CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

Department of Control Engineering

Bachelor's thesis

Development of RC car controlled by STM32 microcontrollers

Kristián Šlehofer

May 2022

Supervisor: Ing. Jan Stejskal

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Šlehofer Kristián**	Personal ID number:	**483516**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Control Engineering**		
Study program:	**Cybernetics and Robotics**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Development of RC car controlled by STM32 microcontrollers

Bachelor’s thesis title in Czech:

Vývoj RC auta řízeného pomocí STM32 mikrokontrolerů

Guidelines:

1) Learn about STM32 microcontrollers, describe their structure and main peripherals.
2) Build the RC car from the functional blocks and explain how they work.
3) Develop the software necessary to control the RC car.
4) Test the behaviour of the designed system in real conditions.

Bibliography / sources:

[1] J. Yiu, 'Definitive Guide to ARM (R) Cortex (R)-M3 and Cortex (R)-M4 Processors,' Elsevier Books, 864 p, 2013, ISBN: 0124080820
[2] T. Martin, 'The Insider's Guide To The STM32 ARM Based Microcontroller,' Hitex Ltd., 106 p, 2009, ISBN: 0-9549988 8
[3] STMicroelectronics, 'STM32F303xB/C/D/E, STM32F303x6/8, STM32F328x8, STM32F358xC, STM32F398xE advanced ARM®-based MCUs,' RM0316 Reference manual [on-line]
[4] Nordic Semiconductor, 'nRF24L01+ Single Chip 2.4GHz Transceiver,' Product specification [on-line]

Name and workplace of bachelor’s thesis supervisor:

Ing. Jan Stejskal Department of Electric Drives and Traction FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **07.02.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

Ing. Jan Stejskal	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt	Student’s signature

Declaration

I declare that the presented work was developed independently ant that I have listed all sources of the information used within it in accordance with the methodical instructions for observing the ethiclal principles in the preparation of university theses.

Prague, May 20, 2022

. . . .
Kristián Šlehofer

Acknowledgement

I would like to express my sincere gratitude to my thesis supervisor Ing. Jan Stejskal for his valuable advice, friendly approach, and help in solving various problems during the project development. I also appreciate all the support I received from my colleagues from Computational Robotics Laboratory and my girlfriend during the thesis writing and experiments. Last but not least, I am also thankful to my family and all my friends for supporting me and keeping me sane during my studies.

Abstract

This thesis focuses on the development of a radio-controlled car powered by STM32 microcontrollers. It describes the building of both the car and transmitter, presents hardware parts and their utilization, and explains the control software in detail.

The controlling system was developed from scratch on top of an old Tamiya chassis and consists of a custom-made control board with modules and necessary control software. The chassis was modified and equipped with more powerful BLDC motor. In order to retrofit the transmitter, a custom PCB was designed. In addition to the essential functions, the car features various sensors and modules with the ability to log the data to the micro SD card. Also, the car status is reported back to the transmitter and visualized on an OLED display.

The developed RC system is tested in real conditions at the end of the thesis. Besides, the maximum controllable range and top speed are measured.

Keywords: STM32, RC car, control software, embedded system, retrofitting

Abstrakt

Tato práce se zaměřuje na vývoj auta na dálkové ovládání poháněného STM32 mikrokontroléry. Popisuje stavbu jak auta, tak i vysílačky, představuje použité hardwarové části a jejich využití a podrobně vysvětluje řídicí software.

Řídicí systém byl od základu vyvinut na starém podvozku Tamiya a tvoří ho na míru vyrobená ovládací deska s moduly a nezbytný řídicí software. Podvozek byl upraven a vybaven výkonnějším BLDC motorem. Aby bylo možné modernizovat vysílačku, byla navržená vlastní deska plošných spojů. Kromě základních funkcí je vůz vybaven různými senzory a moduly s možností záznamu dat na micro SD kartu. Stav vozu je také nahlašován zpět do vysílačky a zobrazován na OLED displeji.

Vyvinutý RC systém je v závěru práce otestován v reálných podmínkách. Kromě toho je změřen maximální kontrolovatelný dosah a maximální rychlost.

Klíčová slova: STM32, RC auto, řídicí software, vestavěný systém, retrofitting

Překlad názvu: Vývoj RC auta řízeného pomocí STM32 mikrokontrolerů

Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Meaning
API	Application Programming Interface
ACK	Acknowledgment packet
ADC	Analog-to-Digital Converter
AHB	Advanced High-performance Bus
APB	Advanced Peripheral Bus
AXI	Advanced eXtensible Interface
BEC	Battery Eliminator Circuit
BLDC	Brushless DC electric motor
CAD	Computer-Aided Design
CRC	Cyclic Redundancy Check
DAC	Digital-to-Analog Converter
DIP	Dual In-line Package
DMA	Direct Memory Access
DSP	Digital Signal Processor
EPROM	Erasable Programmable Read-Only Memory
ESC	Electronic Speed Controller
FPU	Floating Point Unit
GPIO	General-Purpose Input/Output
I^2C	Inter-Integrated Circuit
IMU	Inertial Measurement Unit
LiPo	Lithium-ion Polymer battery
MCU	Microcontroller Unit
Ni-Cd	Nickel-Cadmium battery
NVIC	Nested Vectored Interrupt Controller
OLED	Organic Light-Emitting Diode (display type)
PCB	Printed Circuit Board
PWM	Pulse Width Modulation
RC	Radio Control
SDRAM	Synchronous Dynamic Random Access Memory
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
UART	Universal Asynchronous Receiver and Transmitter
VDDA	Analog supply voltage pin

Chapter 1 Introduction

This thesis focuses on utilizing STM32 microcontrollers to design and build an RC car control board. As the project is Radio Controlled, a control device - a transmitter - had to be developed as well. Both devices are constructed primarily from available modules, which were sometimes modified and combined to form a working RC system.

Therefore, the thesis contains the development of both devices and describes the functional blocks used. The necessary control software, which was developed for both devices, is also presented and explained in detail. Finally, the designed system was tested in real conditions.

The proposed car control system is retrofitted to a particular chassis, but the application of a standard RC control signal using PWM makes it possible to mount the control board to a different chassis with a different servo and motor.

1.1 Motivation

The main motivation was to build an RC car with an STM32 microcontroller that would function and behave like a regular RC car, but at the same time, would be extraordinary. Extraordinary, for example, in terms of range, non-standard features, or user customizability.

Most commercially available RC systems are proprietary and closed and use undocumented components, thus making the modification hard or even impossible. Some hardware modifications are possible with expensive accessories, but the control software customization is usually impossible even with the accessories. This prohibits, for example, advanced driving data reporting and recording or the use of custom sensors.

The solution proposed in this thesis solves the problem, as control boards are built from available and community-known modules, and the control software written in C language is open-source. Additionally, the car control board contains various sensors and a micro SD card slot to record driving data, which can be later visualized and analyzed.

Since the project is not built from scratch and as a base uses an old chassis and transmitter with no spare parts available, another motivation was to make all the modifications with minimal physical intervention to reduce the risk of irreversibly breaking it.

Chapter 2
STM32 microcontrollers

2.1 Introduction

STM32 is a family of 32-bit microcontrollers by STMicroelectronics based on ARM Cortex-M processor, namely Cortex-M3, Cortex-M4(F), Cortex-M0, Cortex-M0+, Cortex-M7F, and Cortex-M33F. The STM32 family consists of numerous series of microcontrollers designed for various use-cases. These can be divided into four main groups: Mainstream, High Performance, Ultra-low-power, and Wireless series. As every series has specific features and peripherals, this section will only focus on the main differences between used ARM Cortex cores. Table 1 shows the utilization of ARM cores in the particular STM32 series.

Table 1: ARM Core utilization in STM32 family

ARM Core	STM32 series
Cortex-M3	F1, F2, L1
Cortex-M4F	F3, F4, G4, L4, L4+, WB, WL[1]
Cortex-M0	F0
Cortex-M0+	G0, L0
Cortex-M7F	F7, H7
Cortex-M33F	L5, U5

2.2 Division by ARM cores

2.2.1 Cortex-M3

Cortex-M3 was the first processor in the Cortex-M generation and the first core used in the STM32 family in the STM32F1 series of microcontrollers. ARM first released this high-performance processor based on ARMv7-M architecture in 2005. It features modified Harvard architecture (with unified memory space) and a 3-stage pipeline. Thanks to 32-bit addressing, 4GB of memory space is supported [1].

2.2.2 Cortex-M4F

The successor to Cortex-M3 was released by ARM in 2010. This new generation called Cortex-M4 is similar to its predecessor in design and features. The instruction set now contains DSP (Digital Signal

[1]Based only on ARM Cortex-M4 (without FPU).

Processor) instructions for more complex data processing, and there is an optional single-precision FPU (Floating Point Unit). Variant with FPU is known as Cortex-M4F [1].

2.2.3 Cortex-M0

Cortex-M0 processor is one of the smallest ARM processors available, optimized for small size and low power consumption. The processor is based on ARMv6-M with a smaller instruction set, uses Von Neumann architecture, and has a 3-stage pipeline. This small core is ideal for general data processing tasks, where low cost and high energy efficiency matter [2].

2.2.4 Cortex-M0+

Cortex-M0+ is based on Cortex-M0, thus adopting small a size and further improving power efficiency and performance. The instruction set and tools are fully backward compatible with Cortex-M0. ARM reduced pipeline to 2 stages to lower power consumption. Furthermore, the processor has more options available, such as vector table relocation and memory protection unit (MPU) [3].

2.2.5 Cortex-M7F

The most powerful processor in the Cortex-M family is Cortex-M7. The processor uses ARMv7-M architecture and features the longest pipeline with six stages. Given the used architecture, the instruction set contains DSP instructions, and there is also an optional FPU, which can be single-precision or even double-precision. Main buses have been enlarged to 64-bit wide. Variant with FPU is known as Cortex-M7F [4].

2.2.6 Cortex-M33F

Cortex-M33 is the newest core used in the STM32 family of microcontrollers. The processor is based on more recent ARMv8-M architecture with Harvard bus architecture. It is conceptually similar to Cortex-M4 with a 3-stage pipeline and optional single-precision FPU. The core is ideal for IoT applications as it has optional TrustZone security instructions for hardware-enforced isolation and memory protection unit (MPU). Variant with FPU is known as Cortex-M33F [5].

2.3 Architecture

ARM as a company does not construct microcontrollers but instead designs the core and other components and licenses them to silicon designers. It is the same concept with the STM32 family of microcontrollers - ST Microelectronics obtains the processor design from ARM as a base for their microcontroller. The company then attaches its own set of memories and peripherals, such as ADC (Analog-to-Digital converter), DAC (Digital-to-Analog converter), clock generation and timers, and various types of connectivity peripherals like UART, SPI, I^2C, etc.

2.3 Architecture

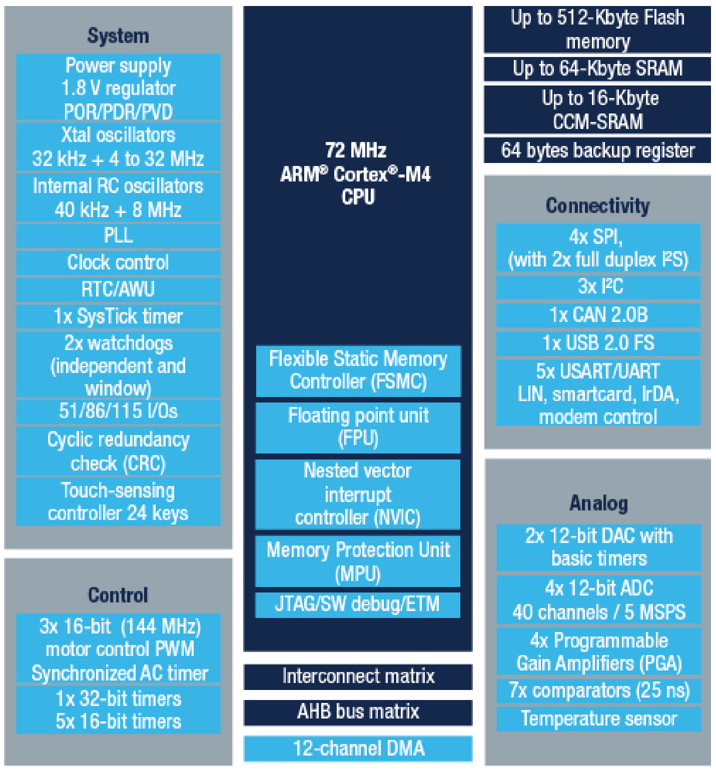


Figure 1: STM32F303 circuit diagram [6]

This approach is illustrated in figure 1, which shows the circuit diagram of the STM32F303CC microcontroller used in this project. It shows the Cortex-M4F processor with all the ST-specific peripherals and memories around.

2.3.1 Clock

The frequency of the processor, buses, and connected peripherals is derived from their clock source, usually an oscillator. STM32 microcontrollers include internal clock sources, or it is possible to connect external clock sources. The frequency from other clock sources can be multiplied by the internal PLL (Phase-Locked Loop) circuit to speed up the microcontroller further and reach higher frequencies. Several prescalers allow finetuning bus and peripherals speed. Internal clock sources have the advantage of providing a clock source at a low cost with no external components. However, being an RC oscillator, the frequency produced is usually not as accurate as an external oscillator.

2.3.2 Interrupts

The main controller responsible for handling interrupt events is Nested Vectored Interrupt Controller, NVIC for short. It is tightly coupled to the Cortex core allowing for low latency interrupt processing. The NVIC features support for up to 240 interrupt inputs with programmable priorities, which can be individually enabled or disabled, along with Non-Maskable Interrupts (NMI). Furthermore, in the case of Cortex M3 and M4, the interrupt latency is only 12 cycles with zero wait state memory [1]. All interrupts can be triggered by software by writing to the appropriate register; this also applies to many system exceptions. Microcontroller vendors determine the total number of interrupts supported by the device with Cortex core when designing the chip.

In the STM32 family of microcontrollers, there is also a hardware device called extended interrupts and events controller (EXTI). This controller is connected to the NVIC and supports up to 36 external/internal interrupt events. Active edge for external interrupts is programmable. This device makes it possible to generate interrupts or wake-up events from external peripherals and GPIO pins.

2.3.3 Buses

The Cortex core has a well-defined on-chip bus structure and protocol that allows the connection of various memory configurations and peripherals. Since most of the Cortex cores use Harvard architecture, multiple bus interfaces enable simultaneous access to instructions and data. Buses are (most of the time) 32-bit wide.

The primary bus interface protocol utilized is AHB Lite (Advanced High-performance Bus). This pipelined protocol allows high operation frequency with a small silicon footprint and is used in program memory and system bus interfaces. Usually, the bus can have more devices that act as a master, for example, DMA (Direct Memory Access) or Ethernet controller. Therefore, the processor includes a bus matrix (or AHB interconnect matrix) that manages transfers from multiple bus masters to different slave segments. Particular bus matrix implementation is shown in figure 2.

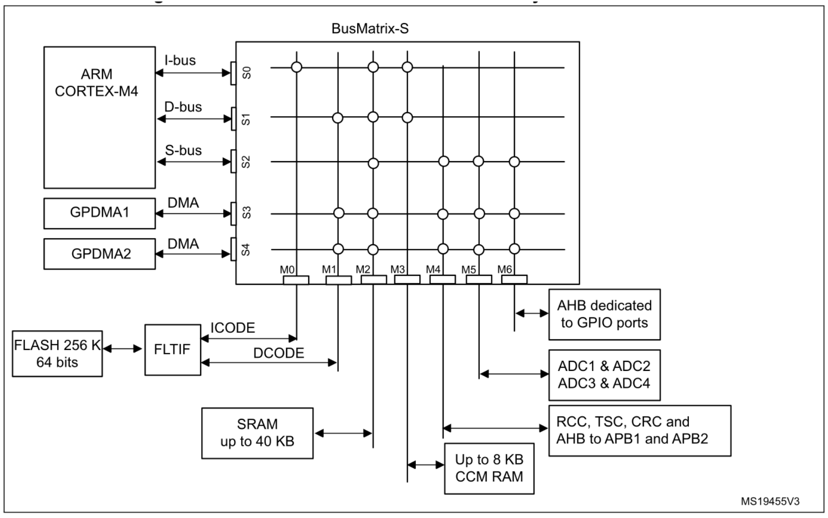


Figure 2: STM32F303CC bus architecture [7]

The secondary bus used mainly for connecting peripherals is APB (Advanced Peripheral Bus). Usually, APB buses are connected to the AHB bus matrix using bridge components. If the peripheral needs higher bandwidth and operation speed, for example, the ADC, it is possible to connect it directly to the faster AHB bus.

Some of the more advanced Cortex processors also use other buses. For example, the Cortex-M7 has an additional 64-bit wide AXI bus that can be used for faster FLASH memory access [4].

2.3.4 Memories

Thanks to the bus architecture, different memories can easily be attached to the AHB bus with suitable memory interface logic. Even though the bus is 32-bit, there are no width restrictions if appropriate conversion hardware is used.

Usually, types like SRAM and FLASH are often used for data memory and program code memory, respectively. However, there is no real limitation on memory type connected to a processor, and different technologies can be utilized, e.g., SDRAM, EPROM, etc. Also, memory size is not limited. The only requirement is that the memory should be byte-addressable, and the interface has to support byte, half-word, and word transfers [1].

2.3.5 Direct Memory Acces (DMA)

A Direct Memory Access is a programmable hardware unit connected to the AHB bus. It reduces processor load by offloading memory transfer operations. Supported transfer directions are peripheral-to-peripheral, memory-to-peripheral, peripheral-to-memory, and also memory-to-memory. There is a small overhead when setting the unit, but afterward, the whole transfer is handled by DMA without processor intervention, thus leaving computing power to other tasks. The unit usually has several independent channels with assignable priorities.

2.4 Peripherals

Every series of STM32 microcontrollers features different peripherals, from the basic ones with little settings to more advanced and configurable ones. This section is not supposed to be an overly exhausting listing of all the peripherals across all STM32 families, but instead, it focuses mainly on the most used ones and those used in this project.

2.4.1 GPIO

The abbreviation GPIO stands for General-Purpose Input/Output. It is a designation for microcontroller pins. The pins are arranged in groups of sixteen and are referred to as GPIOx (for example, GPIOA, GPIOC). Not every pin from the group has to be physically present. Especially for microcontrollers with smaller packages, it is usual that some pins from the group are missing. In figure 3, there is a typical internal structure of the GPIO pin with internal protection diodes.

Each pin can be set as either input, output, or alternate function. Internal pull-up and pull-down resistors are available as well. These features are configured and controlled by software. Most GPIO control registers are 32-bit and have to be accessed as 32-bit words, although some STM32 series support half-word and byte access.

If the pin is in alternate function mode, it is controlled by a connected peripheral, for example, ADC, timer, or USART. Thanks to the highly flexible pin multiplexing, many peripheral functions can be remapped to another pin. Debug pins are in alternate function mode by default after microcontroller reset. Other pins are usually in floating input mode.

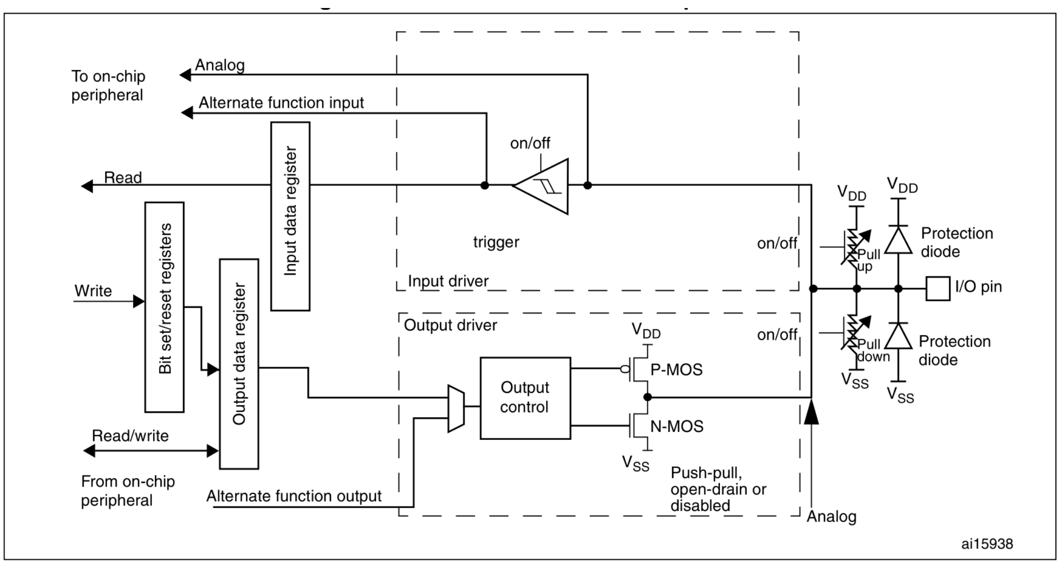


Figure 3: Basic structure of an I/O port bit [7]

When the pin is set as output or alternate function, it can be push-pull or open-drain. Most of the pins can provide ±20 mA of current, making it possible to drive, for example, a LED directly. However, the total current that a microcontroller can source or sink is limited. Connecting pins to voltages higher than the supply voltage is not recommended, yet many pins are 5 V tolerant.

2.4.2 Timers

Timers are one of the basic peripherals. The primary function is counting; it can measure time and do a specific action, for example, toggle a pin. The counter clock frequency is adjustable using a programmable prescaler. Usually, there are many timers in the microcontroller with varying features.

The simplest ones are basic timers. These timers are predominantly used for time-base generation as they feature only a simple up counter, usually 16-bit.

The more feature-rich timers are general-purpose timers. With those, besides the basic functionality, it is possible to generate various waveforms or measure the pulse length of the input signal. Furthermore, there can be support for incremental encoders with quadrature output, which can then be connected straight to the MCU. This feature is handy and makes interfacing encoders pretty straightforward.

As the name suggests, the advanced-control timers are the most advanced. Functionality includes all the features of previously mentioned timers with some added or upgraded. Most of the time, advanced-control timers have more channels that can be synchronized together or using an external signal. This ability makes it possible to generate a three-phase PWM output to control a three-phase motor like BLDC.

The Cortex core contains another 24-bit timer called SysTick timer, dedicated to real-time operating systems. It usually provides OS with regular timer interrupts for time-keeping, but it can also be used as a standard down counting timer.

2.4.3 Analog-to-Digital Converter (ADC)

Another common peripheral is the Analog-to-Digital converter. This peripheral can convert a given voltage (usually between 0 V and VDDA) to a binary number. One possible utilization can be reading a potentiometer voltage to obtain the trigger position.

The particular implementation can vary between the STM32 series, but most of the time, they use successive-approximation ADC or Sigma-Delta ADC. It can be configured to measure either single-ended input or differential input in continuous or single-shot mode. Also, some internal channels are available, for example, connected to an internal temperature sensor and voltage reference.

An internal ADC self-calibration is a convenient feature that can compensate for offset and gain error and significantly improve conversion accuracy. The calibration is triggered by software. If the application has to handle a large amount of data, the DMA unit can serve the ADC memory transfers to free the CPU. Moreover, an analog watchdog feature can monitor the converted voltage and generate an interrupt if it exceeds the programmed thresholds.

2.4.4 Digital-to-Analog Converter (DAC)

The Digital-to-Analog converter does the exact opposite of the Analog-to-Digital converter. It converts the given binary number to a corresponding analog voltage. It is possible to generate complex waveforms or an audio signal with this peripheral. The DAC can therefore be used in many audio applications. If at least two channels are available, the conversion can be independent or simultaneous with external triggers, providing the ability to generate a stereo audio signal. Usually, the DAC also features a noise-wave generation or a triangular-wave generation. Such as with the ADC, the DAC memory transfers can be controlled by DMA, thus freeing up the processor.

2.4.5 Serial Peripheral Interface (SPI)

The serial peripheral interface is a master-slave interface bus used to communicate with smaller devices, such as sensors, memories, displays, or LED drivers. If supported by a particular microcontroller, the peripheral can also be configured as an I^2S (Inter-Integrated Sound) interface, allowing the connection of digital audio devices.

Peripheral settings are rich to support a wide range of compatible devices. It can operate either as a master, providing a clock to connected devices, or as a slave; multimaster mode is also possible. Communication mode and speed can be configured as well as frame size and clock polarity. The DMA unit can help handle a lot of data and achieve high data communication rates with hardware control. Moreover, the SPI peripheral includes a hardware CRC unit with automatic CRC error checking. This feature can be helpful when communicating with SD memory cards.

2.4.6 Inter-Integrated Circuit (I^2C)

The Inter-Integrated Circuit is another commonly used short-range communication bus in embedded systems. It is also a synchronous interface that requires fewer communication wires than SPI but provides lower communication rates. As the SPI interface, the I^2C can be used to communicate with small displays, memories, and other rather lower-speed devices.

The peripheral supports both 7-bit and 10-bit addressing modes, programmable analog and digital noise filters, and configurable multiple speed modes depending on the particular microcontroller. Furthermore, the peripheral includes a hardware CRC unit, and it is designed to support SMBus and PMBus protocols used by PCs and servers. DMA channel connecting I^2C is also available to provide hardware control of transfers.

2.4.7 USART

USART stands for Universal Synchronous/Asynchronous Receiver/Transmitter. Although computers are rarely equipped with serial ports these days (except for the USB), serial communication is still highly used in the embedded world as it is simple and easy to use. This peripheral is flexible and has a wide range of applications - it can be used for simple UART communication; in the synchronous mode, it can act like SPI master/slave; it also features hardware management of some signals from industry standards like RS-232 and RS-485. Moreover, there is support for Smartcard communication, IrDA serial infrared communication, and multiprocessor communication. It is also possible to use the peripheral to implement other buses with hardware control, for example, Modbus or 1-Wire.

The peripheral offers many programmable parameters like baud rate, data word length, the number of stop bits, data order, parity control, or oversampling. Furthermore, USART is the source of many interrupts indicating various statuses and flags. The peripheral is connected to the DMA unit like other serial communication peripherals, significantly facilitating the handling of continuous streams or huge amounts of data.

Chapter 3
Hardware

3.1 Chassis and transmitter

3.1.1 Background

The whole project is built on an old entry-level onroad 1/10 chassis from the *Tamiya Quick Drive* series released on Jul 16, 2003 [8]. The RC car was obtained as a childhood gift. It was pre-assembled and came with a 2-channel transmitter communicating at a frequency of $27\,\mathrm{MHz}$. Later the servo broke, and no repair or upgrade parts were available. Since the repair was nearly impossible, but the rest of the chassis was fully functional, it was the perfect choice for a complete rebuild.

The car featured a sealed gearbox, servo, '280' size brushed motor and $9.6\,\mathrm{V}$ Ni-Cd battery pack. Due to the age of the product, not much technical information can be found. Figure 4 shows the old transmitter PCB.

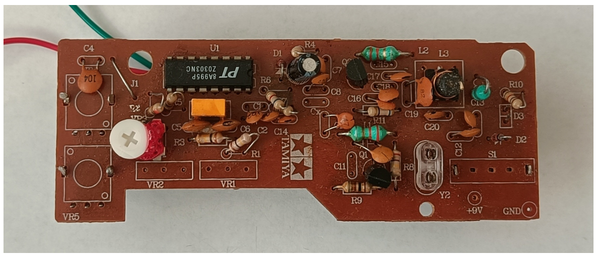


Figure 4: Original transmitter PCB

3.1.2 Modifications

All the electronics were removed, leaving only the plastic chassis. The broken 5-wire servo was replaced with a 'standard' size hobby servo used in most 1/10 RC cars. The old brushed motor was still functional, but it was decided to replace it with a newer and more powerful one. The only constraint was a shaft size, as the old pinion had to be transferred to the new motor to maintain functionality. Therefore a '2435' size BLDC motor with a $25\,\mathrm{A}$ ESC (Electronic Speed Controller) was selected. More information about the servo and BLDC is in sections 3.2 and 3.3. Furthermore, the plastic bearings were replaced by metal ball bearings.

A 2-cell LiPo battery with a capacity of $2200\,\mathrm{mA\,h}$ was chosen to supply power to the electronics. According to the product specification, this battery should withstand 35C of continuous current [9]. The term '35C' means '35 times the battery capacity'. For this particular battery, that means $35 \cdot 2.2\,\mathrm{A} = 77\,\mathrm{A}$ of current, which is more than sufficient.

The transmitter underwent a similar procedure. The original circuit board and the antenna were also removed, retaining only trigger potentiometers and plastic parts. Potentiometers used for throttle and steering trimming were replaced with rotary encoders. Again, the power supply is a 2-cell LiPo battery with a lower capacity of only $900\,\mathrm{mA\,h}$ and 25C of continuous current [10].

3.2 Servo

A servomotor, or servo for short, is an actuator used for precise output shaft positioning. It consists of a motor with position feedback and a control board. RC servos are controlled using a 50Hz PWM signal. The output angle is linearly dependent on a pulse width between $1\,\mathrm{ms}$ and $2\,\mathrm{ms}$. The diagram in figure 5 illustrates typical timing.

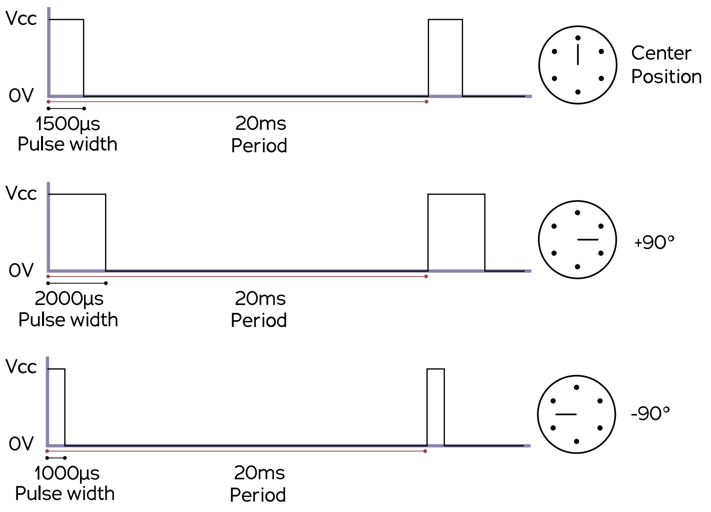


Figure 5: Servomotor timing diagram [11]

The servo used in the car is *EMAX* ES3001. It has plastic gears and operating voltage between $4.8\,\mathrm{V}$ to $6.0\,\mathrm{V}$. Speed and torque depend on the supply voltage. According to the product specifications [12], the servo can achieve an operating speed of $0.17\,\mathrm{s}/60^\circ$ and torque of $3.2\,\mathrm{kg}\cdot\mathrm{cm}$ with the $4.8\,\mathrm{V}$ supply voltage. From the operating speed we can calculate angular velocity:

$$\omega = \frac{d\theta}{dt} = \frac{\frac{\pi}{3}}{0.17} \doteq 6.156\,\mathrm{rad\,s^{-1}}\,, \tag{1}$$

where $\frac{\pi}{3}$ is 60° converted to radians. Since the servo supply voltage is $5\,\mathrm{V}$, both speed and torque values should be slightly higher.

3.3 Motor and ESC

As mentioned in section 3.1.2, the biggest BLDC with a shaft diameter of $2\,\mathrm{mm}$ was selected to replace the old brushed one. The motor was supplied in a set including a $25\,\mathrm{A}$ ESC. The type is

GoolRC 2435 4800 Kv[2] BLDC; available information about the motor is in table 2 and about ESC in table 3.

Table 2: BLDC specifications

Power	300W
Max. voltage	12.6V
Max. current	24A
Rotor poles	4
Kv rating	4800
Length	24mm
Diameter	35mm
Shaft diameter	2mm

Table 3: ESC specifications

Continuous current	25A
Instantaneous current	50A
BEC type	5V, 2A
Battery	2-3 Cell LiPo

Originally the brass pinion was press-fitted to the motor; however, the shaft began to slip inside the pinion. Therefore it was soldered to the shaft with the help of solder paste and a hot-air station.

Since the BLDC motor is directly connected to the ESC and the control board communicates only with the ESC, the BLDC is considered a black box.

RC ESCs are controlled the same way as RC servos; thus, the same timing diagram in figure 5 applies. The only difference is the resulting motion. Pulse width with a duration of 2 ms equals full forward acceleration, pulse width with a duration of 1.5 ms equals neutral, and pulse width with a duration of 1 ms equals full brake/reverse.

Whether the brake or reverse is activated depends on the actual motion of the motor. If the car is going forward (meaning the motor rotates forward) and the pulse width becomes lower than 1.5 ms, first, the brake is activated. The car must be almost static to start reversing; otherwise, the ESC will stay in brake mode. If the car is static or is already reversing, a pulse width lower than 1.5 ms activates reverse mode.

3.4 Control board

Prototyping was mainly done on the breadboard using the *RobotDyn's* BlackPill development board featuring STM32F303CCT6 microcontroller [13]. The board uses a similar size and form factor as the popular BluePill board equipped with STM32F103C8T6 [14]. Both these boards were readily available.

[2]Kv refers to rpm/V with no load attached to the motor shaft.

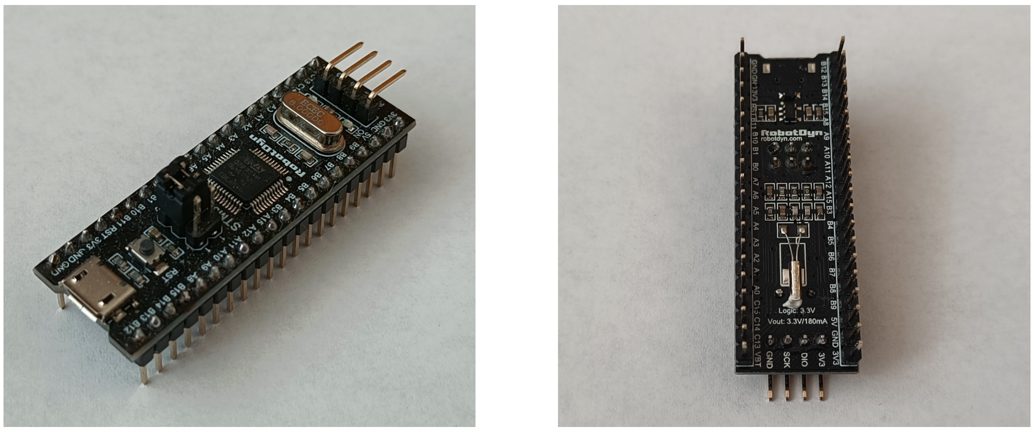

(a) front side (b) back side

Figure 6: RobotDyn BlackPill development board

BlackPill was selected for its better processor based on an ARM M4 core with FPU, larger memories, and more advanced peripherals. Thus, it is more suitable for processing data from more sensors, especially when using floating point arithmetics. The board is shown in figure 6.

3.5 Communication

The project uses wireless modules equipped with nRF24L01+ transceiver chip from *Nordic Semiconductor*, a power amplifier, a low-noise amplifier, and an external antenna. The chip operates in the worldwide $2.4\,\mathrm{GHz}$ ISM band and communicates with the MCU using the SPI bus. Configurable parameters include channel, data rate, and output power. These modules fit all the requirements, and in addition, they are cheap and readily available.

According to the product specification [15], the range should be more than $800\,\mathrm{m}$ line-of-sight. Thanks to the *Enhanced ShockBurst protocol* and its Auto Acknowledgement function [16], attaching a small payload to the ACK packet is possible. This feature is advantageously used to report car status back to the transmitter.

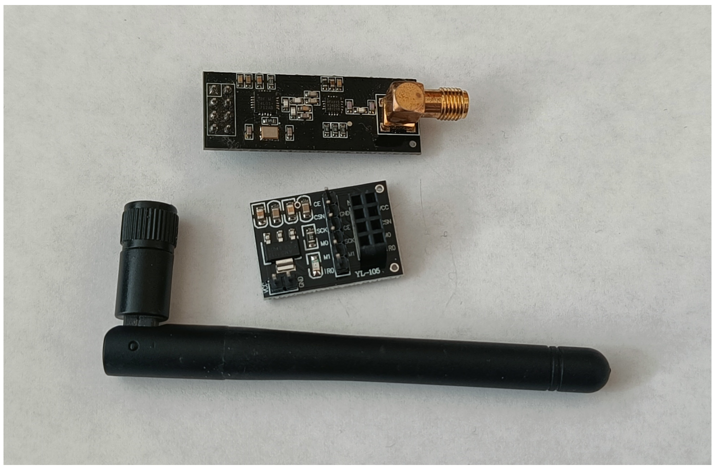


Figure 7: Unmodified nRF24 module with adapter

An additional adapter board is utilized to supply voltage to the nRF24 module. It has the same pinout and includes a voltage regulator with all needed external components. The Adapter board is powered with 5 V, which are then regulated to 3.3 V. This setup ensures a stable power supply for the nRF24 module.

As the adapter board is only equipped with a pin socket for connection, which is unsuitable for use in vibration generating environments, it was removed. The nRF24 module was instead soldered directly to the adapter, as were the wires. The entire communication module is thus more robust and takes up less space. Figure 7 shows the unmodified communication module with the adapter board.

3.6 Sensors

In addition to measuring the voltage, which is necessary for battery monitoring and is realized using a voltage divider, the car also measures current and BLDC case temperature. It is also equipped with 6-axis IMU. In the project's present state, data from sensors are mainly informative and are not used for control purposes.

The transmitter does not use any particular sensor; only battery voltage is measured via the voltage divider.

3.6.1 Current

The current is measured using a module with a 30 A version of an ACS712 chip from *Allegro MicroSystems*. It is a small linear current sensor based on the Hall effect. Current flowing through the sensor's copper terminals generates a magnetic field, which is then sensed by the integrated Hall-effect circuit. The sensor module is in figure 8.

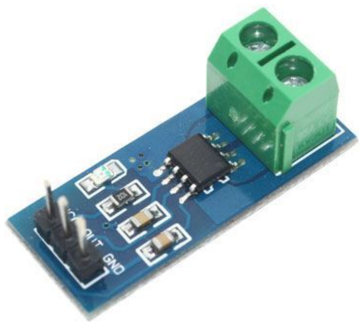


Figure 8: ACS712 current sensor [17]

The output of the sensor is a voltage proportional to the flowing current. If no current is flowing, the sensor output is the supply voltage divided by two. If a positive current flows, the output voltage increases, whereas the output voltage decreases if the current is negative.

Current sense terminals are electrically isolated from the rest of the sensor; therefore, it is possible to measure high voltages without external electrical isolation. The sensor is capable of measuring both AC and DC current with a stated total output error of 1.5% [18].

3.6.2 Temperature

A digital thermometer DS18B20 provides temperature measurement. The sensor is manufactured by *Dallas Semiconductor*, which *Maxim Integrated* later acquired. It can measure temperatures from $-55\,°\mathrm{C}$ to $125\,°\mathrm{C}$. It has a programmable resolution ranging from 9 to 12 bit and $\pm 0.5\,°\mathrm{C}$ accuracy over most of the measuring range. The sensor uses a 1-Wire interface, which requires only one pin for communication. When combined with parasitic power mode, the whole thermometer requires only two pins for operation [19].

The project uses the TO-92 package, which was advantageous because of the limited space around the BLDC. The small size made it possible to mount the sensor on the back of the motor housing with the aid of heat-conducting foil. The sensor is in figure 9.

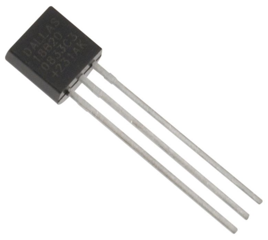


Figure 9: DS18B20 temperature sensor [20]

3.6.3 IMU

For motion measurements, a module with MPU6050 from *InvenSense* was selected. The module is in figure 10. The sensor packs a 3-axis accelerometer, which provides linear acceleration data, and a 3-axis gyroscope, which provides angular velocity data, in one chip. Both the accelerometer and gyroscope have programmable measurement ranges. Communication with the sensor is performed using the I^2C bus. An external 3-axis magnetometer can be attached straight to the sensor, extending the sensor's capability [21].

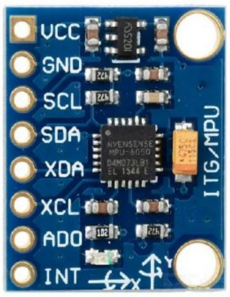


Figure 10: MPU6050 IMU module [22]

The sensor claims it has a DMP (Digital Motion Processor) capable of merging the accelerometer and gyroscope data on-chip. Sadly, this feature is undocumented and available only when using

InvenSense's official *Embedded MotionApps software*. Another possibility would be using *I^2Cdevlib*, where the DMP functionality was reverse-engineered [23]. However, the *I^2Cdevlib* is written for Arduino and would require porting the code to the STM32 with uncertain results. Therefore only raw accelerometer and gyroscope data are used.

3.7 Status info

Neither the RC car nor the transmitter reported any status in the original version. The user had to check the battery state himself; otherwise, the battery could be discharged entirely and damaged. Since LiPo batteries should not be discharged below a certain threshold, the usual rule of thumb across the RC community is not to discharge the battery below 3.1 V per cell, it was necessary to report the battery status.

An RGB LED was chosen for that purpose because it is possible to quickly report the various states to the user using different colors. On the other hand, it is unsuitable for visualizing values such as temperature. Besides, the transmitter has its own battery, whose status should also be reported to the user, and that would need 2 LEDs. The single RGB LED was therefore insufficient for the transmitter.

For this reason, the display seemed like a good alternative and replacement. An OLED display was convenient as its power consumption is low, it is readable in direct sunlight, and thanks to only one color, it is fast. The disadvantage is its small size, which complicates the orientation in the displayed data.

Considering all the factors, both the RGB LED and the OLED display were used. RGB LED serves as a fast state notifier, and further status information can be found on the display.

Both car and transmitter use RGB LED with a common anode and monochrome OLED display module with SSD1306 chip communicating using I^2C bus.

Chapter 4
Realization

Since there are no spare parts available for the transmitter or the car, all modifications have been made with as little structural intervention as possible. This process can be referred to as retrofitting.

4.1 Transmitter

4.1.1 Power supply

As described in section 3.1.2, the transmitter is powered by a 2-cell LiPo battery with a nominal voltage of $7.4\,\mathrm{V}$. The battery is connected via an XT60 connector and a switch to a step-down converter module with an LM2596 chip. The module is needed to regulate the battery voltage to $5\,\mathrm{V}$ to power the control board.

This voltage is used for the RGB LED, communication module, and linear voltage regulator AP2111, which regulates it to $3.3\,\mathrm{V}$ used to power the rest of the components. This regulator was selected because of the low dropout voltage of only $250\,\mathrm{mV}$ and high output current of $600\,\mathrm{mA}$ [24].

In order to measure the battery voltage with the processor's internal ADC, it first had to be lowered to fit within the ADC range, which is $0\,\mathrm{V}$ to $3.3\,\mathrm{V}$. Therefore a voltage divider is soldered to the input terminals of the step-down module. The divider was designed to use resistors with standard values and is pictured in figure 11. The resistor values are R1 = $22\,\mathrm{k\Omega}$, R2 = $10\,\mathrm{k\Omega}$.

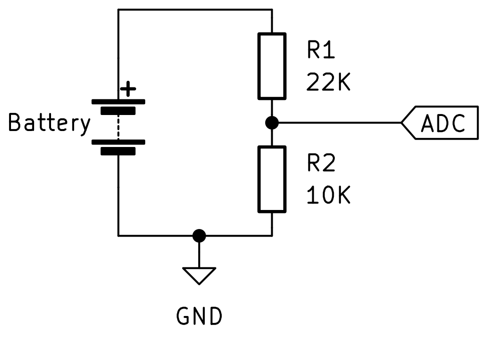


Figure 11: Voltage divider utilized in transmitter

A fully charged 2-cell LiPo battery reaches $8.4\,\mathrm{V}$, then the output of the divider equals

$$U_{adc} = U_{batt} \cdot \frac{R2}{R1 + R2} = 2.625\,\mathrm{V}\,, \tag{2}$$

which is safely within the ADC range.

4.1.2 Controls

The transmitter was supplied with a steering wheel and throttle trigger as $5\,\mathrm{k\Omega}$ potentiometers. Since they are old, a $100\,\mathrm{nF}$ capacitor was added between the output pin of both potentiometers and ground to form a low-pass filter and smooth out the output signal.

Potentiometers used for control trimming were replaced with quadrature rotary encoders with push-buttons. Encoders have screw threads, which fit into the original holes. Since encoders are fixed to the transmitter, a simple connector made from a pin socket was soldered to encoder pins.

4.1.3 PCB design

As the space in the transmitter case was limited, it was decided not to use the perfboard but to create the PCB. It was designed in KiCad and manufactured by JLCPCB. The board consists of connectors, a voltage regulator, an MCU circuit with a crystal oscillator, and an RGB LED driven by N-channel MOSFETs. All traces are rounded to eliminate the possibility of signal reflection. Figure 12 shows the PCB before final soldering to the transmitter, and figure 13 shows the assembled transmitter.

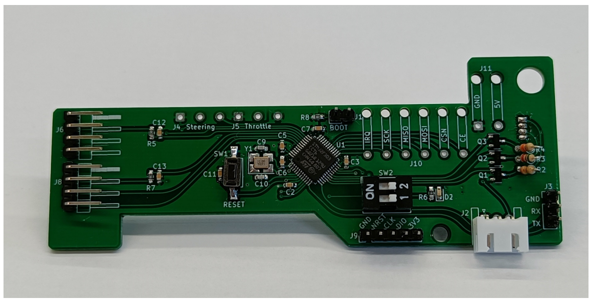
As stated in section 3.4, the prototyping was done using a BlackPill board with a microcontroller STM32F303CCT6. This microcontroller was suitable and was therefore selected also for the final design.

First, the original PCB was measured so that the new PCB could be designed according to these dimensions. A minor modification was done; the new PCB has a cutout on the top to fit the nRF24 communication module.

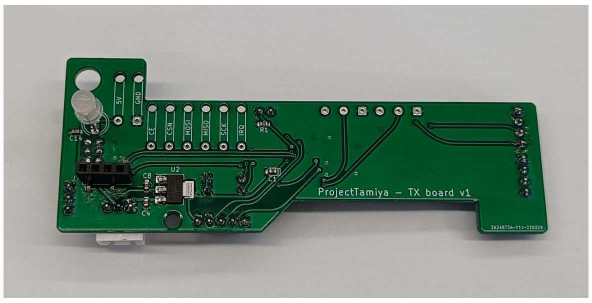
The communication module and control potentiometers are directly soldered to the board. In contrast, the power supply is connected via the JST XH connector, and rotary encoders are connected via simple pin sockets visible on the left side in figure 13a.

Microcontroller circuitry was designed according to Section 7 of [25]. Four $100\,\mathrm{nF}$ ceramic decoupling capacitors were placed as close to the microcontroller as possible. Since the VBAT pin is not utilized, it was connected to the supply voltage (VDD) via another $100\,\mathrm{nF}$ ceramic capacitor. A bigger $4.7\,\mathrm{\mu F}$ decoupling capacitor is placed on the main supply line close to the microcontroller.

The reset circuit was also inspired by [25] and consists of a button and a capacitor connected

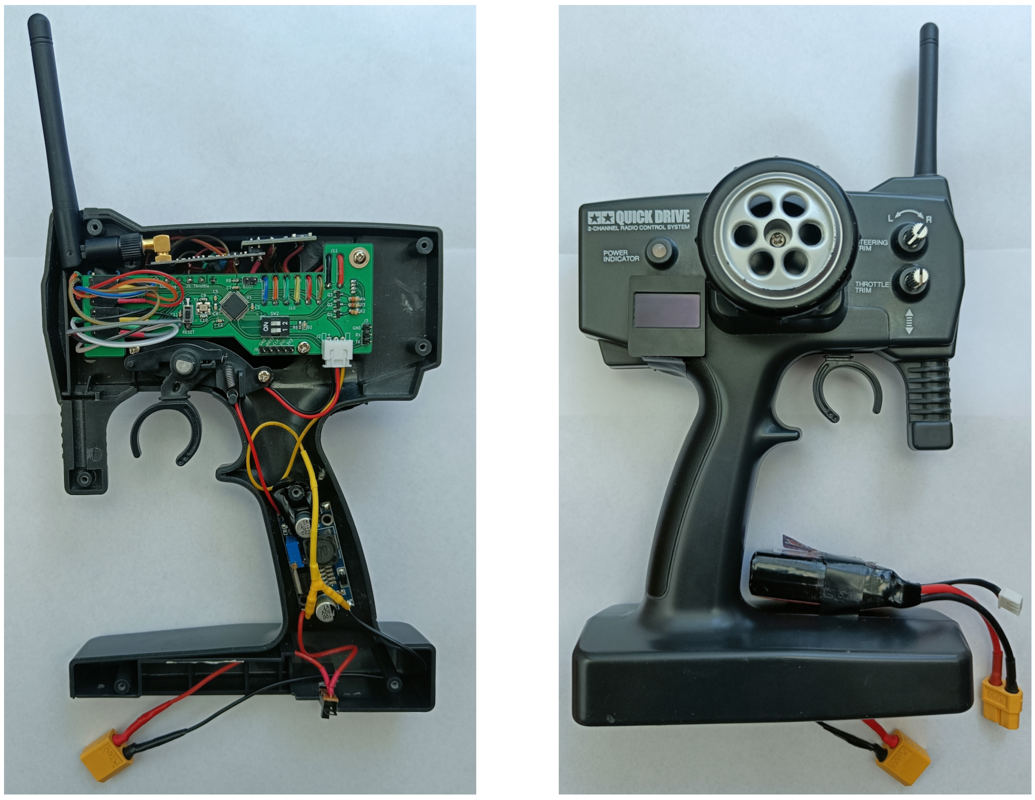


(a) front side



(b) back side

Figure 12: PCB for transmitter



(a) inside view

(b) front side

Figure 13: Assembled transmitter

to reduce the push-button bouncing effect. BOOT0 pin is pulled to the ground with a resistor, but switching the boot mode with a jumper placed on J1 is possible.

External 12 MHz crystal oscillator XT324 with a load capacitance of 20 pF provides the clock for the microcontroller [26]. The circuit of the oscillator is in figure 14. Equation 3 mentioned in Section 3.3 of [27] was used to calculate values of external load capacitances:

$$C_L = \frac{C_{L1} \cdot C_{L2}}{C_{L1} + C_{L2}} + C_S \,, \tag{3}$$

where C_L is the crystal's desired load capacitance, C_{L1} and C_{L2} are values of external capacitors and C_S is the stray capacitance of the printed circuit board and connections. A rough estimate of $C_S = 7\,\mathrm{pF}$ according to Section 6.3.7 of [28] and Section 3.3 of [27] was used for computation. Assuming that $C_{L1} = C_{L2}$, we have

$$C_L - C_S = \frac{C_{L1} \cdot C_{L1}}{C_{L1} + C_{L1}} = \frac{C_{L1}}{2} = 13\,\mathrm{pF} \,, \tag{4}$$

and therefore the external load capacitance should be 26 pF. As this is a non-standard value, 27 pF capacitors were used instead.

The RGB LED resistor values were chosen experimentally to achieve similar luminosity of all channels as no datasheet is available. If all channels are active, the total current through the LED does not exceed the assumed maximal 20 mA.

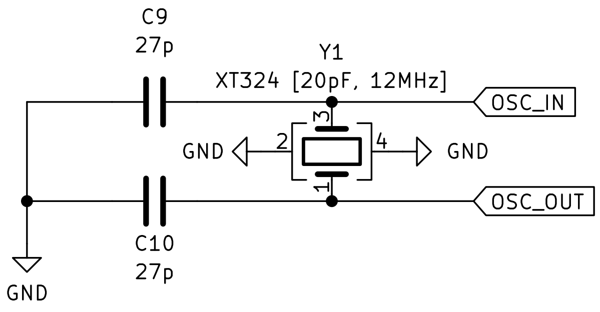


Figure 14: Crystal oscillator circuit

4.2 Car

4.2.1 Power supply

The main power source is a 2-cell LiPo battery discussed earlier in section 3.1.2, which is connected via an XT60 connector to the ESC.

Between the battery and ESC is an ACS712 current measurement module with XT60 connectors on both sides. The module is connected backward as only a positive current is measured. Consequently, the sensor output is within 0.5 to 2.5V, and ADC can safely measure it without extra components.

The control board is powered by ESC's Battery Elimination Circuit (BEC), producing 5 V with a maximum current of 2 A. Therefore ESC's power switch also acts as the main power switch. Moreover, BEC supplies power to the servo, current sensor, and, as with the transmitter, to the communication module and LEDs.

The rest of the electronics is powered by 3.3 V regulated by BlackPill's onboard voltage regulator. Marking on the bottom side of the board claims that the regulator can produce 180 mA of output current, which should be sufficient to power all the components; hence no external voltage regulator was added.

4.2.2 Board design

The Control board for the car was created on the perfboard as it allowed for more flexibility in design and testing, and the space available in the car is quite ample. The schematic was drawn in KiCad. The board looks more complex than the board in the transmitter due to the number of components and consists of necessary connectors, a BlackPill board, an OLED display, an SD card slot, and LEDs.

The main BlackPill board is soldered directly to the perfboard to ensure rigidity. The OLED display and SD card slot were also soldered directly to the board under the BlackPill. Right under the display is a push-button used to wake it up. The UART debug pins are at the bottom left of the display, and to the right is a two-position DIP switch.

Everything not soldered to the board is connected using JST XH connectors. At first, simple pin headers were used as connectors, but the cable was sometimes disconnected from the board since the car moves and generates vibrations. JST XH connectors feature a small locking mechanism to reduce

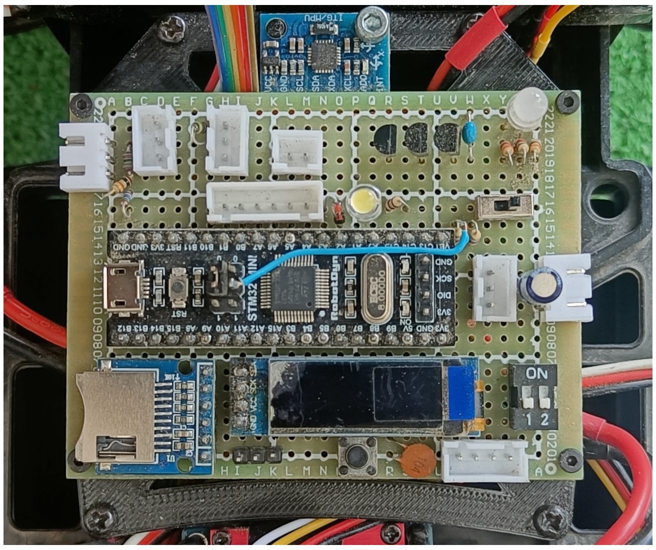


Figure 15: Mounted perfboard for car

the risk of disconnection and are therefore suitable.

The board was designed so that connectors are grouped by type. In the upper left corner are the connectors for measuring the voltage, current, and temperature of the BLDC. More to the right are the connectors for the communication module, 2-pin for power, and 6-pin for SPI communication. To the right of the BlackPill board are connectors for controlling the car; closer to the board is the servo connector, and next to it is the ESC connector, which supplies voltage to the board. The last connector in the lower right corner is for the IMU. The picture of the board is in figure 15.

In the right upper part of the board, there is again an RGB LED with individual channels driven by N-channel MOSFETs. The same resistor values were used since the RGB LED is the same type as the RGB LED in the transmitter. An additional wide-angle white LED controlled by a switch was mounted on the board. The purpose of this LED is to illuminate the car body, so the car is better visible at night.

As the BEC powers the control board with fixed $5\,\mathrm{V}$ output and there is no way to deduce the supply voltage without modifying the ESC, another method had to be found. LiPo batteries use the same JST XH connectors used on the control board to balance the cells when charging. Therefore, the balance connector is advantageously used for battery monitoring. The user only needs to connect the battery balance connector to the appropriate socket on the board.

4.2.3 Parts mounting

The control board was fixed to the chassis simply with tape for the testing phase. As this is not ideal in the long term, it was decided to utilize the original mounting holes present in the chassis and design a frame that would fit into them to avoid drilling.

The frame design was made in *Autodesk's* Fusion360 CAD software and manufactured on the 3D printer. The frame is pretty simple; it serves as an 'adapter' for holes in the chassis and holes in the perfboard. The render is in figure 16. The control board mounting holes are $5\,\mathrm{mm}$ higher than the frame mounting holes to create a small gap.

4.2 Car

There is also a mounting point for the IMU since it should be firmly attached to the chassis for correct measurements. Even though the IMU has two holes, it is mounted with only one screw. That is because the screw head in the second hole would interfere with electrical components on the IMU module.

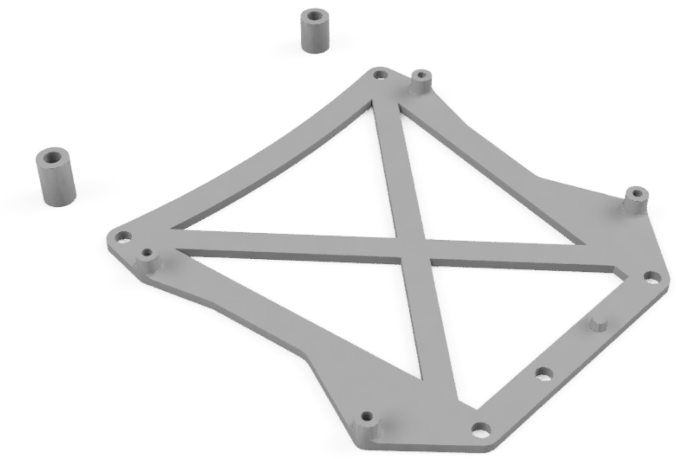


Figure 16: PCB frame

As the mounting holes in the chassis are not at the same height, custom spacers were also designed and 3D printed to level the whole frame. The assembled chassis with the frame is shown in figure 17.

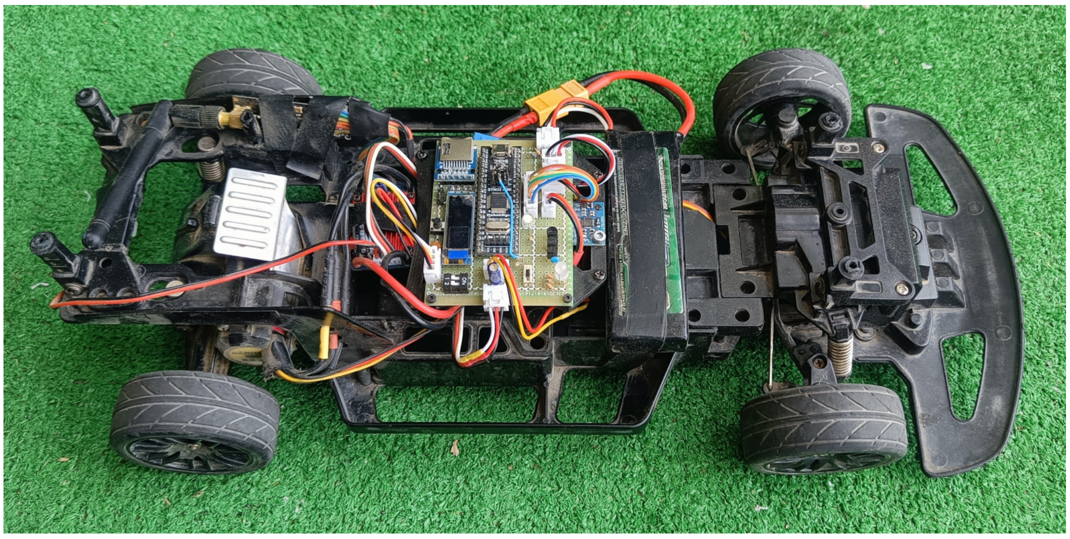


Figure 17: Assembled chassis

Chapter 5
Software

5.1 Introduction

The control program for both the transmitter and the car is written in C language. It utilizes the STM32 HAL (Hardware Abstraction Layer) libraries, and the STM32CubeMX graphical utility was used to set up the necessary initialization and project files and generate code for peripheral configuration. Afterward, the application is compiled using a Makefile under Linux by the GNU GCC cross-compiler for ARM bare-metal targets.

STM32 HAL driver library provides a set of APIs to simplify interaction with all the peripherals and thus the user application implementation. The HAL driver layer implements run-time failure detection to increase robustness. HAL APIs are available for all peripherals. Also, the API ensures high portability across different STM32 families [29].

STM32CubeMX is a graphical tool that provides a convenient user interface to configure the microcontroller. It allows a simple configuration of pin assignments, peripherals, or the clock tree, and the corresponding C initialization code generation. It also packs middleware stacks, such as USB, FatFs, or TCP/IP. This tool notably speeds up the process of microcontroller configuration and allows the user to focus on the application [30].

The control program is almost exclusively interrupt-driven. Emphasis has been placed on program execution speed to ensure a high frequency of control commands and thus smooth car control.

5.2 File structure

Software for both the transmitter and car is in their respective folders. STM32CubeMX generated the code, and to maintain compatibility, the dedicated user code sections were used. The code was generated with CubeMX's basic application structure setting.

Therefore, both project folders in the *'Software'* folder contain the *'Drivers'* folder with ARM and HAL drivers for the board, the *'Inc'* folder with header files, and the *'Src'* folder with source files. The startup file, makefile, linker script, and CubeMX project file are in the root folder. As the car implements FatFs driver, it is located mainly in the *'Middlewares'* folder.

The *'shared_libs'* folder contains shared libraries for both applications, namely the nRF24 module and OLED display libraries, and follows the same file structure; the *'Inc'* folder contains header files and the *'Src'* folder source files. Icons used on the OLED display are in the dedicated *'logos'* folder inside the *'Inc'* folder.

In addition, there are three important files:

- ***'project_tamiya.h':*** Shared header located in the *'shared_libs/Inc'* folder. It contains common libraries and *'platform.h'* file. More importantly, it contains nRF24 communication settings, various macros, and shared function declarations, namely nRF24 and OLED wrapper functions and functions used for UART debug prints.
- ***'project_tamiya.c':*** Source file located in the *'shared_libs/Src'* folder. It consists of function definitions from the appropriate header file.
- ***'platform.h':*** This header file is in both applications' *'Inc'* folder, contains platform-specific macros and settings, and includes platform-specific headers.

The visualization of the file structure with highlighted important files is below.

```
Software
├ Car            Main folder of the car control application
│ └ Inc
│   └ platform.h
├ shared_libs    Shared libraries for both applications
│ ├ Inc
│ │ └ project_tamiya.h
│ └ Src
│   └ project_tamiya.c
└ Transmitter    Main folder of the transmitter control application
  └ Inc
    └ platform.h
```

5.2.1 Communication parameters

Since the communication parameters must be equal in both the car and transmitter to establish the connection, they are placed in the shared *'project_tamiya.h'* header file.

The control packet payload size is only 4 bytes, and the ACK packet payload is only 3 bytes. In addition, 1 byte CRC is used. Payloads are described in figures 18 and 19. CRC and other fields of the *Enhanced ShockBurst* packet are not displayed.

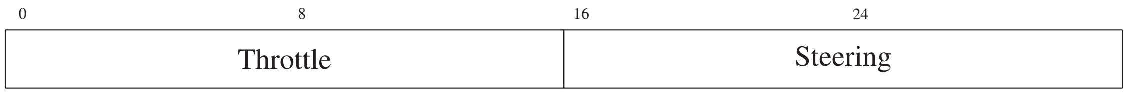


Figure 18: Control payload format

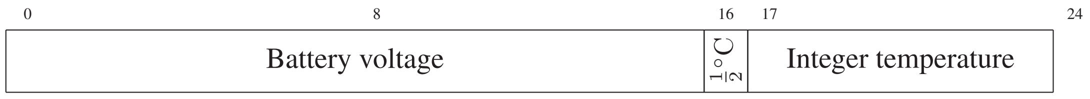


Figure 19: ACK payload format

Bit 16 of the ACK payload is related to the trick described in 5.4.4.

The channel frequency is set to 2515 MHz to reduce interferences with the wi-fi signal, and the data rate is the lowest possible 250 kbit s^{-1}. According to [16], a lower air data rate gives better receiver sensitivity than a higher air data rate. Transmitting power is, on the other hand, the highest possible. This setup ensures the most extended potential range.

5.3 Transmitter

5.3.1 Peripherals and pin configuration

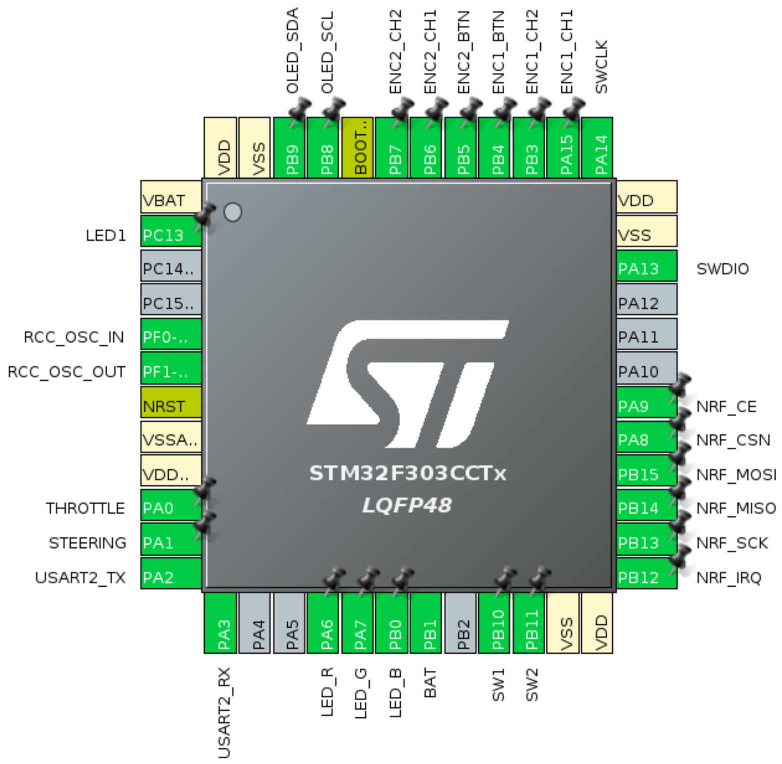


Figure 20: Transmitter pin assignments

The microcontroller runs at 72 MHz with a clock source set to a 12 MHz external crystal oscillator.

ADC1 channels 1 and 2 are used to measure the output voltage from control potentiometers. ADC operates with 12-bit resolution in continuous conversion mode with DMA enabled in circular mode with continuous requests to manipulate the data. Once the ADC conversion is started, it remains active, and DMA repeatedly fills the buffer with measured data. Sampling time is the highest possible 601.5 cycles, and software averaging is used to acquire stable measurements.

ADC3 uses channel 1 to measure the LiPo battery voltage and channel Vrefint to measure the internal reference voltage. Again, the resolution is 12-bit, continuous conversion mode is enabled, and DMA is utilized to transfer data from peripheral to memory. In this mode, the ADC remains active until DMA fills the whole data buffer. The sampling time for channel 1 is 601.5 cycles, and for channel Vrefint 181.5 cycles.

The communication module is connected to the SPI2 bus operating in full-duplex master mode.

Baud rate is set to $9\,\mathrm{Mbit\,s^{-1}}$, and the data size is standard 8 bits. ’NRF_IRQ’ pin is configured in external interrupt mode with rising edge detection. The rest of the ’NRF’ pins are simple outputs.

Quadrature rotary encoders utilize timers TIM2 and TIM4, which are in Encoder mode. This mode is designed specially to interface quadrature encoders with out-of-phase output signals. Encoder push-button pins are configured in external interrupt mode with rising edge detection.

SW1 and SW2 pins belong to the DIP switch and are configured in external interrupt mode with both rising and falling edge detection.

USART2 peripheral is configured as debug serial port. Baud rate is set to $921\,600\,\mathrm{bit\,s^{-1}}$ with 8 bits word length and one stop bit.

I2C1 in Fast Mode Plus with a frequency of $1\,\mathrm{MHz}$ connects the OLED display. LED pins are simple outputs. Furthermore, TIM6 at $1\,\mathrm{Hz}$ and TIM16 at $4\,\mathrm{Hz}$, both timers with interrupts enabled, are used for basic timing purposes.

Figure 20 shows the complete pinout in STM32CubeMX.

5.3.2 Application start

The program starts with auto-generated initialization of all necessary peripherals. Debug UART is enabled after start by default. Afterward, an orange LED visible from the backside of the transmitter is turned on to indicate application initialization.

First, the OLED display is initialized, and the project logo is displayed. At the same time, a greeting is sent via UART, and the RGB LED flashes through each channel. Automatic self-calibration of both ADCs follows, and the Vrefint calibration value is obtained from the appropriate register. This value is later used to calculate VDDA. Then the communication module is initialized in transmitter mode with parameters defined in *’project_tamiya.h’*.

After that, the ADC responsible for reading values of control potentiometers is started in circular DMA mode, as described in section 5.3.1. Processing of the data is explained in section 5.3.4.

In the final part of initialization, all the timers, including the timers used for quadrature encoders, are started in interrupt mode. Then the UART is deinitialized depending on the Switch1 state, the orange LED is turned off to indicate that initialization is done, and the program enters the main control loop.

5.3.3 Main control loop

The control loop is designed to be as non-blocking and interrupt-driven as possible. Therefore, it consists of a set of ’if’ blocks. However, some parts remained blocking. The reason is some of the drivers are blocking and do not allow the use of DMA. Rewriting them was hardly possible or too time-consuming. This is especially true for OLED refreshing, which takes the longest time. Execution of given ’if’ block is controlled by flags, internally boolean values, set by appropriate interrupt routines.

- ***'trigger_data_rdy'*:** The flag is set by ADC interrupt routines and indicates available control values. This block processes and averages measured values from control potentiometers and recalculates them using hardware limits for direct usage in the car. The flag is reset at the end of the block.
- ***'adc_data_bat_rdy'*:** This block is similar to the previous block and indicates available voltage measurements. First, it averages the measured $Vrefint_{meas}$ and battery voltage data. The next step is VDDA calculation:

  $$V_{DDA} = \frac{3.3\,\mathrm{V} \cdot Vrefint_{cal}}{Vrefint_{meas}} \,, \tag{5}$$

  where $Vrefint_{cal}$ is a value acquired during the manufacturing process at $V_{DDA} = 3.3\,\mathrm{V}$ and obtained from the register, and $Vrefint_{meas}$ is measured internal reference voltage. The formula was taken from Section 15.3.32 in [7]. This computed value is used to convert the ADC battery measurement to the actual voltage.

  The corresponding ADC sets the flag, and the program resets it at the end of the block. A $4\,\mathrm{Hz}$ timer triggers the ADC measurement itself.
- ***'sending'*:** This is the first phase of the non-blocking packet transmission routine. It is executed only if the flag is false, indicating that the previous transmission is completed. First, the flag is set, trimming values are added to the control values from potentiometers, and a 4-byte data packet is formed. After that, packet transmission is started. The flag remains set.
- ***'NRF_IRQ'*:** This is the second, final phase of the packet transmission routine. The communication module's interrupt pin sets the flag once it finishes the transmission. The routine starts with checking the transmission status and reading the received payload. If the payload has the expected size, it is parsed into related variables. Then actions depending on the transmission status follow.

  The corresponding counter variable is incremented if the module has reached the maximum retransmit count. This variable is used to detect signal loss as described in section 5.3.5. If the transmission was successful, the previously mentioned variable is reset, the command frequency counter is incremented, and the status LED is toggled.

  The last step is resetting both the *'NRF_IRQ'* and *'sending'* flags.
- ***'display_refresh'*:** This section is responsible for refreshing the information on the OLED display. An interrupt routine of a $4\,\mathrm{Hz}$ timer sets the flag, and the program resets it at the end of the block. Therefore, the display runs with a framerate of about $4\,\mathrm{FPS}$.
- ***'ENCx_IRQ'*:** This section is executed when the rotary encoder's control timer triggers the interrupt routine upon detecting rotation. Depending on the direction of rotation, the trimming value is increased or decreased. The flag is reset at the end of the block.

  Each encoder has its own control block modifying the appropriate trimming value.
- ***'ENCx_reset'*:** This flag is set by the encoder's push-button. The purpose is to reset the appropriate trimming value to zero. At the end of the block, the flag is reset.

5.3.4 ADC data processing

A feature similar to 'double buffering' is implemented to process data from ADC in circular mode. It is not implemented as two separate buffers but as a single 1024 bytes long buffer. It utilizes the ADC's ability to send notifications when the buffer is half-full and when it is full.

ADC is enabled and DMA starts filling the buffer with data. Once the buffer is half-full, the appropriate interrupt is triggered, and MCU is signaled to start processing the first half of the buffer. The DMA still fills the buffer with new data.

When the DMA fills the buffer completely, the appropriate interrupt is triggered, and the DMA starts filling the buffer from the beginning with new measurements. This state is again signaled to the MCU, which starts processing the second half of the buffer while the DMA fills the first half.

Whether the first or the second half is to be processed is controlled by an integer offset that is set in the interrupt routine. Hence, this method is fast as the DMA still fills the same buffer and does not need to be reconfigured, which would add additional overhead.

This approach is repeated, ensuring that the new measurements do not overwrite the values already in the buffer and, therefore, do not affect the value obtained.

5.3.5 Signal quality and loss detection

The nRF24 communication module does not support any detailed form of RSSI (Received Signal Strength Indicator). The only information obtainable from the module is whether the signal during the last transmission was below or under a certain threshold. Therefore, a simple signal quality indicator is implemented in the software.

After every successful packet transmission, the counter variable *'tx_freq_cnt'* is incremented. During the callback routine of a 1 Hz timer, this value is copied into *'tx_freq'* variable and displayed. This value, expressing the number of packets per second, is used to measure the signal quality. Thresholds were determined experimentally based on the maximum observed number of packets per second.

Since the detection of signal loss by the transmitter is not as important as in the case of a car and serves mainly for information purposes, the implementation is quite simple.

It depends on the counter variable *'retransmits_in_row'*, which is incremented whenever the communication module reaches the maximum number of packet retransmits and reset if the transmission was successful. When this variable is greater than 15, it is evaluated as a signal loss in the 4 Hz timer interrupt routine, and the corresponding flag is set.

This status is visualized on the OLED screen and by changing the color of the status LED to blue.

5.3.6 Status visualization

Both the RGB LED and OLED display are used to visualize the current status. The reasons for the choice are discussed in section 3.7.

The RGB LED is clearly visible and serves as a quick notification of the communication and the transmitter's battery status. Color options and their meaning are described in table 4.

The small OLED display is more demanding for quick orientation in the displayed data. On the other hand, it provides more information about the transmitter and even the car. The display interface

Table 4: Explanation of LED status

RGB LED	Communication status	Battery status
flashing green	OK	$> 6.8\,\mathrm{V}$
flashing red	OK	$< 6.8\,\mathrm{V}$
blue	signal lost	–

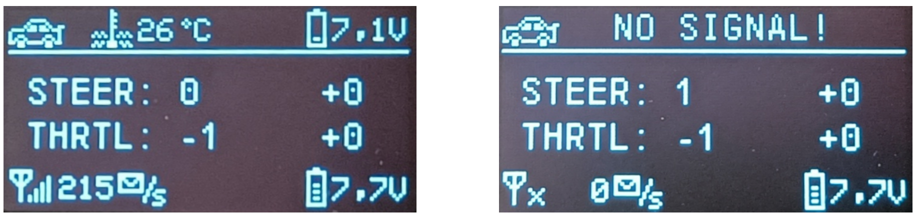
is divided into two main parts.

The first in the upper part of the display resembles a status bar and provides information about the car, as indicated by the icons: the BLDC temperature and battery state. In case of signal loss, this information is displayed here.

The rest of the screen is separated by a line and belongs to the transmitter status. Information about the current position of control potentiometers is displayed in the middle, along with the trim values. The position of potentiometers is not expressed relatively in percentage. Instead, the actual value sent to the car is displayed. The bottom part offers information about packets per second and the transmitter's battery status.

Both battery indicators are interactive and, depending on the voltage, up to 4 bars are displayed. If the battery voltage drops below the threshold of 6.8 V, a flashing exclamation mark is displayed in the battery logo instead. The signal quality indicator is also interactive.

All logos were created in GIMP and exported as XBM files. These were later 'converted' to header files by changing the file extension. Examples of the display interface are in figure 21.



```
26°C            7.1V
STEER:  0        +0
THRTL: -1        +0
215/s           7.7V
```

```
   NO SIGNAL!
STEER:  1        +0
THRTL: -1        +0
0/s             7.7V
```

Figure 21: OLED display interface in transmitter

5.3.7 UART debug prints

One of the UARTs is utilized for printing debug information. Corresponding pins are available on the connector J3; configuration is in section 5.3.1.

The printing is enabled by default after the application start and prints the initialization progress. Afterward, the printing is dependent on the state of the Switch1 (the first lever on the DIP switch SW2).

The UART is deinitialized before entering the main control loop if the switch is in the off position or stays initialized otherwise. If debug prints are enabled, the transmitter informs about:

- LiPo battery voltage and VDDA voltage whenever new data is available,
- encoder rotation,
- switch position change,
- no signal state.

Debug printing can be enabled or disabled anytime during the application execution.

5.4 Car

5.4.1 Peripherals and pin configuration

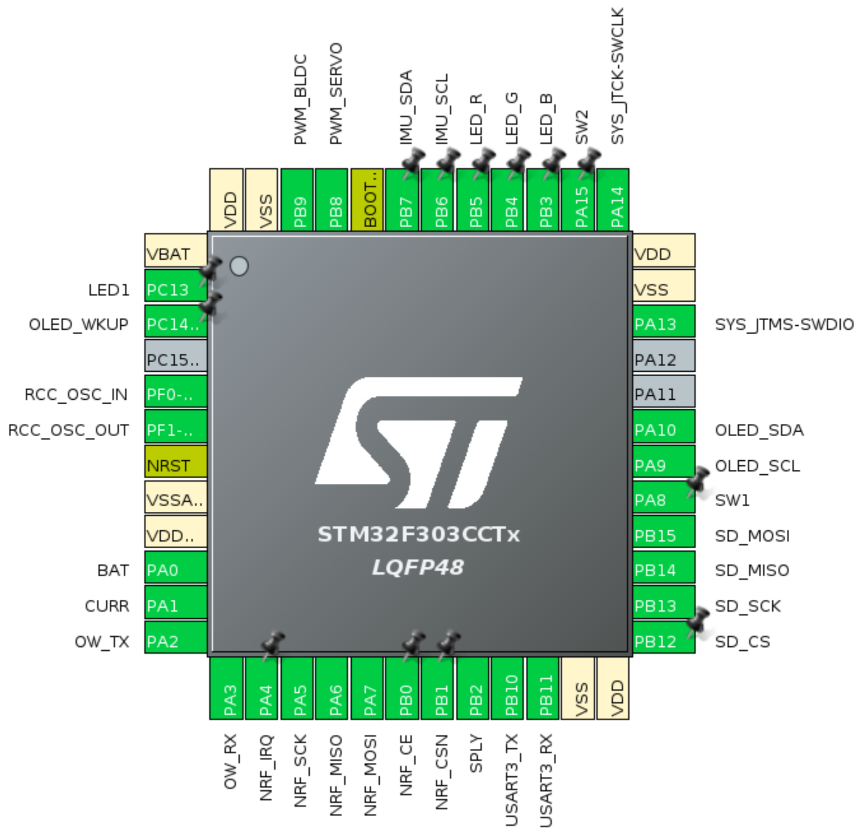


Figure 22: Car pin assignments

Some of the settings are similar to transmitter settings since both microcontrollers are the same STM32F303CCT6. The microcontroller runs at 72 MHz with a clock source set to an 8 MHz external crystal oscillator.

Channels 2 and 3 of timer TIM8 are used to generate PWM and therefore control the car. Prescaler and counter period values are calculated to represent the pulse width in microseconds. That means the counter value of e.g. 1500 equals 1500 µs pulse width, which is convenient for control application implementation.

ADC1 channels 1 and 2 are used to measure the LiPo battery voltage and output voltage from the current sensor. ADC operates with 12-bit resolution in continuous conversion mode with DMA enabled. Sampling time is again the highest possible 601.5 cycles to acquire stable measurements.

ADC2 measures the Vrefint channel to obtain the internal reference voltage and channel 12 to obtain the BEC output voltage. The resolution is also 12-bit, and continuous conversion mode with DMA is enabled. The sampling time for channel Vrefint is 181.5 cycles and 601.5 cycles for channel 12.

The communication module is connected to the SPI1 bus operating in full-duplex master mode. Baud rate is set to $9\,\mathrm{Mbit\,s^{-1}}$, and the data size is standard 8 bits. 'NRF_IRQ' pin is configured in external interrupt mode with rising edge detection. The rest of the 'NRF' pins are simple outputs.

As STM32F303 does not support the SDIO interface, the SD card is connected via the SPI2 peripheral. The baud rate is only $281.25\,\mathrm{kbit\,s^{-1}}$ after reset, but once the SD card is initialized, it is increased to $9\,\mathrm{Mbit\,s^{-1}}$.

I2C2 in Fast Mode Plus with a frequency of $1\,\mathrm{MHz}$ connects the OLED display. IMU unit is served by I2C1 in Fast Mode with a frequency of $400\,\mathrm{kHz}$.

1-Wire bus necessary to communicate with the DS18B20 temperature sensor is implemented using USART2 peripheral and DMA. This method is described in [31], and the DMA version was implemented with the help of [32]. Since RX and TX pins are tied together, and TX pin is configured as push-pull, it is protected with a diode. This method is much simpler and requires fewer external components than the approach in [31] with a discrete open-drain buffer.

SW1 and SW2 pins belong to the DIP switch and are configured in external interrupt mode with both rising and falling edge detection.

USART3 peripheral is configured as debug serial port. Baud rate is set to $921\,600\,\mathrm{bit\,s^{-1}}$ with 8 bits word length and one stop bit.

LED pins are simple outputs. Pins PA11 and PA12 are not used in the project's current state but are reserved for a USB peripheral for possible future expansion with a control computer.

Figure 22 shows the complete pinout in STM32CubeMX.

5.4.2 Application start

The application for controlling the car is written similarly to the transmitter. The program starts with the initialization of all peripherals. Debug UART is enabled after start by default, too, and a blue LED on the BlackPill board indicates application initialization.

First, the OLED display is initialized, and the project logo is displayed. At the same time and a greeting is sent via UART. If the Switch2 is turned on, IMU is initialized, and the flag controlling SD card logging is set. The next step is enabling servo and motor PWMs, initializing 1-Wire, and flashing individual channels of RGB LED.

Automatic self-calibration of both ADCs follows, and the Vrefint calibration value is obtained from the appropriate register. This value is later used to calculate VDDA. Then the communication module is initialized in receiver mode with parameters defined in *'project_tamiya.h'*.

After that, depending on the Switch2 position, the SD card is initialized and mounted. This process is explained in more detail in section 5.4.6.

Lastly, all the timers are started in interrupt mode. Timer TIM15 controlling the display timeout is immediately stopped afterward. This is because the timer triggers an interrupt when started for the first time even though it is not supposed to, which interferes with the display timeout feature.

Then the OLED display is turned off, UART is deinitialized depending on the Switch1 state, and the blue LED is turned off to indicate that initialization is done. The run time in milliseconds is stored just before entering the main control loop for possible later calculation of the data timestamp.

5.4.3 Main control loop

The following section describes the control loop. Similar to the transmitter control loop, it is designed to be as non-blocking and interrupt-driven as possible and consists of a set of 'if' blocks accordingly. The biggest bottleneck is the FatFs driver used for SD card logging, which is too complex to rewrite in DMA mode and takes the longest execution time. Individual 'if' blocks are controlled by their appropriate flag.

- ***'NRF_IRQ':*** The flag is set by the communication module and indicates the module's status change. If the status represents newly received data, an ACK packet is first prepared and sent back to the transmitter to confirm successful reception. If the received data has the expected size, processing follows.

  Received values are checked against the defined chassis mechanical limits and clipped if they exceed the allowed range. After that, values are recalculated to pulse widths and set into PWM generating timers. Lastly, the safety timer is reset, and the command frequency counter is incremented. The flag is reset by the program at the end of the block.
- ***'temp_conv_ready':*** Although it may seem like the temperature sensor controls this flag, it is actually controlled by a 1 Hz timer. The temperature sensor datasheet states that temperature conversion with 12-bit resolution takes 750 ms and, therefore, one-second intervals are sufficient.

  Nevertheless, this section indicates that the temperature conversion is done, and the reading process can be started. If SD logging is enabled, it also flushes cached data to the SD card with the *'f_sync'* function.
- ***'temp_received':*** This is the section controlled by the temperature sensor. The flag is set by a 1-Wire interrupt and indicates that the measured data transfer from the sensor has finished. Thus the raw register data is converted to the actual temperature. The process is mentioned in section 5.4.4.

  Lastly, the flag is reset, and the following temperature conversion is initialized.
- ***'adc_data_bat_rdy'* and *'adc_data_sply_rdy':*** Appropriate ADC callback routines set flags. Both of them must be set to execute this block, which is responsible for processing data. First, measurements from both ADCs are averaged to obtain stable values. The VDDA calculation and conversion of values to the relevant physical quantities follows. Then if SD logging is enabled, data from IMU is read, and all the values are formatted and written to the SD card.

  At the end of the block, both flags are reset. A 50 Hz timer triggers ADCs themselves.

- ***'log_data'* and *'unmount_sd'*:** This block ensures safe unmount and disconnection of the SD card. The file is truncated, closed, and the SD card is unmounted. If Switch2 is turned on after the start of the application, the *'log_data'* flag is set. The *'unmount_sd'* flag is set if it is turned off during execution. Both flags are reset at the end of the block.
- ***'display_refresh'* and *'display_wakeup'*:** This section is responsible for refreshing the information on the OLED display if it is turned on. An interrupt routine of a $4\,\mathrm{Hz}$ timer sets the *'display_refresh'* flag, and the program resets it at the end of the block. The *'display_wakeup'* flag is set for 5 seconds by pressing the push-button under the display, which also activates it.

5.4.4 Integer arithmetics

Although the selected MCU contains the FPU, all program variables are integers. The problem is not the speed of floating-point operations. Floating-point operations are fast thanks to the FPU and are used in some parts of the program, mainly during converting ADC value to actual voltage.

The problem is that formatting floats for output is slow and, in addition, requires a special flag for the linker and increases the final file size considerably. Hence, float variables are multiplied by 1000 to maintain at least some accuracy and saved as an integer. This approach applies mainly to the accelerometer data. The battery voltage value is already calculated in millivolts to avoid float variables altogether.

An easy 'trick' is utilized to display the BLDC temperature as a decimal number without float variables. The raw register data is converted to the integer part of the temperature using bits 11 to 4, according to Figure 4 in [19]. Then the program checks the value of bit 3 associated with $2^{-1} = \frac{1}{2}°\mathrm{C}$ and sets another integer variable to 5 or 0 accordingly. Afterward, the temperature is displayed as two integers with a point between them.

5.4.5 Signal quality and loss detection

It is expected from an RC vehicle that it safely stops or at least stops the motor in case of signal loss to prevent damage or injury. Since pulse width values controlling the servo and motor are only updated when a new control packet is received, loss of the signal would mean that the last command remains set in PWM generating timer and the car uncontrollably drives away.

One of the timers is used as a 'safety timer' to solve this behavior. The timer is set to $4\,\mathrm{Hz}$ with interrupts enabled, and its counter is reset whenever a command packet is successfully received. If no new command is received within $250\,\mathrm{ms}$, the timer triggers an interrupt, resetting the motor and servo to their 'neutral' position and setting the *'no_signal'* flag.

Moreover, the associated interrupt callback routine is the only one that directly interferes with car control, which ensures the motor is always reset after exactly $250\,\mathrm{ms}$ independently of the current program state if no new command is received.

Theoretically, one of the watchdogs could have been used instead of the timer as a more robust safety measure. However, not reloading it in time would mean restarting the MCU and interfering with SD card data logging.

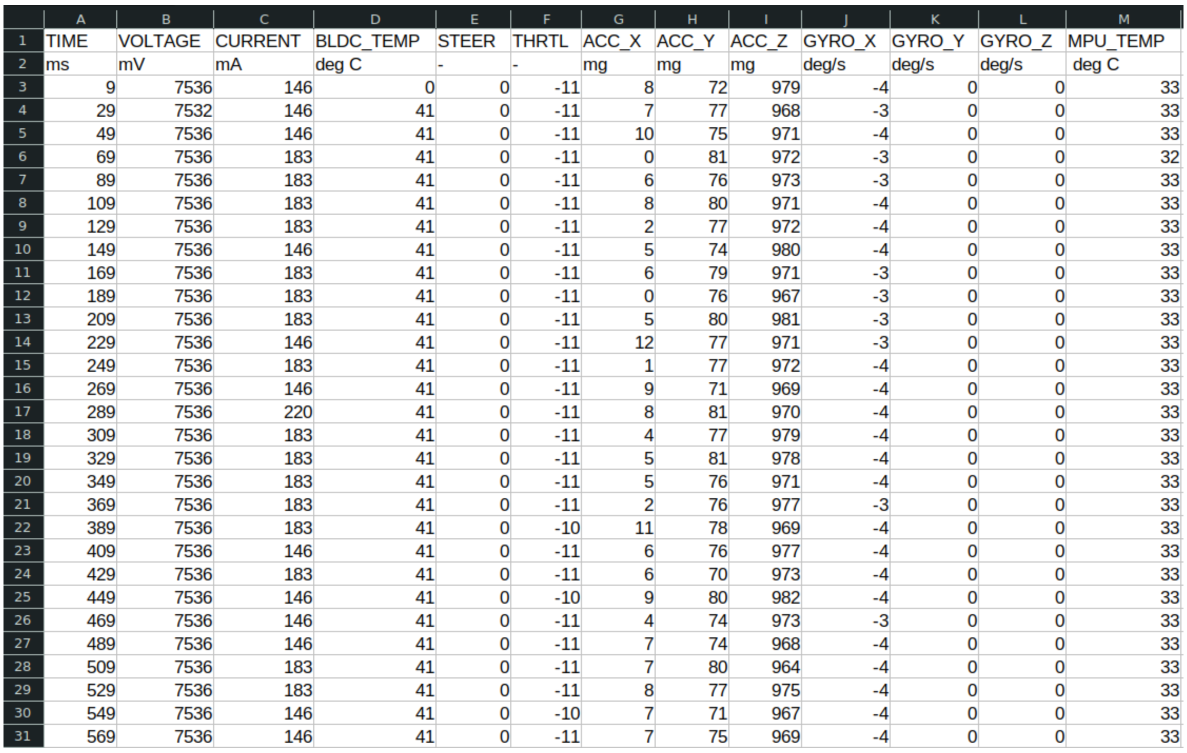
The signal quality indicator is implemented identically to the transmitter. Each time the packet is successfully received, a counter variable is incremented. The value of the counter variable is copied to another variable and reset every second by a 1 Hz timer. This value is used to estimate signal quality, which is then visualized on the OLED display.

5.4.6 SD logging

A micro SD card slot is ready to use to record driving data. The application implements *ChaN's* FatFs driver to interface the SD card, and therefore, any FAT-formatted micro SD card can be used. Driving data are saved with a frequency of 50 Hz in CSV file format.

The logged data includes a timestamp, voltage and current measurements, BLDC temperature, actuator control values, and IMU data. The exact format and corresponding units of quantities are always specified in the first two lines of the CSV file. An example of the data output is shown in figure 23.

Data logging is enabled if Switch2 is turned on during the application start. Unlike UART prints, it cannot be enabled anytime. Depending on the Switch2 position, the SD card mounting and preparation procedure is performed right after communication module initialization.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	TIME	VOLTAGE	CURRENT	BLDC_TEMP	STEER	THRTL	ACC_X	ACC_Y	ACC_Z	GYRO_X	GYRO_Y	GYRO_Z	MPU_TEMP
2	ms	mV	mA	deg C	-	-	mg	mg	mg	deg/s	deg/s	deg/s	deg C
3	9	7536	146	0	0	-11	8	72	979	-4	0	0	33
4	29	7532	146	41	0	-11	7	77	968	-3	0	0	33
5	49	7536	146	41	0	-11	10	75	971	-4	0	0	33
6	69	7536	183	41	0	-11	0	81	972	-3	0	0	32
7	89	7536	183	41	0	-11	6	76	973	-3	0	0	33
8	109	7536	183	41	0	-11	8	80	971	-4	0	0	33
9	129	7536	183	41	0	-11	2	77	972	-4	0	0	33
10	149	7536	146	41	0	-11	5	74	980	-4	0	0	33
11	169	7536	183	41	0	-11	6	79	971	-3	0	0	33
12	189	7536	183	41	0	-11	0	76	967	-3	0	0	33
13	209	7536	183	41	0	-11	5	80	981	-3	0	0	33
14	229	7536	146	41	0	-11	12	77	971	-3	0	0	33
15	249	7536	183	41	0	-11	1	77	972	-4	0	0	33
16	269	7536	146	41	0	-11	9	71	969	-4	0	0	33
17	289	7536	220	41	0	-11	8	81	970	-4	0	0	33
18	309	7536	183	41	0	-11	4	77	979	-4	0	0	33
19	329	7536	183	41	0	-11	5	81	978	-4	0	0	33
20	349	7536	183	41	0	-11	5	76	971	-4	0	0	33
21	369	7536	183	41	0	-11	2	76	977	-3	0	0	33
22	389	7536	183	41	0	-10	11	78	969	-4	0	0	33
23	409	7536	146	41	0	-11	6	76	977	-4	0	0	33
24	429	7536	183	41	0	-11	6	70	973	-4	0	0	33
25	449	7536	146	41	0	-10	9	80	982	-4	0	0	33
26	469	7536	146	41	0	-11	4	74	973	-3	0	0	33
27	489	7536	146	41	0	-11	7	74	968	-4	0	0	33
28	509	7536	183	41	0	-11	7	80	964	-4	0	0	33
29	529	7536	183	41	0	-11	8	77	975	-4	0	0	33
30	549	7536	146	41	0	-10	7	71	967	-4	0	0	33
31	569	7536	146	41	0	-11	7	75	969	-4	0	0	33

Figure 23: Example of raw data

It starts with mounting the disk and printing SD card statistics. Then the application search for the *'logs'* folder, and if it is not found, it creates one. Afterward, the files in the folder are counted to determine the log number, and a new file is created.

Then pre-allocation of 10 MiB of memory follows, which should suffice for approximately an hour-long log file. This step is critical as it dramatically decreases filesystem overhead when writing to the file. Without the pre-allocation, the *'f_sync'* function sometime took so long that the safety timer triggered and reset the motor.

Every second, the log file is *'f_synced'* to lower its execution time and prevent data loss if an unexpected event happens, such as a shutdown. With this approach, the user should not lose more than the last second of data.

Even though the pre-allocation reduced *'f_sync'* execution time significantly, it still sometimes takes more than 100 ms, meaning that some data are missing from the log.

Switch2 must be turned off to finish data logging properly. After that, the file is truncated and closed, and the whole SD card is safely unmounted.

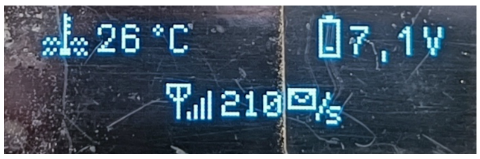
5.4.7 Status visualization

Visualization of the status is consistent with the transmitter and utilizes the RGB LED and a bit smaller OLED display. Again, the RGB LED serves as a quick notification of the communication and the car's battery status. In the case of body shell mounting, it is also visible through it and is the only status indicator. The color scheme is the same as in the transmitter, and thus, the same table 4 applies.

Since the car is usually driven with a body shell mounted, the display is not visible to the user as it is under the body on the control board. For this reason, the display is turned off the majority of the time. It is activated only during the start of the application to display the project logo. The display can be turned on anytime by pressing the push-button under the display, waking it up for 5 seconds.

The display interface is much more plain than the one in the transmitter. Due to the size of the display, the only information presented is the battery voltage, motor temperature, and signal quality indicator with packets per second. Both battery and signal indicators are interactive again.

The signal loss is visualized only by the RGB LED and the zero frequency of the commands on the OLED display; no dedicated error message is shown. Examples of the display interface are in figure 24.



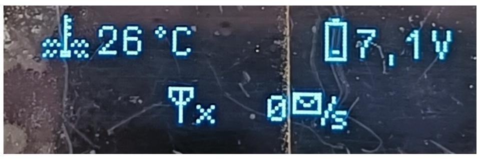


Figure 24: OLED display interface in car

5.4.8 UART debug prints

One of the UARTs is reserved for printing debug information. Corresponding pins are available on the pin header under the display; configuration is in section 5.4.1. The behavior is the same as in the transmitter.

The printing is enabled by default after the application start and prints the initialization progress.

Afterward, the printing is dependent on the state of the Switch1.

The UART is deinitialized before entering the main control loop if the switch is in the off position or stays initialized otherwise. If debug prints are enabled, the car informs about:

- LiPo battery voltage and current, VDDA voltage, 5 V supply voltage, and BLDC temperature whenever new data is available,
- file sync, save, write error if SD logging is enabled,
- switch position changes and push-button presses,
- no signal state.

Debug printing can be enabled or disabled anytime during the application execution.

Chapter 6

Results

6.1 Overall

The built RC system was successfully tested in real-life conditions. The connection between the transmitter and the car is quickly established, even though the antenna is usually hidden under the plastic shell. The driving works flawlessly; the control is responsive and smooth. Trimming encoders can correct any misbehavior, for example, small steering to one side.

Suppose the signal is lost and no command arrives in $250\,\mathrm{ms}$ from the last one. In that case, the dedicated safety-timer resets actuators to their neutral position. The vehicle comes to a slow stop and waits until another command packet is successfully received.

Extra features such as OLED status display with interactive indicators also work as expected and wirelessly provide the user with additional information about the RC car.

Data logging to the SD card is operational, making it possible to record driving data and analyze them afterward. An example of the graphed output data is below in figure 25.

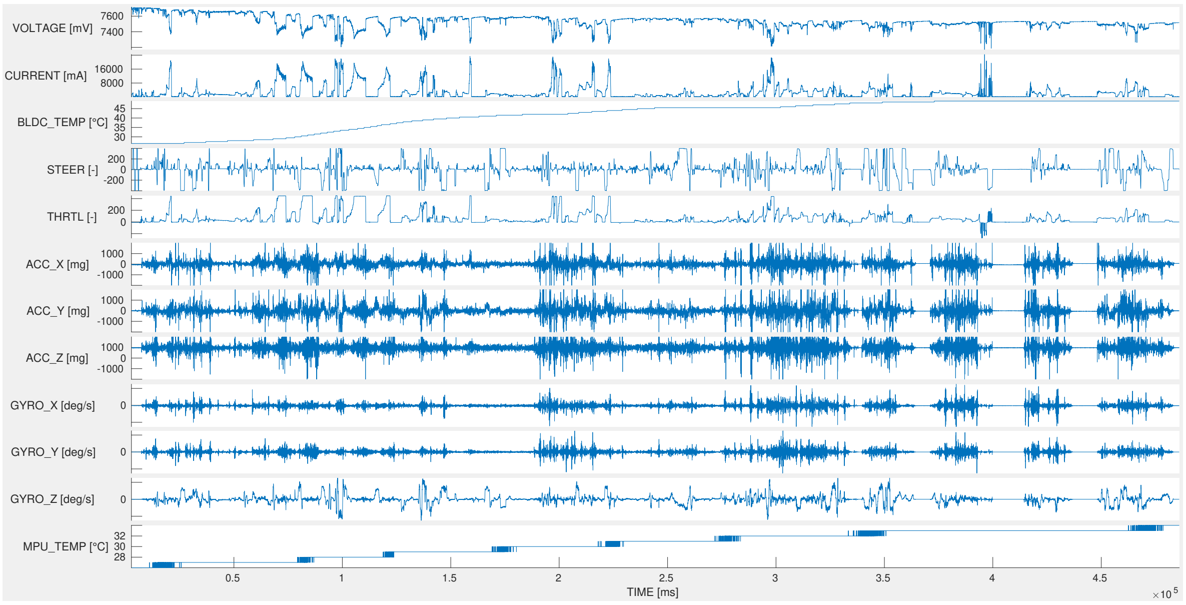


Figure 25: Example of graphed data

The battery life was not measured as it depends on a lot of factors. However, during the testing, the car was able to drive for about an hour. The battery life of the transmitter was not estimated because the battery hardly needed to be charged.

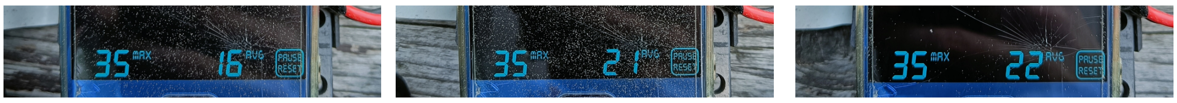
Furthermore, two experiments were realized to obtain maximum speed and communication range.

6.2 Speed

The car's maximum speed was tested in Ladronka park on a flat asphalt cycle path. The speed was measured by GPS and consisted of three test runs. An Honor 9 smartphone running GPS speedometer app DigiHUD was taped to the chassis. Before each run, the app's statistics were reset.

The car accelerated quickly and drove in a straight line with the full throttle until it stopped accelerating visually. Then the throttle was released, and the car performed a U-turn and began accelerating again. This process was repeated at least two times in each run.

Data logging to the SD card was enabled for all three test runs to monitor battery voltage, as the motor's maximum speed depends on its supply voltage. Although the battery voltage dropped by approximately $0.1\,\mathrm{V}$ between runs, during all three of them, the car reached a maximum speed of $35\,\mathrm{km/h}$. Photos of the results of the runs are shown in figure 26.



(a) test run 1 (b) test run 2 (c) test run 3

Figure 26: Results of the speed test

6.3 Range

The range test was also performed in Ladronka park. The test is mainly informative since the car is barely visible at about $250\,\mathrm{m}$. The expected maximum range given by used communication modules was $800\,\mathrm{m}$; therefore, the distance was calculated from GPS coordinates.

The test utilized the vehicle's signal loss detection feature. The transmitter was configured to send the steering command to keep the front wheels turned. The signal loss was easily recognizable since the signal loss detection feature resets the motor and straightens the servo. Both antennas were in a vertical position during the test for the best signal reception.

First, coordinates of the transmitter's initial position were acquired. Then the chassis was moved in a straight direction to keep the transmitter in sight until the signal began to fade.

Table 5: GPS coordinates and distances from the initial position

	GPS coordinates	**Calculated distance**
initial position	50.0792036N, 14.3716383E	–
occasional outages	50.0790850N, 14.3663733E	$375.9\,\mathrm{m}$
frequent outages	50.0793289N, 14.3631853E	$603.3\,\mathrm{m}$
signal 'lost'	50.0794572N, 14.3609497E	$763.2\,\mathrm{m}$

After $375\,\mathrm{m}$, the servo started to straighten occasionally, indicating that no command packet was received within $250\,\mathrm{ms}$ from the last one. Even though the safety-timer sometimes reset the car, it was still controllable.

After $603\,\mathrm{m}$ from the initial position, the signal suffered frequent outages. The car was in a 'no signal' state more than 50% of the time, making the control difficult.

The maximum range was measured at $763\,\mathrm{m}$. At this distance, the signal was not completely lost. Infrequently, the command packet was successfully received, but the car remained in a 'no signal' state most of the time, making the control almost impossible.

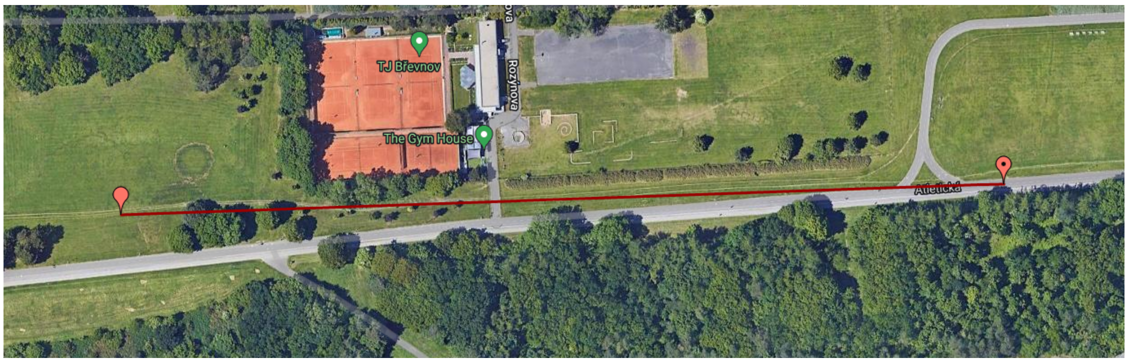


Figure 27: Controllable distance

GPS coordinates along with the calculated distance from the initial position are in table 5. The measured control distance is visualized in figure 27.

Chapter 7

Conclusion

The thesis aimed to utilize the STM32 microcontrollers to develop and control an RC car. STM32 families of microcontrollers were summarized, and their architecture and the most commonly used peripherals were briefly described.

The whole system was developed from scratch, aside from the chassis and other plastic parts. It utilizes readily available parts, including an RC servo and BLDC motor with the controller, sensor boards, and modules. These functional blocks were introduced, and their purpose and actual use were explained. Since it is an RC car, meaning 'Radio Controlled', the control transmitter was developed as well.

A custom control board was designed for both platforms. The car's control board is made on a perfboard and consists primarily of modules and connectors. This approach is simple, quick, flexible, and certainly sufficient for the use case. However, space constraints inside the transmitter asked for a custom PCB design that would provide better space utilization. Therefore, the circuit board was designed according to the original, slightly modified to fulfill current needs, and manufactured.

The resulting RC system was to be unconventional in terms of available features. Accordingly, both the transmitter and the car received an OLED display and RGB LED to report the status. In addition, the car is equipped with various sensors. A micro SD card slot is available to record all the data.

After the hardware design, a control software had to be developed. As the software controls real devices in real time, it is primarily interrupt-driven and implemented with an emphasis on speed to guarantee periodic execution of the control loop and thus smooth control of the car. Nevertheless, it is also designed to be user-friendly and encompasses interactive battery voltage and signal strength indicators that should simplify navigation on display along with other icons.

At the end of the thesis, the RC system is tested in real conditions with successful results. Besides the essential RC car functions, all the unconventional functions also proved to work. The control is responsive. The car's top speed is 35km/h, and the maximum controllable range is 375m.

All the schematics, source code, and 3D models are available at a GitHub repository[3] and on the attached DVD.

Even though the feature is not implemented in the project's current state, microcontroller pins PA11 and PA12 on the car's control board are reserved for the USB peripheral. Therefore, a single-board computer like RaspberryPi running a control software can possibly control the car through the USB, and the chassis can, for example, become a platform for testing autonomy.

The future work can possibly be replacing the commercial ESC controlling the motor with some custom-made solution that would offer a more advanced motor control algorithm. This upgrade would increase the top speed, prolong battery life, and provide the user with more driving data.

[3]https://github.com/lunakiller/project_tamiya

Another future step would be replacing the present 6-axis IMU with a 9-axis IMU with an incorporated magnetometer to obtain the precise position in space and utilize those data for control or mapping purposes. One possibility would be to utilize those data to implement ’stability control’ to help steer the car in a skid or at high speeds, where the car is tricky to control.

References

[1] Joseph Yiu. *Definitive Guide to ARM (R) Cortex (R)-M3 and Cortex (R)-M4 Processors*. Elsevier Books, third edition, 2013. ISBN 0124080820.

[2] ARM Developer: Cortex-M0. [online] https://developer.arm.com/Processors/Cortex-M0. Accessed: 2022-05-17.

[3] ARM Developer: Cortex-M0+. [online] https://developer.arm.com/Processors/Cortex-M0+. Accessed: 2022-05-17.

[4] ARM Developer: Cortex-M7. [online] https://developer.arm.com/Processors/Cortex-M7. Accessed: 2022-05-17.

[5] ARM Developer: Cortex-M33. [online] https://developer.arm.com/Processors/Cortex-M33. Accessed: 2022-05-17.

[6] STMicroelectronics: STM32F303CC Circuit Diagram. [online] https://www.st.com/en/microcontrollers-microprocessors/stm32f303cc.html#overview. Accessed: 2022-01-14.

[7] STMicroelectronics. RM0316: STM32F303 Reference manual. [online] https://www.st.com/resource/en/reference_manual/rm0316-stm32f303xbcde-stm32f303x68-stm32f328x8-stm32f358xc-stm32f398xe-advanced-armbased-mcus-stmicroelectronics.pdf, 2017. Accessed: 2022-01-11.

[8] Tamiya model database - 46028. [online] https://tamiyabase.com/tamiya-models/46028. Accessed: 2022-05-05.

[9] Bighobby - NANO Tech 2200mAh 2S 35C (70C). [online] https://www.bighobby.cz/bighobby-nano-tech-2200mah-2s-35c--70c-/, . Accessed: 2022-05-05.

[10] Bighobby - NANO Tech 900mAh 2S 25C (50C). [online] https://www.bighobby.cz/bighobby-nano-tech-900mah-2s-25c/, . Accessed: 2022-05-05.

[11] Wikimedia Commons. Servomotor timing diagram. [online] https://commons.wikimedia.org/wiki/File:Servomotor_Timing_Diagram.svg, 2021. Accessed: 2022-05-06.

[12] Emax ES3001 37g Plastic Analog Servo For RC Model. [online] https://emaxmodel.com/products/emax-es3001-37g-plastic-analog-servo-for-rc-model. Accessed: 2022-05-19.

[13] STM32-base - RobotDyn Black Pill. [online] https://stm32-base.org/boards/STM32F303CCT6-RobotDyn-Black-Pill.html. Accessed: 2022-05-06.

[14] STM32-base - Blue Pill. [online] https://stm32-base.org/boards/STM32F103C8T6-Blue-Pill. Accessed: 2022-05-06.

[15] GM electronic. RF24L01+PA+LNA datasheet. [online] https://www.gme.cz/data/attachments/dsh.775-034.1.pdf. Accessed: 2022-05-13.

[16] Nordic Semiconductor. nRF24L01+ Preliminary Product Specification. [online] https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf, 2008. Accessed: 2022-05-13.

[17] Hadex. Proudový senzor 20ADC ACS712. [online] https://www.hadex.cz/m439a-proudovy-senzor-20adc-acs712/. Accessed: 2022-05-13.

[18] Allegro MicroSystems. ACS712 Datasheet. [online] https://www.allegromicro.com/-/media/files/datasheets/acs712-datasheet.pdf, 2022. Accessed: 2022-05-07.

[19] Maxim Integrated. DS18B20 Datasheet. [online] https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf, 2019. Accessed: 2022-05-06.

[20] GM Electronic. Číslicový teplotní sensor, -55..125°C, THT, TO-92 DS18B20+. [online] https://www.gme.cz/ds18b20. Accessed: 2022-05-13.

[21] InvenSense. MPU-6000/MPU-6050 Product Specification. [online] https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf, 2013. Accessed: 2022-05-07.

[22] Makerlab Electronics. Triple Axis Accelerometer and Gyro Breakout – MPU6050. [online] https://www.makerlab-electronics.com/product/triple-axis-accelerometer-gyro-breakout-mpu6050/. Accessed: 2022-05-07.

[23] I2Cdevlib: MPU-6050 6-axis accelerometer/gyroscope. [online] https://www.i2cdevlib.com/devices/mpu6050. Accessed: 2022-05-07.

[24] BCD Semiconductor Manufacturing Limited. AP2111 Datasheet. [online] https://cz.mouser.com/datasheet/2/115/DiodesInc_10212021_AP2111-2636031.pdf, 2012. Accessed: 2022-05-07.

[25] STMicroelectronics. AN4206: Getting started with STM32F3 series hardware development. [online] https://www.st.com/resource/en/application_note/an4206-getting-started-with-stm32f3-series-hardware-development-stmicroelectronics.pdf, 2015. Accessed: 2022-05-08.

[26] YIC. XT324 Datasheet. [online] https://www.tme.eu/Document/9f9ff37b85cc93101e405f379d11e905/YIC-XT324.pdf. Accessed: 2022-05-08.

[27] STMicroelectronics. AN2867: Oscillator design guide for STM8AF/AL/S, STM32 MCUs and MPUs. [online] https://www.st.com/resource/en/application_note/cd00221665-oscillator-design-guide-for-stm8afals-stm32-mcus-and-mpus-stmicroelectronics.pdf, 2021. Accessed: 2022-05-08.

[28] STMicroelectronics. DS9118: STM32F303xB, STM32F303xC Datasheet. [online] https://www.st.com/resource/en/datasheet/stm32f303vc.pdf, 2018. Accessed: 2022-05-08.

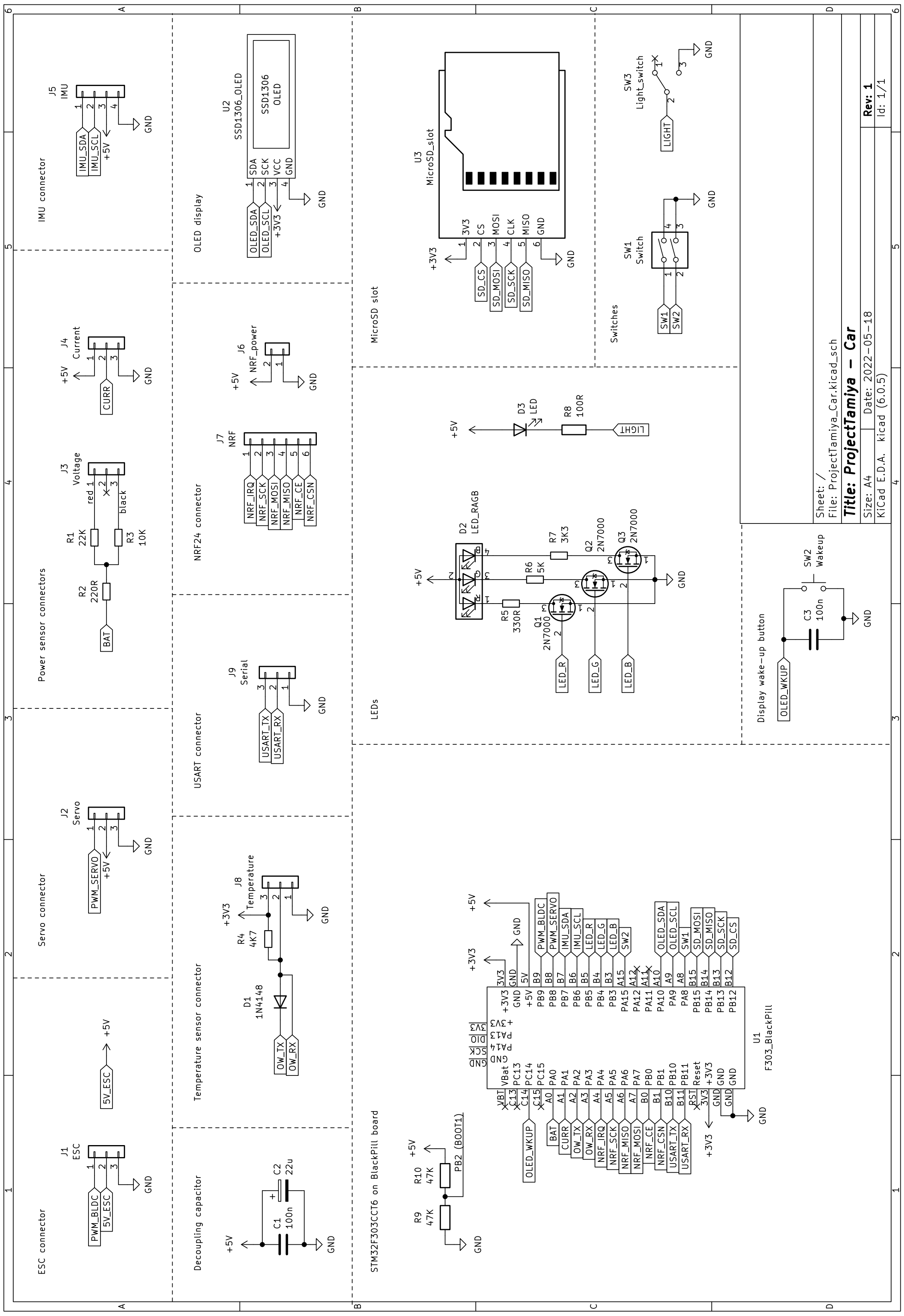
[29] STMicroelectronics. UM1786: Description of STM32F3 HAL and low-layer drivers. [online] https://www.st.com/resource/en/user_manual/um1786-description-of-stm32f3-hal-and-lowlayer-drivers-stmicroelectronics.pdf, 2020. Accessed: 2022-05-09.

[30] STM32Cube initialization code generator. [online] `https://www.st.com/en/development-tools/stm32cubemx.html`. Accessed: 2022-05-09.

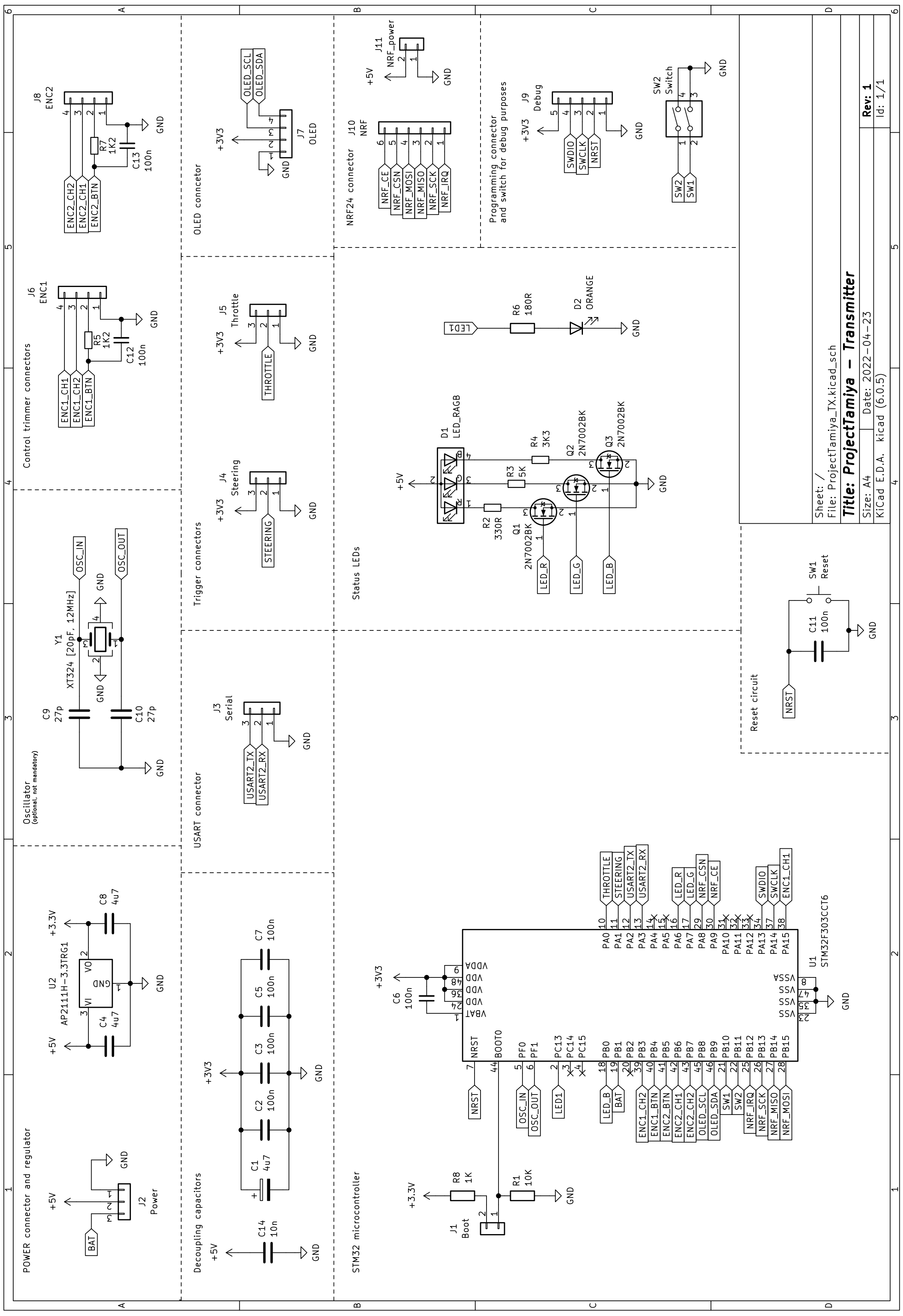
[31] Maxim Integrated. Using a UART to Implement a 1-Wire Bus Master. [online] `https://www.maximintegrated.com/en/design/technical-documents/tutorials/2/214.html`, 2020. Accessed: 2022-05-12.

[32] STM32 - 1-Wire protocol analysis & Implementing of OneWire Protocol using UART peripheral and DMA. [online] `https://cnnblike.com/post/stm32-OneWire/`, 2016. Accessed: 2022-05-12.

Appendix A
Schematics



A. Schematics



Appendix B
Supplementary material

The attached DVD shares the same file structure as mentioned GitHub repository and is described below. Only the folders have been compressed into zip archives.

File	Contents
Software.zip	source codes and drivers for both control applications, shared libraries
Hardware.zip	KiCad schematics and gerber files
3D_printed_parts.zip	STL models for 3D printing
Logs.zip	Two example logs in CSV format