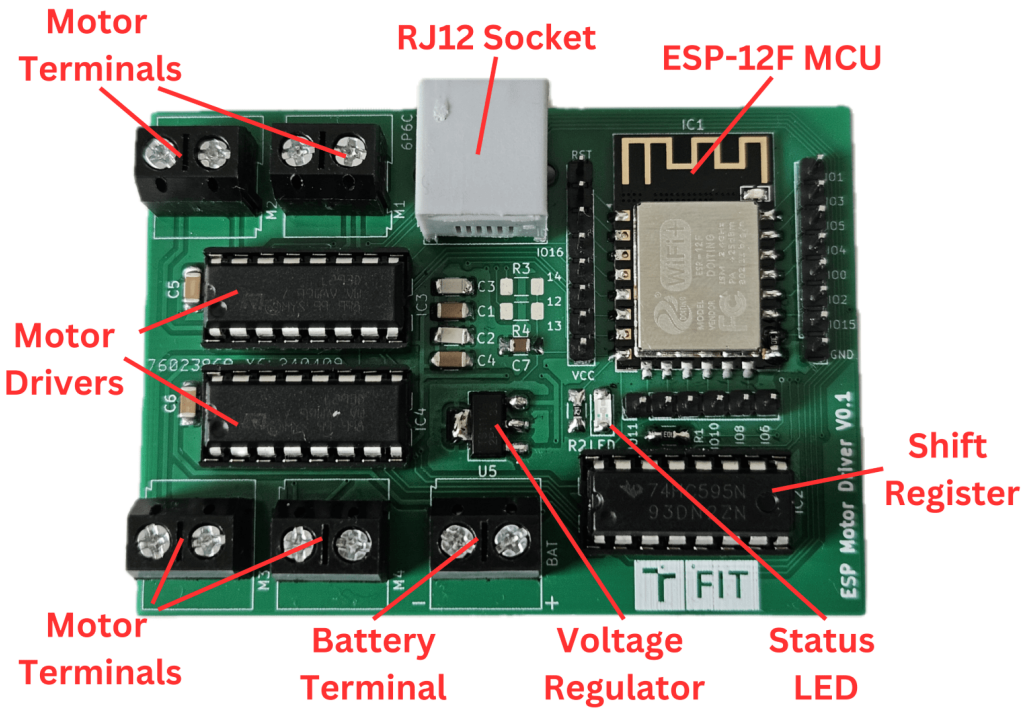

Figure 6.3: Motor Driver PCB with soldered components

6.4 Assembly of the Prototype

After soldering all the components onto the PCB, I used an ESP-Prog development board manufactured by Espressif to flash the program onto the onboard ESP-12F microcontroller, shown in Figure 6.4, as the MCU itself doesn't contain a USB driver. The initial attempts to program the MCU failed as the connection timed out. After analyzing the problem and studying the datasheet of the NodeMCU ESP8266 development board[1]I used in my breadboard prototype, I found out that this microcontroller does not include internal pull-up and pull-down resistors for its pins. This way, the ESP could not be set into the boot mode. To fix this, I soldered pull-up resistors to GPIO pins 2, RST, and ENABLE, and a pull-down resistor to GPIO pin 15. A pull-up resistor is necessary on pin 0 as well, however, it is already present on the board as pin 13 of the shift register is connected to the ESP12's pin 0 and requires a pull-up resistor.

To power the board and the motors, I connected a KAVAN 3-cell Lithium-Polymer rechargeable battery, originally meant for use in remote control vehicles. At 850mAh and 9,4Wh, its capacity is lower compared to six AA batteries, however, there are several advantages to this setup. Where the Power Functions Battery Box struggles to power four motors concurrently and its performance largely depends on the quality of AA batteries, the accumulator with its rating of 11.1 volts and 34 amperes can provide more than sufficient power for all the components of the PCB. At 60mm x 30mm x 20.5mm and 67 grams, it is also significantly more compact and lightweight compared to the battery box, measuring 85mm x 61.5mm x 30mm and weighing 131 grams including alkaline batteries. This allows me to greatly reduce the overall footprint of the prototype.

[1] https://github.com/nodemcu/nodemcu-devkit-v1.0/blob/master/NODEMCU_DEVKIT_V1.0.PDF

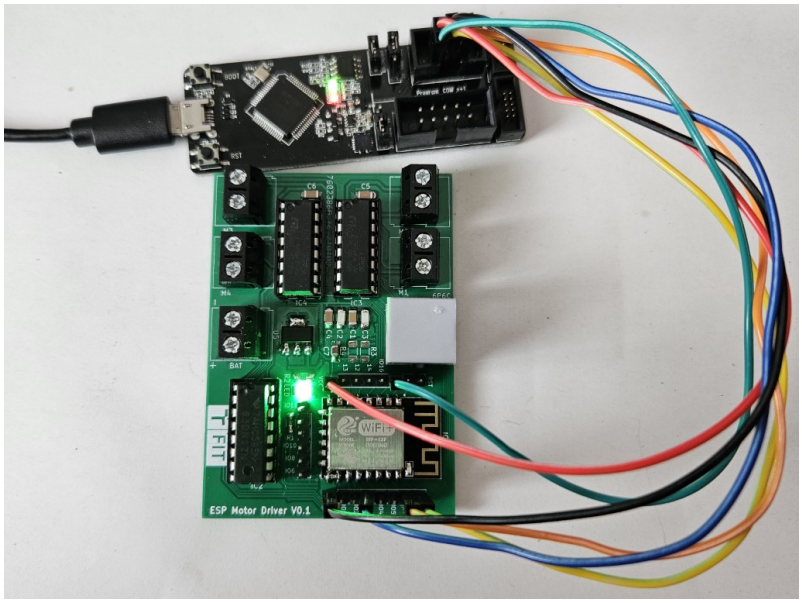

Figure 6.4: Programming the ESP12-F microcontroller using Espressif ESP-Prog

To protect the battery from overly discharging, I used the A0 ADC pin of the MCU to measure voltage. The nominal voltage of a lithium-polymer battery cell is 3.7V, however, the maximum voltage can be up to 4.2V, totaling at 12.6V for the 3-cell battery pack. To make sure that the battery pack does not get damaged, I set the cut-off voltage of a cell at 3.6V, or 10.8V for the whole pack. As the maximum input voltage rating of this pin is 1V, I made use of a voltage divider. Using the formula to calculate the output voltage of a voltage divider, which is $V_{OUT} = V_{IN} * \frac{R_2}{R_1+R_2}$, I calculated the ideal resistance rating ratio of the resistors, 120,000 ohm for resistor R_1 and 10,000 ohm for resistor R_2. In this configuration, the output value of 1V is present with 13V at the input. If the voltage drops below the set value, the microcontroller enters a deep sleep mode, which stops all motors.

The idle consumption of the prototype when no motors are running is 101mA. When the voltage drop past the threshold is detected and the microcontroller enters deep sleep, the consumption drops to 83mA. This value is still rather high, the reason being that the status LED and the motor drivers are powered by the battery, independent of the ESP's state. However, the main function of the ESP entering the deep sleep mode is to stop all the motors, notifying the user of the need to charge the battery pack. The ESP resumes its standard operation after turning the prototype off and back on.

The need for the changes to the original schematic described in this section was discovered during the assembly of the prototype, hence they had to be implemented by hand using mostly THT (Through Hole Technology) components, apart from the pull-down resistor on GPIO15, which is located next to the GND pin of the microcontroller, allowing me to solder an SMD resistor between these two pins. Figure 6.5 displays the bottom side of the board with the modifications applied.

6.5 Designing an Enclosure for the Prototype

I decided to create a 3D printed enclosure for all the components of my prototype. This would allow me to create a custom design that has no limitations on the shape or the

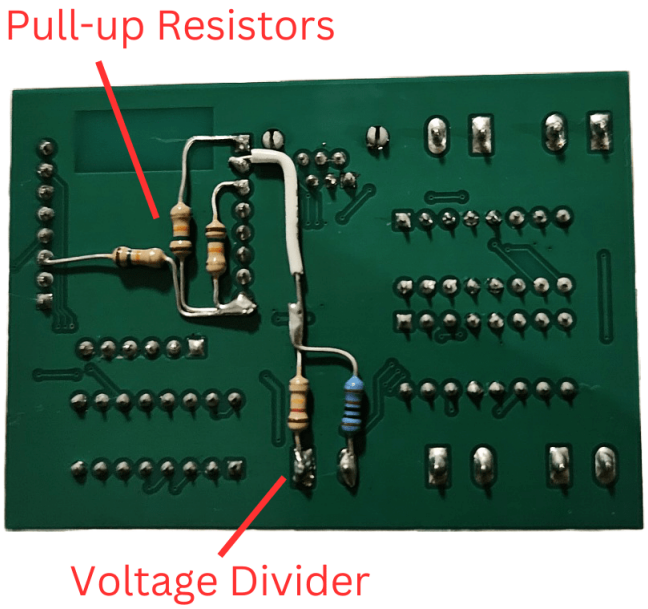

Figure 6.5: Bottom side of the PCB with implemented corrections to the design

specifics of the design. Using Dassault Systèmes' SOLIDWORKS 3D CAD design software, I created a design of the enclosure. It consists of three parts, the bottom segment housing the battery pack, the middle part housing the PCB and the switch for battery power, and the top for connecting motors to the Power Functions connectors. The bottom part includes ten LEGO Technic pin holes on both of its longer sides. It also features a cutout for the battery pack's charging cable and connector. The middle segment has cutouts for the RJ12 connector and a rocker switch for battery power, as well as opening slats on the side of motor drivers, allowing for their passive cooling as these are the components that generate the most heat in this prototype. The top of the enclosure includes four openings for the Power Functions connectors and two rows of 8 by 2 standard LEGO pins at its edges. These, together with the pin holes at the bottom allow the enclosure to be securely connected to LEGO models. The three segments are held together by four screws. I used a cubic infill pattern for the inner parts of the design (Figure 6.6), where only 10% of the volume is filled by the 3D printing filament, reducing the overall weight, as well as saving on the material. The material of the filament used for this prototype is polylactic acid, or PLA for short. Made out of corn starch, this bioplastic has a low environmental impact. The design of the enclosure is shown in Figures 6.7, and 6.8. A picture of the printed enclosure containing the driver components can be seen in Figure 6.9.

6.6 Implementation of the Microcontroller's Code

The code behind the motor driver is built on the Arduino framework. The UART communication is implemented via the *SoftwareSerial* library, the reasoning behind this is elaborated in Section 8.2. Until an ACK byte is received by the microntroller, the ESP repeatedly initiates the handshake procedure with the EV3. Firstly, a break condition is set where the TX pin of the SoftwareSerial communication is pinned to LOW for 500 milliseconds. After

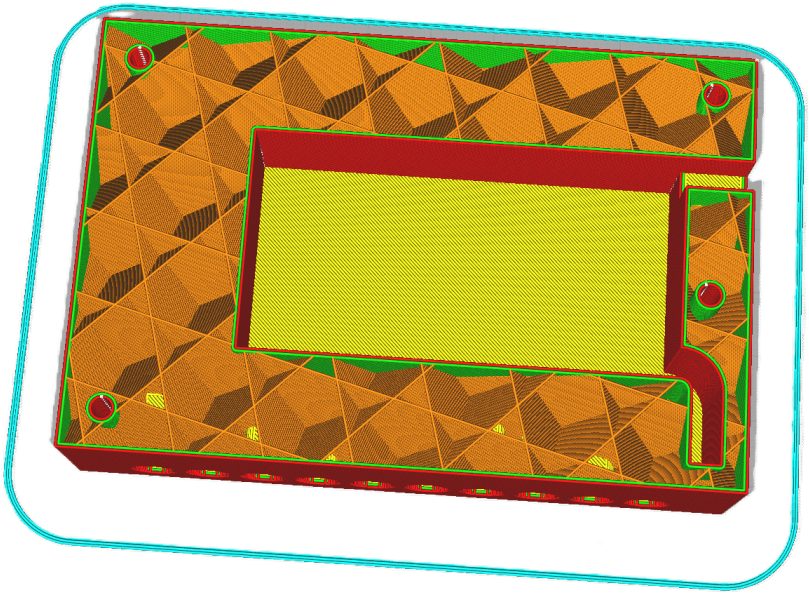

Figure 6.6: Cubic infill inside of the enclosure

this act is finished, the pin has to be set to HIGH for at least one millisecond. Afterwards, the transmission of the handshake data begins.

As described in Subsection 4.4.3, the data is sent in a type - payload - checksum format. TYPE, MODES, FORMAT and optionally SPEED messages are transmitted, ended by an ACK byte (0x04). If the EV3 returns the acknowledgment, data exchange between the two devices begins, setting the agreed-upon baud rate if provided. The ESP replies to the NACK messages with duty cycles for the four motors, with one byte containing a duty cycle of a single motor. This process is further described in Section 8.3. If the user prefers, the microcontroller can be set to also send this data on its own in intervals ranging from 1ms to 100ms. As per the LEGO UART Sensor Protocol, the EV3 sends a WRITE message containing the length of the incoming data. After receiving this message, the ESP determines the length of the data. After receiving a payload byte, the `setMotor()` function is called. Here, the message is parsed and the state, polarity, and duty cycle of a specified motor are set. The `run()` function of the Adafruit motor library to run or release the motor. At the beginning of every data transition from the EV3, a message with a payload length of one and an 0x11 byte as the payload is sent. The purpose of this message is unknown, as it is not described in the protocol. This message would set the polarity of motor 1 to forward and set the duty cycle to 1, but would keep the motor off. As such, this payload sent by a user makes little sense, and thus I can omit this value from being interpreted by the `setMotor()` function. In a case where 1000 milliseconds have passed since the last NACK message, the ESP resets and initializes the handshake again.

Lastly, every two seconds, the battery voltage is measured via the ADC pin of the ESP. The average of the measurements is calculated using a moving average algorithm with a window of ten measurements. If the average drops below the threshold, the ESP enters a deep sleep state. The moving average algorithm ensures that the ESP does not enter the deep sleep state after a single voltage drop caused by the motors starting.

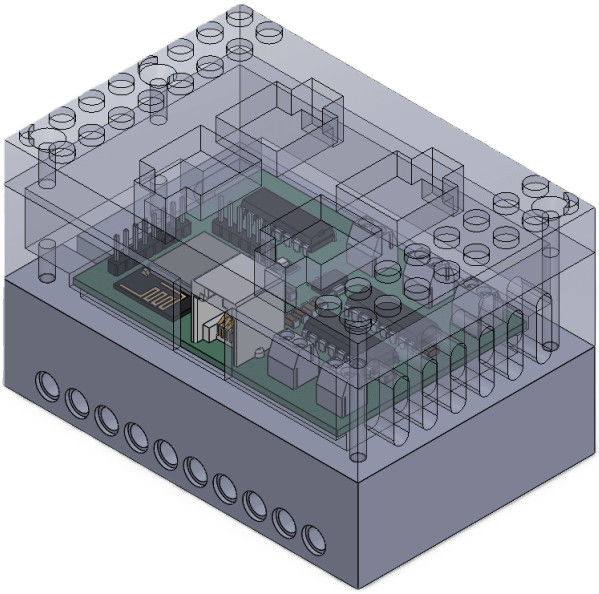

Figure 6.7: Design of the enclosure shown from the profile

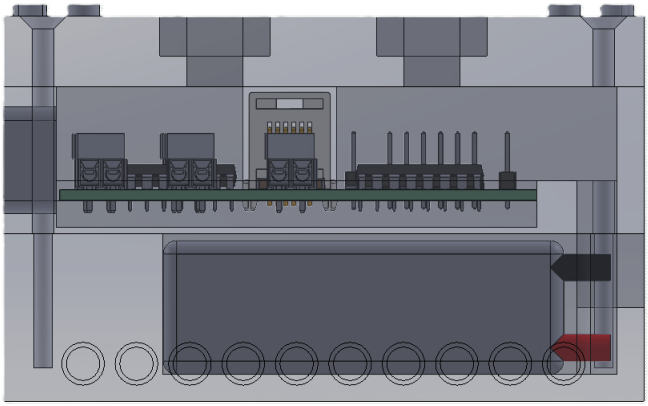

Figure 6.8: Design of the enclosure shown from the side

6.7 Flaws of the Design

During the prototype testing, I discovered a major flaw in the design of the PCB. According to the datasheet of the L293D motor driver, pin 16 acts as a power supply for the inner logic of the driver, with acceptable voltage values of up to 7V [46]. However, due to my design error, this pin is supplied in the same way as the power for the motors on pin 8, connected directly to the power source without any voltage regulation. This means that during the testing of the driver, up to 12.6 volts were supplied to the logic of the drivers. Although these drivers remain functional after testing, this implementation is not acceptable, as drivers can suffer damage at any time. To rectify this error, I decided to design a revised version of the driver board. This board will also include missing components mentioned in Section 6.4, such as pull-up resistors or a voltage divider for battery capacity measurements.

Figure 6.9: 3D printed enclosure for the motor driver

Chapter 7

Implementing a Corrected and Improved Custom Motor Driver

Taking into account the mistakes made while designing the first version of the PCB, as well as the experience gained during the design and implementation process, I set out to improve the characteristics of the driver in its second iteration.

7.1 Design of the Improved Board

I modified the schematic of the board (Figure C.1) to reflect the lack of necessary components in the first version. I added a voltage divider and pull-up resistors. Initially, I planned to add a second voltage regulator to the board to provide a safe logic supply to the motor drivers. However, adding another voltage level would make for a convoluted design. Additionally, adding several components to a complete design with tightly packed parts would be largely difficult. For this reason, I chose to create a completely new design, while adhering to the same principles as with the first board. To avoid the need for two logic voltage levels on the board, I looked for a motor driver the logic of which is compatible with the microcontroller used. I chose to use a **Microchip TC4427COA** MOSFET driver. While this driver can only power and control a single motor, there are several advantages to using this device instead of the L293D drivers. Utilizing TC4427 drivers, with 8 pins per chip in a DIP/SOIC format, allows for two of these drivers to occupy a footprint similar to that of a single L293D. This maintains spatial efficiency on the PCB layout. The peak output current of TC4427 is greater at 1.5A compared to 1.2A of a single channel in the L293D [37, 46]. Although the L293D utilizes two PWM input pins and an enable pin, the TC4427 does not use an enable pin. This greatly reduces the amount of I/0 needed for controlling the drivers, however, the ESP-12E's GPIO options would still not be sufficient, hence why I decided to implement an ESP32-WROOM32E microcontroller instead. This MCU is significantly more powerful and capable with its two-core processor and broader GPIO, while only having a marginally larger footprint and a slightly higher price. Using this MCU with plenty of GPIO pins made the 74HC595 shift register obsolete in this schematic. Removing this component allows me to pursue the maximum efficiency of the design even further.

The board (Figure 7.1) now has a revised component placement. As the board will be enclosed and not used on its own, I did not need to place the motor connecting terminals evenly, instead, they are distributed in a way that minimizes space usage. Another resolved issue lies within the voltage regulator. The area of its thermal relief pad is now almost

twice the size at $126.5mm^2$. The parameters of the pad were also incorrect. As seen in Figure 6.2, the width of the spoke connecting to the pads is just 0.5mm, hindering the ability to efficiently dissipate heat [40]. This is fixed in the second prototype with much larger, 2mm-wide spokes. To be able to withstand the peak current, the routes leading to the driver and the motors are 0.52mm wide. This value was calculated using KiCAD Calculator Tools. The status LED is now connected to a GPIO pin of the microcontroller instead of being constantly powered by the voltage regulator's output. This gives me the option to use the LED to indicate the state of the driver board instead of just signaling whether the driver is powered or not.

The dimensions of this PCB are 69.00mm x 39.00mm. This makes the board 2.5mm shorter and 12mm narrower, decreasing the area of the board by 26.2%. An image of the completed revised PCB with component descriptions is shown in Figure 7.2

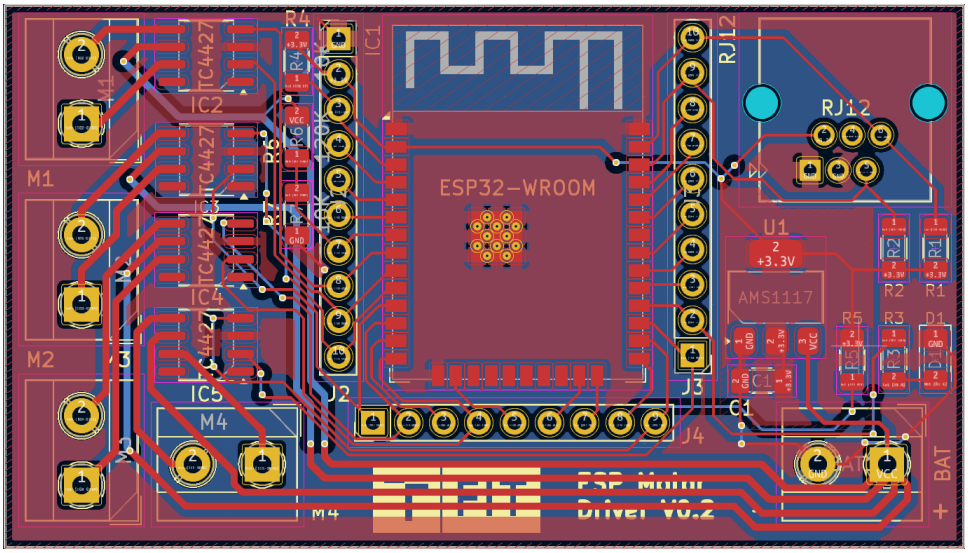

Figure 7.1: Design of the second driver board iteration in KiCAD 8.0

7.2 Program Modifications

To make the program compatible with the ESP32 microcontroller, modifications were necessary. As the shift register is now not present on the driver board, I can omit the usage of the Adafruit motor driver library and use my own classes and functions to control the motors. The *AF_DCMotor* class is replaced by the *Motor* class. Although the class' methods carry the same name, the implementation of the methods differs from the ESP-12F code variant. While the pulse width modulation functionality was carried out by the (PinMode) and `AnalogWrite()` functions when the ESP-12F was used, the ESP32 requires several functions to initialize and generate PWM signals.

A PWM channel is initialized by the `ledcSetup()` function. I set the PWM frequency to 1200Hz as it is the frequency used by the original Power Functions control devices [8], although pulse width modulation for the motors works without any issues when using higher frequencies, such as 5kHz as well. As per the communication protocol between the EV3 and the driver, I set the resolution of the PWM to four bits, allowing for 16 duty cycle values. A pin is attached to the PWM channel in the `ledcAttachPin()` function. The

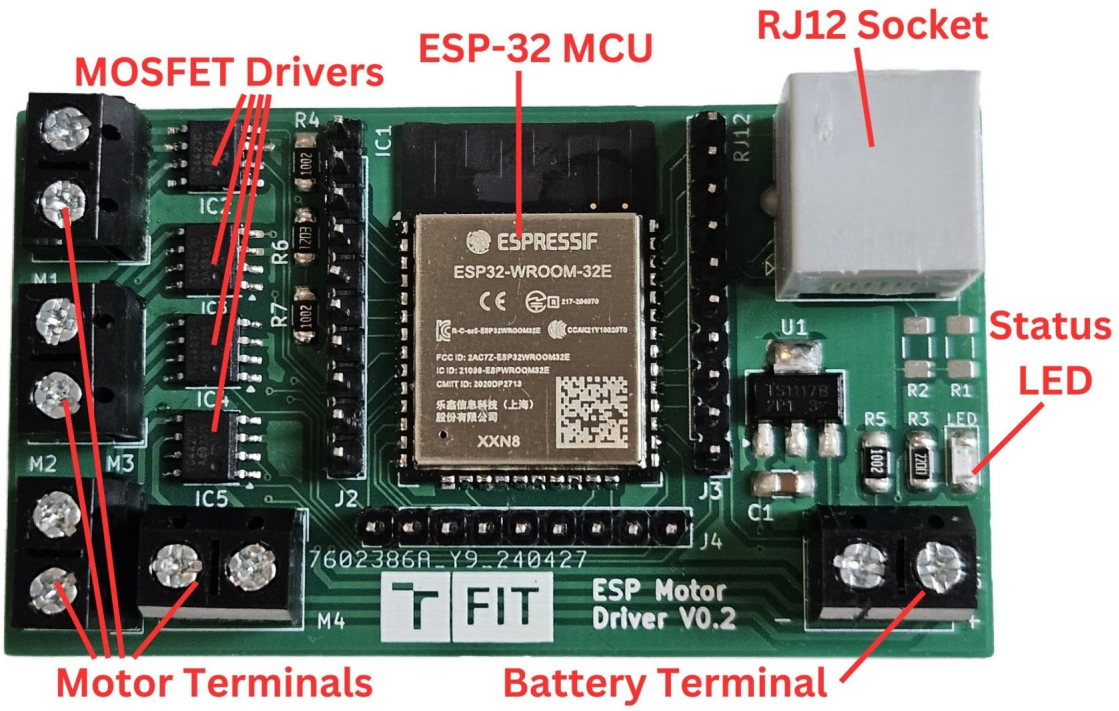

Figure 7.2: Revised PCB with soldered components

motor control code utilizes 8 of the 16 available PWM channels. These functions are called by the constructor of the *DC_Motor* class in the `Init_PWMX()` functions, where X is the number of the motor. To generate the PWM signals, I utilize the `ledcWrite()` function, providing the PWM signals to one of the driver's input pins, depending on the set polarity of the motor.

Connecting the status LED to the GPIO of the microcontroller allowed me to broaden its functionality. The LED operates in three modes, it flashes slowly while the driver is trying to connect to the EV3, and it is lit with a PWM duty cycle of 20% during regular operation. If the voltage reading function determines that the battery voltage is low, the LED flashes rapidly for 10 seconds before the ESP enters deep sleep mode. The pulse width modulation of the LED GPIO pin is bound to PWM channel 9.

7.3 Updated Case Design

For the design of the case for the second iteration of the driver (Figure 7.3), I chose to stay faithful to the design decisions made for the first version of the driver. The layout of the components remains identical, although the smaller driver PCB allowed me to reduce the dimensions of the enclosure. The dimensions of the casing are now 94mm by 48mm by 57.75mm, reducing the width down by 25 percent from the original enclosure's 64mm width. This downsizing resulted in the side walls of the bottom part not being thick enough to accommodate Technic pin holes. Consequently, I had to move these pin holes to the upper part of the assembly, which now contains two rows of 6x2 LEGO pins and four instances of two Technic pin holes, placed at the edges. Additionally, I added LEGO pins the to cutouts for Power Functions connectors to better secure them in place.

The case now consists of two parts instead of three, as I merged the lower and middle parts together to make the assembly easier. In addition, I revised the connection system of the assembly. While the three parts of the first version of the case were held together by four screws, I designed a system of rails for the second iteration, allowing the upper part to slide onto the lower part and hold together solely by friction (Figure 7.4). A picture of the assembled driver is shown in Figure 7.5.

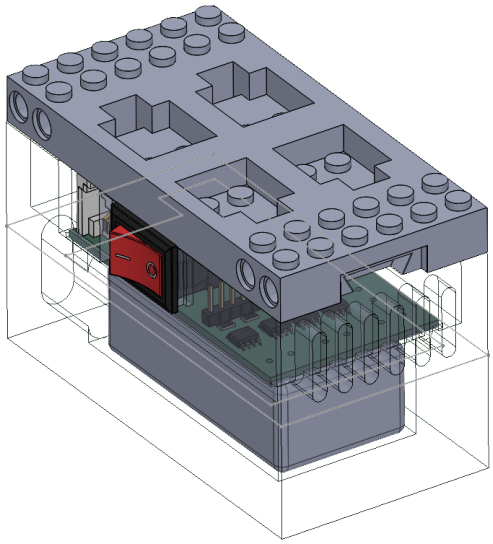

Figure 7.3: Second iteration enclosure design, shown from profile

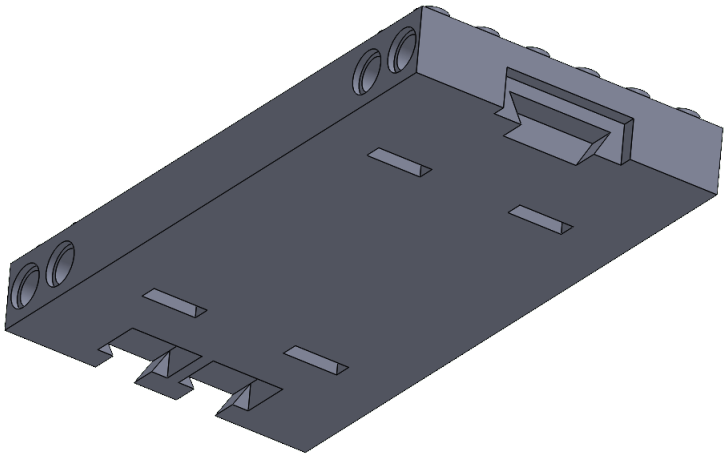

Figure 7.4: Upper part of the enclosure with visible rail system

7.4 Stress Testing the Driver

I used a laboratory-grade adjustable power source to perform driver tests. Firstly, I measured the current draw of the driver in idle and deep sleep states. During the idle state with the status LED powered by a 100% duty cycle, the consumption was 66mA, while the driver drew 8mA in the deep sleep mode. In comparison to the values of the previous iteration,

Figure 7.5: 3D printed enclosure for the second driver iteration

stated in Section 6.4, the idle state consumption decreased by nearly 35%, while the deep sleep current draw dropped by more than 90%. These savings were achieved by connecting the LED to the GPIO instead of being connected directly to the voltage regulator output, being on even during deep sleep mode, and by replacing the outdated L293D drivers with a more efficient solution.

The first problems arose during attempts to stall the motors to generate their peak current. A TC4427 driver was destroyed by the stall current of a motor. This motor was not an original LEGO product, rather it was a non-licensed copy of the Power Functions L motor. Measurements showed that while the stall currents of the original PF M and L motors were 1.03A and 1.53A respectively at 11.1 volts, the unlicensed L motor drew up to 2.3A when stalled. This high current draw permanently damaged the driver, which lacks internal overcurrent protection and only features reverse current protection of up to 0.5A [37]. As external overcurrent protection is rather complex for an implementation of this caliber, it would be ideal to use drivers with internal protection. However, due to time and local market constraints, I was unable to obtain such drivers. For these reasons, I strongly advise to only use licensed LEGO Power Functions M and L motors, whose current characteristics are within the driver specifications. The PF XL motor, with 1.8A of stall current at 9 volts [15], should only be used in applications where a stall condition should not occur.

7.5 Driver Implementation in LEGO Creations

To demonstrate the possibilities that my motor driver design opens up, I created two LEGO models. The first model is a diorama of a wind farm with three turbines (Figure 7.6). My driver allows for simple and convenient control of the motors, their direction, and speed. Additionally, its compact size and included connection pins and holes allow the driver to be integrated into builds with ease. As the driver only uses a single sensor port, up to 16 DC motors can be individually controlled by the EV3, which might prove useful in large LEGO creations with numerous motorized parts.

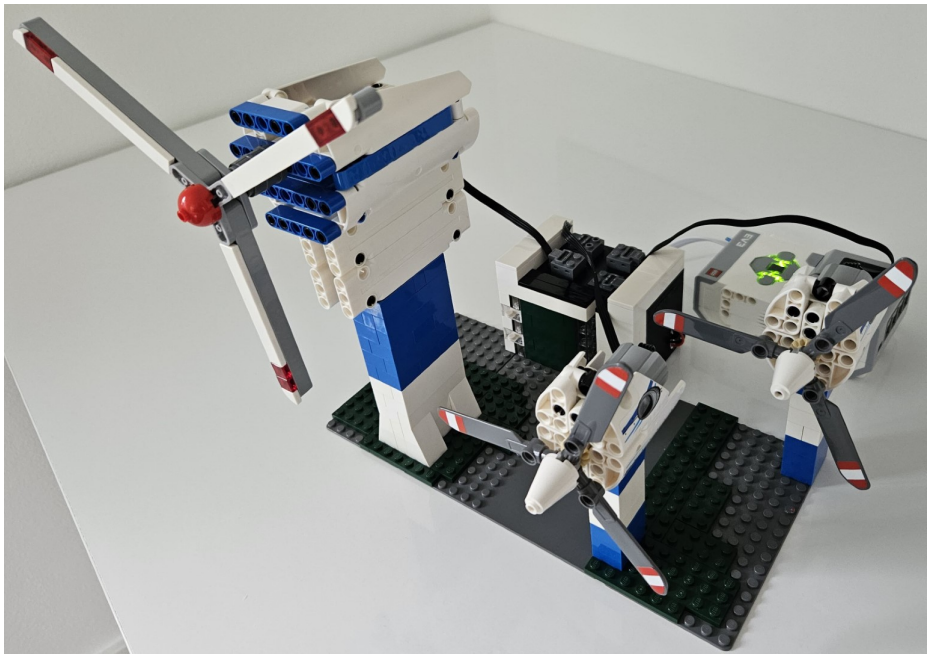

Figure 7.6: Motorized LEGO wind farm diorama

The second model is a chassis of a vehicle with independent four-wheel drive (Figures 7.7 and 7.8). Steering is performed using an EV3 Medium motor with rotational encoders, bringing the total number of motors used to five. As the EV3 only contains four motor ports, this setup would not be possible with the use of EV3 motors while also utilizing conventional steering. By using the driver, four additional EV3 motors can be added while also retaining three free sensor ports, allowing for obstacle avoidance or other functionality. Furthermore, the driver can provide more power to the motors than the EV3 could by itself. This drivetrain configuration opens up possibilities for broadened functionality, such as torque vectoring [54] or tank-like movement. As such, the motor driver could find its use in LEGO models of construction vehicles, tanks, or other tracked vehicles.

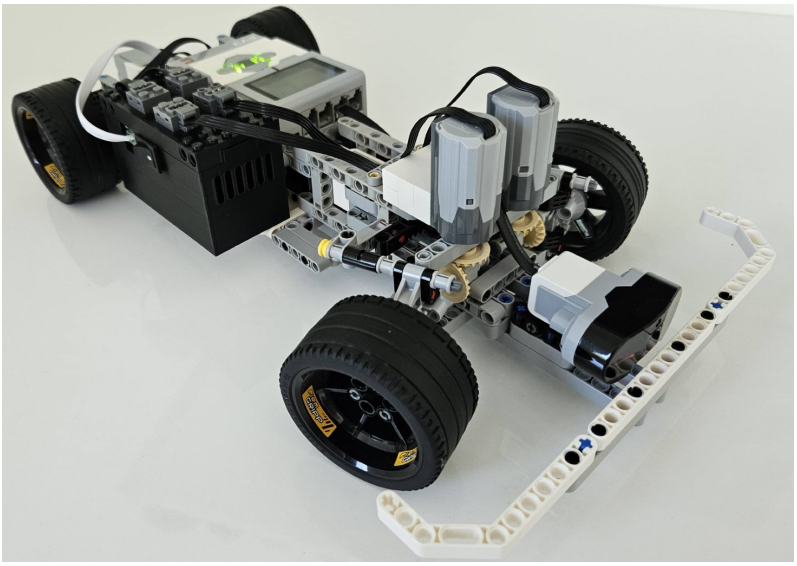

Figure 7.7: Four wheel drive LEGO vehicle chassis, shown from the front

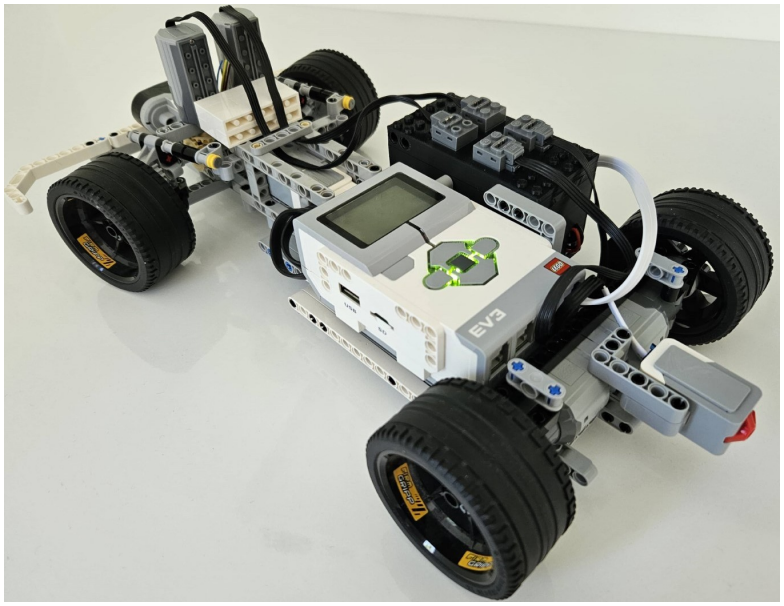

Figure 7.8: Four wheel drive LEGO vehicle chassis, shown from the rear

Chapter 8

Enhancing Communication Capabilities via EV3-G Blocks

The LEGO group releasing a guide for creating custom blocks allowed third-party companies and individuals to create a number of blocks that extend the functionality of the EV3, whether it is new sensors or more advanced functionality within the brick itself. While most sensors and their corresponding blocks utilized the I2C communication protocol to retain compatibility with the NXT brick, few of them used the LEGO UART Sensor Protocol. In addition to the incompatibility with older products, the usage of the protocol itself is more complicated. Developers behind Dexter Industries created a block allowing general I2C communication not reserved to a single sensor, however, I was not able to find such blocks for communicating via UART. I set out to create these blocks since doing so would close the functionality gap between the stock EV3 and the ev3dev software.

8.1 Creating Blocks for the LEGO UART Sensor Protocol and Repairing Existing I2C Blocks

While creating the block mode for reading UART communication was a fairly straightforward process, setting up the communication for the block to read the bytes correctly was not. I used the **PBrickInputReadySI** primitive, used in official LEGO sensor blocks, to execute the message reading process in the EV3. The output of this primitive is of type *Single*, which causes problems in terms of compatibility between the input and the output. Even after numerous tweaks to the setup and the handshake data, I was unable to receive more than a single byte in the EV3 in a data message. Even then, displaying the data proved to be problematic.

The number shown was often in the form of a 2's complement, or a single byte was shown as a 16-bit number made up of the sent byte and the Data Message byte of the protocol [18]. This was partially fixed by sending two bytes of data payload instead of a single byte, which stabilized the output. This solution too, however, is not without issues. The number 191 (0xBF) and numbers 247 to 254 (0xF7 to 0xFE) are not accepted by the EV3, which stops sending the **NACK** (0x02) byte, prompting the connected device to reset and initialize the handshake again. Moreover, the value 255 (0xFF) is displayed as -56 (0xFFC8), once again including the Data Message byte in the value [18]. It should be noted that when initially sending a "compatible number" and incrementing or decrementing

to any of these problematic bytes, they display correctly. The reason behind these problems remains unknown to me.

Although a "grey blob" for writing UART data was not present in the Block Developer Kit, I found it used in a single instance, inside of a reset mode of the Gyro sensor block. Using this, it was possible to send a single byte of data, though sending one data byte at a time is largely inefficient. To send multiple bytes of data at once, I took the approach Dexter Industries did. I created an array of constants and used LabView's **ReplaceArraySubset** function to replace these constants with a variable from the block.

This is where I discovered an issue, when numbers 128 and larger would be capped off and sent as 0x7F, or 127 in decimal. After searching for more information, I found this to be a known issue also present in Dexter's Industries' I2C block[1]. Some believe it to be an issue with the EV3 interpreting bytes as signed integers and presented a workaround by subtracting 256 from numbers larger than 127, making them into a negative number, therefore conforming to the signed data type. This, however, posed an another issue where the number 128 (0x80) could not be interpreted correctly. I refused this interpretation of the problem since a single byte of data in a variable can be sent with no issues and so can an array of constants, therefore I believed the issue to lie within the act of replacing constants of the array with a variable. I tried replacing the **ReplaceArraySubset** function with **InitializeArray** and **BuildArray** functions in an attempt to avoid replacing array elements, but to no avail as the EV3 was unable to run this code and threw an error. After analyzing the code of Dexter Industries', HiTechnic's, and my blocks, I noticed that while I used the **UInt32** data type for variables due to my inspiration by Dexter's code structure, HiTechnic used the **Byte** type in their input variables[2]. Switching to this data type did indeed solve the issue, and values 0 to 255 are sent as unsigned bytes when using UART. A representation of the UART block is present in Figure 8.1. The functions of this block are as follows:

- **UartRead** - Reads a single byte from a connected UART device
- **UartWrite** - Writes a single byte to a connected UART device
- **UartWrite8Bytes** - Writes 8 bytes of data to a connected UART device

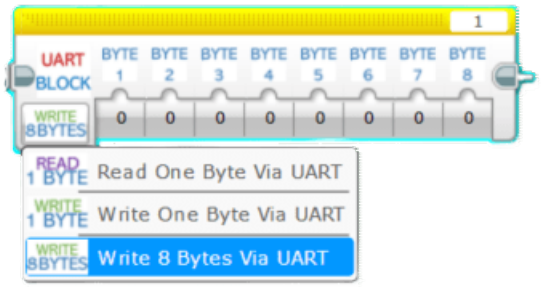

Figure 8.1: UART Control block in the EV3 programming interface

This success with UART blocks motivated me to try to fix Dexter Industries' I2C functionality. As the **PBrickInputSetupI2C** primitive requires the address of the device

[1] https://forum.dexterindustries.com/t/problem-if-i-send-a-value-greater-than-127-using-the-i2c-block/748

[2] https://modernroboticsinc.com/download/hitechnic-ev3-i2c-blocks/

to be sent along with the data, the act of creating an array and replacing its elements is necessary even for sending a single byte. I modified the input variable type in the **WriteI2C_1B** and **WriteI2C_8B** block modes. The data type change worked here as well, and transmission of bytes with values higher than 127 via the I2C protocol can now be performed without any issues.

8.2 UART Communication Speed Testing

After successfully creating UART communication blocks, I set out to find the maximal communication speed between the two devices. I found the maximum baud rate of the communication to be 201,000. This rate is likely limited by the ESP microcontroller rather than the EV3 since the Hardware Developer Kit states that the EV3 can handle up to 460kbit/s on ports 1 and 2 and up to 230kbit/s on ports 3 and 4 [27]. The ESP-12F MCU code utilizes the *SoftwareSerial* library instead of the standard *Serial* functions. This is due to the fact that the handshake protocol requires the TX pin to be set to low for 500 ms before initiating the data sequence and the *Serial* library does not allow this to be successfully implemented [18, 29]. The *SoftwareSerial* UART emulation library allows for more control over the GPIO of the microcontroller at a cost of a limited rate of communication speed. Although speeds of up to 201,000 baud are attainable, the microcontroller does not react to NACK bytes fairly often, as it is likely at its limit of communication speed. Lowering the speed to 200,000 baud yields a more stable connection, however, only rates of 196,000 baud and lower are completely stable.

Although the EV3 can communicate at high baud rates, the total communication speed is rather limited. When using the *PBrickInputWrite* primitive of the EV3, only a single byte can be sent in the span of a millisecond. If the write time of this byte is shorter, the pin is held at HIGH for the rest of the time. The messages sent from the EV3 also contain information and checksum bytes, making a message with a payload of 8 bytes 10 bytes long. A NACK byte is sent at intervals of 100 milliseconds. The maximum transfer rate is 1,000 bytes per second, or 8 kbit/s, and as such, the transfer speed of payload data is

$$(1000 - N_n) * \frac{N_p}{N_p+2} = 792 \text{ B/s} = 6.336 \text{ kbit/s}$$

where N_n is the number of NACK bytes transmitted in a second and N_p is the maximum length of a payload in a single message, with the maximum value being 8 bytes. As such, communication rates higher than 57600 yield no actual advantages.

8.3 Blocks for Controlling the Motor Driver

Another task was to utilize previous blocks to control the motor driver. Using a similar approach as in section 8.1, the block uses import parameters of motor number, its power state, polarity, and duty cycle to construct a message. This functionality is implemented in both the I2C and UART protocols. To mimic the functionality of the Micropython functions in Subsection 5.2.5, I created block modes that send messages setting all four motors globally (Figure 8.2), again for both the I2C and UART communication protocols.

The last of these modes is used to read feedback data from the motor driver when using UART. Due to the reasons elaborated in Section 8.1, I was unable to send multiple bytes of data, each containing the motor duty cycle. To send duty cycle data of every motor and utilize just a single byte of data at a time, I made use of multiplexing. The ESP iterates

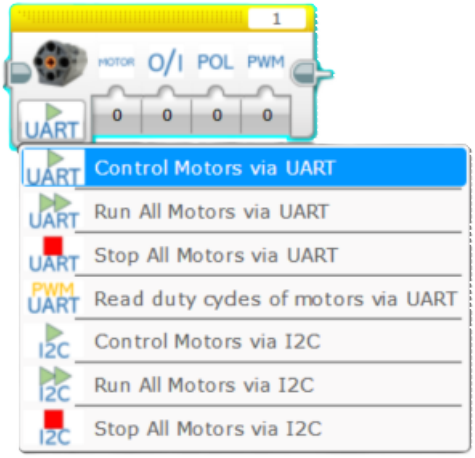

Figure 8.2: Motor Control block in the EV3 programming interface

over the motors indefinitely, transmitting the duty cycle of a single motor in each separate message. To identify this data, I made use of the communication protocol of Subsection 5.2.4. The first four bits of the data once again contain the duty cycle, though here I have opted out of the State and Polarity information, now deemed unnecessary. Their place of bits 5 and 6 is now utilized by the motor number, thus reducing the maximum transmitted value to 63 (0x3F), avoiding the problematic high byte values. The block implementation checks the value, and if it is within the desired range, the duty cycle value is written to the appropriate output. As the value inside the output is kept until it is overwritten, the act of multiplexing does not reset the values, and the output is consistent when displayed. The implementation of this block is shown below in Figure 8.3. The list of functions within the Motor Control block is as follows:

- **MotorControlUART** - Controls a single motor and its state, polarity and duty cycle via the LEGO UART Sensor Protocol
- **RunAllMotorsUART** - Sends four bytes of data, setting the same state, polarity and duty cycle of all four motors at once
- **StopAllMotorsUART** - Sends four bytes of data that immediately stop all motors
- **UARTReadMotorState** - Reads the messages coming from the driver, setting the value of the four outputs depending on the message payload by using multiplexing
- **MotorControlI2C** - Controls a single motor and its state, polarity and duty cycle via the I2C protocol, sent to the specified I2C slave address
- **RunAllMotorsI2C** - Sends four bytes of data, setting the same state, polarity and duty cycle of all four motors at once using the I2C protocol
- **StopAllMotorsI2C** - Sends four bytes of data that immediately stop all motors via the I2C protocol

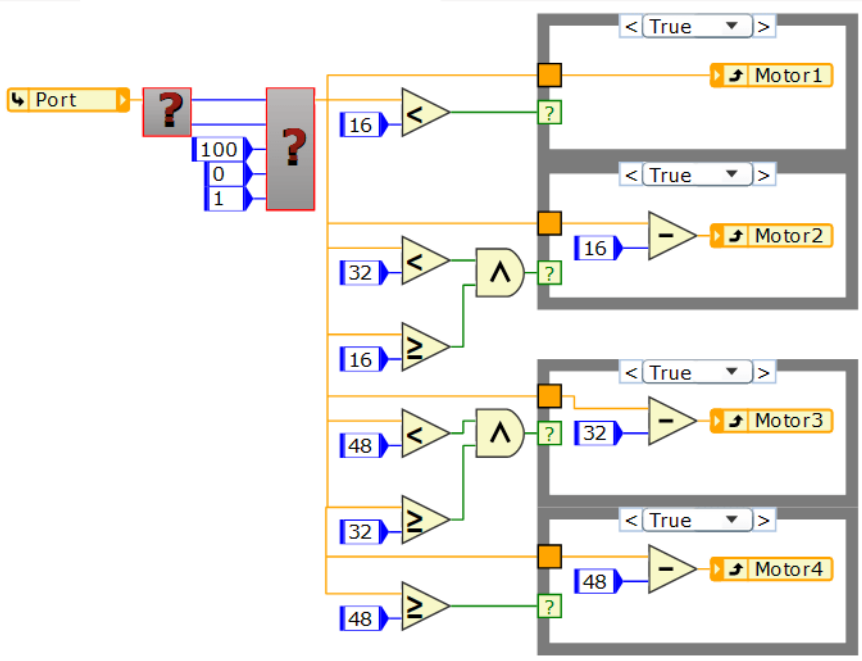

Figure 8.3: Implementation of the Motor Duty Cycle mode of the **MotorControl** Block

8.4 Summary of Contributions to the Code Block Interface

These two blocks, together with the revised Dexter Industries I2C block, broaden the possibilities of communication between the Mindstorms brick and the devices attached to it. This implementation allows users to carry out both general communication and targeted data exchange based on the motor control communication protocol. These functionalities are accessible through the I2C and LEGO UART Sensor Protocol and bring the abilities of the stock EV3 software closer to enhanced third-party software solutions, such as ev3dev or leJOS, while retaining the simplicity and ease of use of coding blocks.

Chapter 9

Conclusion

The goal of this thesis was to propose and implement solutions that would enhance the communication capabilities between the EV3 brick and officially incompatible LEGO motorized pieces. After researching the history of both the programmable bricks and motorized parts, as well as possible areas of improvement, I decided to create a prototype of a motor controller using an Arduino Uno MCU paired with a TI L293D motor driver shield. This prototype communicates with the EV3 via the I2C protocol and can power up to 4 motors independently, using only a single EV3 port. I have created a communication protocol whose messages contain the information to set the state, polarity, and duty cycle for all motors, as well as functions in the Micropython language to construct byte messages according to the protocol.

This prototype was followed by an improved and streamlined iteration. This time, I designed a bespoke printed circuit board that directly integrates an ESP-12F microcontroller and an RJ12 communication port within a simplified and more efficient design. Instead of an alkaline battery box, this version is powered by a 3-cell lithium/polymer rechargeable battery, providing significantly more power to the motors. The design is complemented by a custom 3D printed enclosure with pins and pin holes compatible with standard LEGO and LEGO Technic bricks, allowing it to be securely connected to LEGO creations. This implementation was followed by a next iteration that dealt with issues present in the previous version while being more efficient, compact, and providing more power to connected motors thanks to a more capable ESP32 microcontroller and more efficient TC4427COA driver chips.

In addition, I created two custom LEGO models that demonstrate the functionality of my implementation. The first creation is a diorama that showcases the ability of the EV3 to independently control a large number of motors when using the motor driver, with the motor driver being able to be integrated into these models. The second creation is vehicle with four-wheel drive that demonstrates the broadened capabilities that the driver provides in mobile LEGO creations.

Lastly, I developed EV3-G blocks for general communication with the EV3 using the LEGO UART Sensor Protocol, as well as blocks for controlling the motors connected to the drivers via both I2C and UART protocols and reading their current state while using the UART protocol, utilizing the intricacies of the UART Sensor Protocol. Consequently, I fixed a long-existing issue within the I2C communication block created by Dexter Industries.

I have managed to successfully implement the prototype and test its functionality. I have used LEGO Power Function motors in the implementation due to the greater availability of motors and extension cables compared to the 9V line, though any PWM-controlled

direct-current motor is compatible with my motor driver. This prototype can find uses in various LEGO creations, notably vehicles, where it unlocks the capabilities of independently controlled four-wheel drive, such as torque vectoring or tank-like movement. Other than that, it might prove useful in large-scale motorized LEGO creations and dioramas, as this way, the EV3 can independently control up to sixteen DC motors. The result of my work can be helpful for hobbyists creating custom LEGO creations or young enthusiasts of robotics, since the motor driver is controlled in the beginner-friendly coding block interface.

Future work could bring improvements in terms of performance and safety of the driver, by swapping the currently used TC4427 drivers for more modern and capable drivers that contain internal overcurrent protection. Additionally, a version of the driver for the EV3's successor, the Robot Inventor could be developed, as it features similar interfaces to the EV3. This improved version could also connect to the hub wirelessly via Bluetooth, further expanding the capabilities of the driver and eliminating the limits of wired connections. Additionally, a simple physical interface could be added to the driver that would allow the connected motors to be controlled directly from the driver brick.

Bibliography

[1] The Brickipedia community. *4.5V* [online]. 2019 [cit. 2023-11-17]. Available at: https://en.brickimedia.org/wiki/4.5V.

[2] Alphin, T. *Guide to LEGO 'Powered Up' System* [online]. 2022 [cit. 2023-10-07]. Available at: https://brickarchitect.com/powered-up/.

[3] Barrowman, T., Muir, M., Revere, R. and Cremer, S. *Technic Control I Resource Guide*. 1st ed. LEGO Group, 1991. ISBN 0-914831-74-7.

[4] Beneke, R. *A brief Lego® Motor History* [online]. 2004 [cit. 2023-11-17]. Available at: https://horst-lehner.mausnet.de/lego/ben/9v_12v.html.

[5] Blakbird. *8094 Control Center* [online]. 2008 [cit. 2023-10-28]. Available at: http://www.technicopedia.com/8094.html.

[6] Blakbird. *8479 Barcode Multi-Set* [online]. 2015 [cit. 2023-11-19]. Available at: http://www.technicopedia.com/8479.html.

[7] Fernandez Vicente, M., Calle, W., Ferrandiz, S. and Conejero, A. Effect of Infill Parameters on Tensile Mechanical Behavior in Desktop 3D Printing. *3D Printing and Additive Manufacturing*. 2016, vol. 3, no. 3, p. 183–192, [cit. 2024-05-06]. DOI: 10.1089/3dp.2015.0036. Available at: https://doi.org/10.1089/3dp.2015.0036.

[8] Fučík, J. *Připojení Lego® Power Functions™* [online]. 2019 [cit. 2023-11-17]. Available at: http://www.fucik.name/lego/com/.

[9] *H8/3297 Series Hardware Manual*. 3rd ed. Semiconductor and IC Div. Hitachi, Ltd., 1997 [cit. 2023-09-27]. Available at: https://pdf1.alldatasheet.com/datasheet-pdf/view/249948/RENESAS/H8/3292.html.

[10] Hocker, M. *A History of LEGO Education, Part 1: Strong Foundations* [online]. 2020 [cit. 2024-01-12]. Available at: https://www.brothers-brick.com/2020/01/14/a-history-of-lego-education-part-1-strong-foundations-feature/.

[11] Hocker, M. *A History of LEGO Education, Part 2: Path to Mindstorms* [online]. 2020 [cit. 2023-10-17]. Available at: https://www.brothers-brick.com/2020/01/31/a-history-of-lego-education-part-2-path-to-mindstorms-feature/.

[12] Hocker, M. *A History of LEGO Education, Part 3: Mindstorms over matter* [online]. 2020 [cit. 2023-10-17]. Available at: https://www.brothers-brick.com/2020/02/03/a-history-of-lego-education-part-3-mindstorms-over-matter-feature/.

[13] Hurbain, P. *The Powered Up connector* [online]. 2019 [cit. 2023-11-15]. Available at: https://www.philohome.com/wedo2reverse/connect.htm.

[14] Hurbain, P. *The Powered Up serial link protocol* [online]. 2019 [cit. 2023-11-15]. Available at: https://www.philohome.com/wedo2reverse/protocol.htm.

[15] Hurbain, P. *LEGO® 9V Technic Motors compared characteristics* [online]. 2020 [cit. 2023-11-16]. Available at: https://www.philohome.com/motors/motorcomp.htm.

[16] Hurbain, P. *NXT® motor internals* [online]. 2020 [cit. 2023-09-28]. Available at: https://www.philohome.com/nxtmotor/nxtmotor.htm.

[17] Koehrsen, M. *Understanding the Adafruit Motor Shield Library* [online]. May 2009 [cit. 2024-04-07]. Available at: https://web.archive.org/web/20100724182244/https://docs.google.com/View?docid=dgwf6cmm_2fznx7qgr.

[18] Köhler, S. *UART Sensor Protocol* [online]. February 2015 [cit. 2024-04-08]. Available at: https://sourceforge.net/p/lejos/wiki/UART%20Sensor%20Protocol/.

[19] Lechner, D. *How to use Power Functions with Mindstorms EV3?* [online]. December 2015 [cit. 2024-01-12]. Available at: https://bricks.stackexchange.com/a/7051.

[20] Lechner, D. *LEGO Powered Up UART Protocol* [online]. July 2023 [cit. 2024-04-08]. Available at: https://github.com/pybricks/technical-info/blob/master/uart-protocol.md.

[21] *LEGO Interfacekabel zu den Computern Commodore 64 und 128*. The LEGO Group, 1987 [cit. 2023-10-25]. Available at: https://archive.org/details/lego_c64_manual_basic_comal/page/n17/mode/2up.

[22] *LEGO MindStorms Scout Software Developers Kit User Guide & Reference*. The LEGO Group, 1999.

[23] *MindStorms Robotics Discovery Set Instructopedia* [online]. The LEGO Group, 1999 [cit. 2023-10-23]. Available at: https://www.lego.com/cdn/product-assets/product.bi.core.pdf/4129403.pdf.

[24] *LEGO® Education WeDo™ Teacher's Guide* [online]. 2009 [cit. 2023-11-15]. Available at: https://le-www-live-s.legocdn.com/sc/media/files/user-guides/wedo/wedo-user-guide-80b6e879549d1be595355dc8b6dee075.pdf?la=en-us.

[25] *EV3 Block Developer Kit* [online]. LEGO Group, 2013 [cit. 2024-04-10]. Available at: https://education.lego.com/en-us/product-resources/mindstorms-ev3/downloads/developer-kits/.

[26] *LEGO MINDSTORMS EV3 Firmware Developer Kit* [online]. The LEGO Group, 2013 [cit. 2023-09-29]. Available at: https://assets.education.lego.com/v3/assets/blt293eea581807678a/blt469be1e11ad37696/5f880384f71916144453a49f/lego-mindstorms-ev3-firmware-developer-kit.pdf?locale=en-us.

[27] *LEGO MINDSTORMS EV3 Hardware Developer Kit* [online]. The LEGO Group, 2013 [cit. 2023-09-28]. Available at: https://education.lego.com/en-us/product-resources/mindstorms-ev3/downloads/developer-kits/.

[28] *LMS 2012 UART Device Communication Protocol* [online]. LEGO Group, 2013 [cit. 2024-04-08]. Available at: http://ev3.fantastic.computer/doxygen-all/_uart_protocol.html.

[29] *LMS 2012 UART Device Controller* [online]. LEGO Group, 2013 [cit. 2024-04-08]. Available at: http://ev3.fantastic.computer/doxygen-all/_uart_driver.html.

[30] *LEGO® Education SPIKE™ Prime Technical Specifications* [online]. 2019 [cit. 2023-10-05]. Available at: https://education.lego.com/en-us/product-resources/spike-prime/downloads/technical-specifications/.

[31] *LEGO® Robot Inventor Product Page* [online]. 2020 [cit. 2023-10-05]. Available at: https://www.lego.com/en-us/product/robot-inventor-51515.

[32] *LEGO® Education SPIKE™ Essential Technical Specifications* [online]. 2021 [cit. 2023-10-06]. Available at: https://education.lego.com/en-us/product-resources/45345-spike-essential-resource-page/downloads/technical-specifications/.

[33] Lgauge, T. *LEGO Interface A* [online]. 2011 [cit. 2023-10-24]. Available at: https://lgauge.com/article.php?article=technic/articles/LEGOInterfaceA.

[34] Lgauge, T. *LEGO Interface B* [online]. 2011 [cit. 2023-10-24]. Available at: https://lgauge.com/article.php?article=technic/articles/LEGOInterfaceB.

[35] Lukazi. *LEGO's first programmable product* [online]. 2014 [cit. 2023-10-24]. Available at: http://lukazi.blogspot.com/2014/07/lego-legos-first-programmable-product.html.

[36] Martin, F., Mikhak, B., Resnick, M., Silverman, B. and Berg, R. *To mindstorms and beyond: Evolution of a construction kit for magical machines.* 2000 [cit. 2023-10-15]. Available at: https://www.cs.uml.edu/~fredm/papers/magical-machines.pdf.

[37] *TC4426/TC4427/TC4428 1.5A Dual High-Speed Power MOSFET Drivers* [online]. Microchip, 2014 [cit. 2024-04-27]. Available at: https://ww1.microchip.com/downloads/en/DeviceDoc/20001422G.pdf.

[38] Mindell, D., Beland, C., Chan, W., Clarke, D., Park, R. et al. *LEGO Mindstorms The Structure of an Engineering (R)evolution.* 2000 [cit. 2023-10-17]. Available at: https://web.archive.org/web/20221017071530/http://web.mit.edu/6.933/www/Fall2000/LegoMindstorms.pdf.

[39] *I²C-bus specification and user manual* [online]. NXP, 2021 [cit. 2023-12-11]. Available at: https://www.nxp.com/docs/en/user-guide/UM10204.pdf.

[40] Rangu, M. and Negrea, C. A demystifying study of thermal relief pads: Tradeoff between manufacturing and cooling. In: *33rd International Spring Seminar on Electronics Technology, ISSE 2010.* 2010, p. 390–395 [cit. 2024-05-06]. DOI: 10.1109/ISSE.2010.5547325.

[41] Rinderknecht, M. *Tutorial for Programming the LEGO® MINDSTORMS™ NXT* [online]. 2007 [cit. 2023-09-27]. Available at: https://web.archive.org/web/20140124233424/http://legoengineering.com/wp-content/uploads/2013/06/download-tutorial-pdf-2.4MB.pdf.

[42] Sargent, J. R. *The Programmable LEGO Brick: Ubiquitous Computing for Kids.* 1995. [cit. 2023-10-15]. Master's thesis. Massachusetts Institute of Technology. Available at: https://dspace.mit.edu/bitstream/handle/1721.1/34694/32601552-MIT.pdf?sequence=2&isAllowed=y.

[43] Staple, D. *The Lego RCX, Inside and Out* [online]. 2010 [cit. 2023-09-27]. Available at: https://orionrobots.co.uk/2010/01/19/the-lego-rcx-inside-and-out.html.

[44] *KeyStone Architecture UART User Guide* [online]. Texas Instruments, 2010 [cit. 2023-12-14]. Available at: https://www.ti.com/lit/ug/sprugp1/sprugp1.pdf?ts=1702576347289&ref_url=https%253A%252F%252Fwww.google.com%252F.

[45] *AM1808 ARM Microprocessor* [online]. Texas Instruments, 2014 [cit. 2024-04-11]. Available at: https://www.ti.com/lit/ds/symlink/am1808.pdf?ts=1712817112677.

[46] *L293x Quadruple Half-H Drivers datasheet* [online]. Texas Instruments, 2016 [cit. 2023-12-10]. Available at: https://www.ti.com/product/L293D#tech-docs.

[47] *Connecting the EV3 and the Arduino* [online]. The Dexter Industries Authors, 2014 [cit. 2023-12-10]. Available at: https://www.dexterindustries.com/howto/connecting-ev3-arduino/.

[48] *LeJOS EV3 API Documentation* [online]. The leJOS API Authors, 2015 [cit. 2023-12-13]. Available at: https://lejos.sourceforge.io/ev3/docs/.

[49] *PyBricks 2.0 Documentation* [online]. The Pybricks Authors, 2020 [cit. 2023-12-11]. Available at: https://docs.pybricks.com/en/v2.0/index.html.

[50] TobyMac. *Classic Review: 8485-1 - Control Centre II* [online]. 2020 [cit. 2023-10-25]. Available at: https://rebrickable.com/blog/272/classic-review-8485-1-control-centre-ii/.

[51] Valk, L. *Tutorial: Understanding the difference between NXT set versions* [online]. 2012 [cit. 2023-09-27]. Available at: http://robotsquare.com/2012/02/18/understanding-nxt-versions/.

[52] Valk, L. *EV3 and NXT: Difference and Compatibility* [online]. 2013 [cit. 2023-09-29]. Available at: http://robotsquare.com/2013/07/16/ev3-nxt-compatibility/.

[53] Wood, C. *Lego® RCX presentation* [online]. 2013 [cit. 2023-12-12]. Available at: https://clark.cementhorizon.com/RCX-web-page-2013-09-29.html.

[54] Zhai, L., Sun, T. and Wang, J. Electronic Stability Control Based on Motor Driving and Braking Torque Distribution for a Four In-Wheel Motor Drive Electric Vehicle. *IEEE Transactions on Vehicular Technology.* 2016, vol. 65, no. 6, p. 4726–4739, [cit. 2024-05-06]. DOI: 10.1109/TVT.2016.2526663.

Appendix A

Contents of the Included Storage Media

```
/
├── Tex_Src.zip ..... Source files for the bachelor's thesis text.
├── EV3-G_Blocks
│   ├── Thesis_Block_Src .... Source files for UART and Motor Control blocks.
│   ├── Thesis_Block.ev3b ... Compiled UART and Motor Control block file.
│   ├── Dexter_Mod_Src ..... Source files for the Dexter Industries block.
│   └── Dexter_Mod.ev3b ..... Compiled modified Dexter Industried blocks.
├── Driver_Arduino
│   ├── Uno_Driver.ino ..... Arduino Motor Control program.
│   └── EV3_Driver.py ..... EV3 Micropython code with a MotorDriver class
├── Driver_V0.1
│   ├── Driver_Code
│   │   ├── ESP12_Driver.ino . Code of the Driver v0.1 MCU
│   │   ├── ESP12_Driver.h .. Header file for ESP12_Driver.ino
│   │   ├── AF_Motor.cpp ..... Modified Adafruit library for motor control
│   │   └── AF_Motor.h ..... Header file for AF_Motor.cpp
│   └── Archives
│       ├── V0.1_Proj.zip .... KiCAD PCB project files.
│       ├── V0.1_GBR.zip ..... Gerber files for PCB manufacturing.
│       ├── V0.1_Design.zip . Enclosure design files.
│       └── V0.1_Print.zip .. Files for 3D Printing in .3MF format.
└── Driver_V0.2
    ├── Driver_Code
    │   ├── ESP32_Driver.ino . Code of the Driver v0.2 MCU
    │   ├── ESP32_Driver.h .. Header file for ESP32_Driver.ino
    │   ├── ESP_Motor.cpp ..... Power Functions motor control library
    │   └── ESP_Motor.h ..... Header file for ESP_Motor.cpp
    └── Archives
        ├── V0.2_Proj.zip .... KiCAD PCB project files.
        ├── V0.2_GBR.zip ..... Gerber files for PCB manufacturing.
        ├── V0.2_Design.zip . Enclosure design files.
        └── V0.2_Print.zip .. Files for 3D Printing in .3MF format.
```

Appendix B

ESP Motor Driver V0.1 Schematic

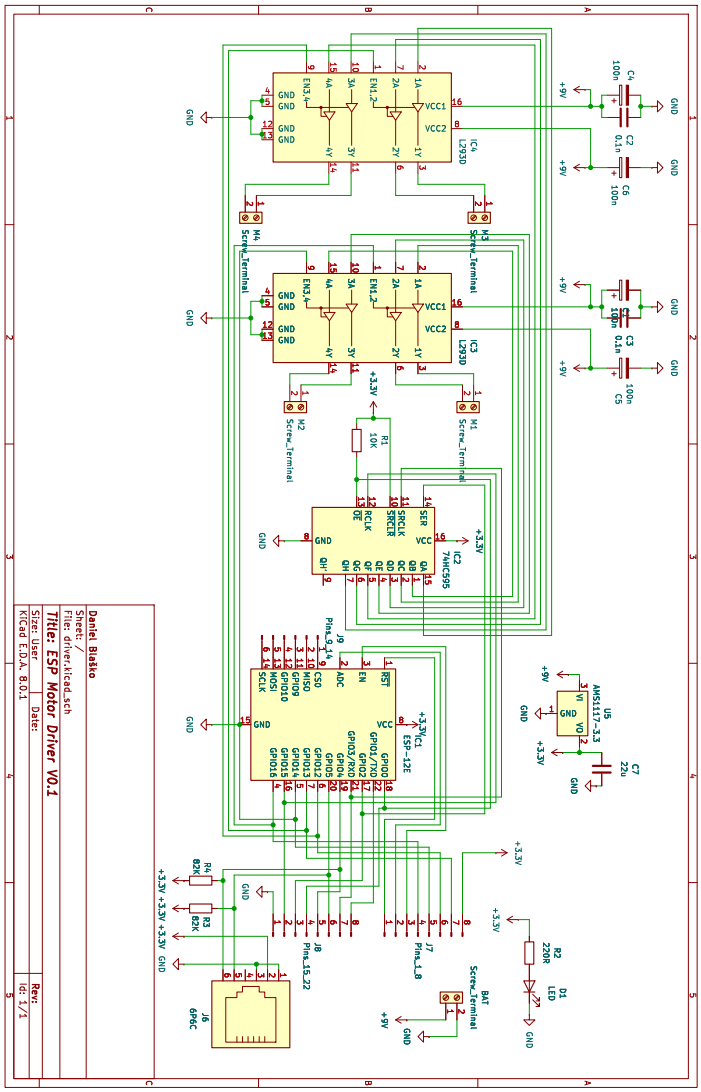

Figure B.1: Schematic of the first custom PCB design iteration

Appendix C

ESP Motor Driver V0.2 Schematic

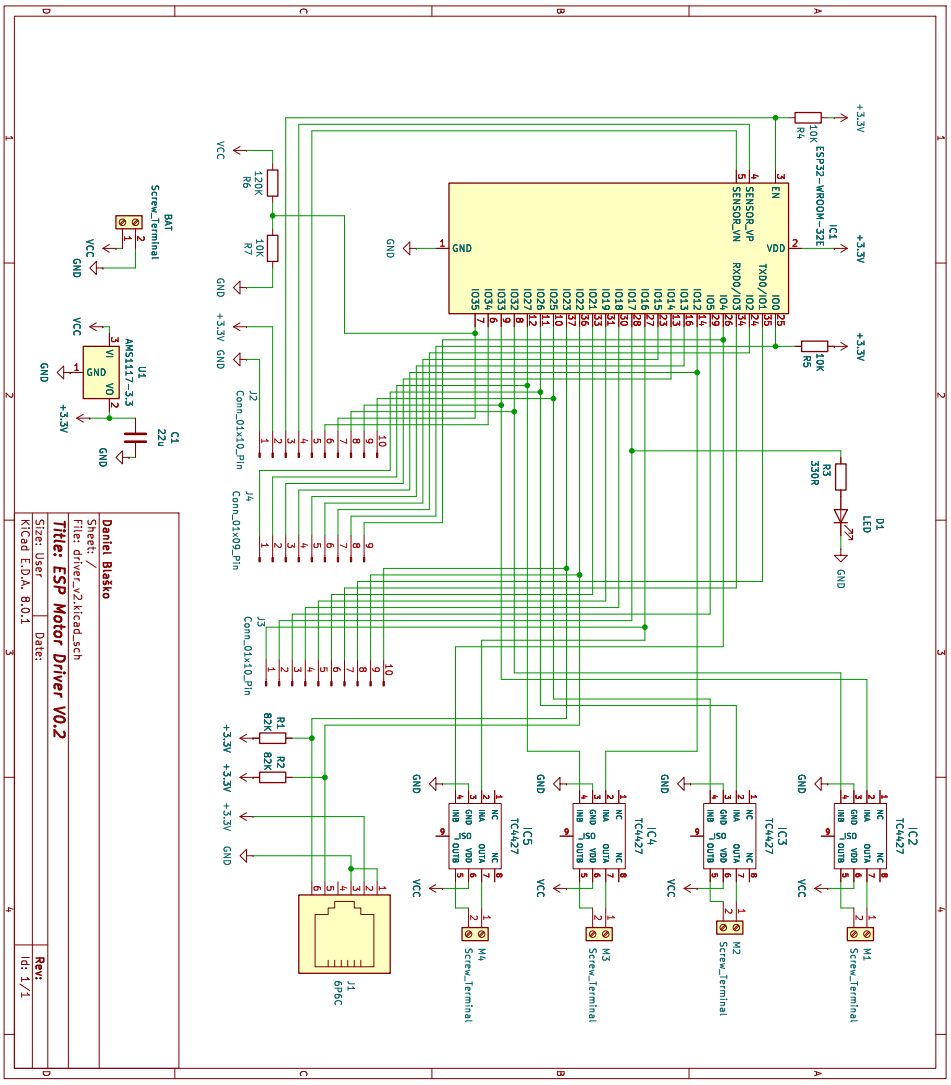

Figure C.1: Schematic of the improved custom PCB design

Appendix D

Manual for Creating and Using the V0.2 Motor Driver

This manual closely describes the process of creating the version 0.2 of the motor driver. Additionally, the manual includes the necessary steps to install custom blocks in the EV3 programming software and use them to communicate with the driver and control motors connected to it.

D.1 Printed Circuit Board

The included storage media (Appendix A) includes the necessary files to get the printed circuit board manufactured in the `Driver_V0.2/Archives/V0.2_GBR.zip` archive. The following components are necessary for the board to be functional:

- ESP32-WROOM-32E microcontroller module
- AMS1117-3.3 linear voltage regulator
- Microchip TC4427COA MOSFET motor driver (x4)
- 22μF SMD capacitor
- 330Ω SMD resistor
- 10kΩ SMD resistor (x3)
- 120kΩ SMD resistor
- 3.3V SMD LED diode
- RJ12 socket with a horizontal layout
- 2-pin screw terminal with a 5mm pitch, capable of handling at least 6A (x5)
- Pin header with a 2.54mm pitch (x6 min., OPTIONAL)
- 82KΩ SMD resistor (x2, OPTIONAL)

The resistors, the diode, and the capacitor are all of an SMD (surface mount device) 1206 footprint. For the component placement, use the schematic in Appendix C.1 and the silkscreen of the PCB for guidance. Begin the soldering process with the SMD components first. Start off with the resistors, the capacitor and the LED, followed by the voltage regulator, motor drivers, and finally the ESP32 MCU. The board has thermal relief polygons connected to the ground, so heating up the GND pins of the components might take longer compared to other pins. After all of the SMD components have been soldered onto the board, solder the THT (through-hole technology) components, beginning with the shortest components, the screw terminals. Soldering pins headers is optional, however, it is strongly recommended as it makes the programming of the microcontroller significantly easier. The minimum number of pin headers that I recommend solder is six, for the GND, 3V3, EN, IO0, TX0, and RX0 pins, however, soldering pin headers for all GPIO pins can help with a diagnostic or debugging process, should such circumstances occur. Lastly, solder the RJ12 socket in place. The R1 and R2 82K pull-up resistors need not be soldered in place, unless you plan to communicate with the EV3 via the I2C protocol. These resistors are necessary for the communication to work correctly according to the EV3 Hardware Developer Kit [27].

D.2 3D Printed Driver Enclosure and Driver Assembly

The `Driver_V0.2/Archives/V0.2_Print.zip` archive contains the **Top.3MF** and **Bottom.3MF** files for printing the enclosure of the driver. Use a freely available software, such as UltiMaker Cura to prepare the files for printing. As this version of the casing has notably narrower walls on it lower portion, I recommend choosing the cubic infill pattern with a 20% infill ratio to retain the structural integrity of the enclosure [7]. If the sliding and locking mechanism does not work correctly, I recommend smoothing the inner surface of the rails with sandpaper. The battery I used for this driver is KAVAN Li-Po 850mAh/11,1V 40/80C 9,4Wh. After placing the battery in the lower part of the driver, place the PCB onto the platform above the battery (Figure D.1). For switching the driver on and off, use a KDC1-101 rocker switch and put it into the opening above the PCB. Solder the positive (red) cable onto one pin and a wire on the second pin. Use a heat-shrinking tube or tape to isolate the connections. Screw the other end of the wire into the positive terminal of the PCB, denoted by the "+" sign of the battery terminal. Afterwards, screw the negative (black) cable of the battery to the "-" terminal.

For the Power Functions connectors on top of the enclosure, cut two PF cables into half. Strip the two middle wires (C1 and C2 in Figure 3.4) and increase the rigidity of the stripped wire by using crimping pliers with wire terminals, or putting a small amount of solder onto the wire. Push the wires through the openings on the upper part of the enclosure, then screw the cables into the M1, M2, M3, and M4 terminals. Finally, slide the driver case closed. After turning on the driver, it automatically enters pairing mode with the EV3, shown by the slow flashing of the status LED. If the device is connected, the LED stays on in a dimmer mode and if the battery of the driver is low, the LED flashes fast for ten seconds before the MCU enters a deep sleep mode. Afterwards, it is necessary to charge the battery. Turning the driver on and off resets the drivers, which then resumes regular operation.

While the EV3 uses RJ12 ports, the latch on the connector is offset to the side. To create a compatible cable, cut the end of an EV3 cable, strip the wires, then place them in an RJ12 connector with a centered latch, and use a crimping tool to secure the cables in

the connector. Alternatively, you can remove the latch of the EV3 connector and use this to connect to the driver, however, note that the connector will not be secured. An image of the driver connected to the EV3 and the motors can be seen in Figure D.3.

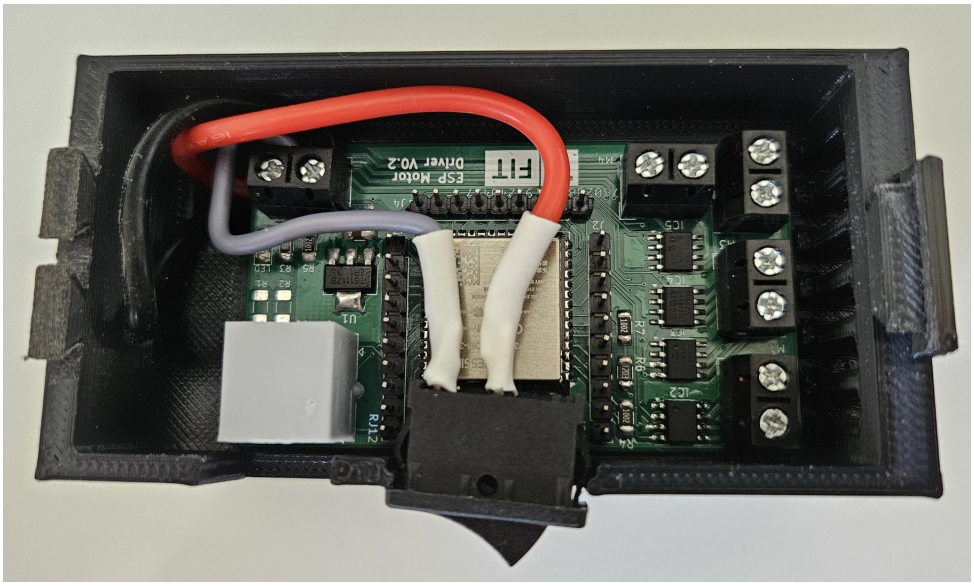

Figure D.1: Assembly of the motor driver

D.3 Programming the Microcontroller

To program the microcontroller, use an Espressif ESP-Prog programmer or an equivalent alternative device. Connect the 6 pins (GND, V_{PROG}, EN, IO0, TX0, RX0) to the corresponding pins of the MCU (Figure D.2). Do not forget to remove the jumper for the V_{PROG} pin if you program the MCU while connected to an external power source. Use Arduino IDE software or the Visual Studio Code editor with *PlatformIO* IDE extension to flash the files located in the `Driver_V0.2/Driver_Code` directory of the attached storage media after choosing the correct device in the board menu. When the ESP-Prog is connected, it uses two COM ports, use the latter one to program the board.

D.4 Using the EV3 Software

Download the EV3 Lab Software[1]and install the app on your device. To import new blocks, firstly create a new project program, then navigate to the **Tools** tab. There, select **Block Import**, find the motor block in the wizard (`EV3-G_Blocks/Thesis_Block.ev3b` in the storage media), then press *Import*. The software then prompts you to restart it. Afterwards, the motor control block will be available for use in the *Sensor* section of the blocks.

Create a program for the brick, which should begin with the Start block and conclude with the Stop Program block. Add the motor block to the program and choose one of the UART modes of the block to control the motors, filling out the desired parameters. Connect

[1] https://education.lego.com/es-mx/downloads/retiredproducts/mindstorms-ev3-lab/software/

the EV3 brick to your computer via cable, Bluetooth, or Wi-Fi, then download the program to the brick and run it. A picture of the EV3 software interface can be seen in Figure D.4. Object number one shows a program example, where all motors are set to run with a duty cycle of 10 out of 15 for ten seconds, then stop for three seconds in an indefinite loop. Number two shows the command block tray with Sensor blocks selected, while number three points to the Motor Control block. Buttons for downloading the program to the brick and running it are located by marker number four. Marker number five shows the *Tools* tab, where the Block Import wizard is located.

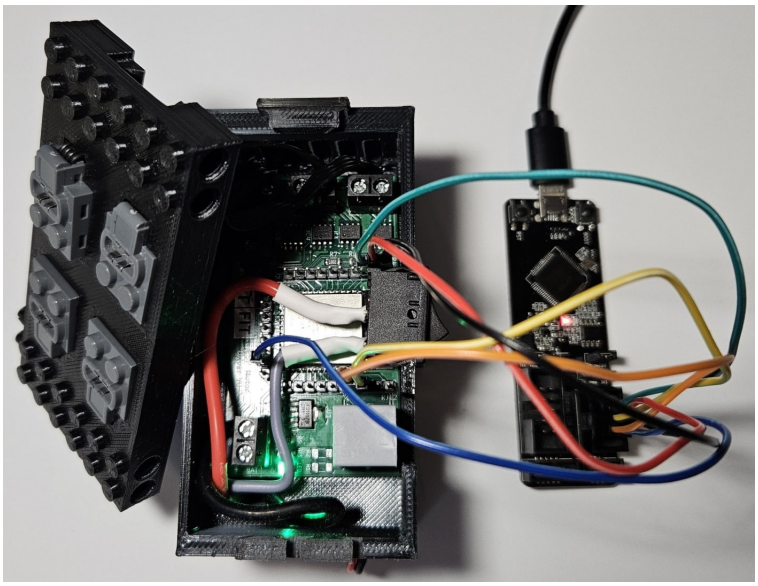

Figure D.2: Programming the motor driver V0.2 with an Espressif ESP-Prog

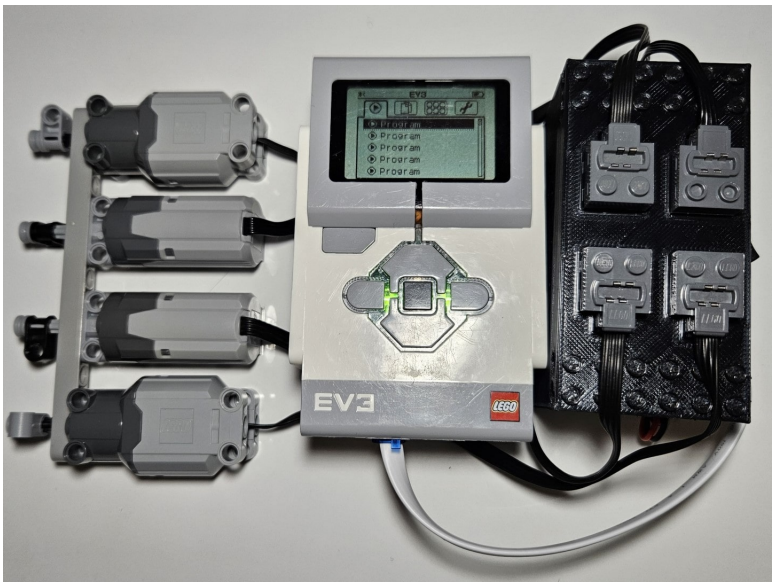

Figure D.3: Driver V0.2 connected to the EV3 and four Power Functions motors

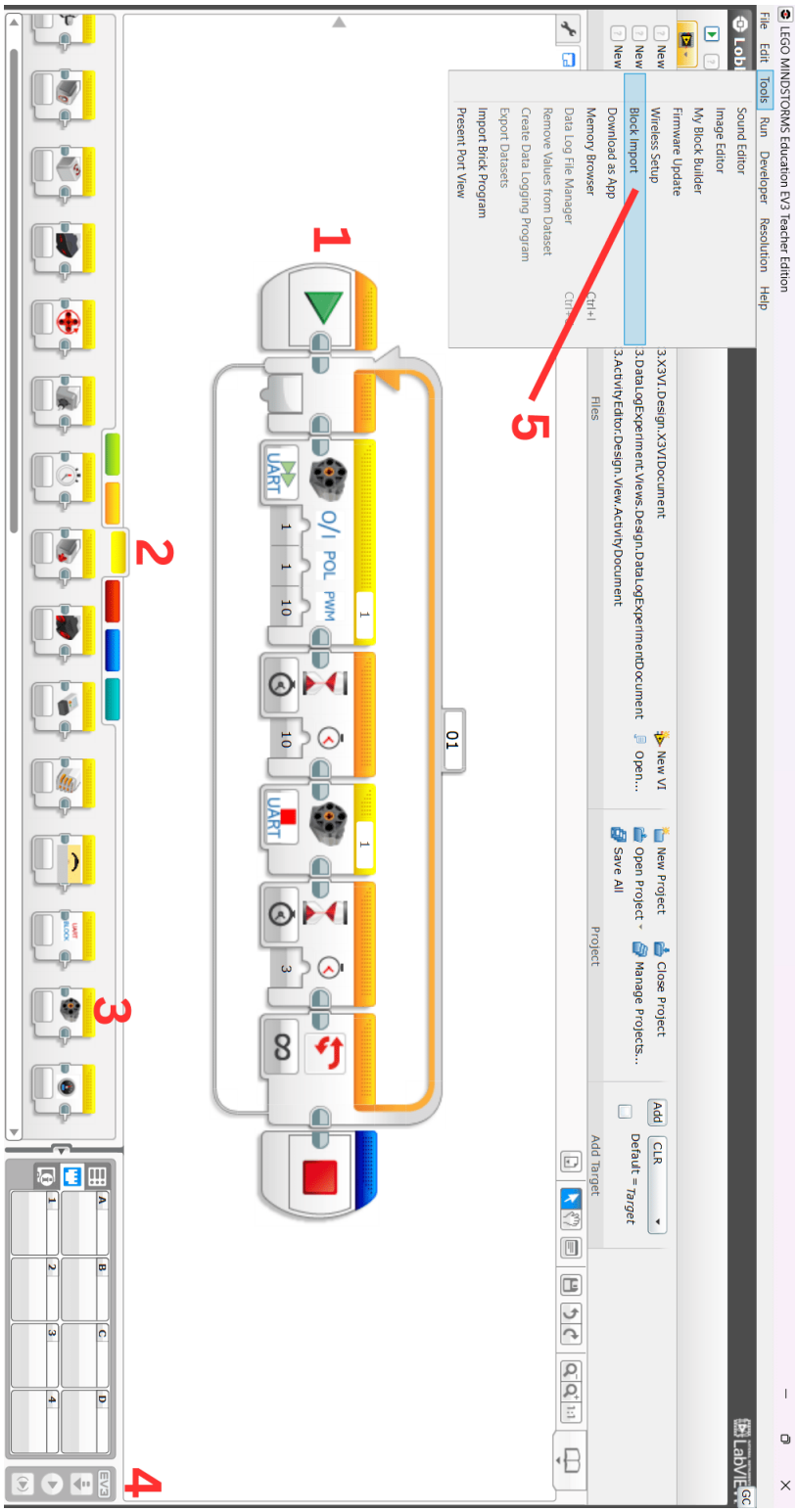

Figure D.4: EV3 Lab Software