

VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

WiFi Module for RC Transmitter

WiFi Modul pro RC Vysílač

Bachelor Thesis Assignment

Student: **Jiří Fišer**

Study Programme: B2647 Information and Communication Technology

Study Branch: 2612R025 Computer Science and Technology

Title: WiFi Module for RC Transmitter
WiFi Modul pro RC Vysílač

The thesis language: English

Description:

The aim of this work is to design an extension for an RC transmitter, adding Internet connectivity through a WiFi module. The task is to design a protocol for communication between the WiFi module and a JETI DC/DS transmitter, to create an application in the transmitter to configure the WiFi module, generate web pages and download data from the Internet. The firmware of the module based on an ESP32 will be modified to allow communication between the transmitter and simultaneously generate webpages. The firmware will provide a webpage in HTML/JavaScript for displaying telemetry information from the transmitter.

Thesis outline:

1. State of the art of the functions of the current high-end RC transmitters.
2. Description of the DC-24 hardware platform and the ESP32 module.
3. Design of the ESP32 firmware, custom communication protocol and the appropriate OS components.
4. Implementation of the proposed design and generation of the web page content.
5. Stress testing of the implementation, results summary.

References:

- [1] Dogan I., Ahmet I.: The Official ESP32 Book, Elektor International Media, 2017. ISBN 978-1907920639
- [2] Tanenbaum, A.S., Bos, H.: Modern Operating Systems (4th Edition), Pearson, 2014. ISBN 978-0133591620
- [3] Wang, K.C.: Embedded and Real-Time Operating Systems, Springer, 2017. ISBN 978-3319515168

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **Mgr. Ing. Michal Krumnikl, Ph.D.**

Date of issue: 01.09.2018

Date of submission: 30.04.2019



Jan Platoš

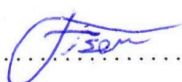
doc. Ing. Jan Platoš, Ph.D.
Head of Department

ka

prof. Ing. Pavel Brandštetter, CSc.
Dean

I hereby declare that this bachelor's thesis was written by myself. I have quoted all the references I have drawn upon.

Ostrava, 29 April 2019

.....


I would particularly like to thank Mgr. Ing. Michal Krumnikl, Ph.D., for willingly supervising my thesis, and for all his help during the making of this work.

I am also deeply grateful to my coworkers at JETI model for providing me with the idea and hardware platform for this work, and the knowledge they gave me.

Abstrakt

Tato bakalářská práce je zaměřená na návrh a výslednou implementaci bezdrátového síťového modulu pro RC vysílač, konkrétně modulu ESP32. Ve výsledku se bude vysílač moct připojit na místní WiFi síť, poskytovat svůj vlastní přístupový bod a hostovat server s webovou stránkou, která bude na koncovém zařízení zobrazovat aktuální informace o telemetrii z vysílače.

Klíčová slova: ESP32, IoT, RC vysílač, WiFi, embedded, web server

Abstract

This thesis focuses on the design and implementation of a wireless network module embedded into a RC transmitter, specifically an ESP32 network module. In result, the transmitter should be able to connect to a local WiFi network, provide its own Access Point, and host a web server with realtime display of useful telemetry data from the transmitter.

Key Words: ESP32, IoT, RC transmitter, WiFi, embedded, web server

Contents

| | |
|---|-----------|
| List of symbols and abbreviations | 8 |
| List of Figures | 9 |
| List of Tables | 10 |
| 1 Introduction | 11 |
| 2 State of the art of the current high-end RC transmitters | 12 |
| 2.1 Graupner MZ-32 | 13 |
| 2.2 Spektrum iX12 | 13 |
| 2.3 JR-28X | 14 |
| 2.4 XPAD3 | 14 |
| 3 Description of the JETI DC-24 transmitter | 15 |
| 3.1 Technical specifications | 15 |
| 3.2 Computational capabilities | 17 |
| 3.3 Operating system | 18 |
| 3.4 Lua programming API for DC/DS-24 | 19 |
| 4 Description of the ESP32 network module | 21 |
| 4.1 ESP32 board in the RC transmitter | 22 |
| 5 Design of the SW components and the used ESP32 firmware | 23 |
| 5.1 Lua programming language | 24 |
| 5.2 AT firmware | 25 |
| 5.3 Communication interface | 26 |
| 6 Implementation and testing of the proposed design | 27 |
| 6.1 WiFi Settings app | 27 |
| 6.2 WiFi module interface | 29 |
| 6.3 Weather app | 34 |
| 7 Web server with realtime telemetry display | 35 |
| 7.1 Enabling and connecting to the web server | 35 |
| 7.2 Processing client requests | 36 |
| 7.3 Generating and displaying dynamic content on the web page | 37 |
| 7.4 Web server limitations and responsivity | 38 |
| 8 Summary | 39 |

List of symbols and abbreviations

| | |
|-----------|--|
| AP | – Access Point |
| API | – Application Programming Interface |
| BLE | – Bluetooth Low Energy - Wireless personal area network technology |
| DNS | – Domain Name System |
| GPIO | – General-Purpose Input/Output |
| GUI | – Graphical User Interface |
| HTTP | – Hypertext Transfer Protocol |
| IoT | – Internet of Things |
| IP | – Internet Protocol |
| JSON | – JavaScript Object Notation - file format for data storage |
| Lua | – Lightweight programming language |
| MAVLink | – Micro Air Vehicle Link - unmanned vehicle communication protocol |
| PSK | – Pre-shared key |
| RC | – Radio Control |
| RF | – Radio Frequency |
| SSID | – Service Set Identifier - WiFi network name |
| TCP | – Transmission Control Protocol |
| TFT | – Thin-film transistor |
| UAV | – Unmanned Aerial Vehicle |
| UI | – User Interface |
| USART | – Universal Synchronous/Asynchronous Receiver/Transmitter |
| WEP, WPA2 | – WiFi security protocols |
| WiFi | – Wireless Fidelity |

List of Figures

| | | |
|----|--|----|
| 1 | Graupner MZ-32 [1] | 12 |
| 2 | Spektrum iX12 [2] | 12 |
| 3 | JR-28X [3] | 14 |
| 4 | XPAD3 [4] | 14 |
| 5 | JETI DC-24 transmitter | 15 |
| 6 | Internal configuration of the DC-24 | 16 |
| 7 | Main screen | 18 |
| 8 | Artificial horizon app [7] | 18 |
| 9 | Model setup | 18 |
| 10 | Cooperative multiplayer game | 18 |
| 11 | User applications menu | 19 |
| 12 | ESP32 development board | 21 |
| 13 | Final board with the ESP32 module, to be serially produced for the DC-24 | 22 |
| 14 | Internal GPIO pins on the rear port for the expansion board | 22 |
| 15 | Layout of the SW components | 23 |
| 16 | Lua logo | 24 |
| 17 | Command line interface for configuring the AT firmware | 25 |
| 18 | List of networks | 27 |
| 19 | WiFi settings | 27 |
| 20 | Network information | 28 |
| 21 | Initialization sequence | 31 |
| 22 | Weather screen | 34 |
| 23 | Forecast screen | 34 |
| 24 | Web page in a desktop browser | 35 |
| 25 | Network monitor | 38 |

List of Tables

| | | |
|---|---|----|
| 1 | Technical specifications of the DC-24 | 15 |
| 2 | Expansion board peripherals | 22 |
| 3 | Public methods of the interface | 29 |

1 Introduction

With ever increasing feature demands for high-end Radio Control (RC) transmitters, manufacturers of high quality electronics for modelers must work hard against worldwide competition in order to keep up with the quickly evolving market. This demand springs from RC model pilots who want to get the most out of their flight experience and set the highest records in all kinds of hobbyist competitions. A high-end RC transmitter with maximum reliability and cutting edge features can easily make the difference between loss and victory in the hands of a skilled pilot.

In fashion of the rapid emergence of Internet of Things (IoT), radios with WiFi and Bluetooth modules have started appearing on the market. Such addition can enrich the radio of several intriguing features, such as wireless headphones connectivity, realtime flight telemetry display, automatic firmware updates or First-Person View (FPV) for streaming image from a forward facing camera mounted on the aircraft. Other high-end radios implementing these features are briefly studied in section 2.

The aim of this work is a custom implementation of an ESP32 network module for a DC-24 transmitter, manufactured by JETI model s.r.o. in Czech Republic. This module is often used in IoT applications for implementing WiFi and Bluetooth connectivity (further described in section 4). It will make the transmitter able to connect to a WiFi network, provide an Access Point, and ultimately host a server on a given IP address with realtime telemetry data being displayed on the web page, which is described in section 7.

Another important goal of this work is understanding communication between the transmitter and an added networking module (through AT commands), and processing multiple clients connected to a server provided by the module.

2 State of the art of the current high-end RC transmitters

Currently, several other RC transmitters with WiFi or Bluetooth capabilities exist on the high-end market. With that in mind, I have decided to work on my own approach to implementing a WiFi module for the JETI DC-24 transmitter, due to its superior reliability, construction quality, ease of use and openness of the platform.

Before I started with my work on implementing the WiFi module, I decided to study other high-end RC transmitters first, in hopes of gaining any useful insight into this area, or finding other transmitters that I could draw any useful inspiration from. Below are four examples of high-end RC transmitters that I found intriguing enough and worthy of mentioning.

While thoroughly searching the market, I did not find any radio that could be considered similar to the JETI DC-24 after it becomes upgraded with the WiFi module, or even provide access to flight telemetry over web, accessed through a WiFi hotspot. Therefore, my implementation is one of its kind.



Figure 1: Graupner MZ-32 [1]



Figure 2: Spektrum iX12 [2]

2.1 Graupner MZ-32

The Graupner MZ-32 (figure 1) is a feature-rich RC transmitter with a 4.3" TFT colour touchscreen. It is the first commercially available RC transmitter with 32 proportional channels in its default configuration, without any extensions. Five different model types can be controlled with this radio: electric, gas or turbine airplanes, helicopters, multirotor aircraft, cars and even boats.

Reliability of the transmitter is greatly improved with its dual redundant 2.4 GHz antenna, and a 9000 mAh LiPo battery safely ensures 10 hours of continuous operation. The gimbals have quad bearing Hall sensors with a resolution of 4096 steps for a smooth flight experience.

The user interface is powered by a custom operating system and provides highly customizable telemetry screens with user selected widgets. The transmitter has an integrated Bluetooth module for connecting with wireless headsets, which can be used for hearing voice alerts or listening to music. Connectivity of the transmitter is further extended with a WiFi module which can be used only for downloading and installing firmware updates.

2.2 Spektrum iX12

The iX12 RC transmitter (figure 2) developed and manufactured by SpektrumTMsports a 4" full-colour touchscreen with an intuitive Android interface and a dedicated quad-core processor. It is also equipped with a Bluetooth module for pairing with wireless headphones or speakers, and thanks to its WiFi module the iX12 gains a much greater amount of features, compared to other transmitters which do not have any networking capabilities.

Users can easily download new apps through a Google Play service and its touchscreen provides a very user-friendly graphical programming interface. WiFi connectivity can be used for streaming music to the device or automatically downloading firmware updates and model setups. However, Android is not always an ideal platform for this purpose due to very little user control over the system, unpredictable reliability and low security.



Figure 3: JR-28X [3]



Figure 4: XPAD3 [4]

2.3 JR-28X

The JR-28X is another RC transmitter with an Android-powered system, featuring a 480 x 273 pixel 4.3" WQVGA-TFT full-colour touch screen with 16-bit colour depth (figure 3). The operating system is powered by Android and allows programming and graphical customizing of telemetry widgets on the home screen. It is the first RC transmitter to feature a 16-bit stick resolution, which was at the time of its inception 16 times more precise than any other RC transmitter.

The transmitter is able to connect to nearby WiFi networks and access the internet through a web browser. User can freely download files to the transmitter and browse them through a file manager. Bundled applications also include an image gallery, video and music player.

2.4 XPAD3

The XLR3 D3, or XPAD3 (figure 4), is a professional RC transmitter that is especially suitable for long range UAV flights. It disposes with a power of 100 to 1000 mW and has a range of up to 200 km and a large 8000 mAh battery. Flight information is displayed on three monochrome displays. For communicating flight telemetry it uses an internal radiomodem with a MAVLink¹ protocol, as a paid optional extension. WiFi and Bluetooth modules are a part of this transmitter by default.

¹Micro Air Vehicle Link - Protocol for communicating with small unmanned vehicles

3 Description of the JETI DC-24 transmitter

The JETI DC-24 is a high-end RC transmitter that is designed and manufactured by JETI model in Czech Republic. It aims to achieve maximum durability and reliability of its mechanical parts, easy servicing and a long lifespan. Its superior construction quality and features easily place it among the top-class of RC transmitters and at the time of writing this work, the DC-24 was the flagship transmitter of JETI model [5].

As a newer generation to the DC/DS-14/16 transmitters, the DC-24 introduced new features such as a 900 MHz backup antenna, 24 channels, MP3 player and FM tuner, or haptic feedback on gimbals. When paired with one of the JETI receivers, it can be used for precise flight control of a RC model of any construction type. The transmitter can store multiple user-defined models of the following four construction types: Fixed-wing, helicopter, multirotor and generic.



Figure 5: JETI DC-24 transmitter

3.1 Technical specifications

The transmitter is a tray-type RC controller with a 2.4 GHz main antenna and a 900 MHz backup antenna. It has three connectors: a charging connector, 3.5 mm audio jack and a USB B mini port for connecting to a PC, which is also used for uploading new programs into the transmitter. A colour 3.5" TFT LCD display with a resolution of 320 x 240 pixels and with easy readability in any light conditions provides an intuitive way to interact with the transmitter [5].

| | |
|-------------------------|-------------------|
| Weight | 1500 g |
| Dimensions | 230 × 270 × 40 mm |
| Output power - 2.4 GHz | 100 mW |
| Output power - 900 MHz | 25 mW |
| Number of channels | 24 |
| Number of controls | 20 |
| Resolution | 4096 steps |
| Operational temperature | −10 to 60 °C |
| Operating time | up to 12 h |

Table 1: Technical specifications of the DC-24

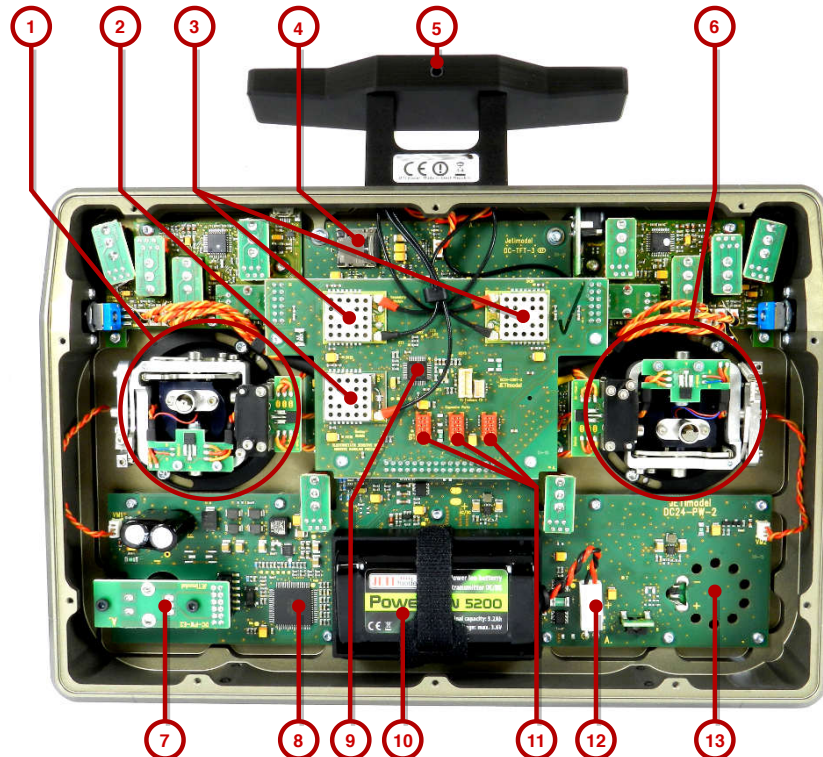


Figure 6: Internal configuration of the DC-24

- 1, 6 - Precise gimbals** Two gimbals with hall sensors and ball bearings to ensure precision movement and an almost unlimited lifespan. Each gimbal has a programmable haptic feedback.
- 2 - Duplex 900 MHz** A backup wireless system. The DC-24 is the first dual-band RC system.
- 3 - Duplex 2.4 GHz** A reliable, frequency hopping, digital data stream system.
- 4 - Memory card slot** Internal micro SDHC card slot for storing user data (up to 32GB).
- 5 - FM antenna, PMM Input/Output** FM tuner allows listening to FM radio stations.
- 7 - 3D button / rotary encoder**
- 8 - Power management CPU**
- 9 - Wireless coprocessor ARM Cortex M4**
- 10 - Li-Ion battery** Primary energy source of the transmitter with 5200 mAh capacity.
- 11 - Extension ports** Will be used to connect the ESP32 board.
- 12 - Battery connector**
- 13 - Microphone** For recording sounds and teaching the transmitter how to respond to voice commands through voice recognition.

3.2 Computational capabilities

The main CPU of the transmitter is a ST Microelectronics microcontroller, chiefly a 32bit processor STM32F439 with an ARM Cortex-M4 core running at 168 MHz. It was chosen due to its great efficiency at achieving a solid performance, while at the same time reaching a very low power consumption. The processor offers slightly better computational capabilities than the STM32F405RGT6 processor, which was used as a main processor for the DC-16 [6].

The transmitter runs at the following internal configuration:

| | |
|------------------------|--|
| MCU | STM32F439 @ 168 MHz |
| Internal memory | 2MB Flash, 256kB SRAM |
| External memory | 8MB (1MB reserved for framebuffer and system resources) |
| SD card support | Up to 32GB micro SDHC |
| Audio playback | MP3 (44.1kHz, 32kHz), WAV (8kHz, 11kHz, 16kHz, 22.05kHz, 32kHz, 44.1kHz; Mono / Stereo) |

The transmitter is highly programmable through third party applications written in Lua. Up to 10 user applications can run simultaneously on the transmitter and each application can register two telemetry entries. If any of them stops executing due to an error, it is killed individually so that other applications are unaffected by it.

Lua applications are placed on the internal SD card, into the /Apps folder. The transmitter loads all applications inside this directory automatically during startup. Each application is identified based on the application filename, from which a unique 32-bit identifier is created so it can be easily referenced by the system.

3.3 Operating system

Paramount functionality of the DC-24 operating system is ensuring a reliable control of a RC model. On top of that, it offers a plethora of features that add to the user's comfort such as programmable alarms, voice output or realtime telemetry display on the main screen.



Figure 7: Main screen

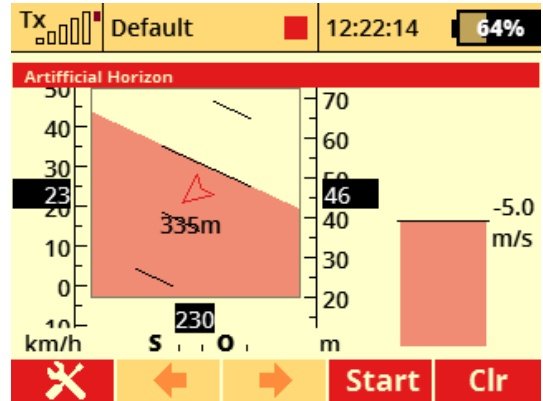


Figure 8: Artificial horizon app [7]

User interface on the LCD display

The UI can be personalized with colour themes. User applications, such as the Artificial Horizon [7] (figure 8), can be displayed on the main screen to be readily accessible during flight. The main screen supports multiple pages for quickly switching between different full-screen applications, or stacking several application or system widgets into a single screen (figure 7).

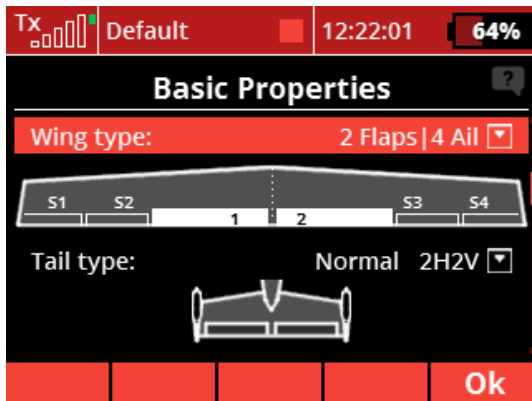


Figure 9: Model setup

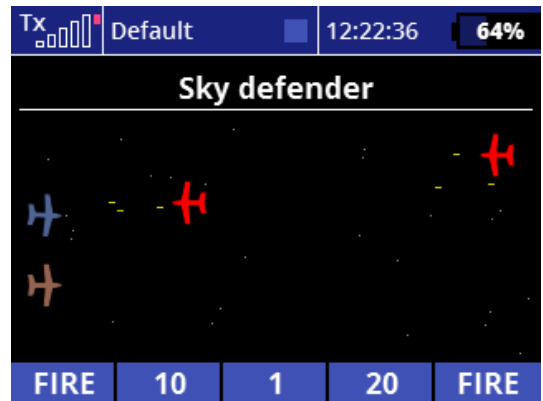


Figure 10: Cooperative multiplayer game

Multiple user profiles for RC models can be defined through an intuitive menu where the user can select the model construction type and define its properties, such as control surfaces of the aircraft (figure 9). Some extra functionalities include a FM radio or audio player for playing music in the background, or games such as Chess or Snake, or other user-made games utilizing realtime 2D drawing (figure 10).

The system acts as a real-time application that is extended of user interface [6]. Therefore, it works as a real time operating system [8]. GUI processing, RC control and audio playback is split into separate threads. All Lua applications run on a GUI thread, therefore a synchronization of all tasks is automatically ensured. The only case where synchronization fail possesses any risk is in reentrant functions. Lua memory allocator is independent of a system memory allocator and is stored in an external SDRAM. The rest of the system uses an internal SRAM.

3.4 Lua programming API for DC/DS-24

Access to system functions of the transmitter is possible through an API, developed specially for the DC/DS-24 transmitter line, with a documentation that can be freely downloaded from the JETI model website. It is constantly updated and extended as new modules and functionalities are added to the transmitter. For example, the addition of the ESP32 module required a new GPIO and serial library to be created.

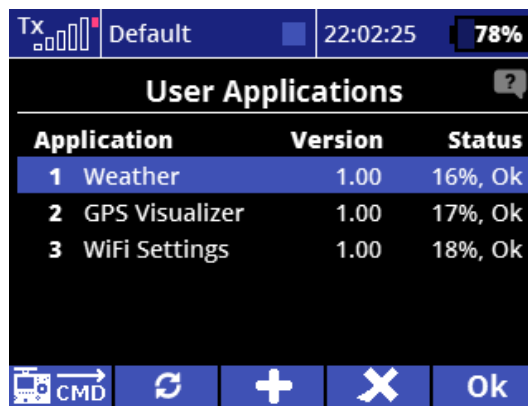


Figure 11: User applications menu

Every application consists of a single Lua script that returns an array describing the application interface. The interface provides callbacks to functions defined in the application that are called when the application is initialized, destroyed (unloaded), and a loop function that is called on every tick. It can look the following way:

```
return {init = {init, destroy = closeApp, loop = loop, author = "JETI model",
version = "1.00", name = lang.appName}}
```

Overview of the installed and active applications can be seen in the Applications menu (figure 11) which shows application name, version, maximum CPU utilization and status. Applications can be installed either from a .lua file or a compiled Lua .lc file, saved in the /Apps directory.

Functions of the API can be accessed through 6 libraries: system, LCD, GPS, form, GPIO and serial library. Below is a brief summary of each library and a showcase of its functions:

3.4.1 System library

Used for retrieving system information such as the current timestamp, CPU utilization, current locale, information about sensors and hardware input; as well as registering application forms, accessing persistent application parameters, controlling media playback and system properties.

3.4.2 LCD library

Encompasses all rendering functions. Used for drawing shapes, displaying text and images on the LCD screen, reading the background colour based on the current colour theme or creating a rendering context. Has an antialiased renderer for drawing graphics using a framebuffer represented by an ImageData object.

3.4.3 GPS library

Created to easily control a GPS module through a precise GpsPoint structure. Contains functions for getting current position of the model, creating GpsPoints from given coordinates, retrieving latitude and longitude from a GpsPoint, converting LCD to GPS coordinates and vice versa.

3.4.4 Form library

Contains functions for creating various UI elements in the application forms, changing their properties, raising question dialogs or configuring buttons on the screen, aswell as a function for reinitializing or closing a form, or for preventing default system behaviour after pressing a key.

3.4.5 GPIO library

Used for communicating with GPIO pins. Has three functions for setting a GPIO pin mode, reading from and writing to a GPIO pin. Internal GPIO pin indexes have a range of 0-8 and can be configured as a digital input, or a digital output with push-pull or open-drain behaviour.

3.4.6 Serial library

Provides access to serial port communication. Defines a function to initialize or deinitialize a serial port, setting a read callback, or listing available ports. It is also used in this work for sending commands to the ESP32 module through a `serial.write(portID, data)` command.

4 Description of the ESP32 network module

To attain network functionality, the transmitter is extended with an ESP32 network module. Created by Espressif Systems, it is a low-cost, low-power system on a chip that was developed as a successor to the older ESP8266 microcontroller. At its heart resides a dual-core or single-core variation of a Tensilica Xtensa LX6 microprocessor, running at up to 240 MHz.

For this work, a clock rate of 160 MHz was chosen, as it is perfectly sufficient to fulfil its tasks. An ESP32 was preferable over the older ESP8266 due to its Bluetooth capabilities and better processing power, with the ability to run a web server on the chip without it slowing down the main CPU of the transmitter.

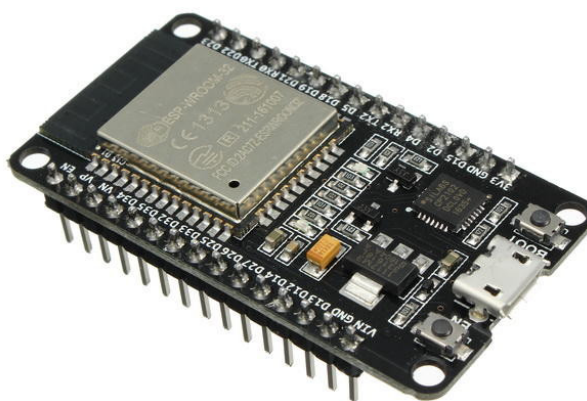


Figure 12: ESP32 development board

The ESP32 can have a wide variety of 3rd party firmwares installed on it, such as the official ESP-IDF² which is programmed in C and includes FreeRTOS³, NodeMCU which is an open source Lua firmware layered on top of the ESP-IDF, MicroPython for programming the module in the Python language, or the AT firmware which uses AT commands for communicating with the module.

Notable technical parameters of the ESP32 module [10]:

- Processors:
 - CPU: Xtensa 32-bit LX6 microprocessor, operating at up to 240 MHz
 - Ultra low power co-processor
- Memory: 520 KiB SRAM
- Wireless connectivity:
 - WiFi: 802.11 b/g/n
 - Bluetooth: v4.2 BR/EDR and BLE

²Espressif IoT Development Framework.

³RTOS - Real Time Operating System [9]

4.1 ESP32 board in the RC transmitter

The ESP32 board is connected to extension ports on the rear side of the transmitter, marked as number 11 on figure 6. The module requires up to 500mA and has two pins connected to unstabilized battery voltage. Voltage is stabilized directly on the board. See attachments for a scheme of the board.

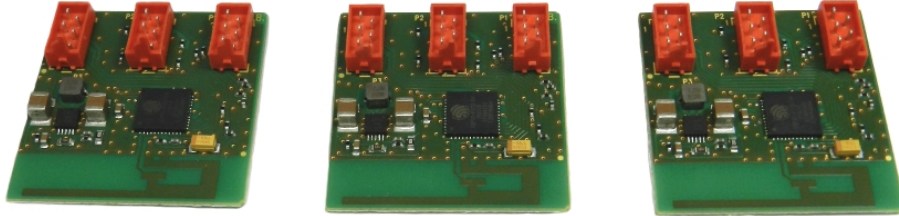


Figure 13: Final board with the ESP32 module, to be serially produced for the DC-24

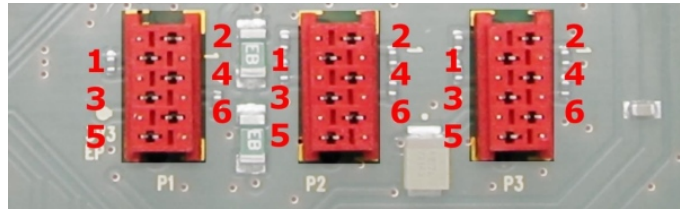


Figure 14: Internal GPIO pins on the rear port for the expansion board

| | | |
|------------------------|----------|--------------------|
| P1-1 GND | P2-1 IO0 | P3-1 IO5 |
| P1-2 VBAT (400 mA) | P2-2 IO1 | P3-2 IO6 |
| P1-3 Serial "COM1" RX | P2-3 IO2 | P3-3 IO7 |
| P1-4 Serial "COM1" TX | P2-4 IO3 | P3-4 IO8 |
| P1-5 GND | P2-5 GND | P3-5 GND |
| P1-6 3V3 (max. 200 mA) | P2-6 IO4 | P3-6 VBAT (200 mA) |

Table 2: Expansion board peripherals

5 Design of the SW components and the used ESP32 firmware

The resulting software implementation can be divided into the following core parts: user application, communication interface, AT firmware and a web page. The diagram below shows all four parts and their relations, grouped by the platform that handles their workload and where are they stored.

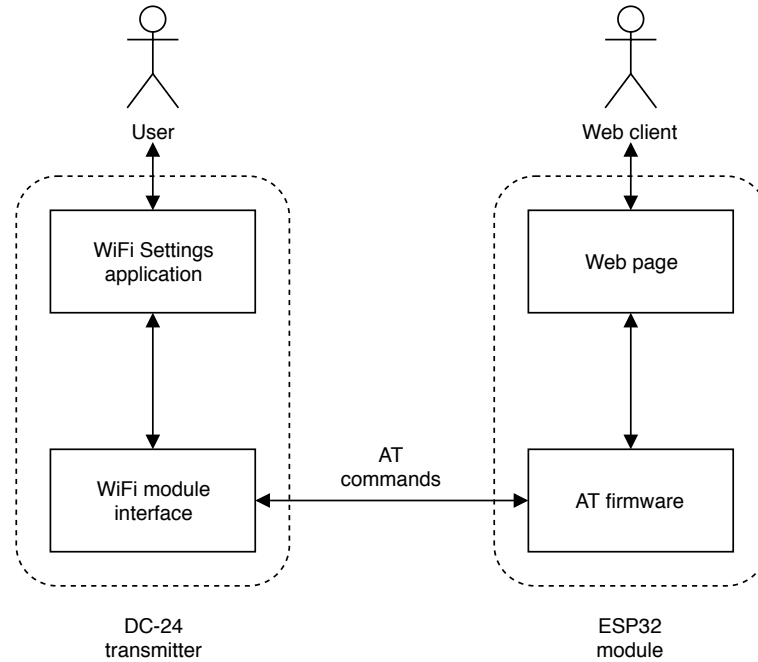


Figure 15: Layout of the SW components

WiFi Settings application Will run continuously in the background, serving as the primary point of interaction with the user. Written in Lua.

WiFi module interface Designed to simplify communication between the application and the module, and to allow other applications to communicate with it as well.

AT firmware The selected firmware to be compiled for the ESP32 module, with customized parameters.

Web page Real-time telemetry output from the transmitter that can be comfortably displayed in a web client from any machine connected to the module.

5.1 Lua programming language

Lua is an efficient, lightweight, simple, yet powerful multi-paradigm scripting language. Unlike many programming languages that help the user write programs with hundreds of thousands of lines, Lua aims to do the same job with only hundreds of lines, if not even less. Due to its small interpreter (taking only 247K when built on Linux), a general ease of use and great performance, it is ideal for usage in embedded applications [11].

The language is typed dynamically and runs by interpreting bytecode through a register-based virtual machine. It also provides automatic memory management and garbage collection, making Lua an excellent choice for prototyping, configuration and scripting.

Lua has found extensive use not only in embedded systems, but also industrial applications and also computer games (such as World of Warcraft or Half-Life), where it is currently the most widely used scripting language. Lua can also be easily integrated and extended with software written in C and C++, as it purposely comes with a very compact default library (taking only 421K).

5.1.1 History

The Lua language was created in 1993 by Roberto Ierusalimschy and his college colleagues Luiz Henrique de Figueiredo and Waldemar Celes, all members of Tecgraf⁴ at the Pontifical Catholic University in Rio de Janeiro, Brazil. Due to a strong policy of trade barriers in Brazil which ensued from 1977 to 1992, it was very difficult for Tecgraf's clients to buy customized software from abroad, which led Tecgraf to implement the basic tools it needed from scratch. Lua 1.0, although never released publicly, has been up and running on 28 July 1993, if not a couple of months sooner. The first public release of Lua was 1.1, which was released on 8 July 1994.

5.1.2 Lua in the DC-24 transmitter

Currently used in JETI DC/DC-24 transmitters is Lua 5.3.1. It is compiled with parameters **LUA_32BITS** which tells the Lua interpreter that the size of integer and floating point numbers is always 32 bits, and **LUA_FLOORN2I** which makes floating point numbers always floored to the nearest integer if the called function requires an integer data type. It has no compatibility with older Lua versions (5.1 and 5.2). The interpreter can also benefit from hardware FPU support.



Figure 16: Lua logo

⁴Computer Graphics Technology Group.

5.2 AT firmware

In order to communicate with the ESP32 module and efficiently write the communication interface, a firmware had to be chosen and flashed to the module. After doing an intensive research, NodeMCU [12] seemed like the most convenient choice due to its easy integration with Lua and good support, but at the time of writing this work it still lacks Bluetooth functionalities which are planned for future expansions of the module. Due to this limitation, I decided to use the AT command set [13], which fulfils all the needs for the project.

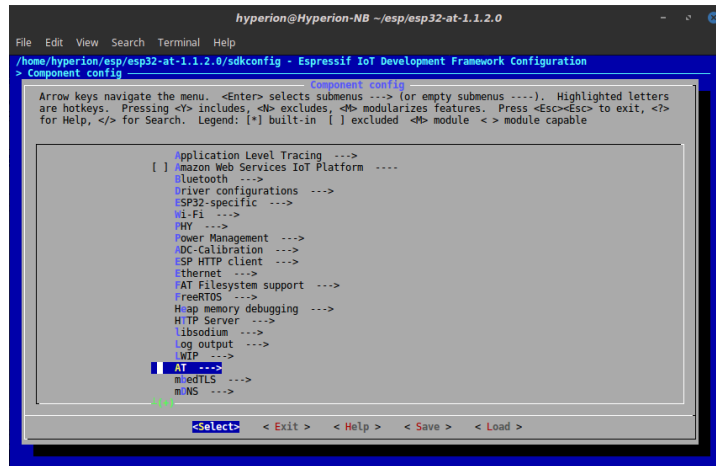


Figure 17: Command line interface for configuring the AT firmware

AT (meaning 'Attention') commands come from the Hayes command set, which was originally developed for a baud modem in 1981, and the syntax is still widely used today. The firmware is developed by Espressif systems and can be widely configured through a command line interface menu, as seen on figure 17.

```
AT+CWJAP?
+CWJAP:"Stargate Command","28:ff:3e:0a:04:0e",1,-37
OK

AT+GMR
AT version:1.3.0.0-dev(2759564 - Nov 12 2018 03:26:27)
SDK version:
compile time:Mar 13 2019 15:04:54
OK

AT+CWMODE=3
OK
```

Listing 1: AT commands communication example

Communication with the module through AT commands is based on sending commands to the module and processing its replies. For example, the `AT+CWJAP?` command asks for which network the module is currently connected to, as can be seen in the AT communication example above. After the sent command is successfully interpreted, the module sends a message informing of success or failure, usually a simple **'OK'** or **'ERROR'** respectively. No new commands can be sent while the module is still processing an older command, and if it happens, a **'busy'** message is returned.

For this project, the release version 1.1.2.0 of the firmware was finally used and compiled through an official ESP toolchain, on a Linux machine. I tested a newer version 1.1.3.0 as well, but I could not establish a stable communication with it. I configured the firmware through menuconfig (figure 17) before compilation, where I changed the baud rate to 921600 and UART Rx and Tx pins⁵ to match the ones connected on the ESP32 board.

5.3 Communication interface

For making user written applications in the transmitter less dependent on the ESP32 firmware used, and to make a possible future replacement for the AT firmware easier, all communications with the transmitter and the module is put into a separate .lua file. This file then acts as an interface and it is necessary to include it in every application that is intending to use any functionalities of the WiFi module. The implementation of the interface is completely encapsulated through local variables, for minimizing the risk of configuring the module incorrectly, freezing all its communications, or making it stuck during the execution of a command.

The aim of the interface for the future is to make the transition to another firmware as smooth as possible, and to strip user applications of any need to communicate with the module directly through serial port commands. In the ideal case, no changes to transmitter applications using the WiFi module will be necessary when such a transition occurs, with only the interface being modified.

⁵Rx - Receiver pin, Tx - Transmitter pin.

6 Implementation and testing of the proposed design

I started implementing the design by first creating the WiFi Settings application and later moving a part of it into the communication interface. I also created another application during my work on this project, for testing IP communication and gathering data from the internet in order to display an accurate weather forecast.

6.1 WiFi Settings app

To control the module from the transmitter, I created a WiFi Settings app. The sole purpose of this relatively simple application is to connect to a WiFi network and configure the WiFi module from the transmitter, similarly to a WiFi settings menu that can be found on an Android phone.

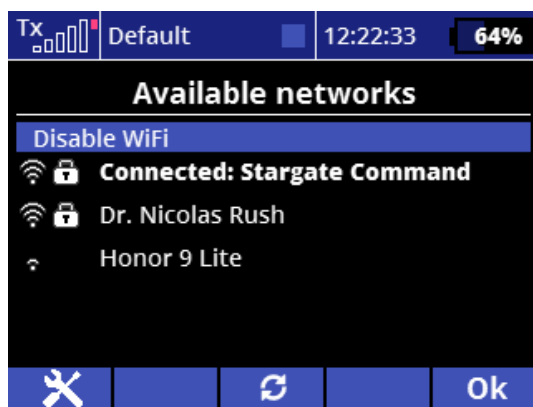


Figure 18: List of networks

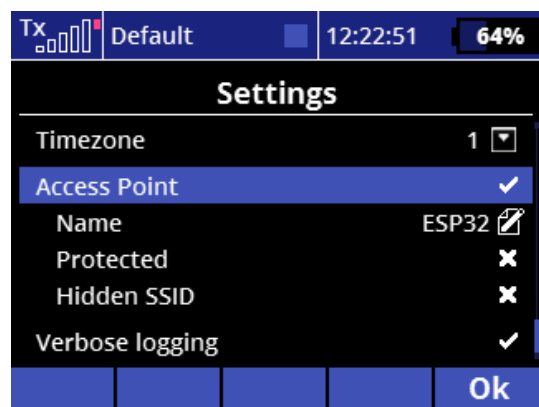


Figure 19: WiFi settings

Screenshots of the WiFi settings app

The interface of the application is very intuitive and easy to understand, and gets the user connected to a network with the least amount of effort. To conserve battery and decrease interference with the transmitter radio antenna, the module's WiFi RF power can be disabled altogether. Access Point can be enabled and configured from the Settings menu (figure 19), where it is possible to configure the time zone as well, which for example affects the calculations of sunrise and sunset times in the Weather app. If the WiFi Settings application is being run through an emulator, a port selector is available to make development and testing for multiple modules easier.

Since only one application is allowed to communicate on a UART port due to safety limitations hardcoded in the transmitter, IP connection requests cannot be handled through any other user made application. To overcome this limitation, I wrote the download request function of the interface in such a way, that each call of the function inserts the request as a new entry into a buffer which is continuously checked through a loop function of the WiFi Settings app. When a request is found in the buffer and the module is not busy, a download is processed. After the download is finished, it returns the result to the application through a callback.

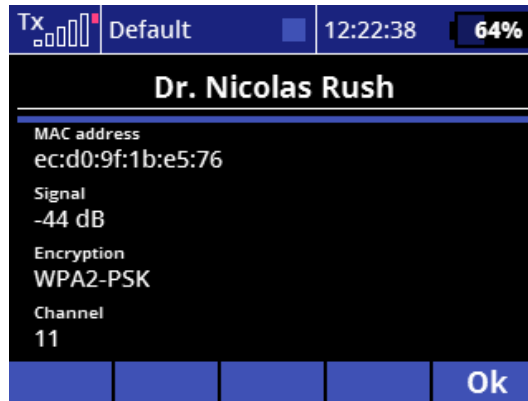


Figure 20: Network information

The interface consists of three main screens. The primary screen (figure 18) shows a scrollable list of networks once the module has finished scanning, and both other screens can be accessed from it through buttons under the LCD screen. If WiFi RF is disabled, only an Enable WiFi button is displayed. The Settings screen (as seen on figure 19) displays a scrollable list of the application and module settings. Password input for the AP is checked for invalid entries before being sent to the module, for containing at least 8 characters, as required by the WPA2 encryption which is used if the AP is protected.

Lastly, an Info screen (see figure 20) can be accessed through the available WiFi networks list, displaying additional information about networks such as its MAC address, signal strength in dB, and the type of password encryption used.

6.2 WiFi module interface

The interface is designed to handle all communication between the transmitter and the module through simple serial port communication commands that were newly added to the DC-24 Lua API. It is saved in a separate file in the WiFi Settings under the name **wifimodule.lua** and must be included in any user application that uses the WiFi module functionalities. In Lua, files are included through the `require` command and are usually stored in a table which acts as their namespace. In the main WiFi Settings application, the interface is included in the very first line of the application code in the following way:

```
local Wifi = require("WiFi/wifimodule")
```

| Method | Description | Used AT commands |
|----------------------|--|------------------------|
| isBusy | Returns true/false whether the module is busy or not | |
| isEnabled | Returns true/false whether the module is enabled or not | |
| isConnected | Returns true/false whether connected to a network or not | |
| getNetworkCount | Returns the count of found WiFi networks | |
| getNetwork(index) | Returns information of a specified network | |
| getNetworks | Returns a table of networks | |
| getNetworkId | Returns ID of the current network | |
| getState | Returns current state of the module | |
| getSignal | Returns a signal strength of the current network | |
| getConfig(param,def) | Returns a config value | |
| setConfig(param,val) | Sets a config value | |
| saveConfig | Saves the config to a JSON file | |
| enable | Enables WiFi RF | AT+CWMODE AT+CWJAP |
| disable | Disables WiFi RF | AT+CWMODE |
| joinAP(ssid,mac,pwd) | Joins an AP | AT+CWJAP |
| quitAP | Quits the current AP | AT+CWQAP |
| scanNetworks | Runs a scan for WiFi networks | AT+CWLAP |
| updateConfig(config) | Updates time zone or AP config | AT+CIPSNTPCFG AT+CWSAP |
| download | Adds a URL to download queue | AT+CIPSTART AT+CIPSEND |

Table 3: Public methods of the interface

6.2.1 Initialization sequence

Upon starting the transmitter or restarting the WiFi Settings application, an initialization sequence of AT commands is run on the module. The sequence sets the module to a default configuration as defined in the application and configuration file. When the application is restarted, the module does not need to be restarted and reinitialized, which will keep it connected to a WiFi network if it was connected to any before the restart. To as a workaround, the first command that is sent to the module upon restart is `AT+CWJAP?` which checks whether the module is connected to a network or not. If it is connected, the module returns the network's SSID, MAC address and encryption. If not, only an empty string is returned.

If the module is connected to a network, the current network variables are updated from the module's response and the initialization sequence is skipped. If an empty string is returned, a sequence timeout occurs or the transmitter is restarted, the module is forcibly restarted and reinitialized to make sure it has a current configuration. After the initialization sequence finishes, the module will automatically scan for networks through the `AT+CWLAP` command. This scan occurs even if the module is already connected to a network.

The module must be initialized with a given baud rate. For the purposes of this project, a baud rate 921600 was chosen. To initialize the module, a `serial.init(port, 921600)` command is used. If the initialization fails for whatever reason, the module is automatically deinitialized with a `serial.deinit(port)` command.

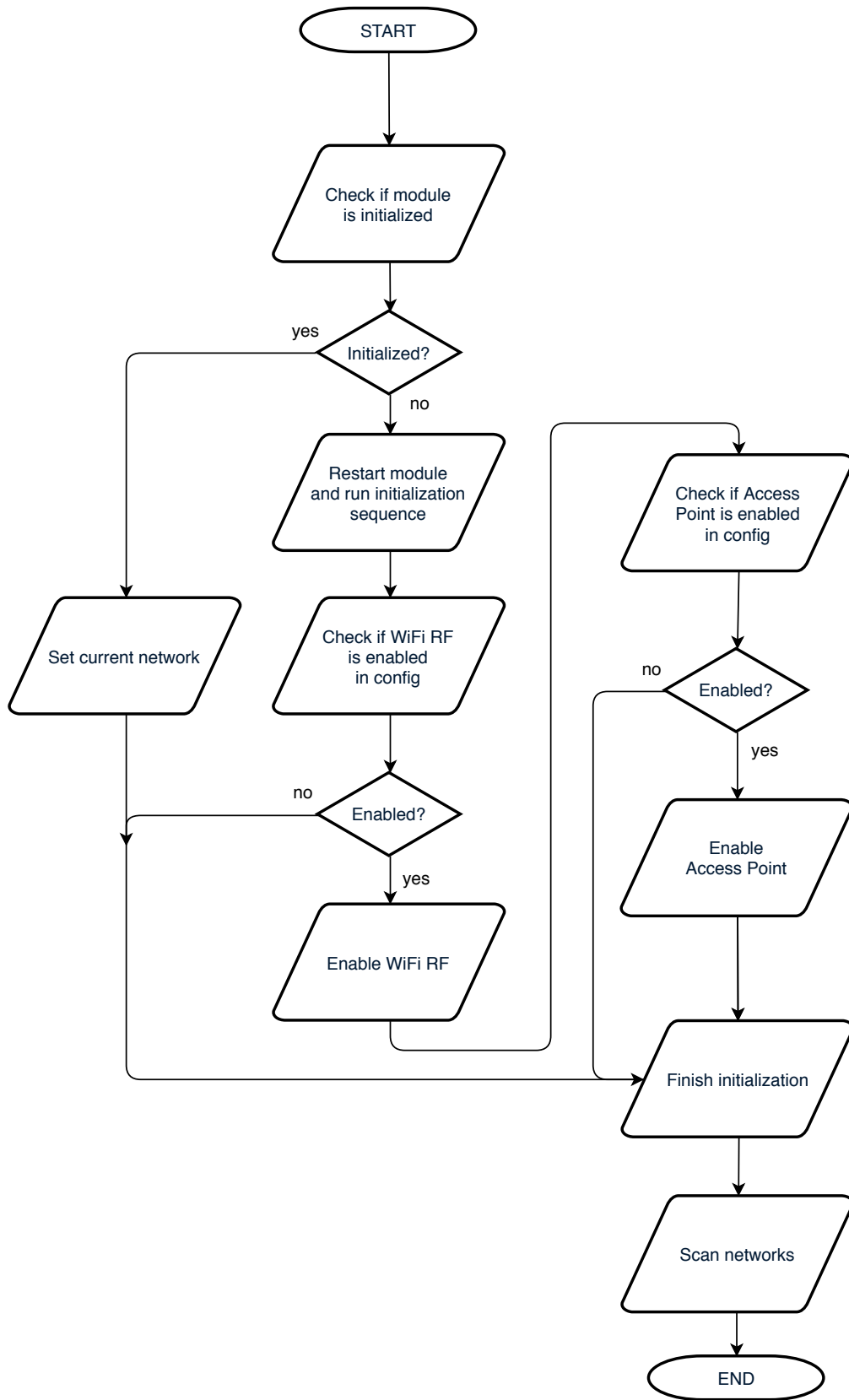


Figure 21: Initialization sequence

6.2.2 Processing a download request

In order to receive data from the server the interface uses a HTTP GET request, for which a sequence of three commands is needed. The first command, `AT+CIPSTART`, establishes a TCP connection with the host on a given socket. The next command, `AT+CIPSEND` tells the module to start receiving data of a given length. Lastly, after building the GET request for which only the **HOST** header is needed in this case, the module starts sending back data. All the received data are put together into a string and if a `1,CLOSED` string is detected, the sequence finishes and a callback is called.

```
local function httpGet(conn)
    print("Downloading data from " .. conn.host .. conn.url)
    httpGetCallback = conn.callback
    busy = true
    payloadPart = 0
    downloadedData = ""

    local seq = {
        {cmd = string.format('AT+CIPSTART=1,"TCP", "%s", 80', conn.host)},
        {cmd = string.format("AT+CIPSEND=1,%d",
            conn.host:len() + conn.url:len() + 25)},
        {cmd = function(line)
            serial.write(
                portId,
                string.format('GET %s HTTP/1.0\r\nHOST: %s\r\n\r\n',
                    conn.url, conn.host)
            )
        end, timeout = 20000, success = "1,CLOSED"}
    }
    seq.onFinish = function()
        busy = false
        httpGetCallback(downloadedData)
    end
    seq.onError = function()
        httpGetCallback(nil, true)
    end
    sequences[SEQ_GET] = seq
    sequenceId, sequencePos = processSequence("", SEQ_GET, 0)
end
```

Listing 2: Function for downloading online data through a HTTP GET request

6.2.3 Configuring an Access Point

Configuring and enabling the Access Point can be done easily from the WiFi Settings app, in the Settings menu (see figure 19). For the module to host the AP, it must first be set into a **'SoftAP'** or a **'SoftAP + Station'** mode, which enables it to host an AP as well as allowing it to connect to a WiFi network, which enables user applications to download data from the internet. This is achieved by sending the command `AT+CWMODE=3` to the module, which recognizes the following modes:

- 0 WiFi RF disabled
- 1 Station mode
- 2 SoftAP mode
- 3 SoftAP + Station mode

Afterwards, the Access Point can be configured with the following command:

```
AT+CWSAP=<ssid>,<pwd>,<channel>,<encryption>[,<max conn>][, <ssid hidden>]
```

The encryption methods supported are WPA-PSK, WPA2-PSK and WPA / WPA2-PSK and the number of maximum connections can be optionally set within the range of 1 to 10 connections. SSID is not hidden by default but can be hidden by setting the last parameter to 1. Parameters of the AP can be controlled from the Settings menu of the WiFi Settings app, as shown on figure 19.

If the values entered are valid and the **Enabled** setting is checked, the persistent configuration file is updated and a sequence of commands with the stored parameters is started upon exiting the menu. If it finishes without an error, the Access Point is automatically started and can be connected to from any suitable device, such as a PC, phone or a tablet.

6.3 Weather app

To test and showcase simple IP communication functionalities of the module, I have developed a weather telling app. In order to download weather data, the receiver needs to be connected to a WiFi hotspot with internet access. If the conditions are met and the module is not busy, the refresh button becomes enabled and pressing it starts a sequence of commands.

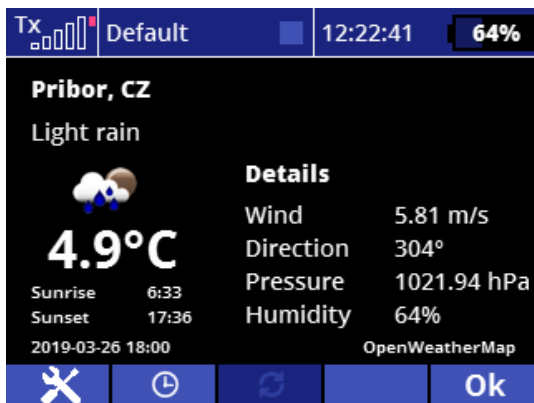


Figure 22: Weather screen

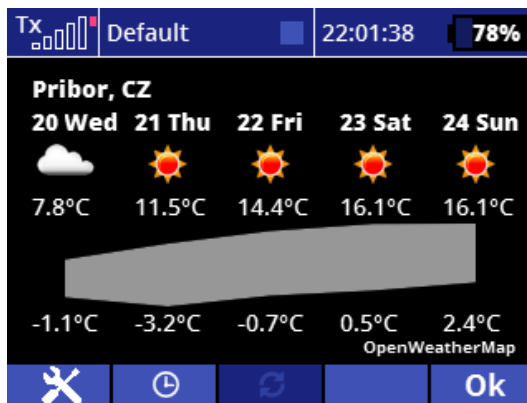


Figure 23: Forecast screen

Main screens of the Weather app

The first screen (as shown on figure 22) shows the current weather, which is calculated by comparing the current timestamp and timestamps in the downloaded forecast file and selecting the nearest one. Pressing the second under-screen button switches to a forecast screen (figure 23), which displays weather for the following four days, together with a minimal/maximal temperatures graph. The settings screen is for configuring the location by inputting the desired latitude and longitude, and changing the temperature unit.

As a weather data provider I decided to use openweathermap.org API through which I can receive weather forecast in a JSON file for a specific location, defined either by city ID or by a geographic coordinate system. In order to retrieve data from the server, a download function of the interface is called in the following way:

```
Wifi.download("api.openweathermap.org", string.format(
"/data/2.5/forecast?lat=%.3f&lon=%.3f&appid=45cae51742b71e4d5e0aa8fa2c3831bd",
config.latitude, config.longitude), downloaded)
```

The command takes a host part "api.openweathermap.org" followed by a formatted URL part and lastly the callback `downloaded`. After the download finishes, the callback function is called which refreshes the data on the screen and stores the received data in a more compact JSON file, which contains only the information that is used in the application.

7 Web server with realtime telemetry display

The web server runs on the ESP32 module, with client requests being processed automatically by the WiFi interface. In this work, the main purpose of the server is to display useful telemetry from the module in realtime, on any device connected to the transmitter-hosted AP. Nonetheless, the user can host any custom website, by replacing the default web page files inside the **WiFi/website** directory.



Figure 24: Web page in a desktop browser

7.1 Enabling and connecting to the web server

To enable the web server, an Access Point must be running on the module (see chapter 6.2.3). The IP of the server is 192.168.4.1 by default and can be retrieved at any time by running the **AT+CIFSR** command and reading the **APIP** value from it. It is run automatically during the initialization sequence and its output can be seen in the application console. The output of the command looks as follows:

```
+CIFSR:APIP,"192.168.4.1"  
+CIFSR:APMAC,"24:0a:c4:12:6b:ed"  
+CIFSR:STAIP,"0.0.0.0"  
+CIFSR:STAMAC,"24:0a:c4:12:6b:ec"
```

7.2 Processing client requests

When a client connected to the Access Point tries to connect to the server's IP address, a GET request message is automatically sent by the AT firmware to the application. When connecting from an Android device, the message can look as follows:

```
0,CONNECT
+IPD,0,169:GET / HTTP/1.1
User-Agent: Dalvik/2.1.0 (Linux; U; Android 7.1.2; Redmi 4X MIUI/V10. ...)
Host: 192.168.4.1
Connection: Keep-Alive
Accept-Encoding: gzip
OK
```

The first line `0,CONNECT` indicates that a connection with link ID 0 has been established. The number is retrieved and saved into a variable for distinguishing which ID to send the following response to, as multiple links can be connected to the AP at the same time. Each link ID presents a single file of a web page. The `+IPD,0,169:GET / HTTP/1.1` line is a standard IP packet message sent by the AT firmware, indicating the link ID, message length in chars, and HTTP version which is later used for constructing the response message as well.

After the message is processed, the request is saved into a queue. When the module is not currently processing any request, a next request is selected from the queue for which a response message is constructed. The `AT+CIPSEND,0,1057` command is then used to initiate a response message, with two arguments: link ID number and the data length of the message in bytes, with a maximum of 2048 bytes. If more bytes are needed for the response, a sequence of several `AT+CIPSEND` commands is constructed for sending the response spliced into several packets.

The response is constructed of a preamble and the web page content, read from a **index.html** file by default, unless a directory is specified. The preamble contains only the HTTP version, matching the GET request message of the client, after which a **200 OK** code is inserted, indicating a successful connection. After the specified number of bytes is written and the transmission was successful, the module returns a **SEND OK** message and the connection can be closed with `AT+CIPCLOSE=1`. If the request queue is empty, the response processing is finished.

7.3 Generating and displaying dynamic content on the web page

The web page has a simplistic layout with a single heading and a table of the received parameters below it, as seen on figure 24. When the **index.html** file has finished sending, two other files are automatically sent afterwards: a jQuery library⁶ and a **script.js** file. Once the web page can be safely manipulated, which is checked through a `$(document).ready()` block, a JSON file containing the desired telemetry data is requested through another GET request, which is then periodically requested every second.

The WiFi interface listens for such request and upon catching it generates a **data.json** file with the updated telemetry data (overwriting the old version if necessary). This file is then sent back and upon receiving it, the HTML table is filled with current data from the JSON file.

```
$(document).ready(function() {  
    var table = $("#telemetry > table");  
  
    $.refresh = function() {  
        $.getJSON('data.json', function(data) {  
            table.empty();  
            $.each(data, function(key, value) {  
                table.append('<tr><td>' + key + '</td><td>' + value + '</td></tr>');  
            });  
        });  
    }  
  
    $.refresh();  
  
    setInterval(timer, 1000);  
    function timer() {  
        $.refresh();  
    }  
});
```

Listing 3: jQuery script file

⁶Open-source JavaScript library designed to simplify HTML web page manipulation [14]

7.4 Web server limitations and responsiveness

The number of connections to the web server is limited only by the maximum number of connection IDs in the `AT+CIPSEND` command, which goes from 0 to 4. Therefore, not more than 5 connections can run simultaneously. Since sending each file of the web page costs one link ID, the design of the web page should be kept relatively simple and possibly without any unnecessary image files.

Once the web page has finished loading, only one connection is required to be free for the given client, for periodically sending the `data.json` file. With this limitation, only up to 5 clients shall be connected at once.

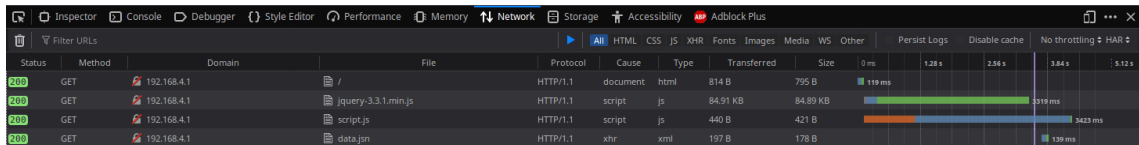


Figure 25: Network monitor

Another bottleneck of the web page is the rate at which the data transmission occurs. At a baud rate of 921600, it takes around 3 seconds to process the jQuery file (see figure 25), which possesses a bottleneck as it is by far the largest file of the web page (taking around 85 kB). The total loading time for the web page is only a little over 3 seconds.

8 Summary

The goal of this work was extending a RC transmitter with an ESP32 network module, creating a custom communication interface between the transmitter and the module, and hosting a web server on the module for displaying realtime telemetry data.

To gain some insight into other, possibly similar implementations, I first studied other radios in section 2 and compared them to the DC-24 platform, which I described in detail in section 3.

Another task was to get familiarized with the ESP32 network module, which is summarized in section 4. I have studied its capabilities and tested various firmwares, from which I selected the most appropriate firmware, which communicates using AT commands. I have described this firmware in section 5.

In order to communicate with the module through AT commands effectively I created an interface which safely takes care of all serial communication through USART port as well as processing AT command sequences.

For connecting the transmitter to a local WiFi network and configuring the Access Point I created an user application, and for testing IP connection and downloading data from the internet I wrote another user application, for downloading and displaying current weather forecast. Both applications and the interface is written in Lua and described thoroughly in section 6.

Lastly, I implemented a logic for automatically processing client requests received by the web server and generating current telemetry data from the transmitter in realtime, which is displayed on the web page. This is described and stress tested in section 7.

Thanks to this work, the JETI DC-24 RC transmitter can now connect to a local WiFi network, download data from the internet, provide an Access Point and host a website with dynamically refreshed telemetry data from the transmitter, that can be displayed in a computer browser or a handheld device.

In the future, the module will be used to process Bluetooth communication as well, which will be used for pairing wireless Bluetooth headphones with the radio (for playing music and voice alerts). My implementation can be further improved by expanding the telemetry web page, and optimized by offloading the main CPU of the transmitter and storing web pages directly on the ESP32 module instead. The module also provides the possibility of automatically downloading firmware updates over the internet, but that is beyond the scope of this thesis.

I will continue further perfecting and extending my work with new functionalities, which will be used in the JETI DC-24 transmitters to serve pilots all over the world.

References

1. *Graupner USA* [online]. 2019 [visited on 2019-03-30]. Available from: <https://www.graupnerusa.com/mz-32-32-CH-HoTT-Color-TFT-Radio-System-S1024.html>.
2. *Spektrum iX12* [online]. 2019 [visited on 2019-03-23]. Available from: <https://www.spektrumrc.com/Products/Default.aspx?ProdID=SPM12000>.
3. *JR Americas 28X* [online]. 2019 [visited on 2019-04-14]. Available from: <http://www.jramericas.com/28x/>.
4. *XLRS D3* [online]. 2019 [visited on 2019-04-14]. Available from: https://d3.xlrs.eu/en/xlrs_d3/.
5. *JETI model s.r.o.* [online]. 2019 [visited on 2019-03-20]. Available from: <http://www.jetimodel.com/en/>.
6. FALTIČKO, Martin. *Implementation of Software for RC Radio*. 2012-05. Master's thesis. Brno University of Technology.
7. *Official JETI model Git repository for Lua applications* [online]. 2018 [visited on 2019-03-22]. Available from: <https://github.com/JETImodel/Lua-Apps/>.
8. TANNENBAUM, Andrew S.; BOS, Herbert. *Modern Operating Systems*. Fourth edition. London: Pearson, 2015. ISBN 978-0133592184.
9. WANG, K.C. *Embedded and Real-Time Operating Systems*. NY: Springer, 2017. ISBN 978-3-319-51517-5.
10. DOGAN, Ibrahim; AHMET, Ibrahim. *The Official ESP32 Book*. Elektor International Media, 2017. ISBN 978-1-907920-63-9.
11. *Official website of the Lua language* [online]. 2018 [visited on 2019-04-14]. Available from: <https://www.lua.org/>.
12. *NodeMCU Documentation* [online]. 2019 [visited on 2019-03-28]. Available from: <https://nodemcu.readthedocs.io/en/master/>.
13. *Git repository of AT Command Core for ESP32 ESP-IDF* [online]. 2018 [visited on 2019-03-22]. Available from: <https://github.com/espressif/esp32-at/>.
14. *jQuery library website* [online]. 2019 [visited on 2019-04-15]. Available from: <http://jquery.com/>.